

Imperial College London
Imperial College Business School

Artificial Intelligence in Operations Management: AI-Driven Techniques for Decision Making

Hanzhao Wang

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy of Imperial College and
the Diploma of Imperial College, June 2025

Statement of Originality

I, Hanzhao Wang, declare that this thesis and the work reported herein was composed by and originated entirely from me, except where acknowledged. No part of this thesis has been submitted before for any degree or examination at this or any other university.

Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC). Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose. When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes. Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Abstract

Artificial intelligence (AI) has demonstrated substantial success across diverse fields, revolutionizing predictive and prescriptive analytics. Motivated by these advancements, this thesis explores the application of artificial intelligence within Operations Management (OM), specifically targeting challenges related to choice modeling, assortment optimization, and sequential decision-making problems. The thesis has three primary contributions. First, a neural network-based discrete choice model is introduced, tailored for single-choice scenarios, capable of accurately capturing complex customer behaviors without reliance on traditional model assumptions. This model can be used successfully for assortment optimization through integer programming methods. Second, the Transformer Choice Net is developed to handle single, sequential, and multiple-choice contexts within a unified transformer-based architecture, significantly enhancing prediction accuracy and scalability across diverse consumer decision settings. Third, an Operational Management Generative Pre-trained Transformer (OMGPT) is presented, adapting GPT-style transformer networks to general sequential decision-making problems, such as dynamic pricing, inventory management, and queuing control. OMGPT provides a robust, data-driven, model-free approach, demonstrating state-of-the-art performance and strong generalization capabilities across varying operational tasks. Collectively, this thesis underscores the transformative potential of artificial intelligence to enhance decision-making and analytics in OM, advancing both theoretical understanding and practical capabilities.

Acknowledgements

After nearly six years at Imperial College London, this place has become my second home. Buildings may change, and science will continue to advance, but my love for this institution and its people will always remain the same.

I am deeply grateful to my supervisor, Prof. Kalyan Talluri. He has not only guided me in conducting research but has also taught me what it means to be a great mentor—responsible, meticulous, and patient. His passion for learning is truly inspiring, and I will always appreciate the time and effort he dedicated to refining our papers, as well as the encouragement he gave me during difficult times. We have experienced both highs and lows, rejections and acceptances, relaxing holidays and intense workloads. Every shadow and every light has shaped these years into a beautiful night.

I would also like to express my gratitude to the faculty members of the Analytics and Operations group. Their dedication to organizing events has provided invaluable opportunities for us to learn about advanced research, enhance our presentation and interview skills, and grow as scholars. Their wisdom and encouragement have kept my passion alive, even when the future seemed uncertain. I am especially fortunate to have received guidance from Xiaocheng Li, who guides me in both research and life. He has shown me what it means to be not only an exceptional researcher but also a dedicated advisor and a loving husband. My sincere thanks also go to Guanting Chen for being a wonderful coauthor and for his invaluable support during my PhD.

I am thankful to my fellow PhD students in the Analytics and Operations group. We have walked this path together—past, present, and future. With you, I have learned, been encouraged, and shared countless moments of joy. We have exchanged both struggles and triumphs, reflected on our pasts, and dreamed of our futures. If my journey in research is reaching for the deepest sky of science, then you are the shining stars lighting my way.

Lastly, my deepest appreciation goes to my parents and my girlfriend, Jiahong Shi. Over the past ten years, I have moved and traveled across different countries, but my starting point and destination are always about you.

People come and go, new stories become old, but the echoes of this moment will be told...

Contents

Statement of Originality	3
Copyright Declaration	4
Abstract	5
Acknowledgements	6
1 Introduction	18
2 A Neural Network based Choice Model for Assortment Optimization	22
2.1 Introduction	22
2.2 Literature Review	25
2.2.1 Neural Network Models in Utility Modeling	25
2.2.2 Assortment Optimization	27
2.3 Problem Setup	28
2.3.1 Random Utility Models (RUM) and Beyond	29
2.4 Neural Choice Model	34

2.4.1	A Feature-free Neural Choice Model	34
2.4.2	Theoretical Analysis	37
2.4.3	Feature-based Neural Choice Models	40
2.4.4	Comparison on Real Datasets	43
2.5	Assortment Optimization for Neural Choice Model	46
2.5.1	Integer Programming Formulation of the Neural Choice-based Assortment Optimization	47
2.5.2	Numerical Study of Joint Estimation and Optimization	48
2.6	Extensions and Discussions	52
2.6.1	Assortment Distribution Shift in the Training Data	52
2.6.2	Depth and Width of the Neural Networks	52
2.6.3	Network Augmentation with Warm Start	53
2.6.4	Neural Choice Model as a Meta Choice Model	53
2.7	Conclusion and Future Directions	54
3	Transformer Choice Net: A Transformer Neural Network for Choice Prediction	57
3.1	Introduction	57
3.2	Literature Review	59
3.3	Problem Setup	61
3.3.1	Preliminaries on Choice Models	61
3.3.2	Learning Task	63

3.4	Transformer Choice Net	65
3.4.1	Attention Mechanism	65
3.4.2	Architecture of TCNet	66
3.4.3	Remarks on TCNet	68
3.5	Expressiveness of TCNet	70
3.6	Numerical Experiments	72
3.6.1	Numerical Performance on Real Datasets	72
3.6.2	Probing into the Representation of TCNet	75
3.6.3	Attention Visualization	75
3.7	Conclusion and Future Directions	78
4	Operational Management Generative Pre-trained Transformer	79
4.1	Introduction	79
4.2	Literature Review	81
4.2.1	Sequential Decision Making	81
4.2.2	Deep Learning in Operations Management	82
4.2.3	Transformer for Reinforcement Learning	83
4.3	Problem Setup	84
4.3.1	Sequential Decision Making	84
4.3.2	Examples	86
4.4	Supervised Learning for a (Good) Algorithm Using OMGPT	87
4.4.1	A (Good) Algorithm as a Sequence Function	87

4.4.2	Preliminaries on (Original) GPTs and OMGPT	90
4.4.3	Supervised Learning for OMGPT	92
4.4.4	Difference from Online/Reinforcement Learning	95
4.4.5	Discussions and Extensions	96
4.5	Understanding Pre-trained $\text{TF}_{\hat{\theta}}$ as an Algorithm	99
4.5.1	$\text{TF}_{\hat{\theta}}$ as a Bayes-optimal Decision Maker	99
4.5.2	Regret Analysis	101
4.6	Numerical Experiments	113
4.6.1	Performance Evaluation	113
4.6.2	Mechanisms Interpretation	125
4.7	Conclusion	131
5	Conclusion	132
	Bibliography	133
A	Appendix for Chapter 2	152
A.1	Appendix for Numerical Experiments without Features	152
A.1.1	Model Details for Table 2.4	152
A.1.2	Benchmark Algorithms	154
A.2	Appendix for Numerical Experiments with Features	155
A.2.1	More Details for Table 2.5	155
A.2.2	More Details on Real Data Experiments	157

A.2.3	Benchmark Algorithms	160
A.3	Proof for Subsection 2.4.2	162
A.4	Appendix for Assortment Optimization	168
A.4.1	Data Generation	168
A.4.2	Estimation Step	169
A.4.3	Optimization Formulation	169
A.5	Appendix for Section 2.6	171
A.5.1	Appendix for Subsection 2.6.1	171
A.5.2	Appendix for Subsection 2.6.2	173
A.5.3	Appendix for Subsection 2.6.3	175
A.5.4	Appendix for Subsection 2.6.4: Hotel Data Example	177
A.6	More Extensions	178
A.6.1	Assortment Effect Processed by Fully Connected Layers	178
A.6.2	Model Calibration	180
B	Appendix for Chapter 3	183
B.1	Proof for Theorem 3.2	183
B.2	Appendix for Section 3.6.1	185
B.2.1	Dataset	185
B.2.2	Models, Hyperparameters, and Training	187
B.2.3	Training Process and Hyperparameters Tuning	189
B.3	Appendix for Section 3.6.2	189

B.4	Appendix for Section 3.6.3	190
B.4.1	Experiment Details	190
B.4.2	More Visualization Examples	190
C	Appendix for Chapter 4	195
C.1	Model Architecture and Alg^*	195
C.1.1	Model Architecture for OMGPT	195
C.1.2	Difference from [LXP ⁺ 24]	197
C.1.3	More Details on Alg^*	198
C.2	More Details on Pre-training	199
C.2.1	Generalization Issue in Pre-training	199
C.2.2	Pre-training Implementation	205
C.2.3	More Experiments on Pre-training	207
C.3	Proofs for Section 4.5	208
C.3.1	Proofs for Subsection 4.5.1	208
C.3.2	Proof of Proposition 2	210
C.3.3	Proof of Proposition 3	212
C.3.4	Proof and Discussions of Theorem 4.2	214
C.4	Appendix for Numerical Experiments	229
C.4.1	More Numerical Experiments	229
C.4.2	Sequential Decision Making Problems and Environment Generation . . .	234
C.4.3	Benchmark Algorithms	239

List of Tables

2.1	Example illustrating the effects of the IIA property of MNL.	30
2.2	Example based on the behavioral experiment in [AJ08].	33
2.3	Example based on the behavioral experiment in [TK85].	33
2.4	Feature-free choice modeling performance.	36
2.5	Choice modeling performance with only product (static) features.	42
2.6	Description of flights and retailing data.	44
2.7	Performance on the airline data and retailing data.	44
2.8	Performance on the SwissMetro and Expedia.	45
2.9	The approximation ratios ($\times 100\%$) of heuristics for assortment optimization without capacity constraint.	51
2.10	The approximation ratios ($\times 100\%$) of heuristics for assortment optimization with capacity constraint.	51
3.1	Comparison on the model complexity.	68
3.2	Performances of TCN and benchmark models.	74
A.1	Extension of Table 2.5 with more architectures.	156

A.2	Features in SwissMetro and Expedia.	159
A.3	Performance on the SwissMetro and Expedia (extended from Table 2.8).	161
A.4	Out-of-sample CE loss for out-of-domain performance of Gated-Assort-Net.	172
A.5	Description of Hotel data ([BFG09]).	177
A.6	Performance on the Hotel data.	178
A.7	Calibration errors on SwissMetro and Expedia by ACE ($\times 10^{-2}$).	181
B.1	Summary statistics of datasets.	185

List of Figures

2.1	Neural choice models.	34
2.2	Feature encoder.	40
2.3	Choice networks with features.	41
2.4	Estimation error on various assortment size.	53
2.5	Validation losses using cold start and warm start.	54
3.1	Architecture of Transformer Choice Net (TCNet).	67
3.2	Probing the latent representation of TCN.	76
3.3	Visualizations of TCN model.	77
4.1	The pre-training phase of OMGPT.	94
4.2	The testing/application phase of the pre-trained OMGPT.	95
4.3	$\text{TF}_{\hat{\theta}}$ nearly matches the optimal decision function Alg^* in dynamic pricing tasks with concentrated deviations.	101
4.4	Performance of OMGPT and other benchmark models.	114
4.5	Performance of OMGPT under various structures.	116

4.6	Performance under horizon generalization in newsvendor problems with uncensored and censored demands.	117
4.7	Performance under different testing noise variances.	118
4.8	Performance under the different levels of out-of-domain shifts.	119
4.9	Performance under the different levels of in-domain shifts.	120
4.10	Performance in stationary environments and non-stationary environments. . . .	123
4.11	Performance in environments with linear demand and environments with square demand.	124
4.12	Out-of-sample demand prediction error comparison.	125
4.13	Visualization of output embedding vectors from different layers.	127
4.14	Probing result.	128
4.15	Results of manipulations.	131
A.1	Network structures and corresponding CE losses.	174
A.2	Effects of fully connected layers in SiwssMetro and Expedia.	178
A.3	Overconfidence of predicting Metro.	181
B.1	Probing the latent representation to identify drink from food.	189
B.2	More visualizations of TCN model (first example).	192
B.3	More visualizations of TCN model (first example).	193
C.1	The architecture of OMGPT.	195
C.2	Performances of $\text{TF}_{\hat{\theta}}$ and oracle posterior algorithms.	200
C.3	Visualizations on the training.	202

C.4	Impact of $\tilde{f}$ and κ	208
C.5	Examples to compare the actions from the transformer $\text{TF}_{\hat{\theta}}$ and the optimal decision function Alg^*	230
C.6	$\text{TF}_{\hat{\theta}}(H_t) - \text{Alg}^*(H_t)$ across different tasks with various numbers of possible environments.	231
C.7	The average out-of-sample regret on tasks with simpler environments.	232
C.8	The average out-of-sample regret on harder tasks.	232
C.9	Comparison of the average out-of-sample regret between using the Transformer and LSTM architectures in OMGPT.	234
C.10	Complete results corresponding to Figure 4.13.	245

Chapter 1

Introduction

Artificial intelligence, especially deep learning, has achieved remarkable success in many fields, including computer vision, natural language processing and large language models, recommendation systems, and even physics. It often outperforms traditional algorithms in performance. Neural networks in particular have been applied with great success to many prediction tasks in industry, owing to their robustness, minimal need for hand-crafted modeling, and standardized training procedures. This prowess of deep learning in other domains is motivating a surge of interest in leveraging similar techniques for complex decision-making problems in operations management (OM). Operations management typically relies on structured analytical models and assumptions, but the growing availability of rich data is creating opportunities for more data-driven decision-making. This highlights the potential of modern AI to enhance OM, needing novel architectures and methodologies tailored to the unique challenges of operational decisions. This thesis addresses that need by developing and applying deep learning models to core OM domains, demonstrating how such models can improve both predictive accuracy and decision quality in complex operational settings.

In Chapter 2, we develop a neural network-based discrete choice model designed for single-choice settings, where each customer selects at most one item from an offered set. It is motivated by the classic assortment optimization problem: a firm must decide which subset of products to offer in order to maximize revenue, which requires accurately predicting customers' choice

probabilities from those assortments. Traditional choice models (e.g. multinomial logit and its variants) are powerful but often require careful hand-specification of utility functions and struggle to capture certain complex behavioral patterns. Here we propose a flexible deep neural network architecture that learns choice behavior directly from data, without requiring a pre-specified utility structure. A key innovation is the integration of the assortment context into the model, enabling it to account for how the presence or absence of products influences choice probabilities. We further formulate an integer programming approach that uses the trained network’s predictions to find the revenue-maximizing assortment. This approach addresses two significant challenges: it incorporates assortment effects into a predictive model and ensures that the “black-box” network can be utilized within an optimization framework. The neural choice model is evaluated on both simulated data (with ground truths from a variety of choice models) and real-world case studies across industries such as retail, airlines, and hospitality. Notably, it achieves predictive accuracy and revenue performance on par with the best traditional choice models in most scenarios, even in cases where customer behavior deviates from common assumptions. These results demonstrate that a carefully designed neural network can serve as a universal choice model for assortment decisions, capturing complex consumer behavior while still allowing for tractable optimization of offerings. This work is detailed in [WCLT23].

In Chapter 3, we introduce the Transformer Choice Net, a novel neural architecture based on the Transformer (self-attention) mechanism [VSP⁺17]. This model is capable of handling single choices, sequential choices, and multiple choices within a single unified framework. Such unification is important because in modern applications, customers may engage in multi-item or sequential purchasing (for instance, adding multiple products to an online cart over time), yet existing choice models are often fragmented—different methods are used for one-shot choices, multiple simultaneous choices, or sequential decision processes. Extending classical discrete-choice models to multi-choice scenarios is notoriously difficult due to the combinatorial explosion in the number of possible item subsets and sequences. The Transformer Choice Net addresses this challenge by leveraging the sequence modeling strengths of Transformer networks: it can effectively capture complex interactions among chosen items and the influence of the offer set and past choices on future selections. In our architecture, each alternative (and the context

of the assortment) is encoded and processed through multi-head attention layers, allowing the model to “attend” to relevant relationships (such as complementary or substitutable items in a basket). This data-driven approach eliminates the need for problem-specific modeling for each choice scenario, providing a single model that can be applied to a wide range of choice prediction tasks. We empirically validate the Transformer Choice Net on a broad set of benchmark datasets spanning different industries and choice behaviors. The results show uniformly superior out-of-sample prediction accuracy compared to leading discrete-choice models in the literature. Notably, our Transformer-based model outperforms these benchmarks in predicting both individual item choice probabilities and entire consideration sets, all without requiring any customized tuning for each dataset or context. This contribution demonstrates the power of advanced deep learning architectures in OM: by adopting the Transformer—an architecture originally developed for natural language and now state-of-the-art in sequence modeling—we achieve a level of flexibility and predictive performance in choice modeling that was previously unattainable with traditional approaches. The Transformer Choice Net thus represents a significant step toward unifying multiple choice paradigms under one methodological umbrella, greatly simplifying the modeling process for complex customer choice phenomena. This work is detailed in [WLT23].

In Chapter 4, we study sequential decision-making problems in operations management, introducing a GPT-style Transformer model called OMGPT (Operational Management Generative Pre-trained Transformer). Many OM contexts—such as dynamic pricing, inventory control, resource allocation, and queueing management—can be formulated as sequential decisions made over time under uncertainty. Traditionally, these problems have been addressed with specialized algorithms (often rooted in dynamic programming or online learning theory) that rely on problem-specific assumptions and structures. In contrast, OMGPT offers a data-driven, model-free paradigm: we cast each operational decision task as a sequence modeling problem in which the goal is to predict the optimal action at each time step given the history of past observations, actions, and rewards. By doing so, we can train a single large-scale Transformer on sequences of decisions, in a manner analogous to how GPT models are trained on language sequences. The Transformer’s ability to capture long-term dependencies and complex patterns

makes it a natural fit for these decision sequences. Once trained, OMGPT can generate near-optimal decision policies for a variety of problems without being explicitly told the underlying dynamics of each environment. This approach marks a shift in how we tackle OM decision tasks: (i) OMGPT leverages knowledge from potentially vast amounts of off-line training data (including heterogeneous scenarios), akin to pre-training in large language models, and (ii) it does not assume any parametric form or simplification of the decision problem, instead learning the policy directly from data. These properties allow OMGPT to generalize across different environments and problem types, in contrast to classical methods that must be re-derived or retrained for each new setting. We evaluate OMGPT on several canonical sequential decision problems – for example, pricing under unknown demand, inventory ordering under uncertain demand, and queueing service rate control – by training the model on a diverse set of simulated task instances. Remarkably, the OMGPT model not only matches or exceeds the performance of state-of-the-art heuristics and reinforcement learning algorithms on each task, but also exhibits strong generalization, successfully adapting to new problem instances it was not specifically trained on. Furthermore, we provide a theoretical interpretation of OMGPT’s learning behavior through a Bayesian lens, showing that under certain conditions it approaches Bayes-optimal decisions and that the variability due to finite training data can act as an implicit exploration mechanism with performance guarantees. Both practical performance and theoretical insight suggest that GPT-style deep networks can serve as powerful universal decision engines for a broad array of operations management challenges.

Chapter 2

A Neural Network based Choice Model for Assortment Optimization

2.1 Introduction

What goes on in the consumer’s mind during the purchase process? How, and why, do they decide to purchase? What features of the product or environment encourage them to purchase? Given the heterogeneity and idiosyncratic thought processes of customers, these questions are unlikely to be answered definitely. Nevertheless these are fundamental questions of interest to marketers, behavioral psychologists, economists, and operations researchers. Hence, for operational and algorithmic purposes, a number of stylized or reduced-form models have been proposed in the literature to explain individual-level decision outcomes as well as collective aggregate purchase behavior.

Discrete-choice models are one such class of models, widely used in marketing, economics, transportation, and revenue management, especially suitable for situations where customers choose at most one of several alternatives. They are typically parsimonious models, based on micro-economic utility maximization principles in a stochastic framework, and expressive enough to incorporate product, customer, and environment features, and tractable enough for estimation and incorporation into optimization models. The classical Multinomial Logit choice

model (MNL) is an early and prominent example that, along with mixed-logit, nested-logit, and probit, has found widespread application in many fields, both in theoretical models as well as in empirical studies. New alternatives to MNL, that are more flexible or more expressive, such as Markov chain choice [BGG16] and Exponential [AS16], have been proposed recently.

One could potentially design more elaborate choice models, folding in many known aspects of the purchase behavior. However, this line of development has some natural limitations as, one, the models have to be specialized and tuned to a particular industry or even to a firm’s purchase funnel; two, would require developing and hand-coding its estimation which may be difficult both in theory and practice; and finally, using the model in an assortment optimization method may be computationally infeasible.

So, given the fundamental indeterminacy in how and why a customer chooses to purchase, our inability to observe all the data required by some of these models, and as the process itself is opaque and liable to change by industry and context, a reasonable research proposition is to search for a robust universal model that requires minimal tuning or expert modeling knowledge and works as well as behavioral models across all contexts and industries. Naturally, machine learning methods, specifically neural networks, come to mind. They have been applied with great success for many prediction tasks in industry precisely because they do not require fine-grained modeling specific to a situation and industry, are robust, and come with standardized training algorithms that work off-the-shelf. With enough data they also have proved to be very good performers in many areas, sometimes proving to be superior even to hand-crafted models.

In this paper, we develop a neural network based choice model suitable for assortment optimization. The firm has to offer an assortment of products, potentially customized to each individual customer, and the customer either chooses one of the offered products or decides not to purchase. This problem arises in a wide variety of industries where customers choose at most one item, such as hotel rooms, airline seats, or expensive one-off purchases in a single category such as automobiles, laptops, or mobile phones. (We leave extensions to multi-item purchases for future research.)

The challenges are the following: First, the data is not at a scale typically required for neural

network training; a hotel for instance has only a few months of seasonal data it can use for predictions. Second, the neural network predictions have to be used subsequently in an assortment optimization problem—essentially revenue-optimal subset-selection—and a neural network output promises no structure to make this optimization efficient. Consequently, we want a network as compact as possible so the optimization—in an integer programming formulation that we develop—is tractable for a reasonably large number of products.

Our contributions in this paper are the following:

1. We tackle the above-mentioned challenges and develop a neural network based choice model under two settings: (i) a feature-free setting where no additional features on products or customers are used in prediction and (ii) a feature-based setting that can utilize both product and customer features.
2. We formulate an integer program over the trained neural network to find an assortment that maximizes expected revenue.
3. We perform extensive numerical simulations that compare the predictive and optimization performance under the following scenarios:
 - (a) Synthetic simulations where the ground truth is generated by a panoply of models in the choice literature—MNL, Markov chain, non-parametric, mixed-logit—choice models and cross-validate them, both on raw prediction of choice probabilities, as well as the quality of the optimized assortment.
 - (b) Situations where the customer behavior does not follow the models’ script or rational economic behavior, but is documented experimentally and empirically in the literature.
 - (c) Real-world assortment and sales transactions data from the following important industries: physical fashion-retail, airline, hotel, and transportation, and compare the predictive performances of the popular discrete-choice models vs. our neural network.

4. We develop a meta-learning trick that allows us to obtain good robust prediction performance from a compact neural network even when trained on limited transactional purchase data. Moreover, we show a well-trained network can be warm-started when adding new products to the pool.

Our numerical results suggest that the neural network gives comparable performance to the best choice model in most cases, both in simulations as well as on real data. Some of the parametric models do perform better in simulations, unsurprisingly, when the ground-truth data generation coincides with the model. We find a similar pattern for real-world data, where on a hold-out sample our neural network gives robust predictions across the different industries. Moreover, we find that a one-layer neural choice model can capture what may be considered “irrational” purchase behavior—documented to occur in experiments and empirical studies—that cannot be explained by random utility models. In summary, shallow neural networks enjoy both prediction power and computational tractability for assortment optimization.

2.2 Literature Review

Probabilistic choice models originated in economics based on a stochastic utility model of the consumer. The oldest and still widely used discrete-choice model is the MNL, with alternates and generalizations proposed to overcome its known limitations, such as the nested-logit and mixed-logit. We refer the reader to the seminal paper [MT00] and the book [BAL85] for background on discrete-choice models and their economic motivation and estimation from data.

We review the literature in Computer Science and Operations Research next on neural network modeling of discrete choices and assortment optimization.

2.2.1 Neural Network Models in Utility Modeling

In the Computer Science community, [BM00] use a neural network to capture the non-linear effects of features on the latent utility. Recently [WMZ20, HZPBA20, SLA20, WF21, AD22,

AKAP23] study different application contexts and explore different neural network architectures for the feature-utility mapping. In addition to the product and customer features, [GT22] also encode history purchase information. [CGT21] and [CM22] take a different route and consider the random forest model as the mapping function. [VCWV⁺22] discuss the potential avenues for further integrating machine learning into choice modeling with a comprehensive review of the existing literature.

We term this line of works as deep learning based utility modeling instead of choice modeling, because these works mainly focus on predicting the unobserved product utilities with the observed features and applying an MNL-style operator for predicting purchase probabilities (except [AD22], who use a mixture of a set of estimated utilities to mimic the mixed-logit model). What is missing is the *assortment effect* on utilities—the interactions between the products within (and possibly outside) the set of offered products. This has two negative implications: First, some of the neural network models require all the training samples to have a fixed-size assortment. This requirement is simply not met in many application contexts such as hotels, airlines, or fashion retail where the assortments change frequently. Second, it fails to capture the effect that the assortment has on the product utilities. Intuitively, a product’s utility is not only determined by product and customer features, but also affected by the assortment offered to the customer which is a distinguishing feature of our neural network.

Another common drawback of the existing deep learning approaches is that they all require the availability of product or customer features. When there are no features available, most of the models degenerate to a uniform choice probability. In contrast, our neural network based choice model is designed to handle the feature-free setting. Importantly, we make a distinction between the feature-free model and the feature-based model not by the availability of the features, but by whether the downstream application is interested in a population-level or personalized choice model for its assortment optimization.

2.2.2 Assortment Optimization

Assortment optimization is an important problem in operations management, used in the airline, hotel, and fashion retail e-commerce. For more on the history and applications of assortment optimization, we refer readers to the books [TVRVR04, GT19]. Here we briefly review the complexity of the problem under some of the aforementioned choice models:

Multinomial Logit Model (MNL): Under the MNL model [TVR04] show that the optimal assortment is a revenue-ordered nested assortment and thus the optimal policy is to order the items by their revenues to form a nested set of assortments and choose the assortment with the largest expected revenue in that nested set. [RT12] show such revenue-ordered policy (RO) is robust against the uncertainty in MNL parameters and solvable under a cardinality constraint.

Markov Chain Choice Model (MCCM): [BGG16] give a polynomial-time algorithm for finding the optimal assortment under this model by repeatedly using a Bellman operator until convergence. [FT17] further prove that the optimal assortment can be found by solving a linear programming. With constraints, [DGSY20] prove that this assortment optimization problem is NP-hard and introduce algorithms with a provable worst-case approximation ratio.

Mixed-Multinomial Logit model (MMNL): [BMDV09] show that even without constraints, the assortment optimization problem under MMNL is NP-hard and then provides a column-generation method to solve the problem. [RSTT14] show that a revenue-ordered policy can obtain the optimal assortment in MMNL when the choice model has some special structures. [MDMBVZ14] propose a branch-and-cut algorithm for solving the optimization problem.

A General Choice Model: When the revenue function underlying a choice model is a black-box function, algorithms for finding the optimal assortment cannot be tailored by utilizing the structure. [Jag14] purposes the ADXOpt algorithm which repeatedly finds the best action in adding (A), deleting (D), and exchanging (X) based on the current assortment with provable approximation ratio under some conditions. [Udw23] defines submodular order functions and develops several heuristics for assortment optimization under various types of constraints. With

mild conditions, the heuristics have bounded approximation ratios.

In this paper, we show that in the context of assortment optimization, (1) a shallow network is enough to recover existing choice models and (2) good performance and robustness are achievable with a standard neural network and integer programming when dealing with different choice models compared to customized heuristics.

2.3 Problem Setup

The firm has a set of n products $\mathcal{N} = \{1, 2, \dots, n\}$ that can be offered to customers. An assortment $\mathcal{S} \subseteq \mathcal{N}$ is a subset of $\mathcal{N}$ that the firm decides to present to the customer. In practice, the assortment can be determined by the product availability or the seller may decide the assortment to maximize profits by limiting the choice of the customer. The choice model gives a prediction of the probability of the customer choosing product i conditional on the assortment $\mathcal{S}$:

$$\mathbb{P}(i|\mathcal{S}) \text{ for all } i \in \mathcal{N} \text{ and } \mathcal{S} \subseteq \mathcal{N}.$$

In particular, we assume that $\mathbb{P}(i|\mathcal{S}) = 0$ for $i \notin \mathcal{S}$, i.e., the customer cannot choose a product not offered in the assortment. In this way, a choice model

$$\mathcal{M} = \{\mathbb{P}(i|\mathcal{S}) : \mathcal{S} \subseteq \mathcal{N}\} \tag{2.1}$$

dictates $2^n - 1$ probability distributions, each of which corresponds to one possible nonempty assortment. In all applicable business contexts, there is usually a “no-purchase” option where the customer chooses not to purchase any of the items from the offered assortment. The no-purchase option can be captured by one product (say, indexed as n) in $\mathcal{N}$ and we can always have $n \in \mathcal{S}$.

A tabular parameterization of the choice model $\mathcal{M}$ generally requires $\Omega(n \cdot 2^n)$ parameters. Additional structures are therefore imposed to facilitate efficient estimation and revenue optimization.

2.3.1 Random Utility Models (RUM) and Beyond

The *random utility model* (RUM) is the dominant framework for choice models in practice. It assigns a random utility for each product and models the consumer's choice behavior according to the principle of utility maximization. Specifically, the utility of the i -th product

$$U_i = u_i + \epsilon_i \text{ for } i \in \mathcal{N}.$$

Here u_i is deterministic and represents the mean utility (to the customer) of purchasing the i -th product among the population. The random quantities, ϵ_i 's are assumed to have a zero mean and can be correlated among products, to model unobservable factors as well as heterogeneity across products. We note that customer heterogeneity is also modeled by assuming the random part is specific to the product as well as the customer. Under the RUM,

$$\mathbb{P}(i|\mathcal{S}) := \mathbb{P}\left(U_i = \max_{j \in \mathcal{S}} U_j\right) \quad \text{for } i \in \mathcal{S}. \quad (2.2)$$

For simplicity, we assume ties are broken randomly. Different distributions of ϵ_i 's specialize in this general RUM framework into different choice models as in the following.

Multinomial Logit Model (MNL): The MNL choice model assumes ϵ_i 's are i.i.d. from a mean-zero Gumbel distribution with scale parameter 1. Then the choice probability for $i \in \mathcal{S}$ has a nice closed-form solution:

$$\mathbb{P}(i|\mathcal{S}) := \frac{\exp(u_i)}{\sum_{j \in \mathcal{S}} \exp(u_j)}. \quad (2.3)$$

When there is a feature vector $\mathbf{f}_i$ associated with each product, one can also represent the deterministic utility $u_i = \boldsymbol{\theta}^\top \mathbf{f}_i$ and obtain the linear-in-attributes MNL model. Several recent works model u_i with a neural network function of the feature $\mathbf{f}_i$ [BM00, SLA20, WMZ20, AKAP23].

Although MNL model is simple to estimate and optimize for an assortment, the model's assumptions make it restricted to the *independence of irrelevant alternatives* (IIA) property: for

any $i, j \in \mathcal{S} \subset \mathcal{S}'$,

$$\frac{\mathbb{P}(i|\mathcal{S})}{\mathbb{P}(j|\mathcal{S}')} = \frac{\mathbb{P}(i|\mathcal{S})}{\mathbb{P}(j|\mathcal{S})}.$$

This means the ratio of two products' choice probabilities is independent of the other products in the assortment. We illustrate the effect of this for prediction and contrast it with the other methods.

Table 2.1 is an example to illustrate the limitation of the IIA property: In case I there is one product A with a no-purchase option. In Case II, when we add another product A' identical to the product A , we assume the customer who would purchase product A in Case I will have an equal probability to purchase A or A' in Case II. From the predicted probabilities, we can see the estimated MNL model cannot capture this purchase behavior due to the IIA property.

Product		True Prob.	Pred. Prob.		
			NN	MNL-MLE	MCCM-EM
Case I	No-purchase	.40	.42	.46	.40
	A	.60	.58	.54	.60
Case II	No-purchase	.40	.42	.32	.38
	A	.30	.28	.38	.32
	A'	.30	.30	.30	.30

Table 2.1: Example illustrating the effects of the IIA property of MNL. Column True Prob. is the true choice probabilities in each case and Pred. Prob. is the predicted probabilities from each method: NN is a one-layer Gated-Assort-Net neural choice model (that we introduce later), MNL-MLE is the MNL model calibrated via Maximum-Likelihood, while MCCM-EM implements the expectation-maximization algorithm [ST18] to fit a MCCM model. All choice models are estimated based on 8000 training samples where each sample's assortment is randomly picked with equal probability.

The following three RUM models (along with many others) were developed to mitigate this.

Mixed-Multinomial Logit Model (MMNL): The MMNL choice model [MT00] is a generalization of MNL to multiple segments. It assumes there are several customer types and each type has its own deterministic utilities; the random terms are still assumed to be i.i.d. and Gumbel distributed. We denote $\mathcal{C}$ as the set of customer types and α_c as the probability (for a customer) of being c 's type. Then the choice probability for $i \in \mathcal{S}$ is:

$$\mathbb{P}(i|\mathcal{S}) := \sum_{c \in \mathcal{C}} \alpha_c \frac{\exp(u_{c,i})}{\sum_{j \in \mathcal{S}} \exp(u_{c,j})}.$$

For a continuous distribution of customer types, the summation is replaced by an integral.

[MT00] show any RUM can be represented by an MMNL choice model under mild conditions and thus MMNL also has a large model capacity. However, the estimation of an MMNL is challenging and many of the methods proposed in the literature fail to recover the latent types' parameters [JSV20, HSLY22].

Markov Chain Choice Model (MCCM): The MCCM [BGG16] defines a discrete-time Markov chain on the space of products $\mathcal{N}$ and models each customer's choice through the realization of a sample path. Specifically, it assumes a customer arrives at product i with probability λ_i (interpreted as the initial state of the Markov chain). Then if the product is in the assortment $\mathcal{S}$, the customer purchases it and leaves. Otherwise, the customer transitions from product i to product j with probability ρ_{ij} . Naturally, $\sum_{i \in \mathcal{N}} \lambda_i = 1$ and $\sum_{j \in \mathcal{N}} \rho_{ij} = 1$ for all $i \in \mathcal{N}$. Let $X_t \in \mathcal{N}$ denote the product (state) that the customer visits at time t , and define the first hitting time $\tau = \{t \geq 0 : X_t \in \mathcal{S}\}$. Then, under the MCCM,

$$\mathbb{P}(i|\mathcal{S}) := \mathbb{P}(X_\tau = i).$$

The MCCM is a generalization of MNL, in the sense MNL arises for a specific form of the λ 's and ρ 's. It has more parameters and hence has a larger model capacity to capture more complex choice behavior. [Ber16] establishes MCCM as a RUM through a random walk argument.

Non-parametric (NP) Choice Model: The NP choice model or random orderings model has origins in classical microeconomic demand theory based on preference rankings [BM59] and recently reintroduced with a new estimation proposal in [FJS09, FJS13]. The model assumes that there exists a distribution $\lambda : \text{Perm}_{\mathcal{N}} \rightarrow [0, 1]$ over the set $\text{Perm}_{\mathcal{N}}$ of all possible permutations of the products. There are $n!$ customer types and each customer type has a preference list of the products corresponding to one permutation in $\text{Perm}_{\mathcal{N}}$. Customers always purchase the most preferred product in the preference list. The choice probability is given by

$$\mathbb{P}(i|\mathcal{S}) := \sum_{\sigma \in \text{Perm}_i(\mathcal{S})} \lambda(\sigma)$$

where the set

$$\text{Perm}_i(\mathcal{S}) := \{\sigma \in \text{Perm}_{\mathcal{N}} : \sigma(i) < \sigma(j) \text{ for all } j \neq i \in \mathcal{S}\}$$

contains all customer types/permutations under which product i is the most preferable product in the assortment $\mathcal{S}$.

The NP choice model has exponentially many parameters and thus a larger model capacity than both MNL and MCCM. Indeed, [BM59] show that any RUM choice model can be represented by an NP model, and vice versa. Moreover, the estimation method proposed by [FJS09, FJS13] needs to solve a linear programming with size $n!$. Although the linear programming enjoys good theoretical properties, it is computationally infeasible when n is large and has to be further simplified or approximated by the sampling or representation method.

Non RUM models: RUM models have a *regularity* property: if we add a new product to the assortment $\mathcal{S}$, then the purchase probability of all currently offered products will never increase. However, as remarked by [CM22], there is an increasing body of experimental evidence which suggests that the aggregate choice behavior of customers is not always consistent with this regularity property, and thus often violates the premise of RUM. For instance, [SPU19] use the seminal experiment of [TK85] and [CM22] quote the behavioral experiment of [AJ08] to motivate the development of choice models beyond the scope of RUMs. Here we use the examples from the above two behavioral experiments to show the limited capacity of RUM, while the neural choice model's potential to pick up such behavior in the data.

- Table 2.2 is an implementation of the experiment in [AJ08]: The price column shows the price of each option. In Case I, an assortment has two available products, while in Case II, a clearly inferior option of Print-Only is added. The addition of Print-Only twists customer utilities for the other two options, which is reflected by the change in the true choice probability.
- Table 2.3 is an implementation of the gambles example in [TK85]: The Win Prob. column shows the winning probability of each gamble and the Payoff column indicates the winning payoff. There exists a preference cycle over three gambles A, B, C with different winning

probabilities and payoffs, where A is preferred to B, B is preferred to C based on the expected payoff. However, C is preferred to A since the winning probability of C is significantly larger than A.

	Option	Price	True Prob.	Pred. Prob.		
				NN	MNL-MLE	MCCM-EM
Case I	No-purchase	—	.14	.14	.14	.14
	Internet-Only	\$59	.57	.58	.43	.43
	Print-&-Internet	\$125	.29	.28	.43	.43
Case II	No-purchase	—	.14	.13	.14	.15
	Internet-Only	\$59	.29	.31	.43	.43
	Print-&-Internet	\$125	.57	.55	.43	.42
	Print-Only	\$125	.00	.01	.00	.00

Table 2.2: The experiment setup is the same as Table 2.1.

	Gamble	Win Prob.	Payoff	True Prob.	Pred. Prob.		
					NN	MNL-MLE	MCCM-EM
Case I	A	1/4	\$ 6	.75	.76	.42	.62
	B	1/3	\$ 4	.25	.24	.58	.38
Case II	B	1/3	\$ 4	.75	.74	.43	.62
	C	1/2	\$ 2	.25	.26	.57	.38
Case III	A	1/4	\$ 6	.20	.20	.36	.24
	C	1/2	\$ 2	.80	.80	.64	.76

Table 2.3: The experiment setup is the same as Table 2.1.

In sum, a parametric model that fits observed purchases in all situations is unlikely. Certainly, more elaborate models of consumer search and decisions exist, but their estimation, based on unobservable factors and limited firm-level data, and subsequent optimization is daunting, and certainly will require a lot of development resources. Hence, it is worth striving for a flexible large-capacity learning engine to predict the purchase probabilities directly without worrying about getting the precise consumer behavior modeling right.

With this in mind, we develop neural network models that allow the utilities to be dependent on the assortment with or without including customer and product features. Hence our neural choice model allows violation of the regularity property, letting the utilities change depending on the assortment that is offered.

2.4 Neural Choice Model

2.4.1 A Feature-free Neural Choice Model

In this section, we introduce our feature-free neural choice model defined simply as a neural network function that maps the assortment input $\mathbf{S} \in \{0, 1\}^n$ to the output $\mathbf{Y} \in [0, 1]^n$:

$$\mathbf{Y} = g(\mathbf{S}; \boldsymbol{\theta})$$

where $\boldsymbol{\theta}$ encapsulates all the model parameters. The input $\mathbf{S} = (s_1, \dots, s_n)$ is the binary encoding of an assortment $\mathcal{S}$ where $s_i = 1$ if $i \in \mathcal{S}$ and $s_i = 0$ if $i \notin \mathcal{S}$. The output $\mathbf{Y} = (Y_1, \dots, Y_n)$ is a vector supported on the probability simplex with $Y_i = \mathbb{P}(i|\mathcal{S})$.

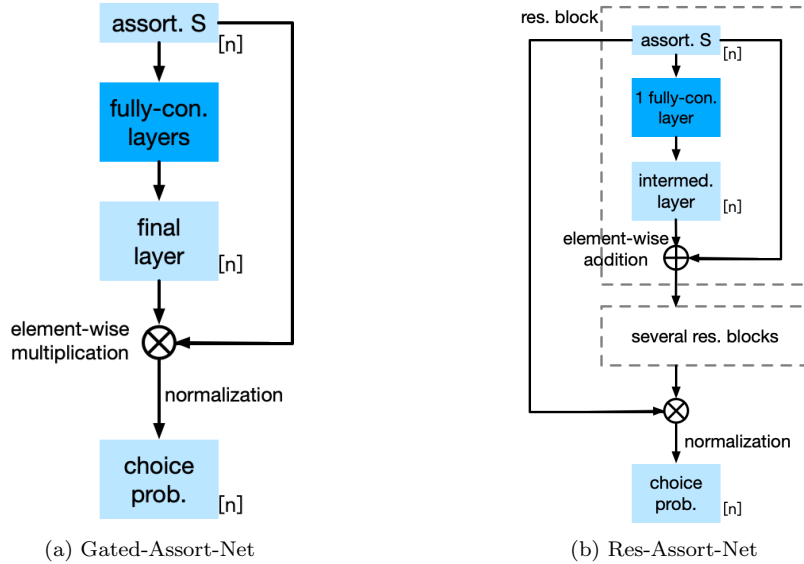


Figure 2.1: Neural choice models. $[n]$ denotes the dimension of the corresponding layer.

We consider two neural network architectures illustrated in Figure 2.1. For both architectures, we let $\mathbf{W}_l \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\mathbf{b}_l \in \mathbb{R}^{n_l}$ be the weight matrix and bias vector respectively for layer l with n_l nodes in the l -th layer, and $n_1 = n_L = n$. We let $\mathbf{z}_l = (z_{l,1}, \dots, z_{l,n_l}) \in \mathbb{R}^{n_l}$ be the output of the l -th layer and the input for the next layer. The parameters L and $\{n_l, l = 1, \dots, L\}$ need to be specified. (But we will show that $L = 1$ and $n_l = n$ can perform well in both the real data and simulated data across different generative models.)

The final layer's output $\mathbf{z}_L = (z_{L,1}, \dots, z_{L,n})$ are to set the choice probabilities given by a gated

operator

$$Y_i = \begin{cases} \frac{\exp(z_{L,i})}{\sum_{i' \in \mathcal{S}} \exp(z_{L,i'})}, & i \in \mathcal{S}, \\ 0, & i \notin \mathcal{S}. \end{cases}$$

While this may appear to be a standard MNL-like function, and hence suffer from the IIA property, etc., it in fact does not; we note an important difference: The $\mathbf{z}_L$ is a function of $\mathbf{z}_0 = \mathbf{S}$, i.e., the assortment effect is encoded into each $z_{L,i}$ and subsequently in Y_i through the neural network layers. (We should hence be representing $\mathbf{z}_L$ as $\mathbf{z}_L(\mathbf{S})$ but for avoiding cumbersome notation, just write as $\mathbf{z}_L$.)

The differences in the architectures are as follows.

2.4.1.1 Gated-Assort-Net (GAsN).

The GAsN model is a fully-connected neural network designed specifically for the discrete-choice model. It adopts the classic structure commonly found in traditional neural networks, making it straightforward to implement. It takes the assortment vector $\mathbf{S}$ as its input layer and runs through a number of fully connected layers. Finally, it uses the assortment $\mathbf{S}$ to create an output gate to ensure $\mathbb{P}(i|\mathcal{S}) = 0$ if $i \notin \mathcal{S}$. We initialize $\mathbf{z}_0 = \mathbf{S}$ as the input layer and for each layer $l = 1, \dots, L$,

$$\mathbf{z}_l = (\mathbf{W}_l \mathbf{z}_{l-1} + \mathbf{b}_l)^+ \quad (2.4)$$

where $(\cdot)^+ = \max\{\cdot, 0\}$.

2.4.1.2 Res-Assort-Net (RAsN).

The RAsN model incorporates the concept of residual learning, which has demonstrated a remarkable enhancement in image recognition compared to the performance of plain networks [HZRS16]. As in the GAsN, it is initialized by $\mathbf{z}_0 = \mathbf{S}$ and finally uses the assortment to create an output gate to ensure $\mathbb{P}(i|\mathcal{S}) = 0$ if $i \notin \mathcal{S}$. However, for each layer $l = 1, \dots, L$ in RAsN,

$$\mathbf{z}_l = (\mathbf{W}_l \mathbf{z}_{l-1} + \mathbf{b}_l)^+ + \mathbf{z}_{l-1}. \quad (2.5)$$

Here $\mathbf{z}_l = (z_{l,1}, \dots, z_{l,n}) \in \mathbb{R}^n$ is the output of the l -th layer and the input for the next layer, $\mathbf{W}_l \in \mathbb{R}^{n \times n}$ and $\mathbf{b}_l \in \mathbb{R}^n$ are the weight matrix and bias vector, parameters of the neural network model. Note we require the dimensions $n_l = n$ for all $l = 1, \dots, L$ for making the operation of adding $\mathbf{z}_{l-1}$ in (2.5) available.

Compared to (2.4), the additional (directly added) $\mathbf{z}_{l-1} \in \mathbb{R}^n$ in (2.5) represents the residual effect. Intuitively, directly adding it in (2.5) can avoid forgetting/distorting the information extracted from previous layers with extreme $\mathbf{W}_l$ (e.g., all entries are close to 0) during the training, which can hopefully further lead to a more robust training process and a better trained network.

2.4.1.3 Computational Experiment Design.

We give a preview of our numerical experiment methodology and also demonstrate the model capacity of the two neural choice model architectures (full details are presented in Appendix A.1).

	# Samples	MNL		MCCM		NP		MMNL	
		$ \mathcal{N} = 20$	$ \mathcal{N} = 50$	$ \mathcal{N} = 20$	$ \mathcal{N} = 50$	$ \mathcal{N} = 20$	$ \mathcal{N} = 50$	$ \mathcal{N} = 20$	$ \mathcal{N} = 50$
Uniform	–	2.51	3.4	2.52	3.44	2.50	3.43	2.48	3.40
MNL-MLE	1,000	1.87	2.65	1.96	0.50	1.73	2.50	1.95	2.41
	5,000	1.86	2.63	1.96	0.50	1.71	2.48	1.96	2.37
	100,000	1.86	2.62	1.95	0.52	1.71	2.47	1.96	2.36
MCCM-EM	1,000	1.86	2.67	1.92	0.54	1.67	2.35	1.86	2.27
	5,000	1.86	2.66	1.88	0.53	1.60	2.35	1.88	2.23
	100,000	1.87	2.66	1.88	0.51	1.62	2.37	1.88	2.22
Gated-Assort-Net	1,000	2.00	3.04	1.97	0.54	1.69	2.71	2.01	2.66
	5,000	1.89	2.81	1.88	0.48	1.55	2.43	1.87	2.30
	100,000	1.86	2.63	1.84	0.40	1.50	2.23	1.82	2.08
Res-Assort-Net	1,000	2.05	3.05	2.03	0.61	1.80	2.79	2.18	2.96
	5,000	1.93	2.82	1.91	0.56	1.56	2.51	1.97	2.53
	100,000	1.86	2.64	1.84	0.41	1.49	2.22	1.82	2.10
Oracle	–	1.86	2.62	1.82	0.38	1.42	2.13	1.80	2.04

Table 2.4: Feature-free choice modeling performance. The training data are generated from the model of MNL, MCCM, NP and MMNL, with randomly generated true parameters. The number of samples refers to the number of training samples, and additional 5,000 samples are reserved to validate the training procedure. The reported numbers are the average out-of-sample cross-entropy (CE) loss over 10,000 test samples and 10 random trials. The row Uniform denotes the performance of a uniform distribution for predicting. MNL-MLE implements the standard maximum likelihood estimation to estimate a MNL model, while MCCM-EM implements the expectation maximization algorithm to fit a MCCM model. GASN and RASN both have one hidden fully-connected layer. The row Oracle denotes the CE loss of the true model.

We first generate training data from four models: MNL, MCCM, NP and MMNL. We then compare the predictive performance of our two neural choice models, GASN and RASN, against

three benchmarks and an oracle that has knowledge of the true generative model (equivalently, the negative entropy of the true distribution). Table 2.4 shows the results.

First, both neural choice models benefit from a larger sample size, while the performances of maximum likelihood estimation (MLE) and expectation maximization (EM) do not improve much as the sample size increases. Second, when the true model is MNL, we know the method of MLE is provably asymptotically optimal and this is unsurprisingly verified from our experiment. But when the true model becomes more complex, such as NP and MMNL, the neural network models show their advantage. Third, the learning of a choice model requires a large sample size when the underlying true model becomes more complex. For example, under the true model of MCCM, NP, or MMNL, none of these methods give a satisfactory performance when the sample size $m = 1000$. Finally, the neural choice models perform consistently well in recovering these true models with a large amount of data, $m = 100000$. They have the model capacity to capture complex structures and they are also capable of fitting a simpler true model such as MNL. In comparison, the MNL choice model suffers from too little model capacity to capture them. Meanwhile, it is widely acknowledged that the models of NP and MMNL are certainly flexible and can model behavior realistically, but lack an effective learning/estimation algorithm given the number of variables that are unobservable in their stories, hence making them ill-suited for operational purposes.

2.4.2 Theoretical Analysis

In this subsection, we briefly discuss some theoretical properties of the neural choice model in terms of its parameter estimation. Throughout this subsection, we will focus on the model of GASn. We remark that the RAsN can be analyzed in the same way. First, we recall that the neural network model can be written as

$$\mathbf{Y} = g(\mathbf{S}; \boldsymbol{\theta}) = (g_1(\mathbf{S}; \boldsymbol{\theta}), \dots, g_n(\mathbf{S}; \boldsymbol{\theta}))^\top$$

which maps the assortment $\mathbf{S} \in \{0, 1\}^n$ to the choice probability vector $\mathbf{Y} \in [0, 1]^n$. The parameters $\boldsymbol{\theta}$ are estimated through *empirical risk minimization* on a dataset

$$\mathcal{D} = \{(i_1, \mathbf{S}_1), \dots, (i_m, \mathbf{S}_m)\}$$

where $(i_k, \mathbf{S}_k)$'s are i.i.d. samples from some unknown distribution $\mathcal{P}$. Specifically, we define the *risk* as the negative log-likelihood function (also known as the cross-entropy loss)

$$r((i, \mathbf{S}); \boldsymbol{\theta}) := -\log g_i(\mathbf{S}; \boldsymbol{\theta})$$

where $g_i(\mathbf{S}; \boldsymbol{\theta})$ gives the probability that the i -th product is chosen under parameter $\boldsymbol{\theta}$.

The estimated parameters $\boldsymbol{\theta}$ are obtained by maximum likelihood estimation as follows

$$\hat{\boldsymbol{\theta}} := \arg \min_{\boldsymbol{\theta} \in \Theta} \hat{R}_m(\boldsymbol{\theta}) := \frac{1}{m} \sum_{k=1}^m r((i_k, \mathbf{S}_k); \boldsymbol{\theta})$$

where the function $\hat{R}_m(\cdot)$ is also known as the *empirical risk*. Here the set Θ is defined as follows:

$$\Theta := \{\boldsymbol{\theta} : \|\mathbf{W}_l\|_\infty \leq \bar{W}, \|\mathbf{b}_l\|_\infty \leq \bar{b}, \text{ for } l = 1, \dots, L\}$$

where $\mathbf{W}_l$ and $\mathbf{b}_l$ are the weight matrix and the bias vector of the l -th layer of the neural network. Here for a matrix $\mathbf{W} \in \mathbb{R}^{n_1 \times n_2}$ we define $\|\mathbf{W}\|_\infty := \max_{i=1, \dots, n_1} \sum_{j=1}^{n_2} |W_{ij}|$ and for a vector $\mathbf{b}_l \in \mathbb{R}^{n_1}$, $\|\mathbf{b}_l\|_\infty := \max_{i=1, \dots, n_1} |b_i|$. The restriction to this bounded set usually improves both the stability of the training procedure and the generalization performance of the neural network.

Also, we define the *expected risk* as

$$R(\boldsymbol{\theta}) = \mathbb{E}[r((i, \mathbf{S}); \boldsymbol{\theta})]$$

where the expectation is taken with respect to $(i, \mathbf{S}) \sim \mathcal{P}$. The expected risk can be interpreted as the expected negative likelihood on some unseen/new data sample. We further define $\boldsymbol{\theta}^*$ as

its minimizer:

$$\boldsymbol{\theta}^* := \arg \min_{\boldsymbol{\theta} \in \Theta} R(\boldsymbol{\theta}).$$

The theorem below shows the gap between $R(\hat{\boldsymbol{\theta}})$, the expected risk of the learned parameter, and the optimal risk $R(\boldsymbol{\theta}^*)$.

Theorem 2.1. *The following inequality holds with probability $1 - \delta$,*

$$R(\hat{\boldsymbol{\theta}}) \leq R(\boldsymbol{\theta}^*) + \frac{4n}{\sqrt{m}} \left(\bar{b} \cdot \frac{(2\bar{W})^L - 1}{2\bar{W} - 1} + (2\bar{W})^L \sqrt{2 \log(2n)} \right) + 5C \sqrt{\frac{2 \log(8/\delta)}{m}}$$

where C is the upper bound of risk function such that $|r((i, \mathbf{S}); \boldsymbol{\theta})| \leq C$ holds for all $(i, \mathbf{S}) \in \mathcal{D}$ and $\boldsymbol{\theta} \in \Theta$.

Theorem 2.1 shows the excess risk of the empirical risk minimizer $\hat{\boldsymbol{\theta}}$. The proof is based on the standard analysis of neural networks [WZZ⁺13, NTS15, GRS18], but we need to customize it to GASN due to the special gated operator in the final layer. We defer the proof to Appendix A.3. By exploring the structure of the risk function, we can further bound the distance between the true distribution of choice probability and the estimated choice probability by $\hat{\boldsymbol{\theta}}$:

Corollary 1. *The following inequality holds with probability $1 - \delta$,*

$$\mathbb{E} \left[KL(\mathcal{P}_{\mathbf{S}}, g(\mathbf{S}; \hat{\boldsymbol{\theta}})) \right] \leq \mathbb{E} [KL(\mathcal{P}_{\mathbf{S}}, g(\mathbf{S}; \boldsymbol{\theta}^*))] + \frac{4n}{\sqrt{m}} \left(\bar{b} \frac{(2\bar{W})^L - 1}{2\bar{W} - 1} + (2\bar{W})^L \sqrt{2 \log(2n)} \right) + 5C \sqrt{\frac{2 \log(8/\delta)}{m}}$$

where C is defined as in Theorem 2.1, $\mathcal{P}_{\mathbf{S}}$ is defined as the distribution of choice probability given assortment $\mathbf{S}$, $KL(\cdot, \cdot)$ is defined as the Kullback–Leibler divergence over two choice probability distributions and expectation is with respect to $\mathbf{S}$.

Proof. Given an assortment $\mathbf{S}$, the Kullback–Leibler divergence over the true choice probability $\mathcal{P}_{\mathbf{S}}$ and $g(\mathbf{S}; \boldsymbol{\theta})$ is

$$KL(\mathcal{P}_{\mathbf{S}}, g(\mathbf{S}; \boldsymbol{\theta})) = \sum_{i \in \mathbf{S}} P(i|\mathbf{S}) (\log(P(i|\mathbf{S})) - \log(g_i(\mathbf{S}; \boldsymbol{\theta}))),$$

where $P(i|\mathbf{S})$ is the choice probability of i -th product given assortment $\mathbf{S}$.

Note that

$$\mathbb{E} \left[- \sum_{i \in \mathcal{S}} P(i|\mathcal{S}) \log(g_i(\mathbf{S}; \boldsymbol{\theta})) \right] = R(\boldsymbol{\theta}),$$

where expectation is with respect to the $\mathcal{S}$, with Theorem 2.1, we complete our proof. $\square$

The term $\mathbb{E} [\text{KL}(\mathcal{P}_{\mathcal{S}}, g(\mathbf{S}; \boldsymbol{\theta}^*))]$ is usually referred as *approximation error*, which depends on the capacity of function family $\{g(\mathbf{S}; \boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\}$, and the remaining two terms with order $O(1/\sqrt{m})$ are usually together referred as *estimation error*, which depends on the number of training samples m and also the capacity of function family through W , $\bar{b}$ and L . In general, with fixed training samples, a larger function family $\{g(\mathbf{S}; \boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\}$ (i.e., deeper and wider networks) will reduce the approximation error but will increase the estimation error.

2.4.3 Feature-based Neural Choice Models

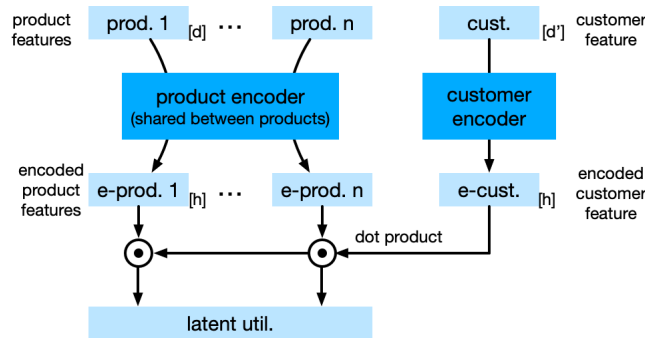


Figure 2.2: Feature encoder. Product features and customer features are encoded and then taken inner product to obtain the latent utility for each product. d is the dimension of the product features, and d' is the dimension of the customer features. Both features are encoded to h -dimensional latent features, and the latent utilities are obtained by the inner product of the corresponding latent features.

Now we extend our neural network models to the setting with features. We first make a distinction between product features and customer features:

- Product feature: there is a feature vector $\mathbf{f}_i^P \in \mathbb{R}^d$ associated with each product $i \in \mathcal{N}$.

We also refer to these features as static features as they usually remain unchanged over all the samples in the training data $\mathcal{D}$.

- Customer feature: Sample in $\mathcal{D}$ represents different customers, each of which is associated with a feature vector $\mathbf{f}_k^C \in \mathbb{R}^{d'}$ for $k = 1, \dots, m$. With these features, the dataset is augmented as

$$\mathcal{D} = \{(i_k, \mathcal{S}_k, \mathbf{f}_k^C), k = 1, \dots, m\}.$$

We refer to these features as *dynamic* features as they may vary across different samples. Modeling-wise, if there are product features that change over time, we can simply view them as dynamic features.

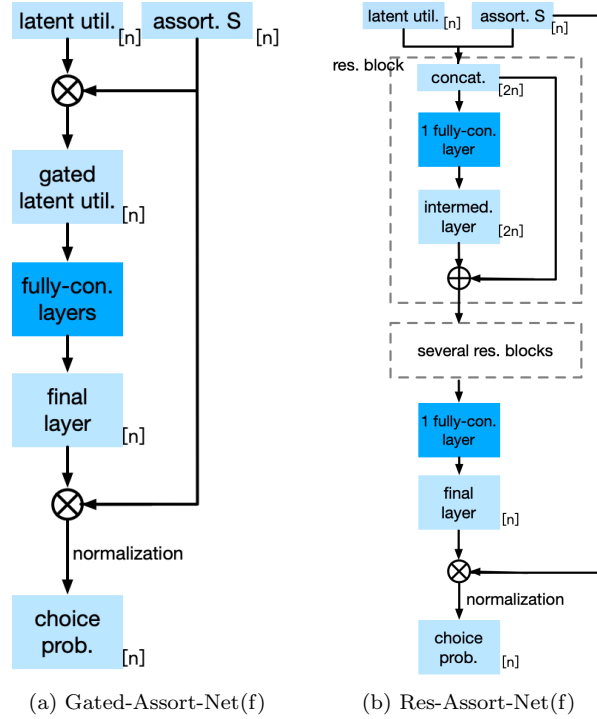


Figure 2.3: Choice networks with features. The networks take (i) the latent utilities from the feature encoder (in Figure 2.2) and (ii) the assortment vector as inputs.

Figure 2.2 and Figure 2.3 describe our choice networks with features. The neural networks inherit the architectures in the feature-free setting, but both take an additional vector of inputs which we call latent utilities. Both feature-based networks of GAsN(f) and RAsN(f) encode product features and customer features to obtain one latent utility for each product. The product encoder is an L -layer fully-connected network shared by all the products (L is 1 or 2 in our experiments). By default, the number of nodes in the intermediate layers of the encoder is the same as the dimension of the encoder's input. When there is no available customer feature, we can simply treat the customer feature as 1 for all the samples.

	MNL(f)	MCCM(f)
MNL(f)-MLE	2.903	2.212
Gated-Assort-Net	2.921	2.010
Gated-Assort-Net(f)	2.918	2.010
Res-Assort-Net	2.914	1.988
Res-Assort-Net(f)	2.914	1.984
Oracle	2.900	1.932

Table 2.5: Choice modeling performance with only product (static) features. The training data (with $n = 50$, $m = 100,000$, and $d = 5$) are generated from the feature-based version of MNL and MCCM. All networks have two-layer structures (for the networks with features, the encoder part has one layer). The reported numbers are the average out-of-sample cross-entropy (CE) loss over 10 random trials. The benchmark method implements MLE for the featured-based MNL model. More details about experiment setup can be found in Appendix B.1.

2.4.3.1 Unnecessity of Static Features.

Table 2.5 presents a synthetic experiment with only product (static) features to illustrate a new insight for choice modeling with features. The experiment first ignores all the product features and trains GAsN and RAsN as in the feature-free setting. Then it takes into account the product features and trains the feature-based version of the two neural networks. We find that both versions of the networks are capable of recovering the true model from the comparison against the oracle performance, and that the two versions of each neural network achieve a similar performance. This might seem counter-intuitive at first sight, because the product features may contain useful information about the product’s utility, and ignoring the features may accordingly hurt the performance. However, we argue that the product utilities are essentially encoded in the choice data $(i_k, \mathcal{S}_k)$ ’s but not in the product features. In fact, the experiment shows that such utilities can be learned by the neural networks of GAsN and RAsN without explicitly accessing the product features. An alternative interpretation is that the product utility here should be viewed as a population-level utility that reflects the preference or popularity of a product over the whole population. Things become different when customer features are also available. In that case, the choice behavior of each data sample is determined by the personalized utilities, and thus the product and customer features become indispensable.

We emphasize that whether to include the (dynamic) customer features in choice modeling is determined not by the availability of the features, but by the downstream application. For

example, the assortment optimization problem aims to find an assortment that maximizes profits under a choice model. For a brick-and-mortar store or an online retailer where the personalized assortment is not allowed, customer features should not be used to fit the choice model even if they are available, because one is more interested in the population-level choice behavior. It also underscores the potential of feature-free networks of GAsN and RAsN despite their simple architectures.

2.4.4 Comparison on Real Datasets

In this subsection, we test the predictive performances of our neural choice model with several benchmarks on four real datasets, two private datasets from revenue management industries without features and two public ones with features. We repeat all experiments 10 times, choosing the train/validation/test sets independently and randomly, and report the averaged results below. In Section 2.5.2 we perform extensive numerical experiments on synthetic datasets to jointly test prediction and optimization performance.

2.4.4.1 Feature-free Neural Choice Model.

We conduct numerical experiments on two private real datasets: airline data and retailing data. The airline data contains three flight markets and the offered products are different bundles, i.e., the bundle of seats, food, etc. The offered assortments are decided by both the selling strategy and remaining seats. The retailing data is collected from a fashion-retailing company, where the products are clothes (from the same category and thus most customers purchase at most one item) and the assortments are decided by the inventories. Both datasets contain records of purchase transactions only, so we add no-purchases as in [ST18]: for each purchase, we create four no-purchases with the same assortment as the original transaction but as choosing the outside no-purchase option. The summary statistics of the dataset are given in Table 2.6.

The testing result is summarized in Table 2.7, where the reported numbers are out-of-sample cross-entropy loss of two benchmark methods (with the same configurations as in Table 2.4)

	Flight 1	Flight 2	Flight 3	Retailing
# Samples	90634	37075	70600	17200
# Products	25	25	25	20
# Train Samples	60000	30000	60000	10000
# Validate Samples	1000	1000	1000	1000
# Test Samples	10000	5000	10000	5000

Table 2.6: Description of flights and retailing data.

and the two neural network models with one layer.

We observe that our neural networks achieve the best performance on these datasets where the generative model is unknown and likely complex. One interesting observation is unlike other numerical results in this paper, MNL-MLE is better than MCCM-EM for the airline markets. One potential reason for the relatively poor performance of MCCM-EM might be the complicated assortment generation: in Subsection 2.6.1, we numerically show the performances of MCCM-EM are correlated with the assortment distributions, which are special in the airline markets due to their nested booking-limits selling strategy.

	Flight 1	Flight 2	Flight 3	Retailing
MNL-MLE	2.798	2.490	2.547	1.073
MCCM-EM	3.972	3.470	3.596	1.063
Gated-Assort-Net	2.738	2.436	2.489	1.063
Res-Assort-Net	2.734	2.434	2.490	1.060

Table 2.7: Performance on the airline data and retailing data. The reported numbers are out-of-sample CE loss.

2.4.4.2 Feature-based Neural Choice Model.

For the case of estimation with features, we perform numerical experiments on two public datasets – SwissMetro and Expedia Search. These two datasets contain dynamic customer/product features; that is, the features associated with each training sample may be different.

The SwissMetro dataset is a public survey dataset to analyze traveler preference among three transportation modes, and the Expedia dataset consists of hotel search records and the associated customer choices. More details of datasets, experiment setup, and benchmarks can be found in Appendix A.2.

	SwissMetro	Expedia
MNL-MLE	0.883	2.827
MNL(f)-MLE	0.810	2.407
TasteNet	0.681	2.436
DeepMNL	0.741	2.374
Random Forest	0.633	2.458
Gated-Assort-Net(f)	0.598	2.403
Res-Assort-Net(f)	0.607	2.325

Table 2.8: Performance on the SwissMetro and Expedia. The reported numbers are out-of-sample CE loss.

We implement a number of benchmark models: MNL, feature-based MNL, TasteNet [HZPBA20], DeepMNL [WMZ20, SLA20], random forest [CGT21]. The MNL and feature-based MNL are learned by MLE. TasteNet and DeepMNL model the product utility u_i as a function of the product and customer features and map the utility to choice probability via MNL (2.3). TasteNet assumes the utility u_i is a linear function of product features and uses the customer feature to determine the coefficients of the linear function, while DeepMNL concatenates the customer and product features and feeds both into a fully-connected network to obtain the utility u_i . Both models bring assortment effects only through the MNL part.

From the results in Table 2.8, our two neural networks give a better performance than the benchmark models. Even though the benchmark models include neural networks in this case, the key difference is that our models explicitly take the assortment as an input. This experiment result shows that even with the presence of product and customer features, incorporating assortment effects can help to better explain customer choices. Comparing the two datasets, the Gated-Assort-Net(f) performs better on SwissMetro, while Res-Assort-Net(f) performs better on Expedia. Recall that the Res-Assort-Net(f) makes a stronger usage of the assortment vector throughout the architecture than Gated-Assort-Net(f). Accordingly, one explanation can be that for the transportation setting, customer choices are less affected by the available options but more by their personal preference, but for the hotel search, the customer choices are more affected by the provided assortment, which gives Res-Assort-Net(f) more of an advantage. As for the random forest model, it is trained as a discriminative model using both features and assortment as input, so we believe it also has better potential with some further model recalibration.

2.5 Assortment Optimization for Neural Choice Model

An important downstream task after the estimation of a choice model is the assortment optimization problem, where the seller aims to find an assortment that maximizes the profits

$$\max_{\mathcal{S} \subset \mathcal{N}} \text{Rev}(\mathcal{S}) := \sum_{i \in \mathcal{N}} \mu_i \mathbb{P}(i | \mathcal{S}) \quad (2.6)$$

where μ_i is the profits/revenue of selling one unit of product i . The problem usually has additional space or cardinality constraints.

The literature on assortment optimization fixes an underlying choice model $\mathcal{M}$ and devises algorithms with provable performance guarantees. However, in practice, the underlying choice model is unknown a priori and a model has to be selected first and its parameters estimated from data $\mathcal{D}$. In this case, the suboptimality of a proposed assortment may come from two sources: (i) the approximation error: the inaccuracy of the choice model estimation, and (ii) the optimization error: the optimality gap induced by solving the assortment optimization problem under the estimated model. The following bound shows the intuition:

$$\text{Rev}(\mathcal{S}^*) - \text{Rev}(\hat{\mathcal{S}}) \leq \underbrace{\bar{\mu} \cdot \text{dist}(\mathcal{M}, \hat{\mathcal{M}})}_{\text{Model Approx. Error}} + \underbrace{\text{Opt. Gap}}_{\text{Opt. Error}}. \quad (2.7)$$

Here $\mathcal{M}$ denotes the true choice model and $\hat{\mathcal{M}}$ is the choice model estimated from the data $\mathcal{D}$. $\mathcal{S}^*$ is the optimal assortment obtained from (2.6) using the true model $\mathcal{M}$, and $\hat{\mathcal{S}}$ is obtained from $\hat{\mathcal{M}}$ via some assortment optimization algorithm, where the term Opt. Gap refers to the suboptimality induced by the algorithm for assortment optimization. The function $\text{dist}(\cdot, \cdot)$ refers to some (pseudo-)distance function between two models, and $\bar{\mu} := \max_{i \in \mathcal{N}} \mu_i$.

We use the inequality (2.7) to emphasize the decomposition of the revenue gap into the approximation error and the optimization error. The existing works on assortment optimization have striven to improve the optimization error, while assuming the approximation error is zero. We make the case that both errors should be taken into account when measuring the performance of an assortment algorithm as the firm is interested in the combined performance. In light of

this, we measure the performance of our neural choice model illustrating the tradeoff between these two errors.

2.5.1 Integer Programming Formulation of the Neural Choice-based Assortment Optimization

In this section, we present a mixed integer programming (MIP) formulation for the assortment optimization problem under the neural choice model. Denote $\boldsymbol{\mu} = (\mu_1, \dots, \mu_n)^\top \in \mathbb{R}^n$ as the revenue vector as in (2.6). Throughout this section, we will focus on the model of GAsN. We remark that the RAsN gives a similar MIP formulation and numerical performance. We formulate the assortment optimization problem under the GAsN as follows,

$$\begin{aligned}
\max \quad & \frac{\sum_{i=1}^n \mu_i z_{0,i} \exp(z_{L,i})}{\sum_{i=1}^n z_{0,i} \exp(z_{L,i})} \\
\text{s.t.} \quad & \mathbf{z}_l - \tilde{\mathbf{z}}_l = \mathbf{W}_l \mathbf{z}_{l-1} + \mathbf{b}_l, \text{ for } l = 1, \dots, L, \\
& \mathbf{0} \leq \mathbf{z}_l \leq M \boldsymbol{\zeta}_l, \mathbf{0} \leq \tilde{\mathbf{z}}_l \leq M(\mathbf{1} - \boldsymbol{\zeta}_l), \text{ for } l = 1, \dots, L, \\
& \boldsymbol{\zeta}_l \in \{0, 1\}^{n_l}, \text{ for } l = 1, \dots, L, \\
& \mathbf{z}_0 = (z_{0,1}, \dots, z_{0,n})^\top \in \{0, 1\}^n,
\end{aligned} \tag{2.8}$$

where the decision variables are $\mathbf{z}_l \in \mathbb{R}^{n_l}$, $\tilde{\mathbf{z}}_l \in \mathbb{R}^{n_l}$, $\boldsymbol{\zeta}_l \in \{0, 1\}^{n_l}$, all for $l = 1, \dots, L$, and $\mathbf{z}_0 \in \{0, 1\}^n$. The decision variables $\mathbf{z}_0$ for the input layer provide a binary representation of the assortment decision, and $\mathbf{z}_l$ represents the values of the intermediate layers of the neural network. The equality constraints describe the forward propagation of the neural network layers, and the inputs $\mathbf{W}_l$ and $\mathbf{b}_l$ are the parameters of the neural network learned from the data. The objective function describes the mapping from the final layer of the network to the revenue under the assortment $\mathbf{z}_0$.

The (auxiliary) decision variables $\tilde{\mathbf{z}}_l$ and $\boldsymbol{\zeta}_l$ implement the big-M method [CCZ⁺14, FJ17] where the value of M is a large positive number obtained by some prior estimate. These two jointly

ensure that

$$\mathbf{z}_l = (\mathbf{W}_l \mathbf{z}_{l-1} + \mathbf{b}_l)^+.$$

It is well known that the choice of M affects the running time of solving the MIP [BBF⁺16]. In our numerical experiments, we follow the methods in [FJ17] to tighten the upper bound. Also, additional linear constraints can be added to incorporate the cardinality or capacity constraints. When there is a no-purchase option, one may introduce an additional constraint of $z_{0,n} = 1$ (n being the index of the no-purchase option).

We remark that the constraints of the MIP (2.8) are all linear, and the objective function resembles that of the MNL model. For a better numerical performance, we replace the exponential function in the objective function with its second-order approximation $\exp(x) \sim 1 + x + x^2/2$ [BGV⁺20]. Though the problem (2.8) cannot be solved in polynomial time as is the case for some of the other models such as MNL and MCCM (when there is no capacity constraint), it does provide a formulation that allows us to take advantage of off-the-shelf integer programming solvers.

2.5.2 Numerical Study of Joint Estimation and Optimization

In this subsection, we provide a comparative study of the four models (MNL, MCCM, 1 and 2-layer GAsN) along with their corresponding estimation methods and assortment optimization algorithms by running them on a total of 9600 different synthetic problem instances generated by a mix of ground-truth strategies. Since there are no computationally efficient estimation methods for NP and MMNL choice models, we do not include them here. Each problem instance is a combination of historical sales and revenue data, and additional capacity constraints (if any), and the algorithm(s) output a recommended assortment following this data-to-decision pipeline:

$$\mathcal{D} = \{(i_k, \mathcal{S}_k), k = 1, \dots, m\}, (\mu_1, \dots, \mu_n), \text{Constraints} \rightarrow \text{Recommended Assortment } \hat{\mathcal{S}}.$$

The numerical experiment is to test performance where the true model is unknown and has to be estimated from data.

We generate the training data $\mathcal{D}$ from four different choice models as ground truth, MNL, MCCM, NP, and MMNL with the number of training samples fixed at $m = 30,000$ for all the trials. The exact optimal assortment $\mathcal{S}^*$ is computed with the knowledge of the generative model, and then the recommended assortment $\hat{\mathcal{S}}$ is evaluated by the optimality ratio

$$\text{Opt. Ratio} := \frac{\text{Rev}(\hat{\mathcal{S}})}{\text{Rev}(\mathcal{S}^*)}.$$

For each problem instance, we consider both unconstrained and constrained settings and implement several benchmark methods as follows. For the estimation part, we implement the maximum-likelihood estimation to estimate the parameters of the MNL model, and the expectation-maximization algorithm to estimate those of MCCM. We provide more details of the experiments in Appendix A.4. Below are the assortment optimization algorithms:

- Unconstrained: The estimated choice model is used to solve for a recommended assortment through the assortment optimization specific to the method.
- Constrained:
 - Revenue-ordered policy (RO): All the n products are ordered in a decreasing order by their revenues to form a nested set of assortments and the feasible assortment with the largest expected revenue is picked from that nested set [TVR04].
 - MNL-MIP: A mixed integer programming is solved based on the estimated MNL model [MDMBVZ14].
 - ADXOpt: The ADXOpt algorithm proposed by [Jag14] is a greedy algorithm to solve assortment optimization under a general choice model. We implement the algorithm on the estimated Markov chain choice model. In our implementation, we increase the removal limit (the maximal number of times for a product to be removed from the assortment) from 1 in [Jag14] to 5 so as to boost the algorithm performance.

- Markov Chain Capacity-Assort (MC-CA): We implement the algorithm given in [DGSY20] on the estimated Markov chain choice model.
- Neural Network Optimization (NN): We implement the MIP (2.8) on the fitted GAsN with 1 or 2 layers (denoted as NN(1) and NN(2)).

The results of our numerical experiments are given in Table 2.9 and Table 2.10. In general, the neural network with MIP formulation performs well in all underlying choice models achieving the best performance in several cases (averaged over different numbers of products), with and without a capacity constraint; and under MNL choice model, the (averaged) approximation ratios of NN(1) are above 95%, both with and without constraint.

The advantage of a neural network is more apparent when the generative models are complicated: with data generated by NP and MMNL and with constraints, the averaged approximation ratios of NN are 92.55% and 84.79%, where the best averaged performance of the other methods are 88.28% and 67.14%. The relatively poor performances of the other methods can be due to the limited expressive power or the hardness of the subsequent optimization. The structural limitations of MNL are well known, and hence is not surprising that it cannot approximate NP and MMNL generated data. For the MCCM, [BGG16] show that the worst case error bound of MCCM to approximate a choice probability of an assortment in MMNL is negatively correlated to “the maximum probability that the most preferable product for a random customer from any segment belongs to that assortment” (Theorem 4.2). Thus, when the offered assortment includes many “preferable products”, the estimated MCCM may not approximate the underlying MMNL choice model well. Also, constraints make approximation ratios worse by making the optimization harder; this can be observed by comparing Table 2.9 and Table 2.10.

Comparing NN(1) with NN(2), NN(1)’s average approximation ratios are better than NN(2)’s in all cases except MCCM with constraint. In fact, we observe the overfitting of NN(2) during the training in some cases, which is a plausible explanation for its worse relative performance.

In summary, when trying to model and optimize the assortment optimization problems, our

finding is that, when evaluating joint estimation and assortment optimization, a shallow network may be preferable to a deep neural network, as it reduces overfitting and leads to a more manageable optimization problem. Of course, if its estimation error is large, one can try adding more layers to enlarge the capacity and expressive power of the neural network.

Model $ \mathcal{N} $	MNL				MCCM			
	20	40	60	Avg.	20	40	60	Avg.
MNL (MLE)	99.94	99.36	99.11	99.47	93.54	94.26	94.59	94.13
MCCM (EM)	99.70	99.44	99.65	99.60	93.88	96.80	97.77	96.15
NN (1-layer)	99.71	98.49	97.80	98.67	96.93	95.78	97.68	96.80
NN (2-layer)	98.03	91.14	86.60	91.92	96.60	90.46	91.83	92.96

Model $ \mathcal{N} $	NP				MMNL			
	20	40	60	Avg.	20	40	60	Avg.
MNL (MLE)	91.10	92.17	92.88	92.05	79.87	88.59	92.17	86.88
MCCM (EM)	92.89	92.88	92.99	92.92	84.36	89.24	91.12	88.24
NN (1-layer)	95.36	93.65	93.84	94.28	89.22	90.81	91.35	90.46
NN (2-layer)	95.80	87.00	78.65	87.15	92.15	83.40	75.28	83.61

Table 2.9: The approximation ratios ($\times 100\%$) of heuristics for assortment optimization without capacity constraint. Each reported value is averaged over 100 trials.

Model $ \mathcal{N} $		MNL				MCCM			
		20	40	60	Avg.	20	40	60	Avg.
MNL (MLE)	RO	83.27	76.82	77.93	79.34	77.44	74.70	90.36	80.83
	MIP	99.92	99.36	99.02	99.43	89.48	92.01	93.22	91.57
MCCM (EM)	MC-CA	92.25	89.98	88.01	90.08	85.83	87.33	95.05	89.40
	ADXOpt	96.91	96.89	96.04	96.61	90.26	92.07	94.48	92.27
NN (1-layer)	MIP	99.14	98.73	98.85	98.91	93.58	92.03	92.71	92.77
NN (2-layer)	MIP	97.49	96.20	94.33	96.01	93.17	93.51	97.08	94.59

Model $ \mathcal{N} $		NP				MMNL			
		20	40	60	Avg.	20	40	60	Avg.
MNL (MLE)	RO	77.82	78.03	74.47	76.77	59.81	47.37	45.46	50.88
	MIP	86.86	90.37	87.20	88.14	75.39	65.44	60.60	67.14
MCCM (EM)	MC-CA	84.96	85.06	84.49	84.84	70.54	55.26	56.17	60.66
	ADXOpt	89.02	88.81	87.02	88.28	73.60	61.70	59.26	64.85
NN (1-layer)	MIP	93.90	93.28	90.47	92.55	92.76	84.99	76.63	84.79
NN (2-layer)	MIP	94.80	91.27	87.53	91.20	91.58	77.59	68.28	79.15

Table 2.10: The approximation ratios ($\times 100\%$) of heuristics for assortment optimization with capacity constraint. Each reported value is averaged over 100 trials.

2.6 Extensions and Discussions

In this section, we discuss several properties of neural choice models gained from our experience with the experiments, including robustness under the assortment distribution shift in the testing data and the effects on the depth and width of the networks. We also list some tricks for deploying neural choice models when we need to add new products or the training samples are not sufficient. We defer the details of all experiments in this section to Appendix A.5.

2.6.1 Assortment Distribution Shift in the Training Data

A very practical but often neglected aspect of learning a choice model is the assortment distribution in the training data $\mathcal{D}$, i.e., the distribution of $\mathcal{S}_k$'s. Figure 2.4 gives empirical evidence that the learning of an MCCM using the EM algorithm (implementing [ST18]) is affected by the assortment size of the training samples. Intuitively, a smaller assortment size means it will take longer for the Markov chain to hit the exiting state, and thus the E-step needs to impute larger unobserved quantities which may result in larger variance. Alternatively, we find the performance of the neural network model is quite robust in terms of the generalization on the assortment domain. Specifically, in Appendix A.5.1, we train the neural network model with one distribution to generate the assortments $\mathcal{S}_k$'s (in training data $\mathcal{D}$) and find that it performs surprisingly well on the test data where the assortments are generated from a different assortment generating distribution (an aspect of out-of-domain performance).

2.6.2 Depth and Width of the Neural Networks

Throughout the paper, we use one-layer or two-layer neural networks for our neural choice models, i.e., $L = 1$ or 2 . We find this provides a sufficiently good fit for both synthetic and real data. For the number of neurons/nodes in the intermediate layers, we set $n_l = n$ for all l 's by default. In Appendix A.5.2, we provide more numerical results that the performance of the neural choice models remains stable with more number of layers and wider width for

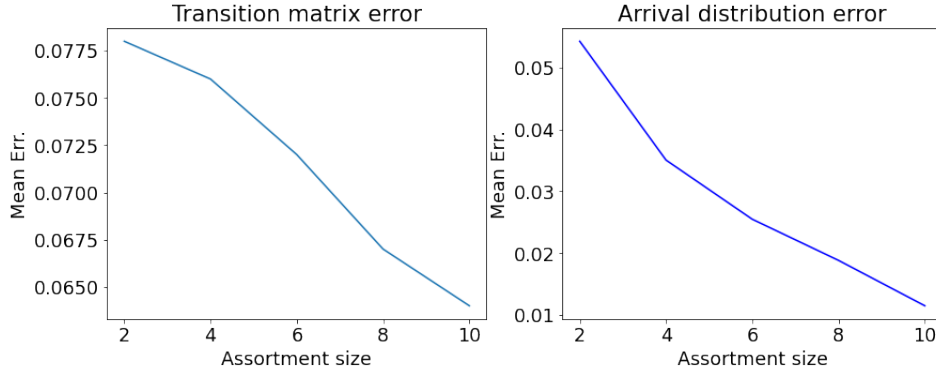


Figure 2.4: Smaller assortment size leads to larger estimation error. In the experiment, the total number of products $n = 20$, and the training data consists of $m = 10,000$ samples with a fixed-size assortment $\|S_k\|$. The x-axis gives the assortment size and the y-axis represents the mean error. The plotted curves are based on an average of 10 independent trials.

the intermediate layers. Generally, it is not recommended to have many layers for the neural network, as it results in a bad landscape for the loss function and consequently, bad local minima that do not generalize well [KMN⁺16].

2.6.3 Network Augmentation with Warm Start

One drawback of the models proposed in this paper is that they all require the universe of available products to be fixed, i.e., $\mathcal{N} = \{1, \dots, n\}$ is fixed. When an additional set of products $\mathcal{N}' = \{n + 1, \dots, n'\}$ is available, the model has to be retrained on the new product set of $\mathcal{N} \cup \mathcal{N}'$. One option is to initialize the weights of the new network according to the previously well-trained network of $\mathcal{N}$ as a warm start. Figure 2.5 shows that the well-trained network of $\mathcal{N}$ can provide a good warm start for retraining the new network.

2.6.4 Neural Choice Model as a Meta Choice Model

From our numerical results in Table 2.4, we gain a new perspective, namely, to use the neural choice model as a meta-choice model. Table 2.4 shows how the neural choice model can benefit from large data sizes but can also be outperformed when the data size is small. In this case, we can use the estimated choice model with the smallest validation error to generate synthetic data to re-train another neural choice model and fine-tune it with the real training data. The

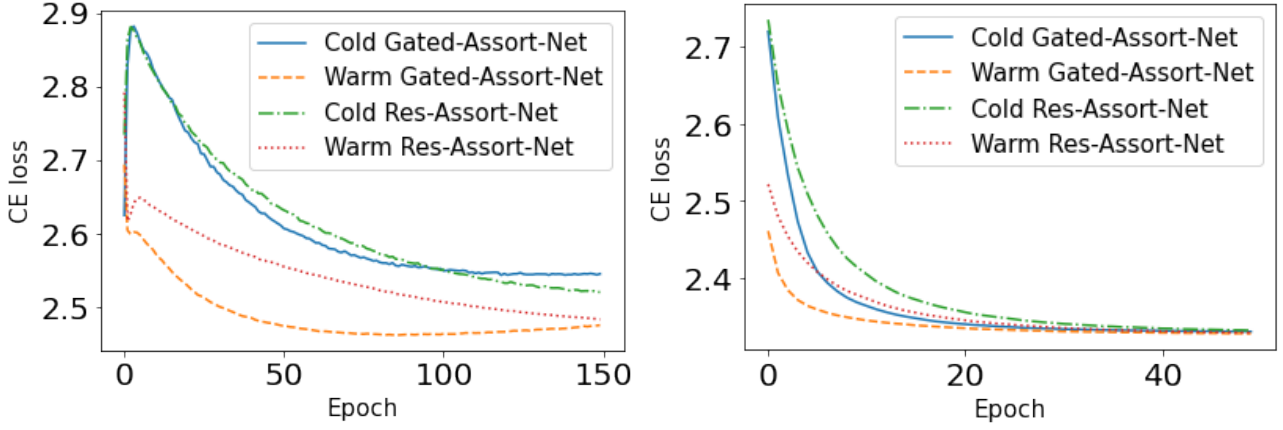


Figure 2.5: Validation losses using cold start and warm start. The left panel uses 2,000 samples for retraining and the right panel uses 100,000 samples.

pseudo-code is given in Algorithm 1. Intuitively, the strong representation ability of the neural choice model can make its performance arbitrarily close to the well-fitted generative model in the re-training step and can be further improved by real data in the fine-tuning step. Since the re-training step is indeed learning the trained choice model $\hat{\mathcal{M}}_{k^*}$ with the *best* validation error from all other trained models $\hat{\mathcal{M}}_k$, which has the *best* training error of the corresponding choice model class $\mathcal{M}_k$, we name this algorithm *Learning the Best of the Bests*. In this way, the neural choice model is not trained as a competitor to the existing choice models, but more as a complementary meta choice model. In Appendix A.5.4, we implement Algorithm 1 on a public Hotel data [BFG09] as an example.

2.7 Conclusion and Future Directions

The existing literature on choice modeling in the feature-free and feature-based settings have been largely separate from each other. In this paper, we propose a neural choice model, a unified neural network framework that applies to both settings through a binary representation of the assortment and feature encoders. In addition, we provide an MIP formulation for the assortment optimization for a trained neural choice model.

From extensive numerical experiments, we gain insights into the performance of the various methods and summarize them into the following three points: First, a single neural choice

Algorithm 1 Learning the Best of the Bests

```

1: Input: Training data  $\mathcal{D}_{\text{train}}$ , validation data  $\mathcal{D}_{\text{val}}$ , a set of choice models  $\{(\mathcal{M}_k, \mathcal{L}_k)\}_{k=1}^K$ 
   where each choice model  $\mathcal{M}_k$  is equipped with a learning algorithm  $\mathcal{L}_k$ .  $(\mathcal{M}_1, \mathcal{L}_1)$  corre-
   sponds to neural choice model.
2: %% Training phase
3: for  $k = 1, \dots, K$  do
4:   %% We omit the training details such as cross-validation
5:   Fit the  $k$ -th candidate choice model using dataset  $\mathcal{D}_{\text{train}}$  and algorithm  $\mathcal{L}_k$ 
6:   Denote the trained model as  $\hat{\mathcal{M}}_k$ 
7: end for
8: %% Comparison phase
9: Compute the test performance of all  $\hat{\mathcal{M}}_k$ 's using  $\mathcal{D}_{\text{val}}$ 
10: Let  $k^*$  be the index of the model with best test performance
11: if  $k^* = 1$  then
12:   Let  $\mathcal{M}_{\text{final}} = \hat{\mathcal{M}}_1$ 
13: else
14:   %% Re-training and fine-tuning
15:   Generate  $m' (= 100,000)$  new training samples as  $\mathcal{D}'_{\text{train}}$  from the model  $\hat{\mathcal{M}}_{k^*}$ 
16:   Use  $\mathcal{D}'_{\text{train}}$  to train (re-train) a neural choice model  $\hat{\mathcal{M}}_{\text{tmp}}$  from scratch
17:   Use  $\mathcal{D}_{\text{train}}$  to further train (fine-tune)  $\hat{\mathcal{M}}_{\text{tmp}}$  and obtain  $\hat{\mathcal{M}}'_1$ 
18:   Let  $\mathcal{M}_{\text{final}}$  be the better one among  $\hat{\mathcal{M}}'_1$  and  $\hat{\mathcal{M}}_{k^*}$ 
19: end if
20: Output:  $\mathcal{M}_{\text{final}}$ 

```

architecture is capable of recovering all the existing choice models, with a standard learning procedure. The neural choice model becomes particularly effective when the underlying model/training data is too complex to be described by a simple model such as MNL and when there is a sufficient amount of training data. Second, the neural choice models are robust under the assortment's distribution shift and can serve as a meta-choice model. Third, the combined calibration and assortment optimization regime based on the neural network can outperform other optimization heuristics, especially when the underlying model is complex and when we have capacity constraints; the drawback nevertheless is that as it is based on a MIP, the size of the optimization cannot scale to more than around a hundred products; we need further research and new optimization algorithms to scale beyond this.

Moreover, we conclude with the following two future directions:

- **Multiple-purchase choice model.** The existing choice models focus exclusively on modeling the single-choice behavior where only one product is chosen from the offered

assortment. The multiple-purchase choice model is usually studied under an inverse optimization framework in the literature of revealed preference [ZR12, ACD⁺15, BLS22]. The deep learning-based choice models provide a natural framework to study the multiple-choice behavior, and they can fit such training data by modifying the loss function.

- **Pricing optimization.** Although we equip our neural networks with an assortment optimization MIP, in a retailing context, the product price is another important factor that affects customer choice. Our current models do not account for this factor but assume prices as fixed. It is an interesting research problem to study optimal pricing under the neural choice model.

Chapter 3

Transformer Choice Net: A Transformer Neural Network for Choice Prediction

3.1 Introduction

Firms are interested in understanding the choice behavior of their customers as well as forecasting the sales of their items. When customers choose at most one item per shopping instance, discrete-choice models estimate the probability of the choice, either at a segment level or individual customer level, based on a latent utility function of the features of the item, the customer, and the provided assortment.

However, there are many situations where customers choose multiple items on a single shopping instance, either from the same category or across categories. The firm may be aware of only the final choices made by the customer (as in physical retail) or the precise sequence of those choices (such as in an e-commerce setting). *Multi-choice models* are used for the former case, to estimate the probability of choosing a subset of items, amongst all possible subsets of the given assortment, considering potential interactions amongst the items and their features. *Sequential*

choice models consider the sequence of choices, taking into account not only the item and customer features but also what the customer has chosen till then to predict the subsequent choice(s).

Modeling and predicting the choice probabilities for these situations is challenging: the complexity of the sequential and multi-choice models is considerably more than in the single-choice case because of combinatorial explosion in the number of possible customer journeys and final choices, and consequently models for multiple choices are less widely adapted in practice.

In this paper, we introduce the Transformer Choice Net, a neural network using the Transformer architecture [VSP⁺17], as a data-driven solution that works under any of the three models: single, sequential, and multiple.

The key contributions of this paper are as follows:

- We develop the Transformer Choice Net, a neural network-based framework capable of encompassing all three choice paradigms—single, sequential, and multiple choices. This unification simplifies the traditionally fragmented approach to choice modeling.
- We empirically validate the prediction performance of our model across diverse industry benchmark datasets. Our architecture shows uniformly superior out-of-sample prediction performance compared to the leading models in the discrete-choice literature, without requiring any custom modeling or tuning for each instance. Moreover, our architecture is scalable, showing an ability to learn and predict multiple choices over a large assortment size, overcoming the limitations of many sequential and multi-choice models.
- We show theoretically the universal representation capacity of the Transformer Choice Net across the three choice paradigms.

3.2 Literature Review

This section reviews the literature for single, sequential, and multiple choices, and also the literature on neural networks for predicting the choice probabilities.

Single-choice models. For general background on discrete-choice models, see [MT00] and the book by [BAL85].

A plethora of models have been proposed for the single-choice case starting from the popular Multinomial Logit (MNL) choice model: to name a few, the Probit [Dag14], the ranking-based choice model [BM59, FJS13], the Negative-Exponential (also called Exponential) model [AS16], and the Markov-chain choice model [BGG16].

From a machine learning perspective, the single-choice case can be viewed as a classification task where we assign probabilities for the various items (classes) to be chosen. From this point of view, researchers have applied random forests [CGT21, CM22], embedding methods [SPU19], and assortment context vectorization [YMCD⁺20, TB21, NJUZ23, BB20] with the latter trying to overcome some known weaknesses of the MNL model. [VCWV⁺22] provide a comprehensive review of the current literature on integrating machine learning into single-choice modeling.

[BM00] give an early effort to use neural networks for single-choice modeling. [WMZ20, HZPBA20, SLA20, WF21, AKAP23] investigate different neural network architectures to capture the relationships between the items' features and utility in the MNL choice model. Several recent works try to capture more complicated assortment effects: the RUMNet [AD22] mimics the mixed-MNL model instead of the MNL model, and the AssortNet [WCLT23] encodes the entire assortment into the choice probability via ResNet architecture. [PGHH22, ROS20] employ the Deep Set architecture [ZKR⁺17] to allow permutation variations in input assortments (see also [WFE⁺19] and §3.4).

Sequential and multi-choice models. Sequential choice is typically based on purchase history data, often from assortments with only a small number of items (≤ 10) [MAG99, Gup88]. Despite the growth of e-commerce and the availability of click-stream data, it is somewhat

surprising that there are relatively few models for sequential choice behavior: The SHOPPER model [RAB20] stands out as a probabilistic representation of consumer choice, aiming to forecast sequential choices based on variational inference. Another noteworthy approach is due to [JFD21], where they compress purchase history data into purchase motivations and then employ variational inference for model deductions. [GT22] design a neural network that takes each customer’s history as input to predict the next choice, but like [JFD21] the architecture does not encode the customer and item features.

There is a slightly richer body of research on multiple choices, i.e., without knowing the sequence of choices [CGWW23, BFS⁺23, LWXX23, JLNZ23, TSKP23]. [BKT18] expand the random utility maximization model from single to multi-choice scenarios. [TSKP23] take a different approach, quantifying pairwise item interactions within each potentially chosen subset through a linear sum. [ASC23] bridge point processes with subset-choice models, encoding pairwise interactions using the log det function. [LLW22] augment the ranking-based choice model [BM59, FJS13] to multiple choices, but they do not explicitly model item or customer features. A common limitation for all these models is scalability: as we have to predict the most likely amongst an exponential number of subsets, the size of the assortment they can handle is small.

Transformer architecture. First introduced by [VSP⁺17], the Transformer architecture has emerged as a ground-breaking deep learning framework. Initially tested on machine translation within the realm of natural language processing (NLP), it has since expanded to a broad range of NLP tasks as exemplified by ChatGPT [Ope23] for text generation.

The Transformer architecture has also excelled in computer vision [HWC⁺22], audio processing [DXX18], and integrated multimodal tasks [RPG⁺21]. For a survey, we refer to the review by [LWLQ22]. Relevant to this paper, [LLK⁺19] design the Set Transformer, which ensures any permutation of the input to yield identical outputs (see §3.4 for more discussions). Recently, [PRZ24] introduce a modified Transformer-based choice model designed explicitly as a simulator for evaluating assortment optimization algorithms within revenue management. This facilitates rigorous assessment and comparison of various decision-making models, includ-

ing Multinomial Logit (MNL) and DeepFM, under controlled yet realistic scenarios. However, their model primarily addresses single-choice scenarios and focuses on model evaluation through simulation rather than direct prediction of consumer choices across multiple paradigms (i.e., single-choice, sequential-choice, and multi-choice models). Consequently, their approach necessitates a different architectural design in deep learning/Transformer frameworks, underscoring a key distinction from the general-purpose Transformer Choice Net proposed in this study.

3.3 Problem Setup

3.3.1 Preliminaries on Choice Models

3.3.1.1 Single-Choice Model

Consider a ground set of n potential items $\mathcal{N} = \{1, 2, \dots, n\}$. For completeness, we can let n denote the no-purchase option which (when selected) means the customer chooses none of the offered items. A seller chooses an *assortment* $\mathcal{S} \subseteq \mathcal{N}$ to offer to the customer, and in the single-choice case the customer will choose at most one item from it. A *single-choice model* prescribes the probability of choosing item i conditional on the assortment $\mathcal{S}$ (where we can assume the no-purchase option is always in the assortment):

$$\mathbb{P}(i|\mathcal{S}) \text{ for all } i \in \mathcal{N} \text{ and } \mathcal{S} \subseteq \mathcal{N}.$$

In particular, $\mathbb{P}(i|\mathcal{S}) = 0$ for $i \notin \mathcal{S}$, i.e., the customer cannot choose an item not offered in the assortment. In this way, a single-choice model

$$\mathcal{M} = \{\mathbb{P}(i|\mathcal{S}) : \mathcal{S} \subseteq \mathcal{N}\} \tag{3.1}$$

dictates $2^n - 1$ probability distributions, each of which corresponds to one possible assortment (excluding when $\mathcal{S}$ is an empty set).

3.3.1.2 Sequential Choice Model

The sequential choice model is an intermediate between the single-choice model and the multi-choice model, and it also serves as a useful model by itself. Consider a customer shopping at an e-commerce who may choose more than one item for purchase and the items are chosen sequentially one after another from an assortment. To capture this scenario, we define the set of currently not chosen items as *candidates* (denoted by $\mathcal{C}$), while the set of initially offered items as *assortment* (denoted as $\mathcal{S}$). Thus $\mathcal{C} \subseteq \mathcal{S}$ and the set $\mathcal{S} \setminus \mathcal{C}$ is the set of chosen items so far. The single-choice model then can be viewed as a special case where $\mathcal{C} = \mathcal{S}$. The sequential choice model describes the probability of choosing item i from candidates $\mathcal{C}$ given the assortment $\mathcal{S}$

$$\mathbb{P}(i|\mathcal{C}, \mathcal{S}) \text{ for all } i \in \mathcal{N}, \mathcal{C} \subseteq \mathcal{S} \subseteq \mathcal{N}.$$

Let $\mathbb{P}(i|\mathcal{C}, \mathcal{S}) = 0$ for $i \notin \mathcal{C}$, i.e., the customer can only choose an item from the candidates set. In parallel to (3.1), the sequential choice model encapsulates the following distributions

$$\mathcal{M} = \{\mathbb{P}(i|\mathcal{C}, \mathcal{S}) : \mathcal{C} \subseteq \mathcal{S} \subseteq \mathcal{N}\}. \quad (3.2)$$

3.3.1.3 Multi-Choice Model

The multi-choice model considers the purchase of a basket of items, i.e., a subset of the given assortment. Hence, we are interested in the following distributions

$$\mathcal{M} = \{\mathbb{P}(\mathcal{B}|\mathcal{S}) : \mathcal{S} \subseteq \mathcal{N}\} \quad (3.3)$$

where $\mathcal{B} \subseteq \mathcal{N}$ denotes the basket of (chosen) items and $\mathcal{S}$ is the given assortment as before. As earlier, we assume $\mathbb{P}(\mathcal{B}|\mathcal{S}) = 0$ if $\mathcal{B} \not\subseteq \mathcal{S}$; in other words, the chosen basket can only contain items from the assortment.

3.3.2 Learning Task

The learning task, the focus of our paper, refers to estimating the choice model $\mathcal{M}$ from m observed samples $\mathcal{D}$:

$$\mathcal{D} = \begin{cases} \{(i_k, \mathcal{S}_k), k = 1, \dots, m\} & \text{single choice,} \\ \{(i_k, \{\mathcal{C}_k, \mathcal{S}_k\}), k = 1, \dots, m\} & \text{sequential choice,} \\ \{(\mathcal{B}_k, \mathcal{S}_k), k = 1, \dots, m\} & \text{multiple choices,} \end{cases}$$

where each observation, depending on the model to be estimated, consists of the final choice(s) (i_k or $\mathcal{B}_k$), accompanied by the assortment information ($\mathcal{S}_k$ or $\{\mathcal{C}_k, \mathcal{S}_k\}$).

Item/product feature. We assume that there is a feature vector associated with each item/product. Such a feature vector encodes characteristics of the product such as color, size, brand, price, etc. When no such feature is available, one can still obtain such a vector by a one-hot encoding of the items.

Reduction to sequential choice model. The goal of our paper is to develop one single transformer model that fits all three choice modeling scenarios. In this light, we discuss here how the single-choice model and the multi-choice model can be reduced to the sequential choice model in terms of both learning and inference so the transformer model can be based only on learning and predicting a sequential choice model.

The reduction of the single-choice model to the sequential choice model is straightforward. As we pointed out earlier, we simply set $\mathcal{C} = \mathcal{S}$, i.e., to predict the first item to be chosen given the initial assortment. For the estimation part, the sequential choice model can be estimated from the single-choice training samples $\{(i_k, \mathcal{S}_k), k = 1, \dots, m\}$ by simply augmenting it into the form $\{(i_k, \{\mathcal{S}_k, \mathcal{S}_k\}), k = 1, \dots, m\}$. For the inference part, one can infer the probabilities of the single-choice model by using the probability $\mathbb{P}(i|\mathcal{S}, \mathcal{S})$ from the sequential choice model.

The reduction from the multi-choice model to the sequential choice model is a bit more complicated. First, we note that the multi-choice model can also be mathematically represented

by the sequential choice model. To see it, we can introduce a sequence σ to denote the order of items being added to the basket. Then we can decompose the choice probability of a basket by sequential choice probabilities:

$$\mathbb{P}(\mathcal{B}|\mathcal{S}) = \sum_{\sigma \in \text{Perm}_{\mathcal{B}}} \prod_{j=1}^{|\mathcal{B}|} \mathbb{P}(\sigma_j | \mathcal{C}_j^{\sigma, \mathcal{S}}, \mathcal{S}), \quad (3.4)$$

where $\text{Perm}_{\mathcal{B}}$ contains all possible permutations over the set $\mathcal{B}$, σ_j is the j -th item in the permutation σ , and $\mathcal{C}_j^{\sigma, \mathcal{S}} = \mathcal{S} \setminus \{\sigma_i : i < j\}$, i.e., the candidates when adding j -th item to the basket.

With this representation in mind, we first consider the learning of a sequential choice model from multi-choice training samples $\{(\mathcal{B}_k, \mathcal{S}_k)\}, k = 1, \dots, m\}$. Since the sequence in which the items are added to the basket $\mathcal{B}_k$ is unobserved, we cannot directly use the data to train a sequential choice model. One remedy involves an iteratively random selection of items from the basket, designating it as the sequential choice, and then constructing its associated candidates. For example, we are given a multi-choice sample $\{\mathcal{B}, \mathcal{S}\}$ and we start the sample generation (for the sequential choice model) by setting $\mathcal{C} = \mathcal{S}$. Say, the first randomly chosen item from $\mathcal{B}$ is $i^{(1)}$; then we obtain one sample of $\{i^{(1)}, (\mathcal{S}, \mathcal{S})\}$. Then we move on to generating the second sample, $\{i^{(2)}, (\mathcal{S} \setminus \{i^{(1)}\}, \mathcal{S})\}$ where $i^{(2)}$ is randomly picked from $\mathcal{B} \setminus \{i^{(1)}\}$. By continuing this process, we generate training samples for the sequential choice model from the original multi-choice sample $\{\mathcal{B}, \mathcal{S}\}$. A similar sampling method is also applied in [RAB20].

To predict the multiple choices of a given assortment $\mathcal{S}$ from a sequential choice model, the set of predicted choices can be sequentially generated by reversing the above sampling method. Specifically, we can start with $\mathcal{C} = \mathcal{S}$ and either sample a product $i^{(1)}$ following the distribution $\mathbb{P}(\cdot | \mathcal{C}, \mathcal{S})$ or select the item with the largest choice probability, and then remove $i^{(1)}$ from $\mathcal{C}$. By continuing this process, we can sample multiple items and add them to the predicted set of choices, until a stopping rule is met. This stopping rule can be enforced either when a pre-determined size of the basket is met or a virtual “stopping item” is sampled.

3.4 Transformer Choice Net

Now we introduce the Transformer Choice Net (TCNet) for the sequential choice model (which as we pointed out covers both single and multi-choices). Mathematically, the TCNet is denoted as f^{TCN} , a function that maps the candidate set $\mathcal{C}$ and the assortment $\mathcal{S}$ to the choice probability $f_i^{\text{TCN}}(\mathcal{C}, \mathcal{S})$ for each item $i \in \mathcal{C}$.

3.4.1 Attention Mechanism

The attention mechanism has become a fundamental component of modern deep learning to solve various tasks of natural language processing and computer vision. In this subsection, we provide some preliminaries on this mechanism (see [NZY21] for more details) and show how it can be adapted for choice modeling.

An *Attention* function $\text{Att}(Q, K, V)$ takes three matrices as input: the matrix of queries $Q \in \mathbb{R}^{S \times d}$, the matrix of keys $K \in \mathbb{R}^{S' \times d}$, and the matrix of values $V \in \mathbb{R}^{S' \times d}$

$$\text{Att}(Q, K, V) = \phi(QK^\top)V,$$

where d is the feature dimension. The function ϕ is an activation function to $QK^\top$, with Softmax being the most prevalent choice. The matrix $QK^\top$ is termed the (unnormalized) score matrix, the (i, j) -element of which quantifies the influence of item j on item i . Intuitively, the attention mechanism empowers a model to selectively focus on specific segments of its input, guided by the score function, to generate an output.

Self-attention is an exemplification of the attention mechanism where all the three matrices above depend on the input feature matrix X :

$$f^{\text{Att}}(X; W_Q, W_K, W_V) = \text{Att}(XW_Q, XW_K, XW_V), \quad (3.5)$$

where $W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$ are trainable parameters. For simplicity, we abbreviate the param-

eters and use $f^{\text{Att}}(X)$ to denote this self-attention function.

3.4.2 Architecture of TCNet

Figure 3.1 illustrates the architecture of the Transformer Choice Net f^{TCN} . The network's input comes from two sources: the assortment set $\mathcal{S}$ of size S and the candidate set $\mathcal{C}$ of size C . As noted in §3.3, we suppose each item i is associated with a feature vector $x_i \in \mathbb{R}^d$ (either product feature or one-hot encoding of the product id). Then the input matrices of TCN are $X^{\mathcal{S}} \in \mathbb{R}^{S \times d}$ and $X^{\mathcal{C}} \in \mathbb{R}^{C \times d}$, and the output is a vector of choice probabilities for each item $i \in \mathcal{C}$.

The TCN consists of three main parts: assortment encoder, candidates encoder, and utility decoder.

The *assortment encoder* processes the input $X^{\mathcal{S}}$ and encodes both the original feature $X^{\mathcal{S}}$ and the interaction effects (from the assortment) into a latent feature matrix $\tilde{X}^{\mathcal{S}} \in \mathbb{R}^{S \times d_v}$. Here each item i is associated with a latent feature vector $\tilde{X}_i^{\mathcal{S}} \in \mathbb{R}^{d_v}$. In this feature transformation, the assortment encoder can contain L layers (which is a hyperparameter), and each with three sub-layers:

First, an optional embedding sub-layer processes the original input $X^{\mathcal{S}}$ by an element-wise fully connected feed-forward network layer applied to each item in the input assortment matrix separately, via two linear operations and an activation function ϕ :

$$f^{\text{FFN}}(x) := W_2 \phi(W_1 x + b_1) + b_2,$$

where $W_1 \in \mathbb{R}^{d_v \times d}$, $W_2 \in \mathbb{R}^{d_v \times d_v}$ and $b \in \mathbb{R}^{d_v}$ are trainable parameters. The dimension d_v controls the embedding's complexity.

Second, a self-attention sub-layer implements the attention mechanism defined in (3.5). This layer captures the interaction effects between the items within the assortment.

Third, a fully connected feed-forward network layer $f^{\text{FFN}}(x)$ (similar to the first part) is applied to each item separately to further process each item's latent features.

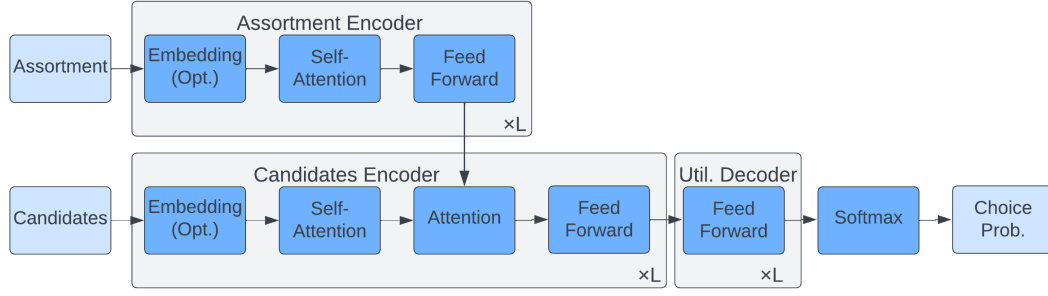


Figure 3.1: Architecture of Transformer Choice Net (TCNet). We omit the residual connections within each sub-layer.

The *candidates encoder* processes the original input X^c in a similar manner as the assortment encoder. It processes the original feature and captures the interaction effect between items within the candidate set through three modules as the assortment encoder: an optional embedding sub-layer, a self-attention sub-layer, and a fully-connected feed-forward layer. One special point is an additional cross-attention sub-layer that mingles the assortment set and the candidate set. Specifically, this assortment-candidate attention sub-layer takes (i) the output of the assortment encoder $\tilde{X}^S \in \mathbb{R}^{S \times d_v}$ and (ii) the output X of the self-attention sub-layer of the candidates encoder as input, and then maps them to a latent feature matrix $\tilde{X}^{c,S} \in \mathbb{R}^{C \times d_v}$ that is aware of the assortment context. Mathematically, this is represented by

$$f^{\text{AC}}(X) = \text{Att}(XW_Q, \tilde{X}^S W_K, \tilde{X}^S W_V)$$

where $W_Q, W_K, W_V \in \mathbb{R}^{d_v \times d_v}$ are trainable parameters.

The *utility decoder* processes the output latent features from the candidates encoder into some latent utilities for each item in $\mathcal{C}$. Ideally, the output latent feature $\tilde{X}_i^{c,S}$ from the candidates encoder has encoded both the candidate set effect and the assortment set effect. The utility decoder transforms each of this latent feature vector $\tilde{X}_i^{c,S}$ into a (scalar) latent utility $\tilde{u}_i$, and the transformation shares the same weight/parameter across different items. Here we apply a fully-connected feed-forward network, which can be replaced by other network architectures. Lastly, an MNL model/softmax layer converts these latent utilities into choice probabilities.

3.4.3 Remarks on TCNet

We make the following remarks on the TCNet.

Assortment size flexibility. One advantage of the attention mechanism is that it can handle input (e.g., assortment set and candidate set) with varying sizes. This is useful in a choice modeling context because the assortment and the candidate sets may have variable cardinality. Moreover, the number of parameters in TCNet does not depend on the total number of items $|\mathcal{N}|$ which makes it attractive for the case when there are a large number of items or newly added items. As long as the item/product feature of a new item/product is available, the TCNet can be used seamlessly for assortments with newly launched products.

Permutation equivariance. For any permutation σ applied to the rows of X (such as interchanging the first and second row), the f^{Att} satisfies *Permutation Equivariant* [LLK⁺19], which is defined as $f^{\text{Att}}(\sigma(X)) = \sigma(f^{\text{Att}}(X))$. This ensures that each output choice probability remains invariant to the ordering/position of the input. This is useful for choice modeling context where the items in the assortment/candidate set are presented to the customer all at once. In this case, there is no ordering of the products and thus the customer choice should be permutation equivariant. Of course, when the position/order does matter in determining choice, one can encode the position/order as a feature.

Model	# parameters	Variable size	Assortment effect
TCNet (Ours)	$O(dd_v + Ld_v^2)$	Yes	Yes
DeepMNL[WMZ20, SLA20]	$O(dd_v + Ld_v^2)$	Yes	No
AssortNet [WCLT23]	$O(\ \mathcal{N}\ dd_v + Ld_v^2)$	No	Yes
SDANet [ROS20]	$O(dd_v + Ld_v^2)$	Yes	Yes
FATENet [PGHH22]	$O(dd_v + Ld_v^2)$	Yes	Yes
DLCL [TB21]	$O(d^2)$	Yes	Yes
RUMNet[AD22]	$O(dd_v + Ld_v^2)$	Yes	No

Table 3.1: Comparison on the model complexity. d and d_v are dimensions of item and latent features respectively, L is the number of layers, and $\|\mathcal{N}\|$ is the total number of items.

Comparison against other NN-based choice models. In Table 1, we make a comparison between TCNet and several other neural network-based choice models (for single-choice) from

several aspects. The column “variable size” refers to whether the model requires a fixed assortment size or allows a variable size. The column “assortment effect” refers to whether the model captures the assortment effect, i.e., the interactions between items within the assortment.

In terms of the number of parameters, we can see that TCNet has the same order of the number of parameters as other models, thus its empirical advantage (in the numerical experiment) should be attributed to its architecture rather than model size. In particular, we note that SDANet [ROS20] and FATENet [PGHH22] essentially implement the Deep Set [ZKR⁺17] for choice modeling, and thus capture item-wise interaction on a coarser scale than our TCNet. DLCL [TB21] explicitly models product interactions through linear functions and does not introduce a neural network architecture.

Transformer specializations for choice modeling. The Transformer [VSP⁺17] is originally designed for seq2seq tasks, such as machine translation and text summarization. The model predicts the likelihood of the subsequent word from the vocabulary list. In comparison, the TCNet is designed for a set-to-member task, where the goal is to forecast the item(s) to be chosen from the candidate set and thus it requires the permutation equivariance. This necessitates modifications to the input’s structure and the utility decoder. Similarly, [LLK⁺19] introduce the Set Transformer which is designed to model interactions within an input set. Different from TCNet, the Set Transformer aims for permutation invariance ($f(X) = f(\sigma(X))$ for any permutation σ over an ordered set X) which keeps the whole output invariant through permuting inputs. In the language of choice modeling, permutation invariance assumes that the assortment effect will take place through the assortment set as a whole $\mathcal{S}$ (or $\mathcal{C}$) and keep same for all items, while the permutation equivariance allows a finer interaction between the items.

Training: We use the cross-entropy loss (CE loss) for the training. With dataset $\mathcal{D} = \{(i_k, \{\mathcal{C}_k, \mathcal{S}_k\}), k = 1, \dots, m\}$, the CE loss is defined by

$$L_{\text{CE}} = -\frac{1}{m} \sum_{k=1}^m \log(f_{i_k}^{\text{TCN}}(\mathcal{C}_k, \mathcal{S}_k)). \quad (3.6)$$

Several tricks/techniques are used in the original Transformer [VSP⁺17] such as layer normalization, multi-head attention, residual connection, and dropout. We also implement them for TCNet.

3.5 Expressiveness of TCNet

The seminal work [Man77] shows that any single-choice model can be represented by $O(2^n \times n)$ utility parameters: $\{u_i^{\mathcal{S}}, i \in \mathcal{N}, \mathcal{S} \subseteq \mathcal{N}\}$. Specifically, for all $i \in \mathcal{S}$ and $\mathcal{S} \subseteq \mathcal{N}$,

$$\mathbb{P}(i|\mathcal{S}) = \frac{\exp(u_i^{\mathcal{S}})}{\sum_{j \in \mathcal{S}} \exp(u_j^{\mathcal{S}})} \quad (3.7)$$

where the utility $u_i^{\mathcal{S}}$ represents the *contextualized* utility of the i -th item under the assortment $\mathcal{S}$. [BP85] further show that the utility $u_i^{\mathcal{S}}$ can be decomposed into a sum of item-wise interactions among the subsets of the assortment which is also recently discussed by [SPU19].

Theorem 3.1 ([BP85]). *For any $\{u_i^{\mathcal{S}} : i \in \mathcal{N}, \mathcal{S} \subseteq \mathcal{N}\}$, there exists a unique set of parameters $\{v_i^{\mathcal{S}} : i \in \mathcal{S}, \mathcal{S} \subseteq \mathcal{N}\}$ where $\sum_{i \notin \mathcal{S}} v_i^{\mathcal{S}} = 0, \forall \mathcal{S} \subset \mathcal{N}$, such that for all $i \in \mathcal{S}$ and $\mathcal{S} \subset \mathcal{N}$,*

$$u_i^{\mathcal{S}} = \sum_{\mathcal{S}' \subseteq \mathcal{S} \setminus i} v_i^{\mathcal{S}'}.$$

The above theorem shows that to estimate a single-choice model, it is enough to estimate the interaction effects of each item i , i.e.,

$$\{v_i\}, \{v_i^{\{j\}} : j \in \mathcal{N}, j \neq i\}, \dots, \{v_i^{\mathcal{N} \setminus \{i\}}\}.$$

Here the constraints $\sum_{j \notin \mathcal{S}} v_j^{\mathcal{S}} = 0, \forall \mathcal{S} \subset \mathcal{N}$ ensure the uniqueness of the parameters (noting the scale-invariance of the utilities in (3.7)). Based on the level of interaction, we call $\{v_i\}$ as 0-th order interaction, $\{v_i^{\{j\}}\}$ as 1-st order (or pairwise) interaction, and so on. The classic MNL model captures only 0-th order interaction, and several studies [SPU19, BB20, YMCD⁺20, TB21, NJUZ23] explore diverse formulations to capture 1st-order interactions $\{v_i^{\{j\}}\}$. In this

sense, TCNet can capture L -th order interactions with L the number of layers in the self-attention module. Thus L provides a statistical handle for controlling the model complexity and also a modeling handle for controlling the level of item-wise interactions.

In parallel, one can parameterize any sequential choice model with utility parameters:

$$\mathbb{P}(i|\mathcal{C}, \mathcal{S}) = \frac{\exp(u_i^{\mathcal{C}, \mathcal{S}})}{\sum_{j \in \mathcal{C}} \exp(u_j^{\mathcal{C}, \mathcal{S}})}, \quad (3.8)$$

where the utilities can be further decomposed by

$$u_i^{\mathcal{C}, \mathcal{S}} = \sum_{\mathcal{C}' \subseteq \mathcal{C} \setminus i} v_i^{\mathcal{C}', \mathcal{S}}.$$

Here the term $v_i^{\mathcal{C}', \mathcal{S}}$ captures the interaction effects of (i) the candidate subset $\mathcal{C}'$ towards item i and (ii) the assortment effect from $\mathcal{S}$. The interaction from the subset $\mathcal{C}'$ can be viewed as a candidate interaction effect, while the interaction from $\mathcal{S}$ can be viewed as a background contextual effect. This decomposition provides intuition for why the TCNet architecture is suitable for the choice modeling task: the assortment encoder encodes the global contextual effect from $\mathcal{S}$, and the candidate encoder encodes the local candidate interaction effect.

Theoretically, this utility decomposition also provides a guarantee on the model expressiveness of TCNet for the choice modeling task as follows.

Theorem 3.2. *Any sequential choice model can be represented by a Transformer Choice Net. Specifically, for any sequential choice model defined by (3.2), there exists a Transformer Choice Net f^{TCN} such that*

$$\mathbb{P}(i|\mathcal{C}, \mathcal{S}) = f_i^{TCN}(\mathcal{C}, \mathcal{S})$$

holds for all $i \in \mathcal{C}, \mathcal{C} \subseteq \mathcal{S}$ and $\mathcal{S} \subseteq \mathcal{N}$.

The proof is deferred to Appendix B.1. It is inspired by [LLK⁺19] which proves the approximability of the Transformer Set for the permutation invariant function family (but not the permutation equivariant needed here for choice modeling). Combined with our discussion in §3.3, we yield the following result.

Corollary 2. *Any single-choice model or multi-choice model can be represented by a Transformer Choice Net.*

3.6 Numerical Experiments

In this section, we present numerical experiments to demonstrate the performance of TCNet. Then we give visualizations and interpretations of the learned TCNet.

3.6.1 Numerical Performance on Real Datasets

We evaluate TCNet’s performance against multiple benchmark models for the three choice modeling tasks we consider in this paper: single-choice, sequential choice, and multi-choice. We emphasize that for all the models and datasets, we use a single architecture of TCNet to showcase its power of “one-size-fits-all”.

Each experiment is conducted 5 times, with training, validation, and test data randomly split at ratios of 60%, 20%, and 20%, respectively. The reported numbers are based on the average values over the test data, accompanied by the standard deviations. Further details on the datasets, the implementation, and the hyperparameter tuning are given in Appendix B.2.

Single-choice prediction. We use the cross-entropy (CE) loss (3.6) to train the TCNet and also as the performance metric to report the test performance in single-choice prediction. We compare the TCNet against several benchmark models: DeepMNL [WMZ20, SLA20], AssortNet [WCLT23], SDANet [ROS20], FATENet [PGHH22], Mixed-MNL [MT00], and DLCL [TB21]. These benchmarks represent the state-of-the-art methods of single-choice modeling. Our experiment includes the public datasets used in these existing works: Car [MT00], Sushi [Kam03], Expedia(book), Hotel [BFG09], SFwork and SFshop [KB06]; as well as two private datasets Flight, and Retail.

Sequential choice prediction. We also use the CE loss as the training objective and the performance metric in sequential choice prediction. We consider two public datasets com-

monly used in existing works: Expedia(click) and Bakery. These two datasets are originally multi-choice datasets without purchasing sequence information. We transform them into the sequential choice data by a method described in Appendix B.2. Though several algorithms are developed for sequential choice prediction, their limitations make them inapplicable with these specific datasets. As a result, for benchmarking purposes, we formulate sequential choice prediction as a single-choice prediction problem and utilize the above-mentioned benchmark models.

Specifically, for a sequential choice dataset denoted as $\mathcal{D} = \{(i_k, \{\mathcal{C}_k, \mathcal{S}_k\}), k = 1, \dots, m\}$, we drop $\mathcal{S}_k$ and recast it as a single-choice dataset: $\mathcal{D} = \{(i_k, \mathcal{C}_k), k = 1, \dots, m\}$ for both the training and testing of the single-choice benchmark models. In essence, these models predict a single-choice from the candidate set $\mathcal{C}$ as the assortment and omits the original assortment $\mathcal{S}$, which may cause the information from what have been chosen unavailable (i.e., the assortment context) and only focus on modeling the candidates effect.

To assess the implications of overlooking the assortment context, we train two variants of the Transformer Choice Net: TCNet(Var) and TCNet, where "Var" denotes the variant version. The former is trained as the single-choice model based on the above method while the latter is trained by minimizing the CE loss (3.6) over the original sequential choice dataset.

Multi-choice prediction. We adopt the F1 score loss as the performance metric for multiple choices prediction, in line with the approach in [PGHH22]. Given the dataset $\mathcal{D} = \{(\mathcal{B}_k, \mathcal{S}_k), k = 1, \dots, m\}$, the F1 score loss is defined by

$$L_{F1} = 1 - \frac{2}{m} \sum_{k=1}^m \frac{|\hat{\mathcal{B}}_k \cap \mathcal{B}_k|}{|\hat{\mathcal{B}}_k| + |\mathcal{B}_k|}.$$

Here, $\hat{\mathcal{B}}_k$ represents the algorithm's predicted multiple choices. This loss provides a balanced measure of model performance on imbalanced datasets, which is the case for most choice modeling data (far fewer chosen item(s) than unchosen ones).

We focus solely on the Expedia dataset here. The Bakery dataset, due to its consistent assortment and unvarying item features, results in identical multi-choice predictions across all

samples when deterministic prediction methods are employed which renders it unsuitable for algorithmic comparison.

As discussed in §3.2, models designed for multiple choices become computationally expensive with increasing assortment sizes. Following the strategy in [PGHH22], we adopt several single-choice models for multi-choice prediction: if an algorithm can produce interim utilities, $u_i^{\mathcal{S}}$ for each item i within assortment $\mathcal{S}$, then the item’s choice probability is $\frac{\exp(u_i^{\mathcal{S}})}{1+\exp(u_i^{\mathcal{S}})}$. And resultant predicted multiple choices are

$$\left\{ i \in \mathcal{S} : \frac{\exp(u_i^{\mathcal{S}})}{1 + \exp(u_i^{\mathcal{S}})} > \mu \right\} \quad (3.9)$$

Here, the threshold μ is a tunable hyperparameter. In this case, the training loss remains the CE loss, yet is applied to each item’s choice outcome independently. We use TCNet to denote the TCNet trained by the method discussed in §3.3.2 and use TCNet(Var) as the one trained by the above method.

Task	Data	TCNet	TCNet(Var)	DeepMNL	AssortNet	SDANet	FATENet	Mixed-MNL	DLCL
Sin.	Car	1.517 ±0.026	-	1.578±0.013	1.585±0.032	1.517 ±0.026	1.521±0.024	1.579±0.029	1.517 ±0.025
	Sushi	1.747 ±0.011	-	1.789±0.007	1.914±0.016	1.747 ±0.013	1.750±0.010	1.764±0.008	1.763±0.007
	Expedia(book)	2.314 ±0.021	-	2.408±0.014	2.389±0.021	2.328±0.020	2.360±0.019	2.351±0.020	2.340±0.017
	Hotel	0.778 ±0.008	-	1.321±0.041	0.851±0.008	0.778 ±0.006	0.803±0.006	0.882±0.004	0.885±0.003
	SFwork	1.539 ±0.031	-	1.902±0.167	1.667±0.026	1.542±0.027	1.570±0.033	1.664±0.022	1.683±0.014
	SFshop	0.821 ±0.006	-	1.265±0.140	0.928±0.024	0.821 ±0.010	0.831±0.015	0.909±0.022	0.894±0.011
	Flight	0.412 ±0.007	-	2.819±0.051	2.751±0.049	0.427±0.007	0.477±0.006	0.476±0.005	0.477±0.006
	Retail	0.982 ±0.021	-	1.429±0.073	1.032±0.008	0.992±0.022	0.996±0.023	0.997±0.023	0.997±0.023
Seq.	Bakery	2.738±0.024	2.710 ±0.029	3.817±0.006	2.767±0.016	2.792±0.009	3.808±0.006	3.557±0.001	3.811±0.006
	Expedia(click)	2.863 ±0.017	2.890±0.023	2.913±0.020	2.922±0.015	3.067±0.023	3.074±0.023	2.924±0.024	2.937±0.012
Mul.	Expedia(click)	0.576 ±0.005	0.583±0.003	0.659±0.003	0.653±0.005	0.593±0.002	0.639±0.005	-	-

Table 3.2: Test performances ($\downarrow$) on three tasks: single-choice (Sin.), sequential choice (Seq.), and multi-choice (Mul.).

Results. The performances on the test sets across all the experiments are summarized in Table 3.2. The Transformer Choice Net (TCNet) exhibits superior performance across all three tasks and datasets. One interesting observation is about the sequential choice prediction on the Bakery data: the TCNet(Var), which is trained without feeding the assortment information outperforms the standard TCNet, which enjoys the assortment information. One explanation is that the uniformity of assortments in all Bakery data samples allows an implicit encoding of this invariant assortment information. Conversely, in the Expedia(click) dataset, where

assortments largely vary across samples, the provision of the additional assortment information appears beneficial, enhancing the performance of the standard TCNet. Also, as noted in the last section, the TCNet has a comparable number of parameters as the benchmark networks, so its advantage should be attributed to its explicit modeling of the item interaction.

3.6.2 Probing into the Representation of TCNet

We train a TCNet on the Bakery dataset which contains a total of 50 items, including food (e.g., cakes with different flavors, and snacks like cookies) and drinks (e.g., different coffees). The original dataset contains 4 item features. However, we train the model without these features and only use a one-hot encoding for each item. We then extract the latent features of each product from the assortment encoder and do linear probing [Bel22] of the model through running linear logistic regression for two tasks: (1) identify cakes from snacks; (2) identify drinks from food. Due to the limited space, we only show the first here and defer the second to the Appendix.

Figure 3.2 showcases our results: Figure 3.2a shows the predicted probabilities to be the cake, and Figure 3.2b uses t-SNE [VdMH08] to visualize the top-10 latent features with largest absolute coefficients from the logistic regression. Both figures are based on the testing samples. We note that the latent representation indeed has the ability to identify Cake from Snack, although this is not explicitly included as the feature (only implicitly encoded in the choice). This shows that by learning the choice model from customer behaviors, the latent features can uncover useful properties of items, such as their categories and characteristics.

3.6.3 Attention Visualization

The attention scores in TCNet across various attention sub-layers reveal the degree of "attention" allocated by the model to each item (whether from an assortment or candidates set) while processing a specific item of interest. Essentially, these scores give insights into the influences and correlations (such as substitutes) of items on the final choice prediction. Moreover, they

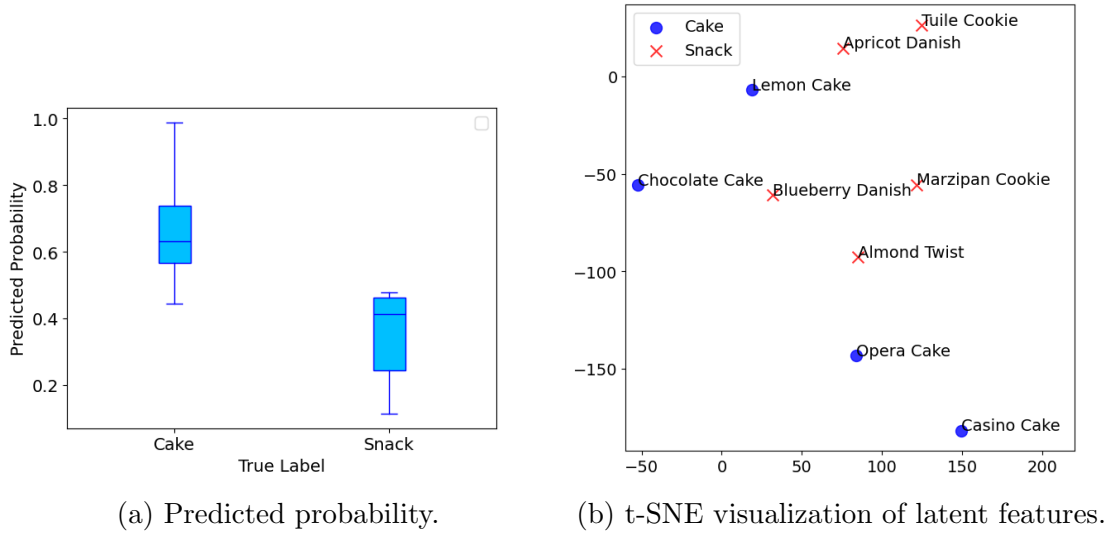


Figure 3.2: Probing the latent representation of TCN.

can help to verify whether the model is concentrating on the important items as we expect. In this subsection, we visualize the attention scores from a trained TCNet as an example to demonstrate how we can understand and explain them.

The TCNet is again trained on the Bakery dataset. Figure 3.3 shows two sets of self-attention scores (Figure 3.3a and 3.3b) and two cross-attention scores (Figure 3.3c and 3.3d) from the candidates encoders. We make the following observations:

Comparing Figures 3.3a and 3.3b, we can see Cake 1 and Cake 2 put extremely high weights on the lemonade in Figure 3.3b in contrast to the Cake 3 offered in Figure 3.3a. This aligns with one’s intuition when making the choice: Lemonade, as a drink, represents a new category differing from the other candidates and can impose a substantial influence on the final choice. For instance, a thirsty customer might randomly choose one cake in Figure 3.3a but will definitely choose the lemonade when it is offered as in Figure 3.3b.

In Figures 3.3c and 3.3d, the first observation is consistent with the above: all three cakes assign substantial weights to Coffee 1, which is a drink, in both figures. Another observation is on the weight of Coffee 1: its weight on Coffee 2 in Figure 3.3c is double of that on the Pie in Figure 3.3d. Given that both Coffee 2 and Pie are purchased items, this implies that buying Coffee 2 has a greater influence on choosing Coffee 1 than purchasing the Pie. This is reasonable, as consumers typically do not buy two coffees but may purchase a coffee alongside

a pie.

In summary, examining the attention scores enables us to validate whether the model adheres to our common sense and allows us to get insights into the relationships between items.

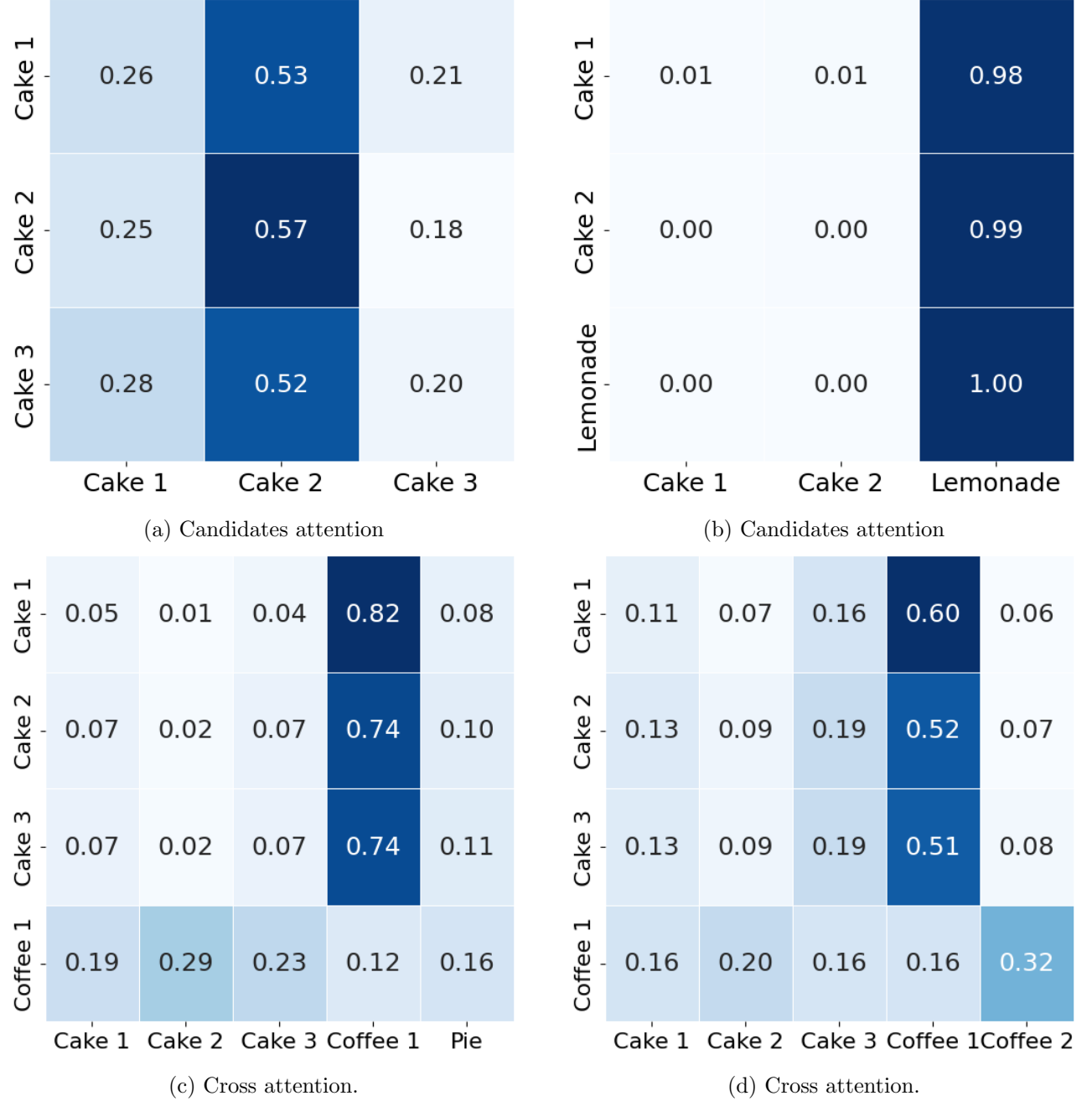


Figure 3.3: Candidates encoder's (self) attention and cross-attention scores. For the second row, the side items are from candidates $\mathcal{C}$ and the bottom items are from assortment $\mathcal{S}$. Each row indicates the normalized attention scores for the side item.

3.7 Conclusion and Future Directions

The existing literature on single-choice model, sequential choice model, and multi-choice model have been largely separate from each other: the current sequential choice models have their own limitations, and the current multi-choice models become impractical when dealing with a large assortment.

While the research on single-choice models is vast, the literature on predicting sequential and multi-choice is noticeably less developed. Our paper introduces the Transformer Choice Net as a unified neural network framework capable of addressing all three types of choice behaviors. Notably, it is the first Transformer-based architecture tailored for sequential choice(s) prediction tasks. We prove its universal representation capacity and, through extensive numerical experiments, demonstrate its superior performance over several benchmarks. We conclude with the following future directions.

Assortment optimization. One important downstream application of choice prediction is assortment optimization, which optimizes the offered assortment to maximize the expected revenue for the seller. Given the Transformer Choice Net’s strong prediction performance, an intriguing avenue to explore would be the optimization of assortment offerings for a trained Transformer Choice Net model.

Exploring alternative Transformer-based structures. The Transformer architecture has increasingly shown its significance within the deep learning realm, showcasing its adaptability across diverse tasks. While the Transformer Choice Net represents an initial endeavor in integrating the Transformer framework into choice modeling, we believe there is still large potential in exploring other Transformer-inspired architectures.

Chapter 4

Operational Management Generative Pre-trained Transformer

4.1 Introduction

We study the problem of *sequential decision making*, where a decision maker interacts iteratively with an unknown environment. At each time period $t = 1, \dots, T$, the decision maker observes a context vector, makes a decision, and receives a reward and possibly other feedback. The decision maker's objective is to maximize cumulative reward. Examples of such problems include the network revenue management problem with unknown demands [Jas15], the newsvendor problem with unknown demand distribution [HR09], and the dynamic pricing problem with unknown demand function [DB15].

A central challenge in these problems is balancing learning about the environment (exploration) and earning rewards (exploitation). This is often referred to as the exploration-exploitation trade-off. Most existing works tackle this problem through the *online learning* paradigm, where they assume specific structures in the underlying problem to design decision algorithms or policies. These algorithms often come with theoretical performance guarantees and can be optimal under certain conditions and definitions. However, this approach has two limitations: (i) it relies on assumptions of the underlying problem, and more relaxed assumptions often

require the design of more complex algorithms, and (ii) it cannot easily incorporate the decision maker’s prior knowledge about the environment into the decision making process.

Alternatively, in this work, we propose a *supervised learning* paradigm to solve sequential decision making problems, which leverages generative artificial intelligence (GenAI) techniques to train a deep learning model. We re-frame the decision making problem as a sequence modeling task, where the goal is to predict optimal actions based on historical data. Specifically, by utilizing the decision maker’s prior knowledge, we can generate pre-training data and train a transformer/GPT-based neural network, named the *Operational Management Generative Pre-trained Transformer* (OMGPT), to sequentially predict the next optimal action. The key contributions of this work are as follows:

- We propose a new supervised learning paradigm for sequential decision making problems using GenAI techniques. Unlike existing methods, our paradigm does not assume any specific problem structure and can incorporate the decision maker’s prior knowledge. Specifically, we design a GPT-based architecture, OMGPT, with a unique pre-training scheme for solving sequential decision making problems.
- In numerical experiments, OMGPT (i) consistently outperforms well-established benchmarks across multiple operational decision making tasks, (ii) demonstrates strong generalization ability when the tested environments differ from the decision maker’s prior knowledge, (iii) handles non-stationary environments and multiple environment types, showcasing its strong capability, and (iv) can also provide valuable side information beyond predicting optimal actions, such as demand forecasting.
- Analytically, we study the behavior of OMGPT as a decision algorithm, showing that (i) asymptotically, it behaves as a Bayes-optimal decision maker, (ii) the pre-training loss functions can serve as surrogates for regrets, which are defined as the reward gaps between the OMGPT and the optimal decisions, and (iii) prediction errors due to finite pre-training samples can provide an inherent exploration mechanism, leading to performance guarantees.

The remainder of the paper is organized as follows. Section 4.2 presents a brief review of the literature; Section 4.3 gives the formal definition of the problem, along with performance measures and examples; Section 4.4 proposes the supervised learning paradigm using OMGPT, introducing its architecture, pre-training, and testing. Section 4.5 analyzes pre-trained OMGPT as a decision algorithm, focusing on its working mechanism and performance analysis. Finally, Section 4.6 provides numerical experiments to evaluate its performance and capability with interpretation results, and Section 4.7 concludes the paper.

4.2 Literature Review

4.2.1 Sequential Decision Making

Sequential decision making encompasses many well-studied problems in operations management, including revenue management [TVR06, GT19], dynamic pricing [DB15, KZ14], inventory management [Axs15], queuing control [CY⁺01], etc. Below, we briefly review two of them as examples:

- **Dynamic pricing** involves setting prices to discern the underlying revenue function and make revenues, typically defined as the product of demand and price. Algorithms that rely purely on the exploitation of historical data tend to be suboptimal, and they may converge to non-optimal prices with positive probability [KZ14]. Therefore, random exploration is crucial to more accurately understand the demand function and to avoid settling on suboptimal prices. Various algorithms address this exploration-exploitation trade-off to achieve near-optimal regret under different settings. See [HKZ12, DBZ14, BZ15, CLPL20, WTL21] for examples of such algorithms, and [DB15] for a comprehensive overview of dynamic pricing.
- **Inventory management** (or the newsvendor problem) requires the decision maker to decide on the quantity of inventory to order to meet a random demand with an unknown distribution. There are various settings, such as perishable inventory, where the unsold

inventory expires at the end of each period, and non-perishable inventory, where the unsold inventory can be carried to the next period. Additionally, the achieved demand can be censored or uncensored. In an uncensored setting, the decision maker observes the actual demand in each period. In a censored setting, when demand exceeds inventory levels, unmet demand is not observed, leading to lost sales. In this case, the optimal decision policy must balance the exploration of observing the realized demand and the exploitation of reducing the cost, sometimes requiring higher inventory levels to gather more demand data, but not so high that exploration costs become excessive. Near-optimal policies with $O(\sqrt{T})$ regret bounds have been developed under various assumptions about lead times and demand models, as shown in works such as [BS00, GP01, ZCS20, AJ22].

4.2.2 Deep Learning in Operations Management

Another related line of literature is about using deep learning tools to address problems in operations management. For example, [BM00, AD22] apply neural networks to study customer choice behavior and predict customer decisions, and [WCLT23] develop a neural network architecture to both predict customer choice and optimize assortment decisions. Additionally, [QSQ⁺23, GBVMZ22, ONST22] develop deep learning frameworks for solving inventory control problems, [LWW⁺24] adapt deep reinforcement learning to dynamic assortment planning, [MCG24] propose using reinforcement learning for choice-based network revenue management, and [HHS23] provide a theoretical framework for analyzing theoretical guarantees in terms of excess risk bounds in newsvendor problems. [CLM22] propose a general framework that leverages deep neural networks to solve data-driven stochastic optimization problems.

The main contribution of our proposed method to this body of literature is that our neural network is trained as an algorithm, meaning it does not require any data from the testing environment and can operate across different environments. In contrast, while the works mentioned above may also tackle sequential decision making problems like inventory management, they typically rely on collecting data from the testing environment and learning a policy that works only in that specific environment. In this sense, our model is more aligned with generative arti-

ficial intelligence: it can sequentially generate different (good) decisions for various (unknown) environments.

4.2.3 Transformer for Reinforcement Learning

The earliest efforts to apply transformers and supervised learning approaches to reinforcement learning can be found in the area of offline reinforcement learning [JLL21, CLR⁺21]. In this paradigm, offline reinforcement learning is effectively converted into a supervised learning problem through pre-training, where actions are predicted based on trajectories in the offline dataset. During the evaluating/testing phase, actions are generated by the pre-trained transformer, conditioned on the so-called “return-to-go”, which is often chosen as the cumulative reward from “good” trajectories in the dataset. This approach allows the transformer to retrieve information from training data trajectories with similar return-to-go values and perform imitation learning from high-reward offline trajectories. Building on this idea, several works further examine the validity of return-to-go conditioning and explore its mechanisms [EEKL21, ADG⁺22, BBB⁺22, LBM23], and proposed alternative methods to improve or extend the approach to other settings [LWO⁺22, BFBNB24, ZZG22, WWH24].

The most closely related work to ours is [LXP⁺24], which also proposes utilizing transformers to predict optimal actions for reinforcement learning problems. However, our proposed paradigm differs in several key aspects, including the studied underlying tasks, the data generation/pre-training step, and the neural network architecture. These differences are tailored specifically to operational decision makings and provide us with the ability to analyze the working mechanisms and performance. We elaborate more discussions in Appendix C.1.1.

4.3 Problem Setup

4.3.1 Sequential Decision Making

In this section, we introduce a general formulation for sequential decision making problems. A decision maker makes decisions sequentially in an environment with a *horizon* T . At each time $t = 1, \dots, T$, the decision maker first observes a *context* vector $X_t \in \mathcal{X}$ (also known as covariates or side information), takes an *action/decision* $a_t \in \mathcal{A}$, and then receives an *observation* $O_t \in \mathcal{O}$, where $\mathcal{X}$, $\mathcal{A}$, and $\mathcal{O}$ denote the spaces of contexts, actions, and observations, respectively. Specifically, at each time $t = 1, \dots, T$,

1. The decision maker observes a context $X_t \in \mathcal{X}$.
2. Based the historical information H_t (including the context X_t) available at t ,

$$H_t := (X_1, a_1, O_1, \dots, X_{t-1}, a_{t-1}, O_{t-1}, X_t),$$

the decision maker makes a (non-anticipatory) decision/action $a_t \in \mathcal{A}$ following an *algorithm* f , i.e., $a_t = f(H_t)$.

3. Next, the decision maker receives an observation $O_t \in \mathcal{O}$ as a random outcome of X_t and a_t .

We remark that the algorithm f is very general in that it takes the whole history which is of a varying length as its input. Also, unlike the development of many online learning algorithms the implementation and analysis of which require the horizon T to be large, our model still works even when $T = 1$, i.e., a static decision making problem.

Environment Dynamics.

Throughout the paper, we assume contexts X_t 's are independently and identically distributed

(i.i.d.) across t and observations O_t 's follow a conditional distribution given X_t and a_t

$$X_t \stackrel{\text{i.i.d.}}{\sim} \mathbb{P}_\gamma(\cdot), \quad O_t \sim \mathbb{P}_\gamma(\cdot | X_t, a_t), \quad (4.1)$$

where the *environment* parameter γ encapsulates all the parameters for these two probability distributions. In particular, the parameter γ is unknown to the decision maker.

At each time t , the decision maker collects a random *reward* $R_t = R(X_t, a_t)$ which depends on X_t , a_t , and also some possible exogenous randomness. The observation O_t includes R_t as its first coordinate and it may or may not include other observed information as well. The goal of the decision maker is to maximize the cumulative reward over the decision horizon

$$\sum_{t=1}^T R_t = \sum_{t=1}^T R(X_t, a_t).$$

We define the expected reward by $r(X_t, a_t) = \mathbb{E}[R(X_t, a_t) | X_t, a_t]$. Specifically, this reward function $r(\cdot, \cdot)$ is unknown to the decision maker and it also depends on the underlying environment γ .

Performance Metric.

The performance of the decision maker's algorithm f is measured by the notion of *regret*, which is defined by

$$\text{Regret}(f; \gamma) := \mathbb{E} \left[\sum_{t=1}^T r(X_t, a_t^*) - r(X_t, a_t) \right] \quad (4.2)$$

where the action a_t^* is the optimal action that maximizes the expected reward

$$a_t^* := \arg \max_{a \in \mathcal{A}} r(X_t, a) \quad (4.3)$$

and the optimization assumes the knowledge of the underlying environment, i.e., the parameter γ . In the regret definition (4.2), the expectation is taken with respect to the underlying probability distribution (4.1) and possible randomness in the decision algorithm f . We use the arguments f and γ to reflect the dependency of the regret on the algorithm f and the

environment γ . Also, we note that the benchmark is defined as a dynamic oracle where a_t^* may be different over time.

4.3.2 Examples

Here we provide two well-studied sequential decision making problems as examples.

- **Dynamic pricing:** The context vector X_t describes the market-related information at time t . Upon the reveal of X_t , the decision maker takes the action $a_t \in \mathcal{A} \subset \mathbb{R}^+$ as the pricing decision. Then the decision maker observes the demand $D_t = d(X_t, a_t) + \epsilon_t$ where ϵ_t is some zero-mean noise random variable and $d(\cdot, \cdot)$ is a demand function unknown to the decision maker. The random reward $R(X_t, a_t) = D_t \cdot a_t$ is the revenue collected from the sales at time t , and the observation $O_t = (R_t, D_t)$. The expected reward $r(X_t, a_t) = \mathbb{E}[R(X_t, a_t)|X_t, a_t] = d(X_t, a_t) \cdot a_t$. The parameter γ governs the generation of X_t 's and also contains information about the demand function d and the noise distribution.
- **Newsvendor problem:** As the dynamic pricing problem, the context vector X_t describes the market-related information at time t . Upon its reveal, the decision maker takes the action a_t which represents the number of inventory prepared for the sales at time t . Then the decision maker observes the demand $D_t = d(X_t) + \epsilon_t$ where ϵ_t is some noise random variable and $d(\cdot)$ is a demand function unknown to the decision maker. The random reward is the negative cost $R(X_t, a_t) = -h \cdot (a_t - D_t)^+ - l \cdot (D_t - a_t)^+$ where $(\cdot)^+$ is the positive-part function, h is the left-over/overage cost, and l is the lost-sale/underage cost. The observation $O_t = (R_t, D_t)$ and the expected reward $r(X_t, a_t) = \mathbb{E}[R(X_t, a_t)|X_t, a_t]$ with the expectation taken with respect to ϵ_t . The parameter γ governs the generation of X_t 's and encodes the demand function d , the noise distribution, and the (possibly known) cost parameters h and l .

4.4 Supervised Learning for a (Good) Algorithm Using OMGPT

In the literature on sequential decision making problems, a common paradigm, often known as *online or reinforcement learning*, is to assume some structure for the underlying environment γ and/or the reward function r . Based on these assumptions, decision algorithms are then accordingly *designed*. This means that different algorithms are crafted for different sets of assumptions. In contrast, this work proposes a paradigm for *learning* policies instead. In this section, we introduce the Operations Management Generative Pre-trained Transformer (OMGPT) as a method for solving sequential decision making problems. It is important to note that OMGPT is not merely a model or an algorithm; rather, it represents a paradigm for potentially solving all sequential decision making problems using a supervised learning approach. Similar to a common supervised learning task like regression, this approach involves optimizing a family of GPT models to predict the optimal actions a_t^* based on the historical data, and the optimization is based on minimizing the empirical training loss over the generated training samples.

4.4.1 A (Good) Algorithm as a Sequence Function

Recall that at each time step t , the decision a_t made by an algorithm f is given by $a_t = f(H_t)$. Thus, an algorithm f can be viewed as a *sequence function* whose input is H_t , a sequence of any length. To illustrate this, we first consider the case where the environment γ is known and given. Ideally, we aim to find a sequence function f^* such that for any $t = 1, \dots, T$ and any possibly sampled H_t as defined in Section 4.3,

$$f^*(H_t) = a_t^*,$$

i.e., f^* can predict the optimal action a_t^* based on any possible H_t at each t . Under the assumption the environment γ and thus the reward function r is known, this can be satisfied

by setting $f^*(H_t) = \arg \max_{a \in \mathcal{A}} r(X_t, a)$ (recall X_t is included in H_t).

More generally, we can formulate a sequence function optimization task aimed at predicting optimal actions. Given a candidate family $\mathcal{F}$ of sequence functions, our goal is to find a sequence function $\hat{f} \in \mathcal{F}$ that minimizes the divergence between the predicted action $\hat{f}(H_t)$ and the target (optimal) action a_t^* :

$$\hat{f} = \arg \min_{f \in \mathcal{F}} \sum_{t=1}^T \mathbb{E}_{(H_t, a_t^*)} [l(f(H_t), a_t^*)], \quad (4.4)$$

where the expectations are with respect to (H_t, a_t^*) 's (recall a_t^* depends on X_t in H_t) and $l(a, a')$ is the loss function to measure the divergence between a and a' : it should be non-negative and only be zero when $a = a'$. For example, the L_2 loss (i.e., $l(a, a') = \|a - a'\|_2$) or the absolute (L_1) loss (i.e., $l(a, a') = \|a - a'\|_1$).

In an environment where the reward function is “smooth” with respect to the action—meaning that the single-period regret incurred at time t decreases as the action a_t approaches the optimal action a_t^* —a sequence function $\hat{f}$ which is optimal in minimizing the prediction loss for the optimal actions should also perform well in terms of regret. In particular, if $f^* \in \mathcal{F}$, we have $\hat{f} = f^*$ and $\text{Regret}(f^*; \gamma) = 0$.

However, in sequential decision making problems, the environment γ is unknown, making the formulation in (4.4) invalid. To address this, one approach is to consider a distribution $\mathcal{P}_\gamma$ such that $\gamma \sim \mathcal{P}_\gamma$. Here, $\mathcal{P}_\gamma$ can be viewed as the decision maker's prior knowledge or belief about the environment. The existence of such prior knowledge is a typical assumption in operational sequential decision making problems (e.g., [BÇRS09, HKZ12, FSLW18]). In this case, we define the expected loss of the decision function f under the environment specified by γ as

$$L(f; \gamma) := \sum_{t=1}^T \mathbb{E}_{(H_t^{(\gamma)}, a_t^{(\gamma)*})} [l(f(H_t^{(\gamma)}), a_t^{(\gamma)*})] \quad (4.5)$$

where $(H_t^{(\gamma)}, a_t^{(\gamma)*})$'s are derived with respect to the environment γ and f . Then we can formu-

late the sequence function optimization task as:

$$\hat{f} = \arg \min_{f \in \mathcal{F}} \mathbb{E}_{\gamma \sim \mathcal{P}_\gamma} [L(f; \gamma)]. \quad (4.6)$$

The redefined $\hat{f}$ should approximate the optimal actions for each possible environment $\gamma \sim \mathcal{P}_\gamma$ (as much as possible) to become the minimizer of (4.6), thereby handling unknown environments. However, there are still several challenges for applying (4.6) to find a (good) algorithm $\hat{f}$:

1. **Choosing $\mathcal{F}$:** We need $\mathcal{F}$ to include a sequence function $\hat{f}$ that can accurately predict the optimal actions for different and unknown γ 's. However, even with a known γ , mapping a high-dimensional and unstructured input H_t approximately to a_t^* is complex, requiring $\mathcal{F}$ to have a large capacity. Additionally, $\mathcal{F}$ must be a family of sequence functions, which can handle variable-length input sequences.
2. **Solving (4.6):** There are multiple challenges in solving (4.6). First, this stochastic optimization problem involves complex randomness in both the environment sampling $\gamma \sim \mathcal{P}_\gamma$ and the environment dynamics H_t 's. Second, the dynamics H_t 's themselves depend on the decision variable f , and this dependence is also complex. Third, optimizing over the complex function family $\mathcal{F}$ is difficult. These difficulties collectively make it challenging to even approximately solve (4.6).
3. **Understanding $\hat{f}$:** It is crucial to understand the algorithm we intend to use, particularly to avoid extreme decisions and to evaluate its performance. Even if we assume that (4.6) can be solved, we still need to know what the resulting algorithm $\hat{f}$ will be and how it performs in terms of regret.

In the following subsections, we will address these challenges.

4.4.2 Preliminaries on (Original) GPTs and OMGPT

In this work, we select the GPT architecture, which stands for Generative Pre-trained Transformer [RNS⁺18], as the sequence function family $\mathcal{F}$ to learn a good algorithm. GPT, a variant of the transformer architecture for deep learning [VSP⁺17], was originally designed as a language model due to its two key properties: high expressiveness and the ability to model sequences. Specifically, GPT can be efficiently scaled to billions of parameters, providing a high degree of expressiveness and the capacity to capture complex patterns, such as those in natural language. Additionally, its structure allows it to handle inputs of varying lengths, making it well-suited for sequence modeling. These properties align precisely with the requirements for the function family $\mathcal{F}$ in (4.6). Here, we provide a brief introduction to the original GPT architecture, followed by an overview of the OMGPT architecture, with detailed information deferred to Appendix C.1.1.

Attention Mechanism. The key component of the GPT architecture is the *attention mechanism/function*. The input to an attention function is a set of embedding vectors $\{z_\tau\}_{\tau=1}^t$, all sharing the same dimension, and the output is a set of processed embedding vectors with the same dimension, denoted by $\{\tilde{z}_\tau\}_{\tau=1}^t$. Let $Z = [z_1, z_2, \dots, z_t]^\top$ be the matrix composed of the embedding vectors $\{z_\tau\}_{\tau=1}^t$. The attention function is then defined as

$$f^{\text{Att}}(Z; W_Q, W_K, W_V) := \text{Softmax}(Z^\top W_Q W_K Z) Z^\top W_V.$$

Here, the Softmax function is applied to each row of the matrix $Z^\top W_Q W_K Z$, and W_Q, W_K, W_V are trainable square parameter matrices, commonly referred to as the query weight, key weight, and value weight, respectively, which have the same dimensions as the embedding vectors z_τ 's.

Intuitively, the attention mechanism connects each element from the input sequence using the Softmax function and the weight matrices, allowing each output vector to embed information from the entire sequence. Additionally, the attention mechanism supports variable-length inputs.

GPT architecture. The GPT architecture is built with multiple layers that share the same

structure. Each layer consists of an attention function f^{Att} followed by an element-wise, fully connected feed-forward network sub-layer. Each embedding vector $\tilde{z}_\tau$ from the attention function, for $\tau = 1, \dots, t$, is individually processed through this sub-layer, which uses trainable matrices W_1, W_2 and vectors b_1, b_2 , all matching the input dimension of $\tilde{z}_\tau$:

$$z'_\tau = W_2 \phi(W_1 \tilde{z}_\tau + b_1) + b_2, \quad (4.7)$$

where ϕ is an activation function to add the non-linearity (like ReLU or GELU [HG16]) and the outputs z'_τ 's form the input sequence for the next layer. By stacking multiple layers, GPT gains the capacity to “understand” more complex relationships within the data.

The GPT architecture also includes a layer at the beginning to transform input elements (e.g., words/tokens in natural language or actions in sequential decision making tasks) into embedding vectors, and an output layer at the end to convert these embedding vectors (typically the last one) into the desired output elements (e.g., tokens or actions). These layers can be structured as linear layers similar to (4.7), with dimensions of trainable parameters adjusted to match the inputs and outputs.

OMGPT. Unlike the original GPT, which is designed for natural language modeling with a single type of input and output (words/tokens), OMGPT is tailored for sequential decision making tasks and must handle different types of elements as input (actions, contexts, observations) and output only actions. Therefore, modifications to the OMGPT architecture are required to accommodate these different elements. Details on these modifications are provided in Appendix C.1.1. Additionally, other neural network architectures, such as Long Short-Term Memory (LSTM) [Hoc97], can also be applied as $\mathcal{F}$, as they are also capable of sequence modeling and can handle inputs of varying lengths. In Appendix C.4.1.3, we evaluate the performance of OMGPT by replacing the transformer with LSTM, and the results demonstrate that the transformer performs better. This finding is consistent with observations from the literature on natural language modeling/large language models [ZZL⁺23].

4.4.3 Supervised Learning for OMGPT

We denote the OMGPT architecture by TF_θ , where θ encapsulates all the trainable parameters (we assume that the hyperparameters, such as the number of layers and the dimension of the embedding space, are given and not included in θ). We assume there exists a parameter space Θ for θ and set $\mathcal{F} = \{\text{TF}_\theta\}_{\theta \in \Theta}$. The sequence function learning task (4.6) for the OMGPT architecture can then be reformulated as:

$$\min_{\theta \in \Theta} \mathbb{E}_{\gamma \sim \mathcal{P}_\gamma} [L(\text{TF}_\theta; \gamma)]. \quad (4.8)$$

As mentioned earlier, this stochastic optimization problem is challenging to solve (even approximately) due to the complex randomness in γ and H_t , as well as the high dimensionality of the parameter space Θ , which can consist of millions or even billions of dimensions. To address this, we can adopt a supervised pre-training approach, similar to the training of large language models (e.g., ChatGPT), to learn a parameter $\hat{\theta}$ that serves as an approximate solution to (4.8).

4.4.3.1 Pre-training Data Generation

The supervised pre-training first constructs a pre-training dataset

$$\mathcal{D}_{\text{PT}} := \left\{ \left(H_1^{(\gamma_i)}, a_1^{(\gamma_i)*} \right), \left(H_2^{(\gamma_i)}, a_2^{(\gamma_i)*} \right), \dots, \left(H_T^{(\gamma_i)}, a_T^{(\gamma_i)*} \right) \right\}_{i=1}^n.$$

In particular, we assume the parameter γ (see (4.1)) that governs the generation of X_t and O_t are generated from the environment distribution $\mathcal{P}_\gamma$. Then, we generate n environments denoted by $\gamma_1, \dots, \gamma_n$. There are two key aspects regarding the generation of $\mathcal{D}_{\text{PT}}$:

- **Target action** $a_t^{(\gamma_i)*}$: For this paper, we generate $a_t^{(\gamma_i)*}$ as the optimal action specified by (4.3), which requires knowledge of the underlying environment γ_i . A common property of all these sequential decision making problems is that the optimal action can be often easily or analytically computed from the generated γ_i 's. Alternatively, $a_t^{(\gamma_i)*}$ can also be

generated based on expert algorithms such as Thompson sampling or upper confidence bound (UCB) algorithms [LS20, WTL21]. In this case, the action $a_t^{(\gamma_i)*} = \mathbf{Alg}(H_t^{(\gamma_i)})$ where $\mathbf{Alg}(\cdot)$ represents the algorithm that maps from $H_t^{(\gamma_i)}$ to the action without utilizing the knowledge of γ_i . These target actions can be viewed as “labels” to predict in the supervised learning task.

- **History** $H_t^{(\gamma_i)} = (X_1^{(\gamma_i)}, a_1^{(\gamma_i)}, O_1^{(\gamma_i)}, \dots, X_{t-1}^{(\gamma_i)}, a_{t-1}^{(\gamma_i)}, O_{t-1}^{(\gamma_i)}, X_t^{(\gamma_i)})$: Here, $X_\tau^{(\gamma_i)}$ ’s and $O_\tau^{(\gamma_i)}$ ’s are generated based on the parameter γ_i following (4.1). The action $a_\tau^{(\gamma_i)}$ is generated based on some pre-specified decision function $\tilde{f}$ in a recursive manner:

$$a_t^{(\gamma_i)} = \tilde{f}(H_t^{(\gamma_i)}), \quad H_{t+1}^{(\gamma_i)} = (H_t^{(\gamma_i)}, a_t^{(\gamma_i)}, O_t^{(\gamma_i)}, X_{t+1}^{(\gamma_i)}) \quad (4.9)$$

for $t = 1, \dots, T$. It is important to note that $a_t^{(\gamma_i)}$ can differ from the target action $a_t^{(\gamma_i)*}$. The pre-specified decision function $\tilde{f}$ can be chosen to randomly select an action or as another expert algorithm. In general, the history $H_t^{(\gamma_i)}$ generated by $\tilde{f}$ should be representative of the histories encountered during the testing phase where we will deploy the learned/pre-trained OMGPT. Intuitively, the pre-training data should resemble the testing situations to ensure that the pre-trained OMGPT generalizes well during testing. These histories can be viewed as “features” in the supervised learning task. Further discussion on the choice of $\tilde{f}$ and its influence on pre-training $\mathbf{TF}_\theta$ is provided in Appendix C.2.

Formally, we define a distribution $\mathcal{P}_{\gamma, \tilde{f}}$ that generates H_t and a_t^* as

$$X_\tau \sim \mathbb{P}_\gamma(\cdot), \quad a_\tau = \tilde{f}(H_\tau), \quad O_\tau \sim \mathbb{P}_\gamma(\cdot | X_\tau, a_\tau), \quad H_{\tau+1} = (H_\tau, a_\tau, O_\tau, X_{\tau+1}) \quad (4.10)$$

for $\tau = 1, \dots, t-1$ and a_t^* is specified by (4.3). The parameter γ and the function $\tilde{f}$ jointly parameterize the distribution $\mathcal{P}_{\gamma, \tilde{f}}$. The parameter γ governs the generation of X_t and O_t just as in (4.1). The decision function $\tilde{f}$ maps from the history to the action and it governs the generation of a_τ ’s in the sequence H_t . In this way, the pre-training data is generated with the

following flow:

$$\mathcal{P}_\gamma \rightarrow \gamma_i \rightarrow \mathcal{P}_{\gamma_i, \tilde{f}} \rightarrow \left\{ \left(H_1^{(\gamma_i)}, a_1^{(\gamma_i)*} \right), \dots, \left(H_T^{(\gamma_i)}, a_T^{(\gamma_i)*} \right) \right\}.$$

4.4.3.2 Supervised Pre-training

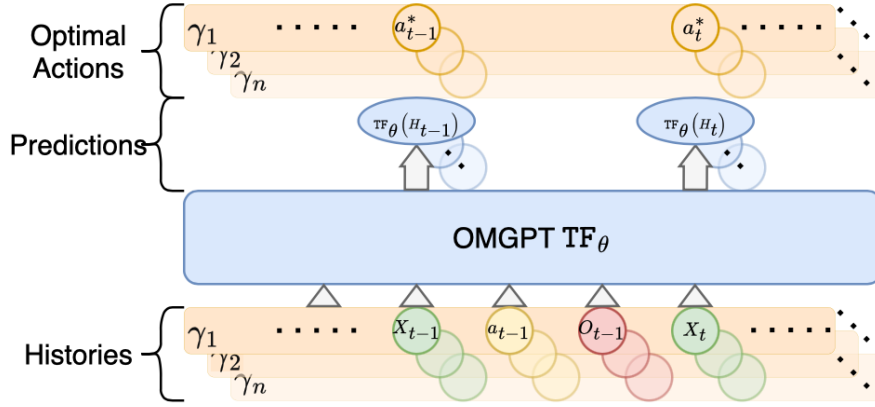


Figure 4.1: The pre-training phase of OMGPT. We first generate different environments γ_i . For each γ_i , we generate the corresponding histories and optimal actions. The histories serve as the inputs to the OMGPT, which then uses them to predict the corresponding optimal actions. The goal of the pre-training is to minimize the empirical prediction errors.

Figure 4.1 illustrates the supervised pre-training phase of OMGPT. Based on the dataset $\mathcal{D}_{\text{PT}}$, the supervised pre-training refers to the learning of the sequence function $\text{TF}_{\hat{\theta}}$ through minimizing the following empirical loss

$$\hat{\theta} := \arg \min_{\theta \in \Theta} \frac{1}{nT} \sum_{i=1}^n \sum_{t=1}^T l \left(\text{TF}_\theta \left(H_t^{(\gamma_i)} \right), a_t^{(\gamma_i)*} \right) \quad (4.11)$$

where the loss function $l(\cdot, \cdot) : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$ specifies the prediction loss. Thus the key idea of supervised pre-training is to formulate the sequential decision making task as a *sequence modeling* task that predicts the optimal action $a_t^{(\gamma_i)*}$ given the history $H_t^{(\gamma_i)}$. Further discussions on how different loss function l influences $\text{TF}_{\hat{\theta}}$ and how the pre-training loss relates to the regret are deferred to Section 4.5.

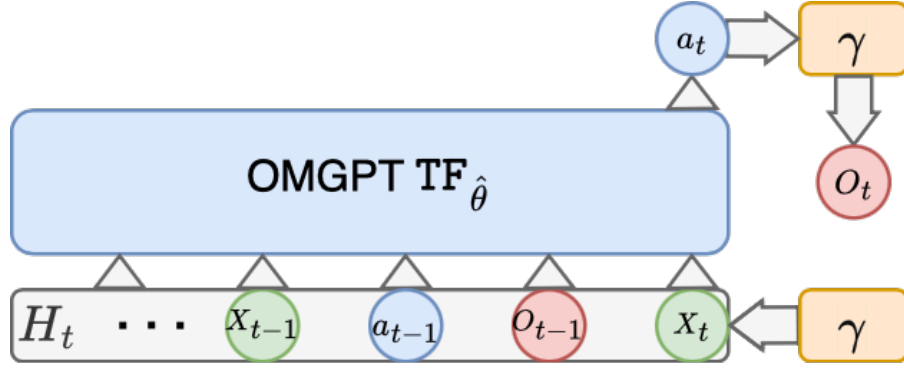


Figure 4.2: The testing/application phase of the pre-trained OMGPT. We iteratively input the history H_t into the pre-trained OMGPT $\text{TF}_{\hat{\theta}}$ to obtain an action a_t . This action is then applied to interact with the testing environment γ , generating a new observation O_t . The cycle is then repeated with the updated history H_{t+1} .

4.4.3.3 Testing

Figure 4.2 illustrates the testing (or application) phase of a pre-trained OMGPT. In the testing phase, a new environment γ (which is unknown) is sampled from $\mathcal{P}_{\gamma}$ and we apply the learned decision function $\text{TF}_{\hat{\theta}}$. The testing procedure is described by the following dynamics:

$$X_{\tau} \sim \mathbb{P}_{\gamma}(\cdot), a_{\tau} = \text{TF}_{\hat{\theta}}(H_{\tau}), O_{\tau} \sim \mathbb{P}_{\gamma}(\cdot | X_{\tau}, a_{\tau}), H_{\tau+1} = (H_{\tau}, a_{\tau}, O_{\tau}, X_{\tau+1}). \quad (4.12)$$

The only difference between the test dynamics (4.12) and the training dynamics (4.10) is that the action a_t (or a_{τ}) is generated by $\text{TF}_{\hat{\theta}}$ instead of the pre-specified decision function $\tilde{f}$.

4.4.4 Difference from Online/Reinforcement Learning

In this subsection, we discuss the key differences between the proposed OMGPT paradigm and traditional online/reinforcement learning approaches for sequential decision makings in the following aspects:

- **Learning-based Paradigm:** OMGPT represents a learning-based approach, while most online/reinforcement learning algorithms are manually designed. Through a supervised pre-training phase, OMGPT learns an algorithm, $\text{TF}_{\hat{\theta}}$, that can function like a decision algorithm when interacting with an unknown environment, and can even achieve better

performances than well-established online algorithms, as demonstrated in Section 4.6. This means OMGPT discovers new algorithms during pre-training and has the potential to offer valuable insights and guidance for the design of future online learning algorithms.

- Utilization of Prior Knowledge and Pre-training Data:** Many online/reinforcement learning algorithms rely on setting hyperparameters and can only utilize limited prior knowledge of the testing environments, such as parameter bounds. They only learn the optimal actions through the data collected from the testing task. This restricts their ability to incorporate more detailed prior knowledge, such as a prior distribution of the testing environment. While some algorithms, like Thompson Sampling, can use a prior distribution, embedding more complex prior knowledge (e.g., demand functions can change unpredictably during the horizon) can be challenging. OMGPT, however, can handle such complex prior knowledge by generating pre-training samples based on simulations. This ensures that the pre-trained $\text{TF}_{\hat{\theta}}$ can effectively leverage the prior knowledge when making decisions based on the data collected from the testing task. Numerical results demonstrating this advantage are provided in Appendix C.4.1.2.
- Flexibility:** Unlike online algorithms designing, which are based on specific assumptions and often require redesign when those assumptions are not met, OMGPT serves as a universal paradigm. Regardless of the assumptions or prior knowledge at hand, the same framework can be applied to pre-train an OMGPT model and utilize it effectively, thanks to its large capacity. This universality means OMGPT can work across various problems without relying on predefined analytical model structures or assumptions. Our numerical experiments in Section 4.6 demonstrate its strong versatility across various problems and structures.

4.4.5 Discussions and Extensions

We provide the following discussions and extensions to the proposed supervised learning paradigm.

4.4.5.1 Extending to Longer Testing Horizon

Our paradigm can be generalized to a longer testing horizon (beyond the length of the pre-training data) by introducing a *context window*. Specifically, we define a context window size W and limit the input to the last $\min\{W, t\}$ timesteps of the input sequence H_t to predict a_t^* during both the pre-training and testing phases. By considering only the latest $\min\{W, t\}$ timesteps, we enable the model to handle testing sequences of any length, even when $t > T$. This technique is commonly used in the literature (e.g., [CLR⁺21]).

In general, a larger context window provides more information for predicting the optimal action but requires more training data, longer training time, and potentially a larger model size. We provide numerical performance results in Section 4.6, and believe that its theoretical analysis (e.g., determining the optimal window size for a random testing horizon) is an important and interesting future research direction.

4.4.5.2 Extending to State-dependent Context

Although in Section 4.3 we assume that the contexts X_t 's are i.i.d. generated, the OMGPT paradigm can be extended to handle Markov Decision Processes (MDPs), where X_t encapsulates the state of the decision process and is sampled through a Markov process:

$$X_t \sim \mathbb{P}(\cdot | X_{t-1}, a_t).$$

For this extension, no changes to the paradigm are required to accommodate such non-i.i.d. X_t 's, and we can follow the same procedure to generate pre-training data, as well as to conduct pre-training and testing. In Section 4.6, we provide numerical experiments that demonstrate the performance of OMGPT in common operational decision making problems involving states, including queuing control and revenue management.

4.4.5.3 Extending to Observation Prediction

In addition to predicting optimal actions, OMGPT can be pre-trained to also predict observations, O_t , with some modifications. These predictions can be valuable for decision makers in certain scenarios. For example, in a dynamic pricing task, while the main target is to find the optimal prices, the decision maker may also need to predict the resulting demands to plan manufacturing or storage.

To enable it, an additional output layer is added to OMGPT to predict observations, while the original output layer continues to predict optimal actions as usual. Specifically, with this added output layer, OMGPT becomes a sequence function that maps a sequence H to both an action $a \in \mathcal{A}$ and an observation $O \in \mathcal{O}$, i.e., $\text{TF}_\theta(H) = (a, O)$.

When predicting observations, the input sequence is modified to $H_t \cup \{a_t\}$, incorporating the taken action to provide the necessary information for the observation prediction, rather than just H_t as used for predicting optimal actions. Additionally, during pre-training, the training loss function must account for the prediction error on observations:

$$\hat{\theta} := \arg \min_{\theta \in \Theta} \frac{1}{nT} \sum_{i=1}^n \sum_{t=1}^T l \left(\hat{a}_t^{(\gamma_i)*}, a_t^{(\gamma_i)*} \right) + l' \left(\hat{O}_t^{(\gamma_i)}, O_t^{(\gamma_i)} \right). \quad (4.13)$$

Here, $l'(\cdot, \cdot)$ is a loss function that measures the prediction error of observations. As defined earlier, $\hat{a}_t^{(\gamma_i)*} = \text{TF}_{\theta,1}(H_t^{(\gamma_i)})$ is the predicted action, and $\hat{O}_t^{(\gamma_i)} = \text{TF}_{\theta,2}(H_t^{(\gamma_i)} \cup a_t^{(\gamma_i)})$ is the predicted observation, where the subscripts 1 and 2 in **TF** indicate the first and second outputs, respectively.

Compared to the original loss function in (4.11), which only accounts for predicting optimal actions, the revised loss function in (4.13) includes an additional term to capture the prediction error on observations, which ensures that the pre-trained OMGPT can effectively predict the observations through the pre-training.

4.5 Understanding Pre-trained $\text{TF}_{\hat{\theta}}$ as an Algorithm

In the previous section, we discuss using a supervised learning paradigm to learn a pre-trained $\text{TF}_{\hat{\theta}}$ as an algorithm. Now, we return to the perspective of sequential decision making and analyze the properties and performance of $\text{TF}_{\hat{\theta}}$ when being deployed as a decision algorithm. The proofs in this section are deferred to Appendix C.3.

4.5.1 $\text{TF}_{\hat{\theta}}$ as a Bayes-optimal Decision Maker

For a given distribution $\mathcal{P}_{\gamma}$, we define the *Bayes-optimal decision function* Alg^* as follows:

$$\begin{aligned} \text{Alg}^*(H) &:= \arg \min_{a \in \mathcal{A}} \mathbb{E}_{\gamma} \left[l(a, a_t^{(\gamma)*}) | H \right] \\ &= \arg \min_{a \in \mathcal{A}} \int_{\gamma} l(a, a_t^{(\gamma)*}) \mathcal{P}(\gamma | H) d\gamma \\ &= \arg \min_{a \in \mathcal{A}} \int_{\gamma} l(a, a_t^{(\gamma)*}) \mathbb{P}(H | \gamma) d\mathcal{P}_{\gamma} \end{aligned} \tag{4.14}$$

for any history $H \in \mathcal{H}$, where

$$\mathcal{H} = \{(X_1, a_1, O_1, \dots, X_{t-1}, a_{t-1}, O_{t-1}, X_t) | X_t \in \mathcal{X}, a_t \in \mathcal{A}, O_t \in \mathcal{O}, t = 1, \dots, T\},$$

is the space of all possible histories.

In the definition of Alg^* , recall $\mathcal{P}_{\gamma}$ is the environment distribution that generates the environment γ , and the optimal action $a_t^{(\gamma)*}$ depends on both the history H (or more precisely, X_t) and the environment γ (as defined in (4.3)). $\mathbb{P}(H | \gamma)$ is proportional to the likelihood of observing the history H under the environment γ

$$\mathbb{P}(H | \gamma) \propto \prod_{\tau=1}^t \mathbb{P}_{\gamma}(X_{\tau}) \cdot \prod_{\tau=1}^{t-1} \mathbb{P}_{\gamma}(O_{\tau} | X_{\tau}, a_{\tau})$$

where $H = (X_1, a_1, O_1, \dots, X_{t-1}, a_{t-1}, O_{t-1}, X_t)$ for some $t \in \{1, \dots, T\}$ and the distributions $\mathbb{P}_{\gamma}(\cdot)$ and $\mathbb{P}_{\gamma}(\cdot | X_{\tau}, a_{\tau})$ are as in (4.1). In the last line of definition, the integration is with

respect to the environment distribution $\mathcal{P}_\gamma$, and each possible environment γ is weighted with the likelihood $\mathbb{P}(H|\gamma)$.

We refer to $\mathbf{Alg}^*$ as the Bayes-optimal because its definition follows Bayes' law (as seen in the last line of (4.14)). And the following proposition, which relates $\mathbf{Alg}^*$ with the pre-training objective, further justifies why it is called the optimal decision function. Recall that the decision function $\tilde{f}$ is used to generate the actions in H_t .

Proposition 1. *The following holds for any decision function $\tilde{f}$:*

$$\mathbf{Alg}^*(\cdot) \in \arg \min_{f \in \mathcal{F}} \mathbb{E}_{\gamma \sim \mathcal{P}_\gamma} \left[\sum_{t=1}^T \mathbb{E}_{(H_t, a_t^*) \sim \mathcal{P}_{\gamma, \tilde{f}}} [l(f(H_t), a_t^*)] \right]$$

where $\mathcal{F}$ is the family of all measurable functions (on a properly defined space that handles variable-length inputs).

Proposition 1 states that the function $\mathbf{Alg}^*$ is one minimizer of the expected loss under any decision function $\tilde{f}$ (given $\mathcal{P}_\gamma$). To see this, $\mathbf{Alg}^*$ is defined in a pointwise manner for every possible H . The probability $\mathcal{P}_{\gamma, \tilde{f}}$ defines a distribution over the space of H . The pointwise optimality of $\mathbf{Alg}^*$ naturally leads to its optimality for any decision function $\tilde{f}$. Furthermore, this implies that $\mathbf{Alg}^*$ is also the optimal solution to the expected training loss (i.e., the expectation of the objective function of (4.11)). This means that if the OMGPT class $\{\mathbf{TF}_\theta\}_{\theta \in \Theta}$ is sufficiently rich to cover $\mathbf{Alg}^*$ (for instance, by increasing the number of layers and embedding dimensions), and with an infinite amount of pre-training data, the supervised pre-training will result in $\mathbf{Alg}^*$ at best. Thus, we can interpret the property of the supervised pre-trained $\mathbf{TF}_\theta$ by $\mathbf{Alg}^*$ as a proxy. Figure 4.3 shows an example of that $\mathbf{Alg}^*$ and $\mathbf{TF}_\theta$ are closely matched. This notion of a Bayes-optimal function and the Bayesian perspective has been explored in various settings within the in-context learning literature for large language models and GPTs [XRLM21, ZZYW23, JLLVR24].

In addition, the corollary below characterizes the exact behavior of $\mathbf{Alg}^*$ (and thus approximately of $\mathbf{TF}_\theta$) under different loss functions l in (4.14). This corollary can serve as a guide when pre-training OMGPT to target one specific algorithm.

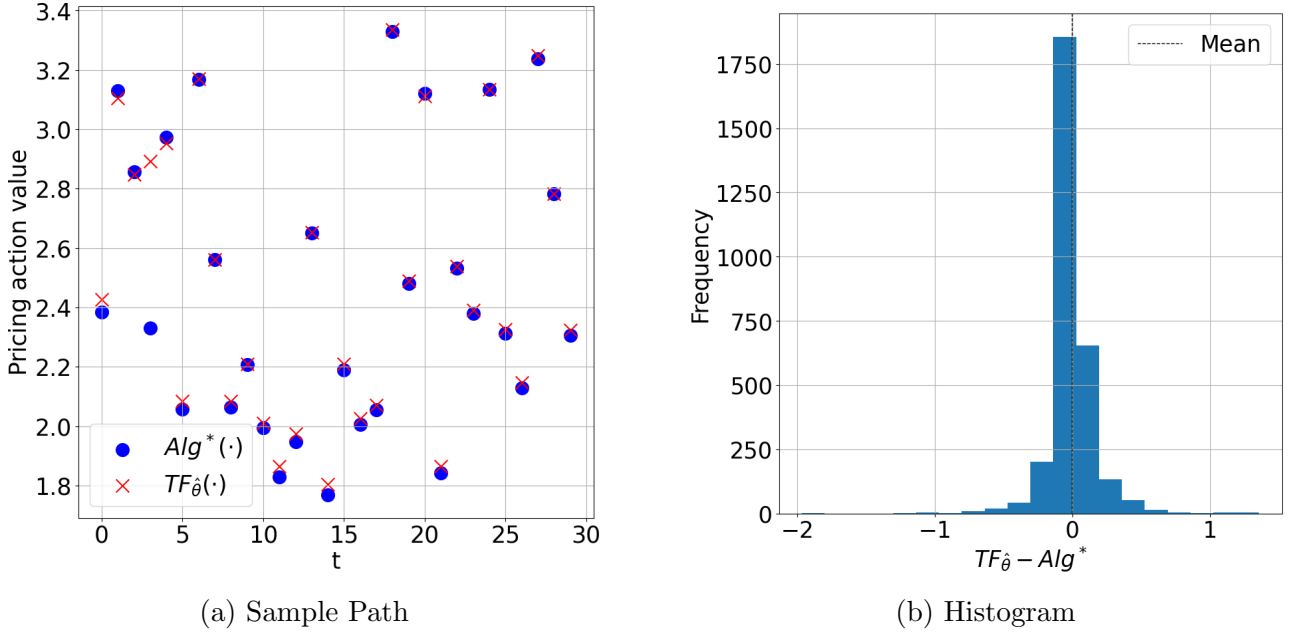


Figure 4.3: $\text{TF}_{\hat{\theta}}$ nearly matches the optimal decision function Alg^* in dynamic pricing tasks with concentrated deviations. Figure (a) shows one decision trial for Alg^* and $\text{TF}_{\hat{\theta}}$, where the optimal actions change over time because X_t 's are different for different time t . Figure (b) shows the histogram of $\text{TF}_{\hat{\theta}} - \text{Alg}^*$ over different environments. The experiment setup and more results are deferred to Appendix C.4.1.1.

Corollary 3. Alg^* behaves as

- (a) posterior sampling under the cross-entropy loss;
- (b) posterior averaging under the squared loss;
- (c) posterior median under the absolute loss;

where the posterior distribution is with respect to γ and it is defined by $\mathcal{P}(\gamma|H) \propto \mathbb{P}(H|\gamma) \cdot \mathcal{P}_{\gamma}(\gamma)$. Detailed definitions of posterior sampling, averaging, and median are provided in Appendix C.1.3.

4.5.2 Regret Analysis

The previous section analyzes the behavior of $\text{TF}_{\hat{\theta}}$. In this subsection, we study its performance in terms of regret, as defined in (4.2).

4.5.2.1 Surrogate Property of the Prediction Error

We begin with the following proposition, which establishes a relationship between the prediction error and the regret of any decision function f (where $f = \mathbf{TF}_{\hat{\theta}}$ is not necessarily required).

Recall we define

$$L(f; \gamma) = \sum_{t=1}^T \mathbb{E}_{(H_t^{(\gamma)}, a_t^{(\gamma)*})} [l(f(H_t^{(\gamma)}), a_t^{(\gamma)*})]$$

as the testing prediction error of f for a specific environment γ .

Proposition 2 (Surrogate property). *We say the loss function $l(\cdot, \cdot)$ satisfies the surrogate property if there exists a constant $C > 0$ such that*

$$\text{Regret}(f; \gamma) \leq C \cdot L(f; \gamma)$$

holds for any sequence function f and γ . Then under mild conditions (see Appendix C.3), We have

- (a) *The cross-entropy loss satisfies the surrogate property for the multi-armed bandits problem.*
- (b) *The squared loss satisfies the surrogate property for the dynamic pricing problem.*
- (c) *The absolute loss satisfies the surrogate property for the linear bandits and the newsvendor problem.*

We provide the definitions of these problems in Appendix C.4.2. Proposition 2 gives a first validity of the supervised pre-training approach. The surrogate property is an attribute of the loss function and is independent of the distribution $\mathcal{P}_\gamma$ or the decision function f . Specifically, it establishes an upper bound on the regret in terms of the action prediction loss. This result is intuitive, as a more accurate prediction of the optimal action naturally leads to lower regret. One key implication is that this result guides the selection of the appropriate loss function for different underlying problems.

4.5.2.2 Linear Regret of Alg^*

Although Proposition 2 provides an upper bound on the regret using the action prediction error for each individual environment γ , it does not imply that the minimizer Alg^* of the expected prediction error with respect to some distribution $\mathcal{P}_\gamma$ will have sub-linear regret in the horizon T . This is illustrated by the following proposition:

Proposition 3. *There exists a linear bandits problem instance and a dynamic pricing problem instance, i.e., an environment distribution $\mathcal{P}_\gamma$, such that the optimal decision function learned under the squared loss incurs a linear regret for every γ , i.e., $\text{Regret}(\text{Alg}^*; \gamma) = \Omega(T)$ for every γ .*

Proposition 3 states a negative result that the optimal decision function Alg^* , albeit being optimal in a prediction sense (for predicting the optimal action), does not serve as a good algorithm for sequential decision making problems. To see the intuition, the decision function is used recursively in the testing phase (see (4.12)). While it gives the best possible prediction, it does not conduct *exploration* which is critical for the concentration of the posterior measure. This result does not contradict Proposition 2. While Proposition 2 provides the action prediction error as an upper bound on the regret, in the instances described in Proposition 3, both the prediction error and the upper bound are linear in T .

This highlights a key difference between sequential decision making and single decision making problems (e.g., [CLM22, HHS23, BR19, FS23]) when applying machine learning or deep learning tools. In single decision making problems, where only a single decision needs to be inferred for a single time (rather than an “entire” algorithm for a horizon), the collected training samples (analogous to the context-action-observation tuples in H_t) are i.i.d. generated, and this results in a convergence/concentration of the posterior to the true model/environment γ . However, in sequential decision making problems, the dynamics of (4.12) induce dependencies across the history H_t , and it may result in a non-concentration behavior for the posterior. This is the key insight behind constructing the linear-regret example in Proposition 3 and the examples are also inspired from and share the same spirit as the discussions in [HKZ12]. It is also worth

noting that such a lack of exploration may not occur in discrete action spaces, as studied in [LXP⁺24]. This is because for discrete action space, the output $\mathbf{TF}_{\hat{\theta}}$ (or $\mathbf{Alg}^*$) can give the posterior distribution as a distribution over the action space. The distribution output will lead to a randomized action and thus can automatically perform exploration.

4.5.2.3 (Sub-linear) Regret Analysis on $\mathbf{TF}_{\hat{\theta}}$

There are simple remedies to address the linear regret behavior of the optimal decision function, such as using an ϵ -greedy version of it and explicitly adding some exploration. However, even without any further modifications to the pre-trained $\mathbf{TF}_{\hat{\theta}}$, we do not observe this linear regret behavior in any of our numerical experiments (as will be demonstrated in Section 4.6). In this section, we provide a theoretical framework to explain the observed sub-linear regrets of $\mathbf{TF}_{\hat{\theta}}$.

We begin by informally stating the regret upper bound of $\mathbf{TF}_{\hat{\theta}}$ to provide intuition on why it performs well without explicit exploration. We assume $\mathcal{P}_{\gamma}$ is a uniform distribution over a bounded support $\Gamma \in \mathbb{R}^d$ and recall $\gamma \in \Gamma$ is the sampled testing environment, which generates H_t through (4.12) and decides the reward function r . For simplicity, we denote the conditional distribution of the observation O_t given the history H_t and $a_t = \mathbf{TF}_{\hat{\theta}}(H_t)$ (or $a_t \sim \mathbf{TF}_{\hat{\theta}}(H_t)$) as $\mathbb{P}_{\gamma'}(\cdot|H_t, \mathbf{TF}_{\hat{\theta}})$ for any environment $\gamma' \in \Gamma$ (which is not necessary the true testing environment γ). Under conditions on restricting smoothness of $\mathbb{P}_{\gamma'}(\cdot|H_t, \mathbf{TF}_{\hat{\theta}})$, we can establish the following result:

Theorem 4.1 (Regret Upper Bound (Informal)). *Suppose there exists constants Δ_{Exploit} and Δ_{Explore} such that for any $\gamma \in \Gamma \subset \mathbb{R}^d$ and H_t , $\mathbf{TF}_{\hat{\theta}}$ and $\mathbf{Alg}^*(H_t)$ satisfies*

- $\mathbb{E}[r(X_t, \mathbf{Alg}^*(H_t)) - r(X_t, \mathbf{TF}_{\hat{\theta}}(H_t))|H_t] \leq \Delta_{\text{Exploit}}$, where the expectation is taken with respect to the possible randomness in $\mathbf{Alg}^*$ and $\mathbf{TF}_{\hat{\theta}}$.
- The KL divergence

$$KL(\mathbb{P}_{\gamma}(\cdot|X_t, \mathbf{TF}_{\hat{\theta}}(H_t))||\mathbb{P}_{\gamma'}(\cdot|X_t, \mathbf{TF}_{\hat{\theta}}(H_t))) \geq \Delta_{\text{Explore}}\|\gamma - \gamma'\|_2^2$$

for any $\gamma' \in \Gamma$, and the log-likelihood ratio between $\mathbb{P}_{\gamma}(\cdot|X_t, \mathbf{TF}_{\hat{\theta}}(H_t))$ and $\mathbb{P}_{\gamma'}(\cdot|X_t, \mathbf{TF}_{\hat{\theta}}(H_t))$ is sub-Gaussian for any $\gamma' \in \Gamma$.

- $\mathbb{E}[r(X_t, a_t^*) - r(X_t, \mathbf{Alg}^*(H_t))|H_t] \leq O(\mathbb{E}_{\gamma'}[\|\gamma - \gamma'\|_2^2|H_t])$, where the expectation on the left side is taken with respect to the possible randomness in $\mathbf{Alg}^*$ and the expectation $\mathbb{E}_{\gamma'}[\cdot]$ on the right side is with respect to the posterior distribution over the possible testing environment γ' given H_t .

Then we have

$$\text{Regret}(\mathbf{TF}_{\hat{\theta}}; \gamma) = O\left(\Delta_{\text{Exploit}}T + \frac{d}{\Delta_{\text{Explore}}}\right)$$

for any $\gamma \in \Gamma$, where the notation $O(\cdot)$ omits logarithm terms in T and d and the parameters in assumptions (e.g., boundedness parameter).

We should first note that Theorem 4.1 provides a worst-case regret bound since it holds for any possible testing environment $\gamma \in \Gamma$. Thus, it also serves as an upper bound for the expected regret $\mathbb{E}_{\gamma \sim \mathcal{P}_{\gamma}}[\text{Regret}(\mathbf{TF}_{\hat{\theta}}; \gamma)]$. To better understand Theorem 4.1, we first examine its three key conditions. The first condition is that $\mathbf{TF}_{\hat{\theta}}$ behaves closely to $\mathbf{Alg}^*$ and the (reward) deviation between these two is bounded by Δ_{Exploit} . The deviation could arise due to finite pre-training data or the limited function space $\mathcal{F}$ during the pre-training phase. This condition can be justified by Figure 4.3. The second condition characterizes the exploration intensity of $\mathbf{TF}_{\hat{\theta}}$, ensuring it can distinguish between the true environment and any other environment by bounding their KL divergence with Δ_{Explore} . The last condition characterizes the performance of $\mathbf{Alg}^*$ in terms of the posterior distribution of the possible testing environment given H_t . As the posterior distribution concentrates on the true testing environment γ , the reward of $\mathbf{Alg}^*(H_t)$ converges to that of the optimal action.

Under these conditions, intuitively, $\mathbf{TF}_{\hat{\theta}}$ can be viewed as the optimal decision function $\mathbf{Alg}^*$ plus some deviation. These deviations can be considered as random noises: while they may distort the performance of $\mathbf{TF}_{\hat{\theta}}$ when the posterior is concentrated (first and third conditions), they can play the role of exploration and helps the posterior to concentrate (second condition).

In particular, $\text{TF}_{\hat{\theta}}$ deviates from Alg^* by an amount of Δ_{Exploit} in terms of the reward, and this deviation accumulates over time into the first term in the regret bound. On the other hand, this deviation also gives an exploration whose intensity is measured by Δ_{Explore} , and the second term captures the regret caused during the concentration of the posterior onto the true environment.

These conditions serve as a stylized model for analyzing the behavior and performance of $\text{TF}_{\hat{\theta}}$. In many online learning algorithms, such as UCB or Thompson sampling, the constants Δ_{Exploit} and Δ_{Explore} should have a dependency on t . Nevertheless, these conditions give us a perspective to understand how the pre-trained $\text{TF}_{\hat{\theta}}$ works as a decision algorithm and why it can have sub-linear regret. For example, when $\Delta_{\text{Exploit}} = \Delta_{\text{Explore}} = O(1/\sqrt{T})$, this regret upper bound is in the order of $O(\sqrt{T})$. In the following part, we will elaborate on the assumptions required for these conditions and provide a formal result for the regret upper bound.

Assumptions We first present the assumptions needed for the regret analysis.

Assumption 1 (Environment Space). *We assume there exists a space $\Gamma \subset \mathbb{R}^d$ of environment parameters such that*

- *There exists a constant $\bar{\gamma}$ such that $\sup_{\gamma \in \Gamma} \|\gamma\|_2 \leq \bar{\gamma}$.*
- *There exists a constant $\bar{r}$ such that $\sup_{x \in \mathcal{X}, a \in \mathcal{A}, \gamma \in \Gamma} r(x, a; \gamma) \leq \bar{r}$, where we use $r(x, a; \gamma)$ to denote the reward function of environment γ .*
- *$\mathcal{P}_{\gamma}$ is a uniform distribution over Γ .*

The above assumptions characterize the space of possible environments. The first two assumptions bound the space Γ and the maximum possible reward, which are commonly applied in the literature (e.g., [DB15, HR09]). The last assumption concerns the decision maker's prior distribution over the testing environment. We remark that this assumption can be relaxed to allow for any general distribution. However, such a relaxation would introduce an additional multiplier in the resulting regret bound. This multiplier would be proportional to the upper

bound of the likelihood ratio between two possible environments in Γ . Intuitively, this represents the (largest) effort needed to correct an incorrect prior. A larger ratio indicates that more exploration is required to concentrate the posterior distribution on the true environment in the worst-case, leading to higher regret.

Assumption 2 (Observation Distribution). *Denote $\mathbb{S}_+^d$ as the space of positive semidefinite $d \times d$ matrices and recall γ is the true environment. Then we assume for any $X \in \mathcal{X}$, $a \in \mathcal{A}$ and $\gamma' \in \Gamma$:*

- *There exists a function $\Delta : (\mathcal{X}, \mathcal{A}) \rightarrow \mathbb{S}_+^d$ such that*

$$KL(\mathbb{P}_\gamma(\cdot|X, a) \| \mathbb{P}_{\gamma'}(\cdot|X, a)) \geq \|\gamma - \gamma'\|_{\Delta(X, a)}^2, \quad (4.15)$$

where $\|z\|_{\Delta(X, a)} = \sqrt{z^T \Delta(X, a) z}$ for $z \in \mathbb{R}^d$.

- *There exists a constant C_{σ^2} such that for any $\gamma' \in \Gamma$, the log-likelihood ratio $\log \left(\frac{\mathbb{P}_\gamma(O|X, a)}{\mathbb{P}_{\gamma'}(O|X, a)} \right)$, where $O \sim \mathbb{P}_\gamma(\cdot|X, a)$, is $C_{\sigma^2} \|\gamma - \gamma'\|_{\Delta(X, a)}$ -sub-Gaussian.*
- *$\log \mathbb{P}_{\gamma'}(O|X, a)$ is concave in $\gamma' \in \Gamma$. Further, there exists $C_\Gamma > 0$ such that for all $X \in \mathcal{X}$, $a \in \mathcal{A}$ and $O \in \mathcal{O}$,*

$$\|\nabla_{\gamma'}^2 \log \mathbb{P}_{\gamma'}(O|X, a)\|_2^2 \leq C_\Gamma.$$

The first two assumptions above characterize the statistical properties required for the log-likelihood ratio, $\log \left(\frac{\mathbb{P}_\gamma(O|X, a)}{\mathbb{P}_{\gamma'}(O|X, a)} \right)$, and the last one assumes the smoothness of the log-likelihood function $\log \mathbb{P}_{\gamma'}(O|X, a)$. Specifically, in the first two assumptions, $\sum_{\tau=1}^t \Delta(X_\tau, a_\tau)$ can be interpreted as the information matrix from (X_τ, a_τ) 's. For example, in dynamic pricing with Gaussian noise and a linear demand function (i.e., $d(X, a) = \alpha^\top X + \beta^\top X \cdot a$ where $\gamma = (\alpha, \beta)$), we have $\Delta(X_\tau, a_\tau) = (X_\tau, a_\tau X_\tau)(X_\tau, a_\tau X_\tau)^\top$. The second assumption is necessary to bound the rate at which the posterior distribution concentrates around the true environment. The last assumption requires the log-likelihood to be smooth with respect to the environment parameters. These assumptions are commonly satisfied in tasks like dynamic pricing and bandits with generalized linear demands or rewards under Gaussian noise.

Assumption 3 (Reward Gaps). *We assume for any $\gamma \in \Gamma$ with its corresponding reward function r :*

- *There exists a constant Δ_{Exploit} such that $\text{TF}_{\hat{\theta}}$ satisfies*

$$\mathbb{E}[r(X_t, \text{Alg}^*(H_t)) - r(X_t, \text{TF}_{\hat{\theta}}(H_t)|H_t)] \leq \Delta_{\text{Exploit}}$$

for all possible H_t , where the expectation is taken with respect to the possible randomness in Alg^* and $\text{TF}_{\hat{\theta}}$.

- *There exists a constant C_r such that Alg^* satisfies*

$$\mathbb{E}[r(X_t, a_t^*) - r(X_t, \text{Alg}^*(H_t))|H_t] \leq C_r \mathbb{E}_{\gamma'}[\|\gamma - \gamma'\|_2^2 | H_t],$$

where the expectation on the left side is taken with respect to the possible randomness in Alg^* and the expectation $\mathbb{E}_{\gamma'}[\cdot | H_t]$ on the right side is with respect to the posterior distribution over the possible testing environment γ' given H_t .

The first condition bounds the reward deviation between Alg^* and $\text{TF}_{\hat{\theta}}$, which can be potentially caused by the finite pre-training samples and is numerically justified by Figure 4.3. The second condition indicates the performance of Alg^* in relation to the posterior distribution (which is further decided by the history H_t): as the posterior distribution of γ' under H_t becomes more concentrated around the true testing environment γ , the performance of Alg^* improves.

Regret Upper Bound

Theorem 4.2 (Regret Upper Bound). *Denote $\lambda_t = \lambda_{\min}(\sum_{\tau=1}^t \Delta(X_\tau, a_\tau))$ as the minimum eigenvalue of the matrix $\sum_{\tau=1}^t \Delta(X_\tau, a_\tau)$ for each $t = 1, \dots, T$, and $t_0 = \min\{t = 2, \dots, T | \lambda_{t-1} > 0\}$ (here we let $t_0 = T + 1$ when $\{t = 2, \dots, T | \lambda_{t-1} > 0\} = \emptyset$). Under Assumption 1, 2, 3, we*

have for any $\gamma \in \Gamma$

$$\begin{aligned} \text{Regret}(\text{TF}_{\hat{\theta}}; \gamma) &\leq \Delta_{\text{Exploit}}T + \bar{r}\mathbb{E}[t_0 - 1] + C_r\mathbb{E}\left[\sum_{t=t_0}^T \epsilon_t\right] + 16C_r(4C_{\sigma_2} + 1)\mathbb{E}\left[\sum_{t=t_0}^T \frac{\log(2K_tT)}{\lambda_{t-1}}\right] + 4C_r\bar{\gamma}^2 \\ &= O\left(\Delta_{\text{Exploit}}T + \mathbb{E}\left[t_0 + \sum_{t=t_0}^T \frac{d}{\lambda_{t-1}}\right]\right) \end{aligned}$$

where $\epsilon_t = \frac{16d}{\lambda_{t-1}} \log\left(\frac{32\bar{\gamma}\sqrt{d}(C_{\Gamma}(t-1) + \lambda_{t-1})}{\lambda_{t-1}}\right)$, $K_t = \max\left\{1, \left(\frac{32\bar{\gamma}\sqrt{d}(C_{\Gamma}(t-1) + \lambda_{t-1})}{\lambda_{t-1} \min\{\sqrt{\epsilon_t}, 1\}}\right)^d\right\}$, and the expectation is with respect to the potential randomness in λ_t and t_0 .

Theorem 4.2 follows the same spirit as Theorem 4.1 and can also be decomposed into two terms: the first term $O(\Delta_{\text{Exploit}}T)$ captures the regret caused by the deviation of $\text{TF}_{\hat{\theta}}$ from the optimal decision Alg^* . The second term $O\left(\mathbb{E}\left[t_0 + \sum_{t=t_0}^T \frac{d}{\lambda_{t-1}}\right]\right)$ represents the regret incurred during the concentration of the posterior distribution towards the true environment γ . Specifically, λ_t is the minimum eigenvalue of the empirical “information matrix” $\sum_{\tau=1}^t \Delta(X_{\tau}, a_{\tau})$, which controls the rate of the posterior distribution convergence to the true testing environment (see Corollary 5). The random time t_0 is required for the availability of $\lambda_t > 0$ and is influenced by both $\text{TF}_{\hat{\theta}}$ and the generation of X_t . The proof of Theorem 4.2 and further discussions on its technical contributions are deferred to Appendix C.3.

Our analysis offers a complementary understanding of how the pre-trained $\text{TF}_{\hat{\theta}}$ works in literature. While [LBM23] provide a statistical learning framework for analyzing the performance of transformer-style models in decision makings, they show that when (i) $a_t^{(\gamma_i)^*}$ ’s in the training data are no longer the optimal actions a_t^* in our setting but generated from some algorithm Alg and (ii) the decision function $\tilde{f} = \text{Alg}$ in the distribution $\mathcal{P}_{\gamma_i, \tilde{f}}$, the pre-trained transformer $\text{TF}_{\hat{\theta}} \rightarrow \text{Alg}$ as the number of training samples goes to infinity. We note that this only implies that the transformer model can *imitate* an existing algorithm Alg . In other words, such setting brings the theoretical benefits of consistency, but it excludes the possibility of using the optimal a_t^* in the pre-training and thus makes it impossible for the pre-trained $\text{TF}_{\hat{\theta}}$ to perform better than the algorithm Alg . To better see it, as shown in Proposition 3, Alg^* can incur linear regret (for numerical results, see Appendix C.1.3), a behavior that is not numerically observed

with $\mathbf{TF}_{\hat{\theta}}$. This discrepancy cannot be explained by [LBM23] when setting $\mathbf{Alg} = \mathbf{Alg}^*$. Our assumptions and analysis provide a stylized explanation for such a sub-linear regret behavior of $\mathbf{TF}_{\hat{\theta}}$, as shown in the following example.

Example: Dynamic Pricing Here we adopt the contextual dynamic pricing problem [QB16, BK21] as an example to show the application of Theorem 4.2.

Example 1. Consider a dynamic pricing problem, where the demand is given by $D_t = \alpha^\top X_t + \beta^\top X_t \cdot a_t + \epsilon_t$. Here $\epsilon_t \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1)$ and $\gamma = (\alpha, \beta) \in \Gamma \subset \mathbb{R}^{2d}$. We suppose that $\mathbf{Alg}^*$ performs posterior averaging. In addition, for each time t , we assume with some constant C , $\mathbf{TF}_{\hat{\theta}}(H_t) = \mathbf{Alg}^*(H_t) + \Delta_t$, where

$$\Delta_t = \begin{cases} CT^{-1/4} & \text{w.p. } 1/2, \\ -CT^{-1/4} & \text{w.p. } 1/2 \end{cases}$$

for any possible history H_t and $t = 1, \dots, T$, and Δ_t is independent of $H_t, \mathbf{Alg}^*(H_t), \gamma$, and previous Δ_τ for $\tau = 1, \dots, t-1$. If we further assume some boundedness on the parameters, then by Theorem 4.2, we have

$$\text{Regret}(\mathbf{TF}_{\hat{\theta}}, \gamma) = O(d\sqrt{T})$$

for all $\gamma \in \Gamma$.

We make the following remarks for Example 1: (i) The derived upper regret upper bound matches the regret lower bound in terms of the order of T , as established in [KZ14]. This illustrates that, with an appropriate deviation between $\mathbf{Alg}^*$ and $\mathbf{TF}_{\hat{\theta}}$, $\mathbf{TF}_{\hat{\theta}}$ can achieve an asymptotically optimal (worst-case) regret bound. (ii) The assumption $\mathbf{TF}_{\hat{\theta}}(H_t) = \mathbf{Alg}^*(H_t) + \Delta_t$ introduces a (random) perturbation term Δ_t to $\mathbf{Alg}^*(H_t)$. These perturbations act as explorations to learn γ , and the assumed rate $\Theta(T^{-1/4})$ effectively balances the exploration and the exploitation to bound the regret.

Regret Upper Bound on Finite Environment Space In the following, we present the regret upper bound for a special case where Γ contains only a finite set of environments.

Theorem 4.3 (Regret Upper Bound on Finite Environment Space). *Assume $\Gamma = \{\gamma_1, \dots, \gamma_n\}$.*

Under Assumption 1, 2, 3, we have for any $\gamma \in \Gamma$:

- **(Problem-dependent bound).** *We denote $t_i = \min\{t = 2, \dots, T \mid \sum_{\tau=1}^{t-1} \|\gamma_i - \gamma\|_{\Delta(X_\tau, a_\tau)}^2 \geq 16C_{\sigma^2} \log(nT^2)\}$ (we let $t_i = T + 1$ when the set is empty), then*

$$\text{Regret}(\mathbf{TF}_{\hat{\theta}}; \gamma) < 4C_r \bar{\gamma}^2 + \Delta_{\text{Exploit}} T + \bar{r} C_r \sum_{\gamma_i \neq \gamma} \mathbb{E}[t_i - 1] + C_r \sum_{\gamma_i \neq \gamma} \mathbb{E} \left[\sum_{t=t_i}^T \frac{4\|\gamma_i - \gamma\|_2^2}{\sum_{\tau=1}^{t-1} \|\gamma_i - \gamma\|_{\Delta(X_\tau, a_\tau)}^2} \right]. \quad (4.16)$$

- **(Problem-independent bound).** *Recall $\lambda_t = \lambda_{\min}(\sum_{\tau=1}^t \Delta(X_\tau, a_\tau))$ as the minimum eigenvalue of the matrix $\sum_{\tau=1}^t \Delta(X_\tau, a_\tau)$ for each $t = 1, \dots, T$ and $t_0 = \min\{t = 2, \dots, T \mid \lambda_{t-1} > 0\}$, then*

$$\text{Regret}(\mathbf{TF}_{\hat{\theta}}; \gamma) < 4C_r \bar{\gamma}^2 + \Delta_{\text{Exploit}} T + \bar{r} C_r \mathbb{E}[t_0 - 1] + n C_r \mathbb{E} \left[\sum_{t=t_0}^T \frac{16C_{\sigma^2} \log(nT^2) + 4}{\lambda_{t-1}} \right]. \quad (4.17)$$

Here the expectation is with respect to the potential randomness in λ_t and t_i , $i = 0, \dots, n$.

Compared to Theorem 4.2: (i) The regret bounds in Theorem 4.3 depend on the number of environments n rather than the dimension of the environment d . This is because, with a finite structure, we only need to identify γ from a finite set, rather than from an infinite set of possible environments as in Theorem 4.2, whose complexity is controlled by d . Technically, the shift from n to d arises from using a covering number technique in Theorem 4.2 to discretize a bounded continuous space into a discrete space, where the covering number depends on d . (ii) It is difficult to say whether Theorem 4.3 provides sharper bounds than Theorem 4.2 without specifying the underlying problem or the structure of Γ . For instance, when $n \gg d$, the bounds in Theorem 4.3 can be worse.

We refer to (4.16) as the problem-dependent bound and (4.17) as the problem-independent bound. This is because (4.16) accounts for specific candidate environments (e.g., the term

$\|\gamma_i - \gamma\|_2^2$), whereas (4.17) does not depend on such specific problem structure. These two bounds are similar to the problem-dependent and problem-independent bounds in multi-armed bandits [LS20]. However, it is important to note that in the multi-armed bandits problem, there is a finite action space, and the problem-dependent bound relates to the action rewards. In contrast, here we assume a finite number of environments, and the bound relates to these environments.

The following example shows that, given a specific problem structure, the problem-dependent bound can sometimes provide a sharper regret upper bound than the problem-independent bound.

Example 2. Consider a dynamic pricing problem without context, where the demand is given by $D_t = \alpha + \beta \cdot a_t + \epsilon_t$. Here $\epsilon_t \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1)$ and $\gamma = (\alpha, \beta) \in \Gamma \subset \mathbb{R}^2$. We further assume that $\mathbf{TF}_{\hat{\theta}} = \mathbf{Alg}^*$ and $\mathbf{Alg}^*$ performs posterior averaging. If we suppose $\Gamma = \{(1, -1), (1, -2)\}$ and assume some boundedness on the parameters, then by applying (4.16), we have

$$\text{Regret}(\mathbf{TF}_{\hat{\theta}}, \gamma) = O(\log T)$$

for all $\gamma \in \Gamma$.

This example illustrates that a regret bound of $O(\log T)$ can be achieved by applying (4.16), which is smaller than the typical regret upper bound of $O(\sqrt{T})$ found in dynamic pricing problems (e.g., [BR12, KZ14, WTL21]). Indeed, through a similar analysis as in Example 1, Theorem 4.2 would also result in a regret upper bound of $O(\sqrt{T})$ for Example 2.

To understand the $O(\log T)$ regret, intuitively, there are no “uninformative prices” [BR12] for identifying γ due to the specific structure of Γ . Consequently, the regret term $\frac{4\|\gamma_i - \gamma\|_2^2}{\sum_{\tau=1}^{t-1} \|\gamma_i - \gamma\|_{\Delta(X_\tau, a_\tau)}^2}$, which typically arises from uncertainty of γ , will decrease at a rate of $O(1/t)$ for this specific space Γ , even when making greedy optimal decisions with $\mathbf{Alg}^*(H_t)$ that involve no exploration. This example also illustrates how $\mathbf{TF}_{\hat{\theta}}$ (or $\mathbf{Alg}^*$) can outperform benchmark algorithms due to the special structure of Γ .

4.6 Numerical Experiments

In this section, we conduct numerical experiments to evaluate OMGPT’s performance from various perspectives and investigate its internal mechanisms when making decisions. All experiment details can be found in Appendix C.4. In all figures, where applicable, the shaded area represents the 90% empirical confidence interval, and the reported values are based on 100 runs.

4.6.1 Performance Evaluation

We first assess OMGPT across several operational decision making tasks and examine its generalization ability in testing environments with longer horizons than those used in the pre-training. Additionally, we evaluate its performance in environments that are generated in a way different from the pre-training ones. We further test its capacity to handle non-stationary environments and environments with multiple demand types, highlighting its potential as a solution to the problem of model misspecification. Lastly, we show that, beyond predicting optimal actions, OMGPT can also serve as a predictor for some side information, such as the demand in dynamic pricing problems, to provide valuable auxiliary insights.

4.6.1.1 Basic Performance

We evaluate OMGPT across a variety of operational decision making tasks, including dynamic pricing, the newsvendor problem (with censored demand), queuing control, and revenue management. The settings of these problems are deferred to Appendix C.4.2. It is important to note that both the queuing control and revenue management problems feature state-dependent contexts, as defined in Section 4.4.5.2. In these tasks, the context or state—such as queue length in queuing control or available resources in revenue management—depends on both the previous context and the actions taken. These tasks are used to validate OMGPT’s ability to effectively handle operational decision making problems with state-dependent contexts.

The pre-trained OMGPT, $\text{TF}_{\hat{\theta}}$, is compared against well-established benchmark algorithms for each task. Figure 4.4 presents a summary of the results and more details (e.g., experiment setup and benchmark algorithms) are deferred to Appendix C.4. It is important to note that while the testing environment is sampled from the decision maker’s prior distribution, $\mathcal{P}_{\gamma}$, which also generates the pre-training data for $\text{TF}_{\hat{\theta}}$, the distribution is assumed to have infinite support. This implies that the testing environment, γ , differs from the pre-training environments, γ_i ’s, with probability 1. This setup ensures a fair comparison and provides a more challenging scenario to assess OMGPT’s robustness and generalization ability.

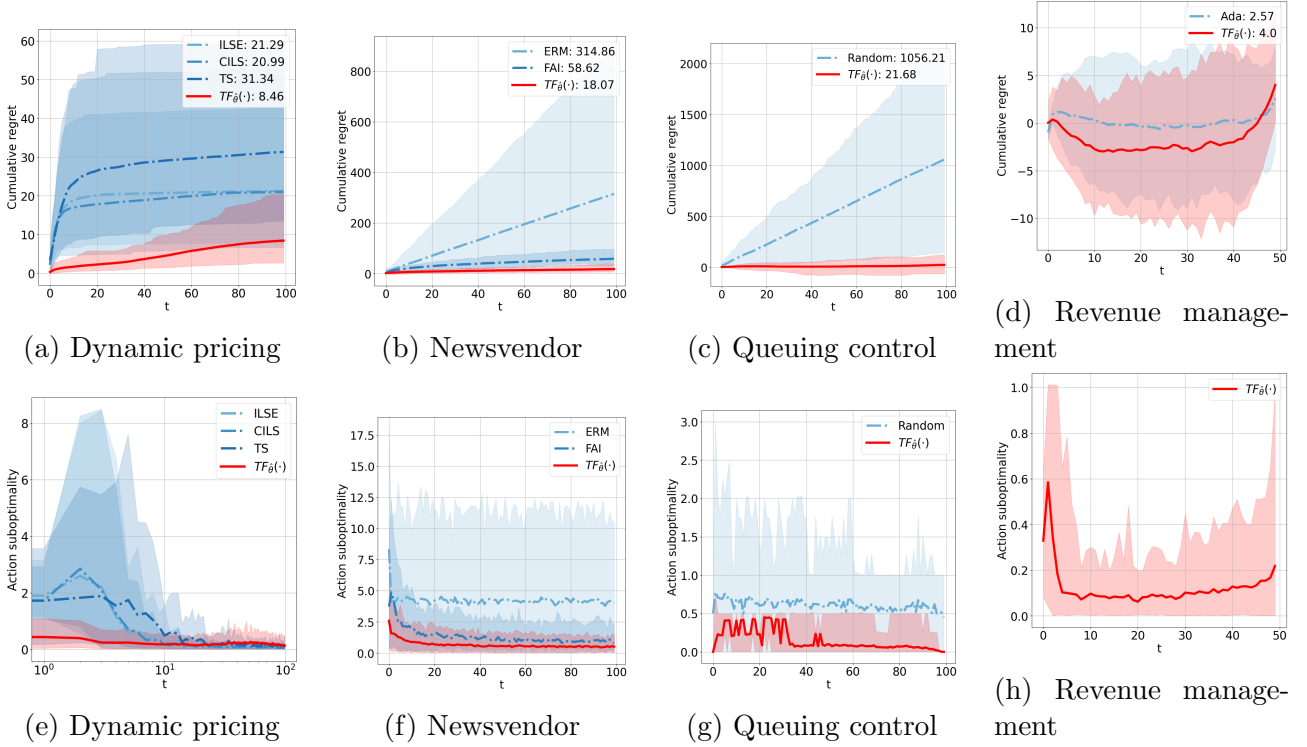


Figure 4.4: The average out-of-sample regret (first row) and action suboptimality, i.e., $|a_t^* - \text{Alg}(H_t)|$, (second row) of $\text{TF}_{\hat{\theta}}$ against benchmark algorithms (see details in Appendix C.4.3). The numbers in the legend bar are the final cumulative regret. The last two tasks can have negative regret during the horizon since they are MDP problems where the optimal actions may achieve a lower single-period reward than the applied algorithms, although they maximize the final cumulative expected reward. For revenue management, since the exact optimal actions are computed at a high computation cost, we use the actions from the Adaptive Allocation Algorithm (Ada) from [CLY24], which can achieve constant regret, to approximate optimal actions in both the pre-training and testing (and thus not show the suboptimality of the Ada algorithm in (h)), and use their upper bound of the optimal cumulative reward to compute (an upper bound) of the regret used in (d).

From Figure 4.4, we observe that OMGPT, $\text{TF}_{\hat{\theta}}$, consistently outperforms the structured benchmark algorithms across all tasks. What makes this result particularly remarkable is that $\text{TF}_{\hat{\theta}}$ does not merely replicate or imitate the performance of the benchmark algorithms; rather, it

uncovers a new and more effective approach. This highlights the effectiveness of using the optimal actions, a_t^* , as the target during pre-training.

We further attribute the OMGPT’s advantage over the benchmark algorithms to two key factors: the incorporation of prior knowledge from pre-training data and its more greedy exploitation strategy. First, the OMGPT effectively leverages the pre-training data, making it well-suited to the decision maker’s prior knowledge of the testing environment (see Section 4.5.1 and Appendix C.4.1.2). In contrast, all the benchmark algorithms face a cold-start problem. Second, while the benchmark algorithms are designed for asymptotic optimality, potentially sacrificing short-term performances, OMGPT takes a greedier approach, similar to Alg^* , which does not explicitly encourage exploration. This greedier behavior can lead to better performances when the time horizon T is short, particularly when $T \leq 100$ as shown in Figure 4.4.

4.6.1.2 Impact of Model Scaling and Task Complexity

In this section, we explore the performance changes when tuning both model size and task complexity. Specifically, we adjust the number of layers to control the model size and the problem dimension (the dimension of the context) in a dynamic pricing task to control the task complexity, where the number of unknown parameters is twice the context dimension. Since the optimal reward may scale differently across dimensions, we provide a relatively fair comparison by evaluating the advantage of OMGPT relative to the best-performing benchmark algorithm. This advantage is defined as the reward improvement rate of OMGPT compared to the best benchmark algorithm (the one among ILSE, CILS, and TS that achieves the highest reward). Thus, a positive value indicates the pre-trained OMGPT $\text{TF}_{\hat{\theta}}$ outperforms all benchmark algorithms, and a larger value indicates a better performance of $\text{TF}_{\hat{\theta}}$. We test three model sizes (4, 8, and 12 layers) and three problem dimensions (4, 10, and 20), while keeping other hyperparameters the same. The results are presented in Figure 4.5.

From Figure 4.5, we observe the following: (i) $\text{TF}_{\hat{\theta}}$ consistently outperforms the benchmark algorithms, and this advantage grows as the complexity of the problem increases. (ii) Increas-

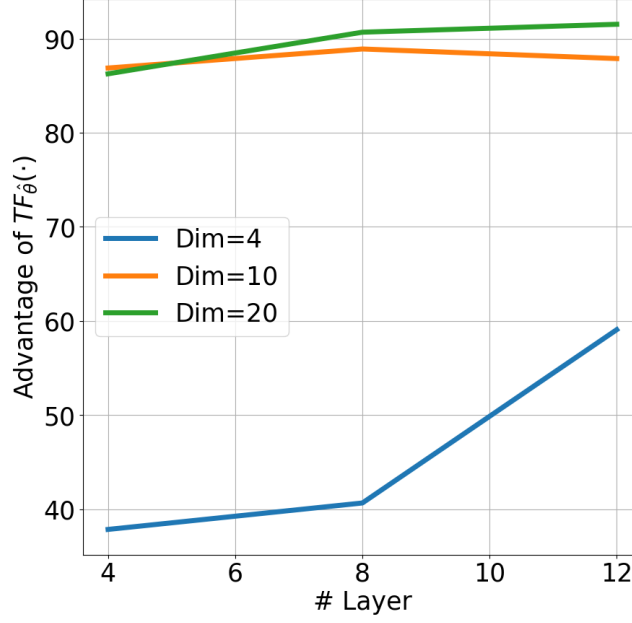


Figure 4.5: The advantage of OMGPT compared to the best benchmark algorithm in dynamic pricing, across different model size (number of layers) and problem complexity (problem dimensions).

ing model size consistently improves the performance of $TF_{\hat{\theta}}$ across all problem dimensions, indicating that larger models are generally preferred. This highlights $TF_{\hat{\theta}}$'s superior ability to handle more complex tasks and these observations align with the scaling law [KMH⁺20] for large language models.

4.6.1.3 Performance under Horizon Generalization

In practice, the testing environment may have a different time horizon from the pre-training data. In this part, we evaluate OMGPT's generalization ability when the testing horizon is longer than that seen during pre-training. As for a shorter testing horizon, OMGPT can simply run until the end of the horizon, and Figure 4.4 demonstrates strong performances in such cases, particularly when the horizon is less than 100.

Technically, during testing for $t > 100$, we limit the input sequence H_t to the last 100 timesteps to predict a_t^* . As discussed in Section 4.4.5.1, this restriction aligns with the input sequence length used during pre-training, enabling OMGPT to effectively handle longer testing horizons. Figure 4.6 shows the performance when the testing horizon is extended from 100 (pre-training

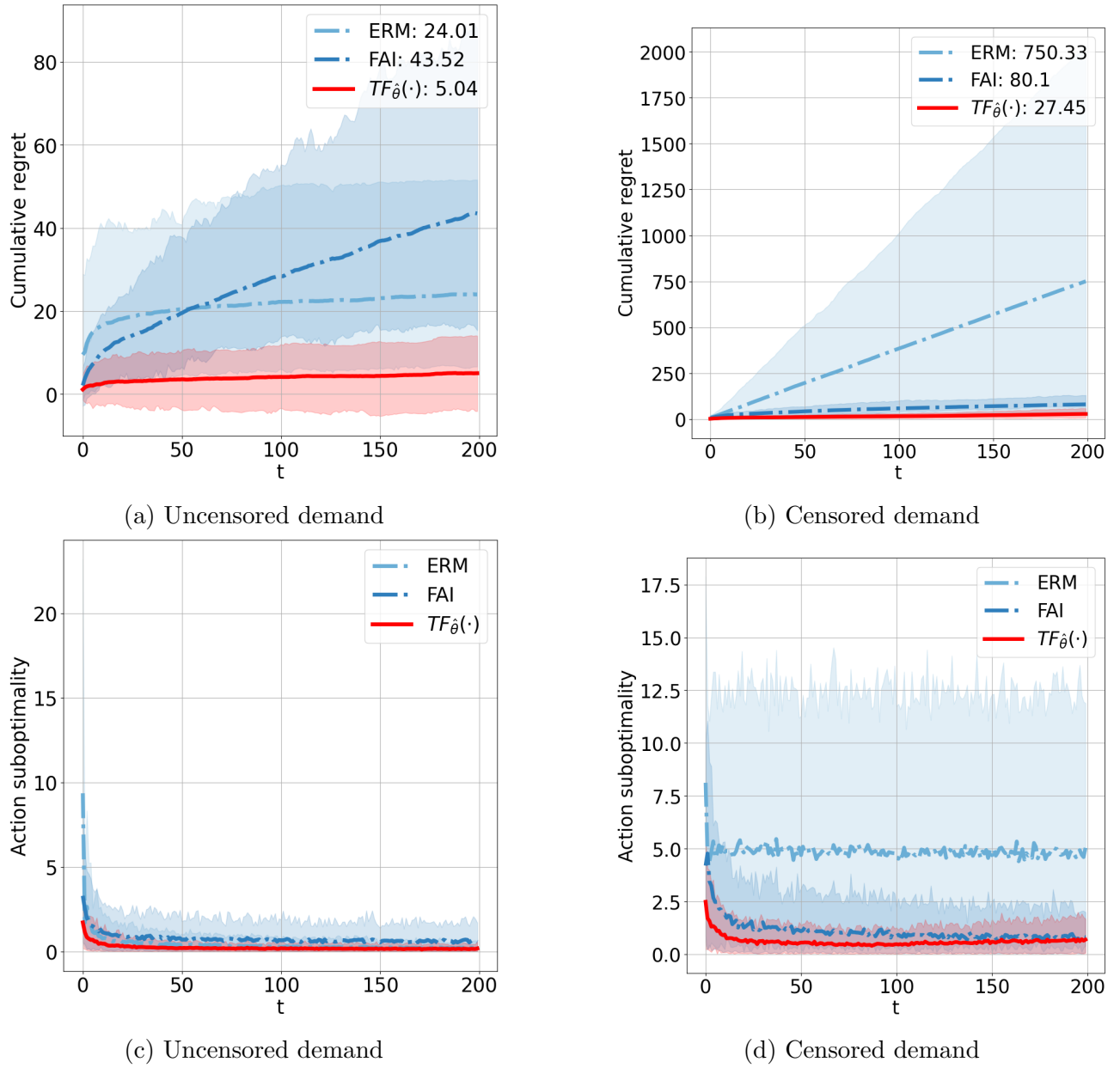


Figure 4.6: Performance under horizon generalization in newsvendor problems with uncensored and censored demands. The pre-training samples have a horizon of 100. It shows the average out-of-sample regret (first row) and action suboptimality, i.e., $|a_t^* - \text{Alg}(H_t)|$, (second row) of $\text{TF}_{\hat{\theta}}$ against benchmark algorithms. The numbers in the legend bar are the final regret at $t = 200$.

horizon) to 200 in the newsvendor problem with uncensored demands (first column) or censored demands (second column). In the censored setting, which requires exploration of actions as discussed in Section 4.2.1, we indeed evaluate whether OMGPT can maintain good performance despite the limited input sequence (i.e., the last 100 timesteps) possibly leading to forgotten explorations. The results demonstrate that OMGPT generalizes well in this extended horizon scenario, even in the censored demand setting where exploration is essential. Its actions remain

nearly optimal beyond $t = 100$, resulting in lower regret compared to benchmark algorithms.

4.6.1.4 Performance under Shifted Environment Distribution

In practice, the prior knowledge of the decision maker, $\mathcal{P}_\gamma$, may not exactly match the distribution of the testing environment. This is often referred to as the out-of-distribution (OOD) problem. In this section, we evaluate OMGPT's generalization ability under such conditions, using dynamic pricing problems as the test case. Specifically, we define the demand functions in the pricing problems as $D_t = \alpha^\top X_t - \beta^\top X_t \cdot a_t + \epsilon_t$, and adjust the generation of testing environments by altering the distributions of the noise term ϵ_t and the (generation of) parameters (α, β) to deviate from those in the pre-training environments.

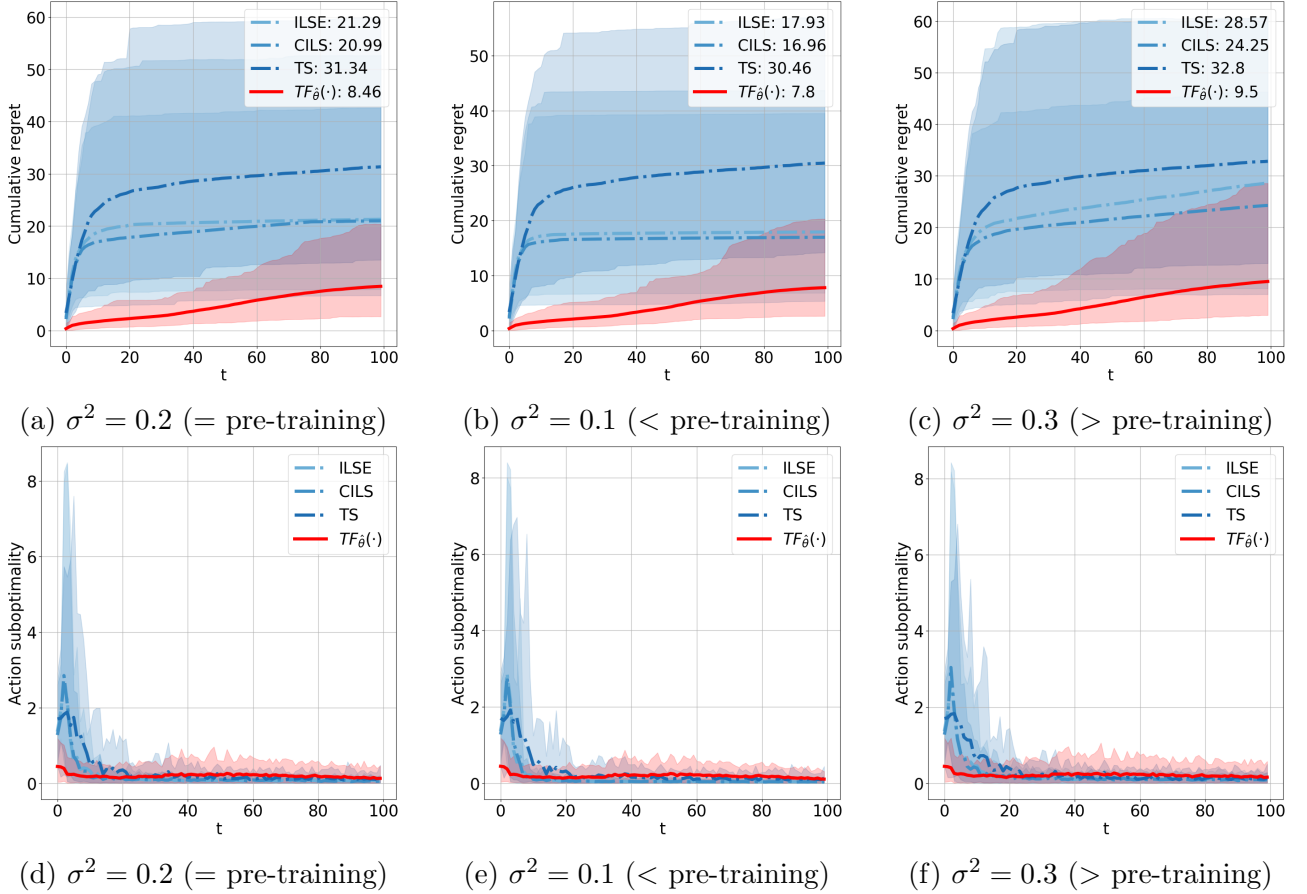


Figure 4.7: Performance under different testing noise variances, which may deviate from the pre-training variance of $\sigma^2 = 0.2$. It shows the average out-of-sample regret (first row) and action suboptimality, i.e., $|a_t^* - \text{Alg}(H_t)|$, (second row) of TF_θ against benchmark algorithms. The numbers in the legend bar are the final regret at $t = 100$.

We first test OMGPT in three different problem settings with varying testing noise distributions

$\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ while keeping the pre-training noise $\epsilon_t \sim \mathcal{N}(0, 0.2)$ for all of them: $\sigma^2 = 0.2$, which matches the noise variance used in pre-training; $\sigma^2 = 0.1$, which is smaller than that used in pre-training; and $\sigma^2 = 0.3$, which is larger. Figure 4.7 provides the results. We do not observe any significant sign of failure in OMGPT’s OOD performance: across all three settings, the benchmark algorithms consistently incur higher regret than OMGPT. Although OMGPT shows some variation in mean regret across OOD settings (specifically, lower regret when $\sigma^2 = 0.2$ and higher regret when $\sigma^2 = 0.3$), this is partially due to the varying “difficulty” of the underlying tasks. As expected, higher noise variances, which need more data for accurately estimating the demand function compared to lower variance cases, lead to worse performance for all algorithms tested. However, these variations should not be interpreted as evidence of OMGPT failing to handle OOD issues. In fact, all benchmark algorithms show similar performance fluctuations under these conditions, further demonstrating OMGPT’s robustness.

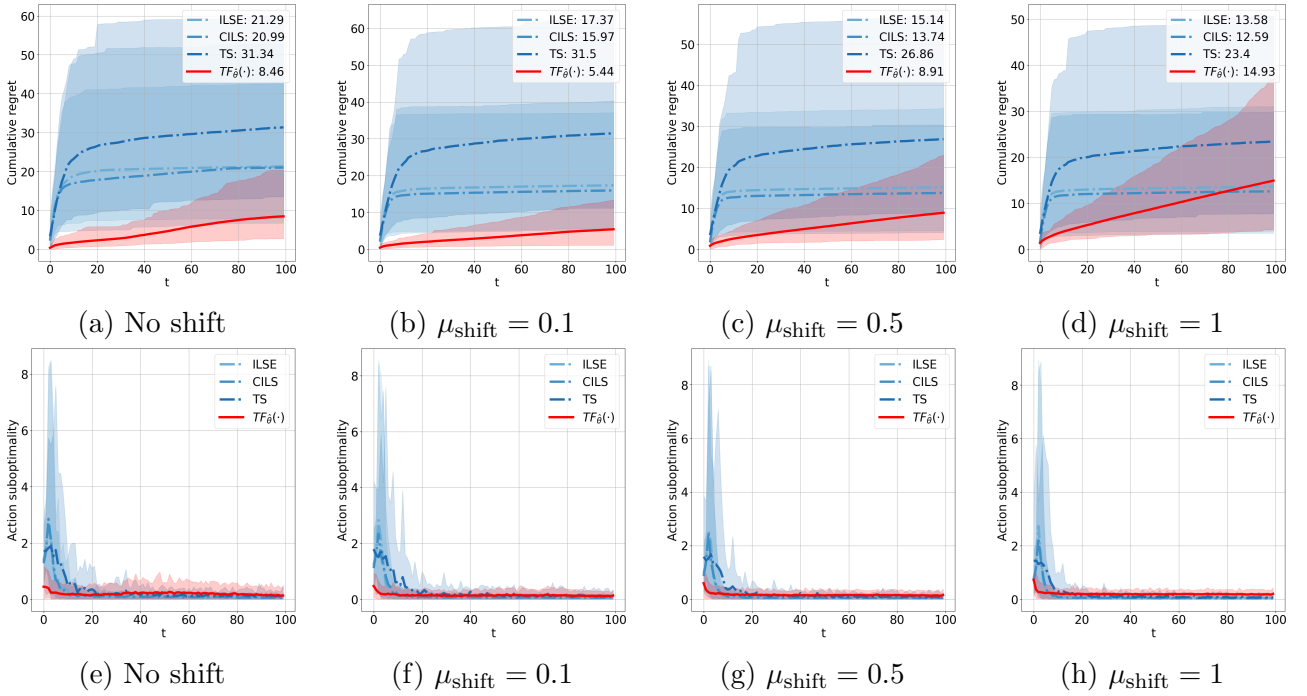


Figure 4.8: Performance under the different levels of out-of-domain shifts μ_{shift} in the testing demand function generation. It shows the average out-of-sample regret (first row) and action suboptimality, i.e., $|a_t^* - \text{Alg}(H_t)|$, (second row) of $\text{TF}_{\hat{\theta}}$ against benchmark algorithms. The numbers in the legend bar are the final regret at $t = 100$.

We further evaluate OMGPT under distributional shifts in the generation of parameters (α, β) , considering both out-of-domain and in-domain shifts. During pre-training, these parameters are sampled per environment as $\alpha \sim \text{Unif}([0.5, 1.5]^6)$ and $\beta \sim \text{Unif}([0.05, 1.05]^6)$. In the testing

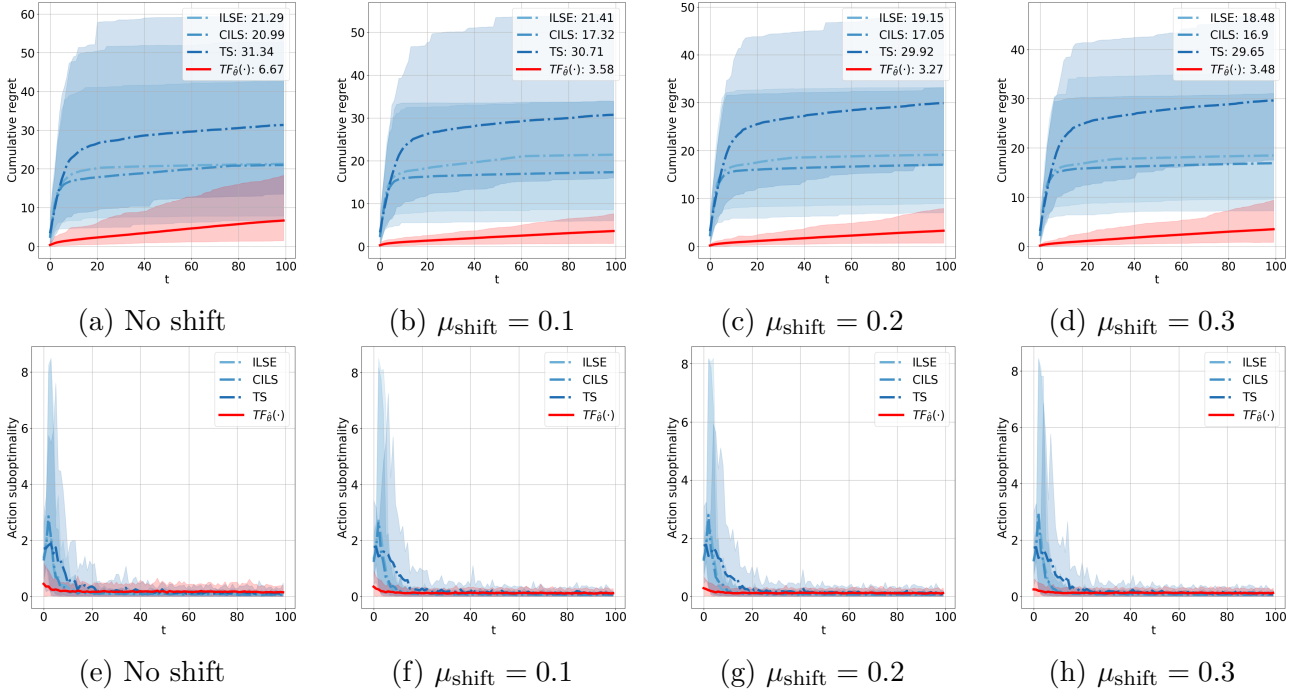


Figure 4.9: Performance under the different levels of in-domain shifts μ_{shift} in the testing demand function generation. It shows the average out-of-sample regret (first row) and action suboptimality, i.e., $|a_t^* - \text{Alg}(H_t)|$, (second row) of $TF_{\hat{\theta}}$ against benchmark algorithms. The numbers in the legend bar are the final regret at $t = 100$.

phase, two types of shifts are introduced, parameterized by the shift level μ_{shift} :

- Out-of-domain shifts:** Here, the test parameters (α, β) can be sampled beyond the original training ranges to simulate scenarios where prior knowledge fails to encompass the true environment space. Specifically, we generate $\alpha \sim \text{Unif}([0.5 + \mu_{\text{shift}}, 1.5 + \mu_{\text{shift}}]^6)$ and $\beta \sim \text{Unif}([0.05 + \mu_{\text{shift}}, 1.05 + \mu_{\text{shift}}]^6)$. Four shift levels are evaluated: $\mu_{\text{shift}} = 0$ (matching the pre-training distribution) and $\mu_{\text{shift}} = 0.1, 0.5, 1$.
- In-domain shifts:** For these shifts, sub-intervals of length $(1 - \mu_{\text{shift}})$ within the original training ranges are randomly selected, and parameters are sampled uniformly within these sub-intervals. Specifically, for a given μ_{shift} , we sample $\kappa_1 \sim \text{Unif}([0.5, 1.5 - (1 - \mu_{\text{shift}})])$ and $\kappa_2 \sim \text{Unif}([0.05, 1.05 - (1 - \mu_{\text{shift}})])$. Then, we draw $\alpha \sim \text{Unif}([\kappa_1, \kappa_1 + 1 - \mu_{\text{shift}}]^6)$ and $\beta \sim \text{Unif}([\kappa_2, \kappa_2 + 1 - \mu_{\text{shift}}]^6)$. This approach simulates a scenario in which the prior knowledge is conservative, assuming a broader feasible space than the true space that generates the testing environment. A larger μ_{shift} corresponds to more conservative prior knowledge, implying a larger expected feasible parameter space than the actual

one. We consider four shift levels: $\mu_{\text{shift}} = 0$ (matching the pre-training distribution) and $\mu_{\text{shift}} = 0.1, 0.2, 0.3$.

Figure 4.8 and Figure 4.9 illustrate OMGPT’s performance under out-of-domain and in-domain shifts, respectively. For out-of-domain shifts (Figure 4.8), we observe a performance decline at higher shift levels ($\mu_{\text{shift}} = 0.5, 1$) in (α, β) . This effect is reasonable, as (α, β) directly determines the optimal price, $\frac{\alpha^\top X_t}{2\beta^\top X_t}$. Therefore, the parameters sampled outside the training distribution’s support may lead to optimal decisions that the model has never encountered during training, resulting in performance degradation. For in-domain shifts (Figure 4.9), we observe no significant indications of failure of OMGPT in this case. Interestingly, the regret performance of OMGPT and benchmark algorithms improves slightly under these shifts. This enhancement may stem from the shifts leading to (α, β) values that are more centered within the range, thereby reducing the occurrence of parameters near the original boundaries that would otherwise be too large or too small. As a result, this shift reduces the likelihood of encountering “corner” environments where optimal decisions involve extreme values.

In practice, we recommend adopting more conservative prior knowledge to generate a broader range of training environments, as this approach minimizes potential distortions in out-of-domain performance, as observed in Figure 4.8, while only potentially incurring little sacrifice as suggested by Figure 4.9.

4.6.1.5 Solution to Model Misspecification

Most sequential decision making algorithms, including all benchmark algorithms used in our experiments, rely on structural or model assumptions about the underlying tasks. For example, in pricing and newsvendor problems, demand functions are typically assumed to be linear with respect to the context. When these algorithms are applied to misspecified environments, where such assumptions do not hold, the performance can degrade significantly.

In contrast, OMGPT provides a potential solution to model misspecification. For example, for newsvendor problems where both linear and non-linear demand functions can appear, by

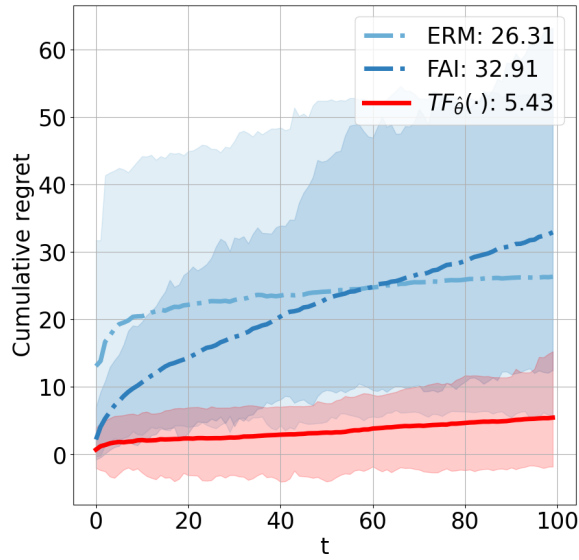
generating pre-training samples from both types of environments, a single pre-trained $\text{TF}_{\hat{\theta}}$ can leverage its large capacity to make near-optimal decisions across different types of environments.

In this section, we test OMGPT in two different scenarios: non-stationary environments and environments with multiple demand types. Specifically, we pre-train and test $\text{TF}_{\hat{\theta}}$ on environments where (i) demand functions may abruptly change at a random time $\tau \leq T$ (non-stationary setting) or (ii) there are two types of demand functions: linear and square (detailed in Appendix C.4.2). We compare $\text{TF}_{\hat{\theta}}$ with benchmark algorithms that are well-studied but specifically designed for stationary, linear demand cases.

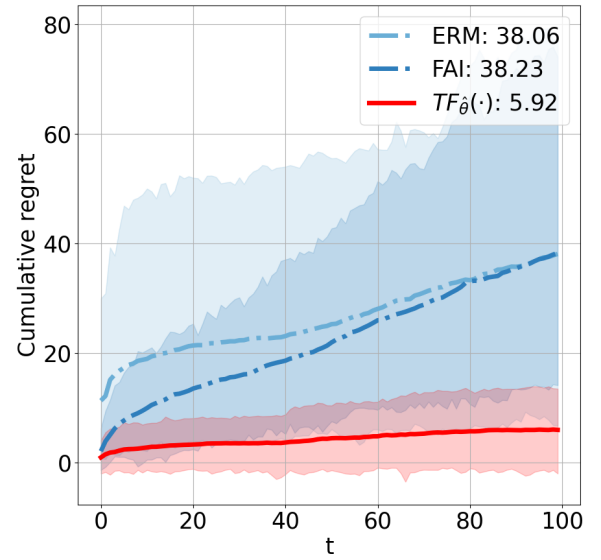
Figure 4.10 illustrates the performance in the newsvendor problem with uncensored demand, tested in environments where (i) the demand functions remain consistent throughout the horizon (first column) and (ii) the demand functions are non-stationary and may change unpredictably during the horizon, causing benchmark algorithms to face model misspecification issues (second column). Figure 4.11 shows the performance in newsvendor tasks in environments where demand functions are either (i) linear-type (first column) or (ii) square-type, where benchmark algorithms also encounter model misspecification issues (second column). The results demonstrate that benchmark algorithms, which are designed for stationary, linear demand environments, struggle when faced with model misspecifications. In contrast, OMGPT leverages its large capacity and pre-training to learn and adapt to changes in the environment or different demand types, effectively handling them during testing. These findings highlight the potential of $\text{TF}_{\hat{\theta}}$ to effectively address model misspecifications and perform robustly in diverse environments.

4.6.1.6 Demand Prediction as By-product

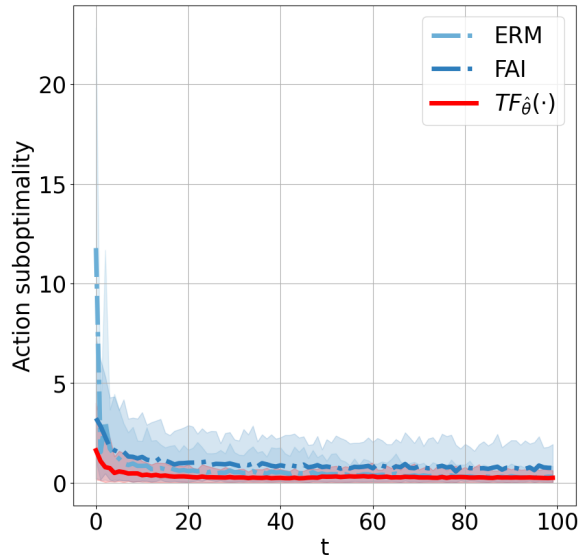
In practice, decision makers may require additional side information beyond the optimal actions to better understand the testing environment. For instance, in dynamic pricing, the decision maker might want to learn the demand given a particular price to manage inventory or plan manufacturing. In this section, we evaluate OMGPT’s ability to predict demand in a dynamic pricing scenario (with 8 unknown parameters), showcasing its potential to provide useful side



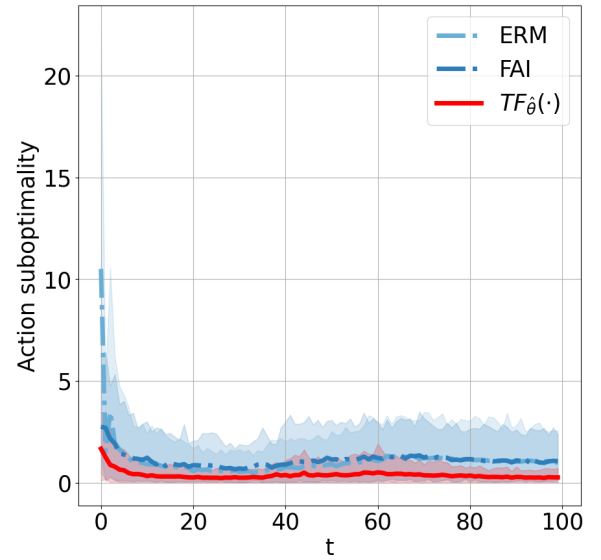
(a) Stationary environment (no misspecification)



(b) Nonstationary environment (misspecification)



(c) Stationary environment (no misspecification)

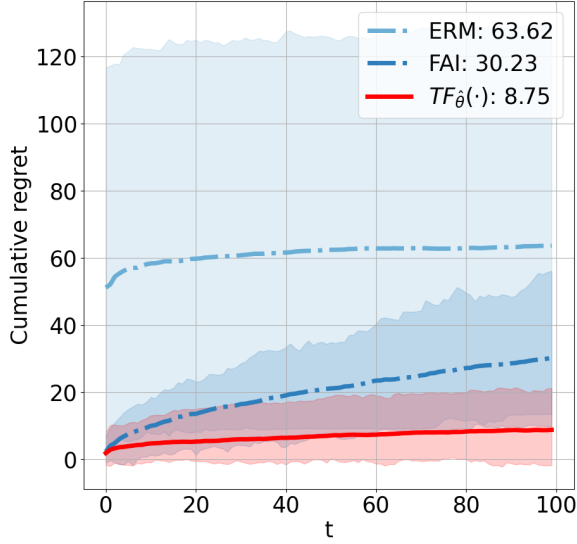


(d) Nonstationary environment (misspecification)

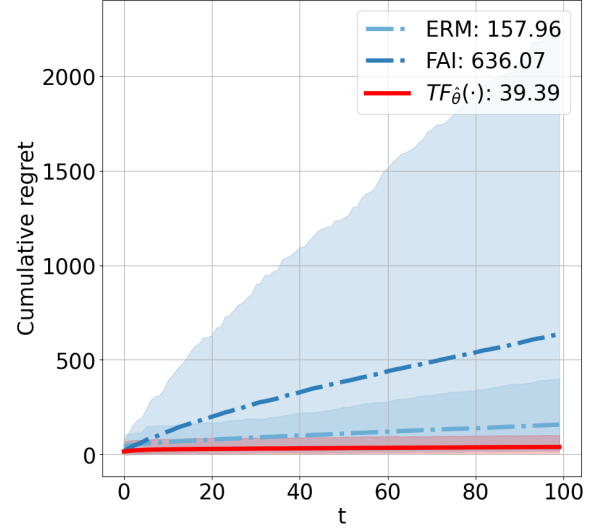
Figure 4.10: Performance in stationary environments ((a), (c)) and non-stationary environments ((b), (d)), where benchmark algorithms encounter model misspecification issues. The same $\text{TF}_{\hat{\theta}}$ is applied in all figures. It shows the average out-of-sample regret (first row) and action suboptimality, i.e., $|a_t^* - \text{Alg}(H_t)|$, (second row) of $\text{TF}_{\hat{\theta}}$ against benchmark algorithms. The numbers in the legend bar are the final regret at $t = 100$.

information.

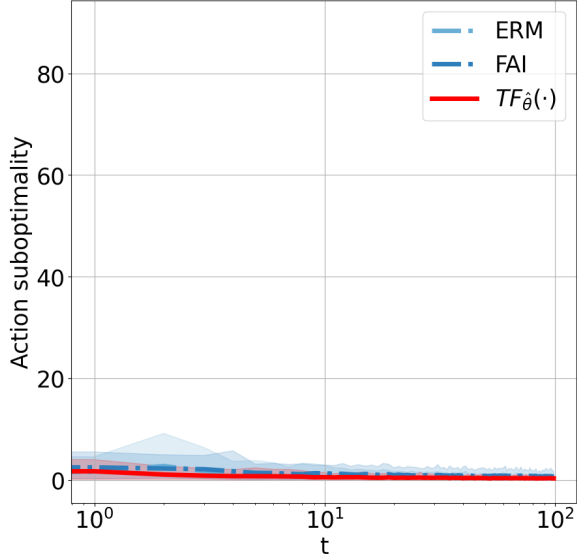
We modify OMGPT to predict demand as discussed in Section 4.4.5.3 and compare its performance to ridge regression (with a regularization term of 0.1) as a benchmark. We use ridge regression instead of ordinary least squares since the number of parameters can exceed the



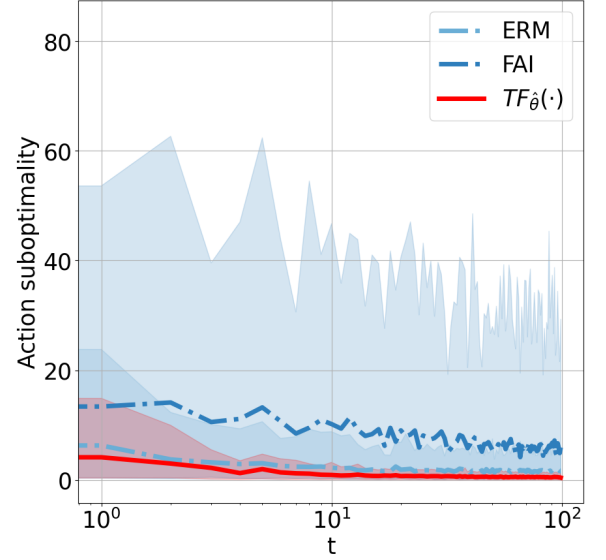
(a) Linear demand (no misspecification)



(b) Square demand (misspecification)



(c) Linear demand (no misspecification)



(d) Square demand (misspecification)

Figure 4.11: Performance in environments with linear demand ((a), (c)) and environments with square demand ((b), (d)), where benchmark algorithms encounter model misspecification issues. The same $\text{TF}_{\hat{\theta}}$ is applied in all figures. It shows the average out-of-sample regret (first row) and action suboptimality, i.e., $|a_t^* - \text{Alg}(H_t)|$, (second row) of $\text{TF}_{\hat{\theta}}$ against benchmark algorithms. The numbers in the legend bar are the final regret at $t = 100$.

number of samples in some cases. We first test the models in environments with linear demands. Figure 4.12a shows the prediction errors with varying input sequence lengths, and consequently, varying numbers of observed $((X_t, a_t), O_t)$ samples in the input (as the training samples in the demand regression task). The results indicate that OMGPT performs similarly to ridge regression when the number of samples exceeds 10. However, with fewer than 10 samples, OMGPT outperforms ridge regression. We attribute this advantage to OMGPT's ability

to leverage prior knowledge from its pre-training.

To further evaluate the capability of (a single) pre-trained OMGPT, we pre-train and test it in a more complex scenario where demand functions can be either square-type or linear-type, each with a probability of $1/2$. Figure 4.12b and 4.12c presents the prediction errors with varying numbers of observed $((X_t, a_t), O_t)$ samples in the input sequence for different demand types, but using the same $\text{TF}_{\hat{\theta}}$. While ridge regression slightly outperforms OMGPT in the linear-type demand cases (Figure 4.12b), it performs significantly worse in the square-type demand cases (Figure 4.12c), where a linear model fails to capture the square demand function accurately. This highlights one single pre-trained $\text{TF}_{\hat{\theta}}$'s superior ability to predict observations across different types of environments.

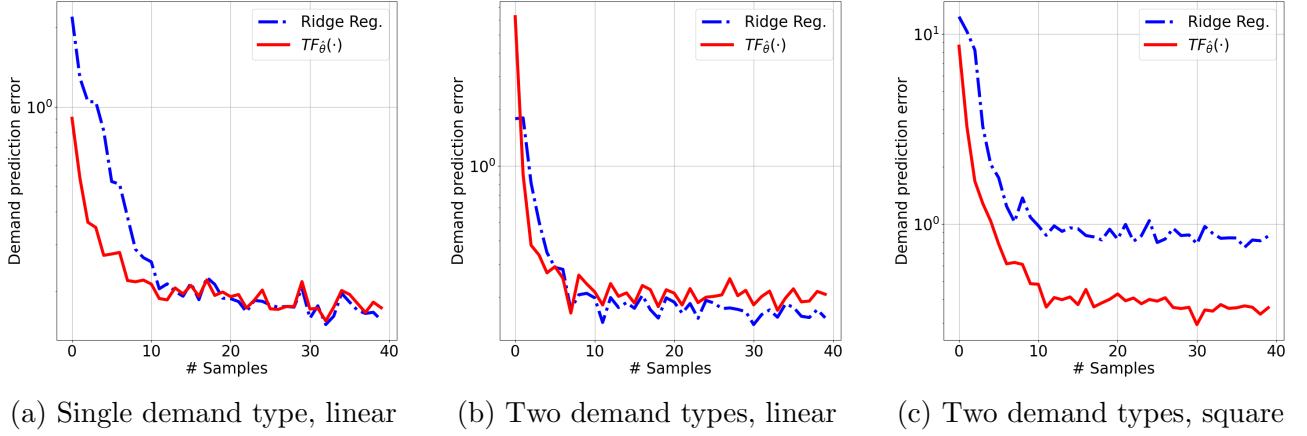


Figure 4.12: Out-of-sample demand prediction error comparison between OMGPT ($\text{TF}_{\hat{\theta}}$) and ridge regression across different scenarios. (a): Single demand type, where $\text{TF}_{\hat{\theta}}$ is pre-trained in environments with linear demand, and both methods are tested on linear demand environments. (b) and (c): Two demand types, where $\text{TF}_{\hat{\theta}}$ is pre-trained in environments with a mix of linear and square demand (50% each). Both methods are then tested on either linear demand (b) or square demand (c).

4.6.2 Mechanisms Interpretation

In Section 4.5, we analytically demonstrate that the pre-trained $\text{TF}_{\hat{\theta}}$ behaves similarly to the Bayes-optimal decision maker Alg^* and numerically observe such alignment between them (see Figure 4.3 and Appendix C.4.1.1). In this section, we further explore and interpret the internal mechanisms of $\text{TF}_{\hat{\theta}}$ when predicting optimal actions.

Specifically, recall that the Bayes-optimal decision maker Alg^* is defined as:

$$\text{Alg}^*(H) = \arg \min_{a \in \mathcal{A}} \int_{\gamma} l(a, a_t^{(\gamma)*}) \mathcal{P}(\gamma|H) d\gamma.$$

It selects the optimal action that minimizes the expected divergence from the true optimal action $a_t^{(\gamma)*}$, with the expectation with respect to the posterior distribution of the environment $\mathcal{P}(\gamma|H)$. This formulation consists of two key components: (i) an environment inference part, which infers the testing environment based on the history H (i.e., $\mathcal{P}(\gamma|H)$), and (ii) an optimal decision inference part, which infers the optimal action $a_t^{(\gamma)*}$ for each environment γ based on the current context X_t .

In this section, we demonstrate that when the pre-trained $\text{TF}_{\hat{\theta}}$ predicts optimal decisions, it implicitly mirrors these two components. This suggests that $\text{TF}_{\hat{\theta}}$ not only (finally) produces actions that align with the Bayes-optimal decision maker Alg^* , but its internal mechanisms also operate in a similar manner. In essence, these findings provide further numerical support for the analysis presented in Section 4.5.

4.6.2.1 Environment Inference

There Exists Environment Inference We first investigate whether environment inference is conducted by $\text{TF}_{\hat{\theta}}$. To explore this, we pre-train a 12-layer $\text{TF}_{\hat{\theta}}$ on dynamic pricing tasks where the demand functions can either be linear or square, each appearing with a 50% probability. After pre-training, we visualize the output embedding vectors from each layer using the t-SNE method (with the 0-layer representing the input vectors to the GPT model, see Appendix C.1.1). The visualization is based on 500 randomly sampled testing histories, each with a length of $T = 100$, using the embedding vector corresponding to X_{100} . Among the 500 samples, half have linear demand and the other half have square demand. Figure 4.13 presents the results. We observe that while the embedding vectors from the input layer and the first layer (Figures (a) and (b)) do not show a clear separation between linear and square demand functions, by layer 2, the vectors are well-separated. This indicates that as the input sequence is processed

through the layers, the pre-trained model begins to infer the environment information.

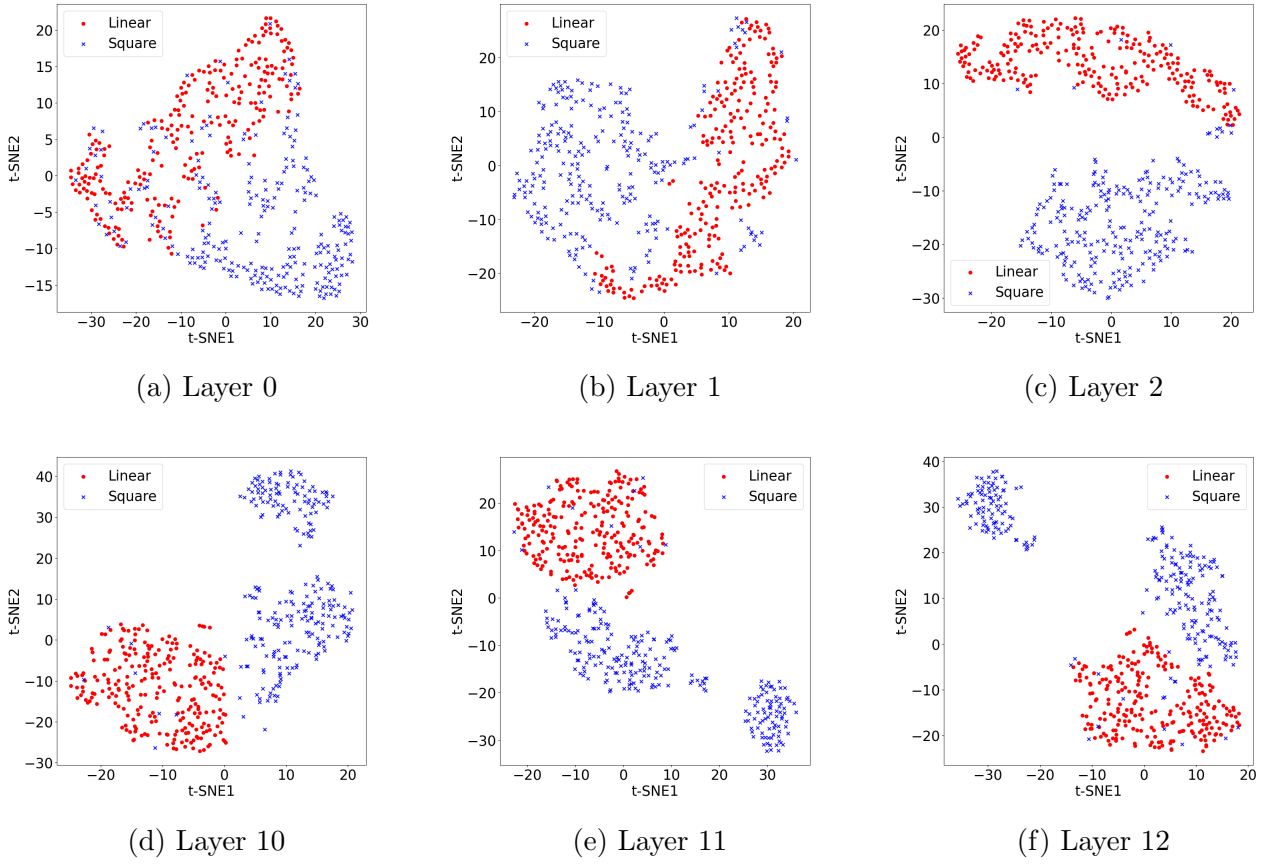


Figure 4.13: The vectors corresponding to different demand function types become clearly separated starting from layer 2, indicating that environment inference occurs during the processing. Additional visualizations for other layers are provided in Appendix C.4.1.4.

Optimal Decision Prediction Aligns with Environment Inference Next, we investigate whether the pre-trained model utilizes the inferred environment when predicting optimal decisions. For this, we use the linear probing technique [Ala16], a commonly used method to understand the internal workings of deep learning models. Similar to visualizing embedding vectors from intermediate layers, linear probing tests whether intermediate layer outputs contain specific information by linearly regressing or classifying quantities of interest (as the target) based on the intermediate embedding vectors (as the input) and analyzing the prediction error. Intuitively, smaller errors indicate that the embedding vectors from that layer can recover the relevant information through simple linear regression/classification, suggesting that the embedding vectors contain such information.

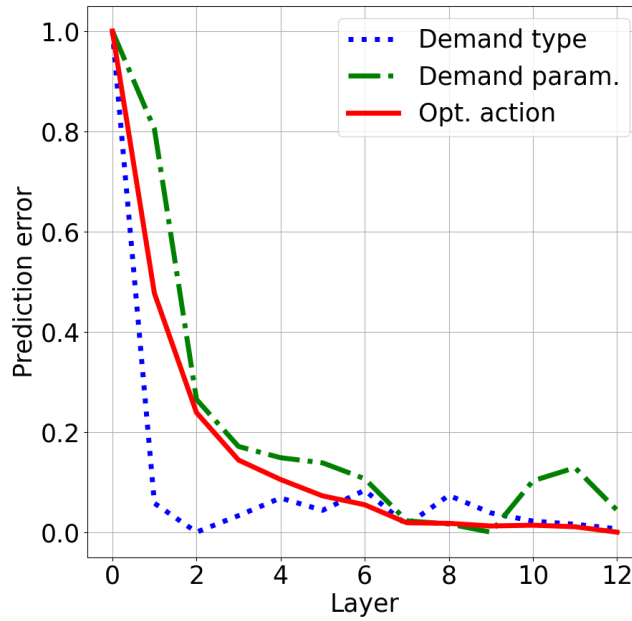


Figure 4.14: Normalized prediction errors of demand type, demand parameters, and optimal actions across layers, using linear probing. The prediction errors for demand/environment parameters (green, dashed line) and optimal actions (red, solid line) follow similar trends, which suggests a strong correlation between environment inference and optimal action prediction.

We apply linear probing to the same pre-trained $\text{TF}_{\hat{\theta}}$ from Figure 4.13 and probe three values: demand function type (i.e., linear or square), demand function parameters (i.e., the unknown parameters of the demand function), and optimal actions. The linear probes are trained using 1024 randomly sampled environments and histories. The input embedding vectors to train linear probes correspond to X_{100} 's in the histories. Among the training samples, half have linear demand and the other half have square demand. The reported values are the averages over 256 randomly sampled (testing) histories, with half having linear demand and the other half having square demand. Figure 4.14 shows the normalized prediction errors from linear probing, with values scaled between 0 and 1 for comparison. The results show that:

- A minor observation is that the prediction error for the demand function type becomes relatively small after layer 1, aligning with our observations in Figure 4.13. This occurs earlier than the layer where the demand function parameter prediction error becomes small (after layer 7), which corresponds to the natural flow of estimation—first identifying the model class (linear or square), then estimating the model parameters.
- More importantly, the prediction errors for both the parameters and optimal actions

follow a similar trend: both decrease quickly before layer 7 (with similar rates across layers) and then stabilize at a relatively low level. It is important to note that the pre-training only requires the OMGPT to predict the optimal actions, not the exact demand parameters. For example, in a linear demand model, the optimal price can be computed using $\frac{\alpha^\top X}{-2\beta^\top X}$ (where α, β are the unknown demand parameters), so exact values for α and β are unnecessary. (The divergence in prediction errors between the demand parameters and optimal actions after layer 9 can be attributed to this “distillation”: model “distills” unnecessary information to better predict optimal actions.) This similar trend between the predictions of parameters and optimal actions suggests that optimal decision prediction is highly correlated with parameter/environment inference. While we cannot establish a causal relationship, the result suggests that optimal decision prediction is closely related to environment inference.

4.6.2.2 Optimal Decision Inference

In this section, we investigate whether the pre-trained $\text{TF}_{\hat{\theta}}$ performs optimal decision inference. Specifically, we rewrite the input sequence of $\text{TF}_{\hat{\theta}}$ as follows:

$$H_t = \{(X_1, a_1, O_1), \dots, (X_{t-1}, a_{t-1}, O_{t-1})\} \cup \{X_t\} = \tilde{H}_{t-1} \cup \{X_t\}$$

where $\tilde{H}_{t-1} = \{(X_1, a_1, O_1), \dots, (X_{t-1}, a_{t-1}, O_{t-1})\}$ contains all previously collected samples before t and can be used to infer the unknown environment γ (which generates H_t). Assuming X_t is generated independently of γ conditional on $\tilde{H}_{t-1}$, we want to verify whether:

$$\text{TF}_{\hat{\theta}}(H_t) = \hat{a}^*(X_t, \hat{\gamma}(\tilde{H}_{t-1})) \quad (4.18)$$

holds for some functions $\hat{\gamma}$ (representing environment inference) and $\hat{a}^*$ (representing optimal action inference). If this holds, it implies that the pre-trained $\text{TF}_{\hat{\theta}}$ does not simply memorize pre-training data. Instead, it follows a decision rule $\hat{a}^*(X_t, \hat{\gamma})$ that maps from the current context X_t and the inferred environment $\hat{\gamma}$ (based on $\tilde{H}_{t-1}$) to the predicted action, thereby

performing optimal decision inference.

To investigate this, we manipulate the history sequence to create $H'_t = \tilde{H}_{t-1} \cup \{X'_t\}$ by altering the original sampled context X_t to X'_t (while keeping γ and $\tilde{H}_{t-1}$ unchanged), and meanwhile we ensure that no similar pairs $(\tilde{H}_{t-1}, X'_t)$ are seen during pre-training. We then compare the behavior of $\text{TF}_{\hat{\theta}}$ in these manipulated histories with that in non-manipulated histories (which are generated in the same way as the pre-training data). If $\text{TF}_{\hat{\theta}}$ behaves similarly in both cases, predicting nearly optimal actions, it suggests that $\text{TF}_{\hat{\theta}}$ performs optimal decision inference rather than simple memorization, as it has not encountered any similar pairs $(\tilde{H}_{t-1}, X'_t)$ during pre-training.

In the experiment, we create such manipulated histories in a newsvendor problem with non-perishable products (details provided in Appendix C.4.2). The context is generated by $X_t = q(\tilde{H}_{t-1})$, and the optimal action is $a_t^* = a^*(X_t, \gamma)$ with some deterministic functions q and a^* . We pre-train OMGPT on un-manipulated data ($X_t = q(\tilde{H}_{t-1})$) and test it on both un-manipulated and manipulated histories. In the manipulated case, we set $X'_t = \epsilon'_t$, where ϵ'_t is independently sampled from a uniform distribution over the range $[0, 10]$ and is independent of $\tilde{H}_{t-1}$. Figure 4.15a shows the distribution of $a^*(X'_t, \gamma) - a^*(X_t, \gamma)$ at $t = 50$, i.e., the changes of optimal actions after manipulation, over 1000 randomly sampled γ 's. It shows most optimal actions are shifted after manipulations (not around 0).

We then compare $\text{TF}_{\hat{\theta}}(H_t) - a_t^*$ before ($a_t^* = a^*(X_t, \gamma)$) and after manipulation ($a_t^* = a^*(X'_t, \gamma)$) to check if there exists a significant change in $\text{TF}_{\hat{\theta}}$'s behavior after manipulations. Figure 4.15b presents boxplots of $\text{TF}_{\hat{\theta}}(H_t) - a_t^*$. While the variation increases after manipulation, the predicted actions $\text{TF}_{\hat{\theta}}(H_t)$'s are concentrated around the optimal actions in both cases, as opposed to the significant shift observed in the optimal actions in Figure 4.15a. This indicates that $\text{TF}_{\hat{\theta}}$ can still perform nearly optimally, even after the manipulation of contexts. These results demonstrate that $\text{TF}_{\hat{\theta}}$ does not rely on memorizing pre-training data but instead (implicitly) performs optimal decision inference.

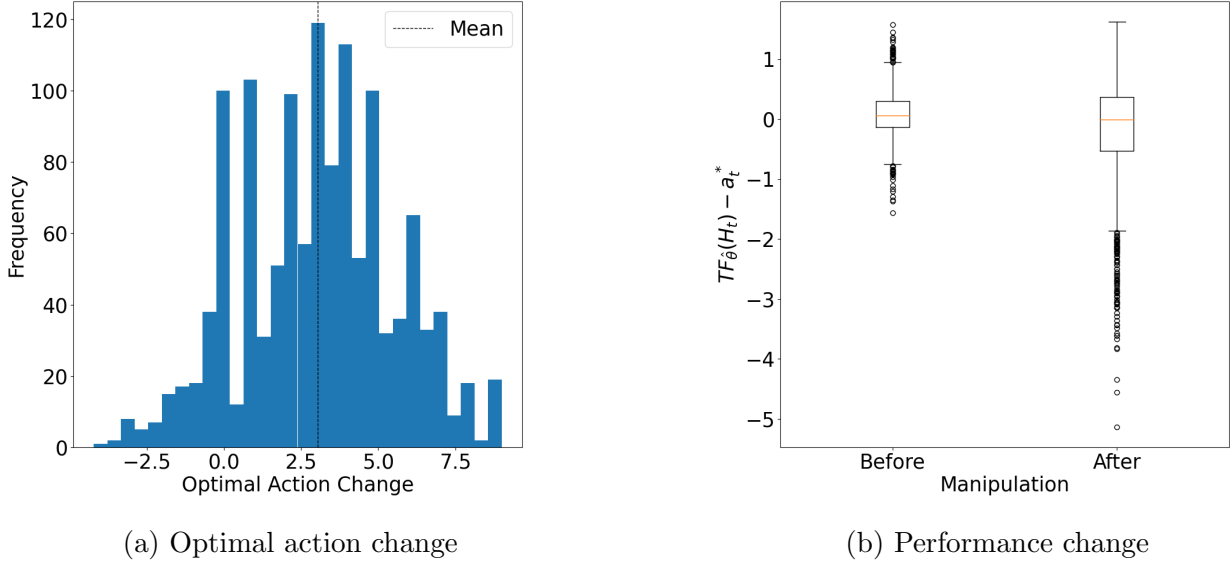


Figure 4.15: (a) Histogram of optimal action changes due to the manipulation of the context X_t . (b) Boxplots comparing $TF_{\hat{\theta}}$'s behavior ($TF_{\hat{\theta}}(H_t) - a_t^*$) before and after manipulations.

4.7 Conclusion

In this work, we frame sequential decision making problems as sequence modeling tasks, where we predict optimal actions based on historical data. We propose a supervised learning paradigm that does not assume any analytical model structure and can easily incorporate the decision maker's prior knowledge. Specifically, we (i) design OMGPT, a GPT-based neural network architecture for sequential decision making problems, (ii) introduce a new pre-training scheme tailored to these problems, and (iii) demonstrate through numerical and analytical results that OMGPT exhibits strong performance and capabilities, positioning it as a promising solution for a wide range of operational decision making applications.

Chapter 5

Conclusion

This thesis has explored the intersection of artificial intelligence and operations management, demonstrating that deep learning models and neural network architectures can significantly advance both predictive and prescriptive analytics within operations management. Through a series of targeted studies, we have demonstrated the applicability and effectiveness of deep learning in choice modeling, assortment optimization, and sequential decision-making tasks.

In the first contribution, we presented a neural network-based discrete choice model specifically designed for single-choice assortment optimization. Unlike traditional parametric models, this approach is capable of capturing complex and nuanced consumer behaviors, explicitly accounting for assortment effects. Our proposed integer programming formulation allowed for effective operationalization of neural network outputs into optimal assortments, achieving competitive performance against well-established choice models across simulated and real-world scenarios.

Building on this foundation, our second contribution developed the Transformer Choice Net, a versatile neural architecture capable of addressing single, sequential, and multi-choice decision paradigms. This unified transformer-based framework significantly outperformed existing benchmarks, simplifying modeling efforts across various consumer choice contexts. The flexibility and scalability of this approach allow it to be broadly applicable, with minimal requirement for customization to specific choice modeling problems.

The third and final contribution introduced OMGPT, a GPT-inspired transformer architecture designed to address general sequential decision-making challenges prevalent in operations management. OMGPT offered a significant paradigm shift from traditional model-based methods towards a fully data-driven approach. By training on extensive simulated decision sequences, OMGPT effectively learned robust decision-making policies, delivering near-optimal solutions across diverse operational problems such as dynamic pricing, inventory control, and queuing management. Additionally, theoretical analysis established connections between OMGPT's predictive accuracy and Bayesian optimality, providing a foundational understanding of its high-performance capabilities.

In summary, this thesis has demonstrated that artificial intelligence can serve as a powerful methodological bridge, enhancing and even surpassing traditional operations management approaches. The integration of data-driven deep learning techniques into operations management practices can enable more accurate modeling, improved decision-making capabilities, and greater adaptability to real-world complexities. Moving forward, the methodologies introduced in this thesis set a foundation for future research directions, potentially expanding into even broader applications and fostering increasingly autonomous, intelligent, and responsive operational decision systems.

Bibliography

- [ACD⁺15] Kareem Amin, Rachel Cummings, Lili Dworkin, Michael Kearns, and Aaron Roth. Online learning and profit maximization from revealed preferences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.
- [AD22] Ali Aouad and Antoine Désir. Representing random utility choice models with neural networks. *arXiv preprint arXiv:2207.12877*, 2022.
- [ADG⁺22] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.
- [AG13] Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *International conference on machine learning*, pages 127–135. PMLR, 2013.
- [AJ08] Dan Ariely and Simon Jones. *Predictably irrational*. HarperCollins New York, 2008.
- [AJ22] Shipra Agrawal and Randy Jia. Learning in structured mdps with convex cost functions: Improved regret bounds for inventory management. *Operations Research*, 70(3):1646–1664, 2022.
- [AKAP23] Ioanna Arkoudi, Rico Krueger, Carlos Lima Azevedo, and Francisco C Pereira. Combining discrete choice models and neural networks through embeddings: Formulation, interpretability and performance. *Transportation Research Part B: Methodological*, 175:102783, 2023.

- [Ala16] Guillaume Alain. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.
- [Apo74] T.M. Apostol. *Mathematical Analysis*. Addison-Wesley series in mathematics. Addison-Wesley, 1974.
- [AS16] Aydın Alptekinoglu and John H Semple. The exponential choice model: A new alternative for assortment and price optimization. *Operations Research*, 64(1):79–93, 2016.
- [ASC23] Sander Aarts, David B Shmoys, and Alex Coy. An interpretable determinantal choice model for subset selection. *arXiv preprint arXiv:2302.11477*, 2023.
- [Axs15] Sven Axsäter. *Inventory control*, volume 225. Springer, 2015.
- [BAL85] M. Ben-Akiva and S. Lerman. *Discrete-Choice Analysis: Theory and Application to Travel Demand*. MIT Press, Cambridge, MA, 1985.
- [BB20] Amanda Bower and Laura Balzano. Preference modeling with context-dependent salient features. In *International Conference on Machine Learning*, pages 1067–1077. PMLR, 2020.
- [BB22] Shane Barratt and Stephen Boyd. Fitting feature-dependent markov chains. *Journal of Global Optimization*, pages 1–18, 06 2022.
- [BBB⁺22] David Brandfonbrener, Alberto Bietti, Jacob Buckman, Romain Laroche, and Joan Bruna. When does return-conditioned supervised learning work for offline reinforcement learning? *Advances in Neural Information Processing Systems*, 35:1542–1553, 2022.
- [BBF⁺16] Pietro Belotti, Pierre Bonami, Matteo Fischetti, Andrea Lodi, Michele Monaci, Amaya Nogales-Gómez, and Domenico Salvagnin. On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, 65(3):545–566, 2016.

- [BÇRS09] Alain Bensoussan, Metin Çakanyıldırım, Andrew Royal, and Suresh P Sethi. Bayesian and adaptive controls for a newsvendor facing exponential demand. *Risk and Decision Analysis*, 1(4):197–210, 2009.
- [Bel22] Yonatan Belinkov. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219, 2022.
- [Ber16] Gerardo Berbeglia. Discrete choice models based on random walks. *Operations Research Letters*, 44(2):234–237, 2016.
- [BFBNB24] Anirudhan Badrinath, Yannis Flet-Berliac, Allen Nie, and Emma Brunskill. Waypoint transformer: Reinforcement learning via supervised learning with intermediate targets. *Advances in Neural Information Processing Systems*, 36, 2024.
- [BFG09] Tudor Bodea, Mark Ferguson, and Laurie Garrow. Data set—choice-based revenue management: Data from a major hotel chain. *Manufacturing & Service Operations Management*, 11(2):356–361, 2009.
- [BFS⁺23] Yicheng Bai, Jacob Feldman, Danny Segev, Huseyin Topaloglu, and Laura Wagner. Assortment optimization under the multi-purchase multinomial logit choice model. *Operations Research*, 2023.
- [BGG16] Jose Blanchet, Guillermo Gallego, and Vineet Goyal. A markov chain approximation to choice modeling. *Operations Research*, 64(4):886–905, 2016.
- [BGV⁺20] Kunal Banerjee, Rishi Raj Gupta, Karthik Vyas, Biswajit Mishra, et al. Exploring alternatives to softmax function. *arXiv preprint arXiv:2011.11538*, 2020.
- [Bie18] M. Bierlaire. 2018.
- [BK21] Gah-Yi Ban and N Bora Keskin. Personalized dynamic pricing with machine learning: High-dimensional features and heterogeneous elasticity. *Management Science*, 67(9):5549–5568, 2021.

- [BKT18] Austin R Benson, Ravi Kumar, and Andrew Tomkins. A discrete choice model for subset selection. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 37–45, 2018.
- [BLS22] John R Birge, Xiaocheng Li, and Chunlin Sun. Learning from stochastically revealed preference. *arXiv preprint arXiv:2206.01484*, 2022.
- [BM59] Henry David Block and Jacob Marschak. Random orderings and stochastic theories of response. 1959.
- [BM00] Yves Bentz and Dwight Merunka. Neural networks and the multinomial logit for brand choice modelling: a hybrid approach. *Journal of Forecasting*, 19(3):177–200, 2000.
- [BMDV09] Juan José Miranda Bront, Isabel Méndez-Díaz, and Gustavo Vulcano. A column generation algorithm for choice-based network revenue management. *Operations research*, 57(3):769–784, 2009.
- [BP85] Richard R Batsell and John C Polking. A new class of market share models. *Marketing Science*, 4(3):177–198, 1985.
- [BR12] Josef Broder and Paat Rusmevichientong. Dynamic pricing under a general parametric choice model. *Operations Research*, 60(4):965–980, 2012.
- [BR19] Gah-Yi Ban and Cynthia Rudin. The big data newsvendor: Practical insights from machine learning. *Operations Research*, 67(1):90–108, 2019.
- [BS00] Apostolos N Burnetas and Craig E Smith. Adaptive ordering and pricing for perishable products. *Operations Research*, 48(3):436–443, 2000.
- [BZ15] Omar Besbes and Assaf Zeevi. On the (surprising) sufficiency of linear models for dynamic pricing with demand learning. *Management Science*, 61(4):723–739, 2015.
- [CCZ⁺14] Michele Conforti, Gérard Cornuéjols, Giacomo Zambelli, et al. *Integer programming*, volume 271. Springer, 2014.

- [CGT21] Ningyuan Chen, Guillermo Gallego, and Zhuodong Tang. Estimating discrete choice models with random forests. In *INFORMS International Conference on Service Science*, pages 184–196. Springer, 2021.
- [CGWW23] Ningyuan Chen, Pin Gao, Chenhao Wang, and Yao Wang. Assortment optimization for the multinomial logit model with repeated customer interactions. *Available at SSRN 4526247*, 2023.
- [CLM22] Ningyuan Chen, Saman Lagzi, and Joseph Milner. Using neural networks to guide data-driven operational decisions. *Available at SSRN 4217092*, 2022.
- [CLPL20] Maxime C Cohen, Ilan Lobel, and Renato Paes Leme. Feature-based dynamic pricing. *Management Science*, 66(11):4921–4943, 2020.
- [CLR⁺21] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.
- [CLRS11] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 208–214. JMLR Workshop and Conference Proceedings, 2011.
- [CLY24] Guanting Chen, Xiaocheng Li, and Yinyu Ye. An improved analysis of lp-based control for revenue management. *Operations Research*, 72(3):1124–1138, 2024.
- [CM22] Yi-Chun Chen and Velibor V Mišić. Decision forest: A nonparametric approach to modeling irrational choice. *Management Science*, 2022.
- [CY⁺01] Hong Chen, David D Yao, et al. *Fundamentals of queueing networks: Performance, asymptotics, and optimization*, volume 4. Springer, 2001.
- [Dag14] Carlos Daganzo. *Multinomial probit: the theory and its application to demand forecasting*. Elsevier, 2014.

- [DB15] Arnoud V Den Boer. Dynamic pricing and learning: historical origins, current research, and new directions. *Surveys in operations research and management science*, 20(1):1–18, 2015.
- [DBZ14] Arnoud V Den Boer and Bert Zwart. Simultaneously learning and optimizing using controlled variance pricing. *Management science*, 60(3):770–783, 2014.
- [DFVR03] Daniela Pucci De Farias and Benjamin Van Roy. The linear programming approach to approximate dynamic programming. *Operations research*, 51(6):850–865, 2003.
- [DGSY20] Antoine Désir, Vineet Goyal, Danny Segev, and Chun Ye. Constrained assortment optimization under the markov chain-based choice model. *Management Science*, 66(2):698–721, 2020.
- [DHR24] Jingying Ding, Woonghee Tim Huh, and Ying Rong. Feature-based inventory control with censored demand. *Manufacturing & Service Operations Management*, 2024.
- [DLD⁺22] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.
- [DXX18] Linhao Dong, Shuang Xu, and Bo Xu. Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5884–5888. IEEE, 2018.
- [EEKL21] Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline rl via supervised learning? *arXiv preprint arXiv:2112.10751*, 2021.
- [FJ17] Matteo Fischetti and Jason Jo. Deep neural networks as 0-1 mixed integer linear programs: A feasibility study. *arXiv preprint arXiv:1712.06174*, 2017.

- [FJS09] Vivek Farias, Srikanth Jagabathula, and Devavrat Shah. A data-driven approach to modeling choice. *Advances in Neural Information Processing Systems*, 22, 2009.
- [FJS13] Vivek F Farias, Srikanth Jagabathula, and Devavrat Shah. A nonparametric approach to modeling choice with limited data. *Management science*, 59(2):305–322, 2013.
- [FS23] Qi Feng and J George Shanthikumar. The framework of parametric and non-parametric operational data analytics. *Production and Operations Management*, 32(9):2685–2703, 2023.
- [FSLW18] Kris Johnson Ferreira, David Simchi-Levi, and He Wang. Online network revenue management using thompson sampling. *Operations research*, 66(6):1586–1602, 2018.
- [FT17] Jacob B Feldman and Huseyin Topaloglu. Revenue management under the markov chain choice model. *Operations Research*, 65(5):1322–1342, 2017.
- [GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [GBVMZ22] Joren Gijsbrechts, Robert N Boute, Jan A Van Mieghem, and Dennis J Zhang. Can deep reinforcement learning improve inventory management? performance on lost sales, dual-sourcing, and multi-echelon problems. *Manufacturing & Service Operations Management*, 24(3):1349–1368, 2022.
- [GH20] Arushi Gupta and Daniel Hsu. Parameter identification in markov chain choice models. *Theoretical Computer Science*, 808:99–107, 2020.
- [GP01] Gregory A Godfrey and Warren B Powell. An adaptive, distribution-free algorithm for the newsvendor problem with censored demands, with applications to inventory and distribution. *Management Science*, 47(8):1101–1112, 2001.

- [GPSW17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.
- [GRS18] Noah Golowich, Alexander Rakhlin, and Ohad Shamir. Size-independent sample complexity of neural networks. In *Conference On Learning Theory*, pages 297–299. PMLR, 2018.
- [GT19] Guillermo Gallego and Huseyin Topaloglu. *Revenue management and pricing analytics*, volume 209. Springer, 2019.
- [GT22] Sebastian Gabel and Artem Timoshenko. Product choice with large assortments: A scalable deep-learning model. *Management Science*, 68(3):1808–1827, 2022.
- [GTLV22] Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.
- [Gup88] Sunil Gupta. Impact of sales promotions on when, what, and how much to buy. *Journal of Marketing research*, 25(4):342–355, 1988.
- [HG16] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.
- [HHS23] Jinhui Han, Ming Hu, and Guohao Shen. Deep neural newsvendor. *arXiv preprint arXiv:2309.13830*, 2023.
- [HKZ12] J Michael Harrison, N Bora Keskin, and Assaf Zeevi. Bayesian dynamic pricing policies: Learning and earning under a binary prior distribution. *Management Science*, 58(3):570–586, 2012.
- [Hoc97] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.
- [Hor91] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

- [HR09] Woonghee Tim Huh and Paat Rusmevichientong. A nonparametric asymptotic analysis of inventory planning with censored demand. *Mathematics of Operations Research*, 34(1):103–123, 2009.
- [HSLY22] Yiqun Hu, David Simchi-Levi, and Zhenzhen Yan. Learning mixed multinomial logits with provable guarantees. *Advances in Neural Information Processing Systems*, 35:9447–9459, 2022.
- [HWC⁺22] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on vision transformer. *IEEE transactions on pattern analysis and machine intelligence*, 45(1):87–110, 2022.
- [HZIPBA20] Yafei Han, Christopher Zengras, Francisco Camara Pereira, and Moshe Ben-Akiva. A neural-embedded choice model: Tastenet-mnl modeling taste heterogeneity with flexibility and interpretability. *arXiv preprint arXiv:2002.00922*, 2020.
- [HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [IYZ21] Zachary Izzo, Lexing Ying, and James Zou. How to learn when data reacts to your model: performative gradient descent. In *International Conference on Machine Learning*, pages 4641–4650. PMLR, 2021.
- [Jag14] Srikanth Jagabathula. Assortment optimization under general choice. *Available at SSRN 2512831*, 2014.
- [Jas15] Stefanus Jasin. Performance of an lp-based control for revenue management with unknown demand parameters. *Operations Research*, 63(4):909–915, 2015.
- [JFD21] Bruno Jacobs, Dennis Fok, and Bas Donkers. Understanding large-scale dynamic purchase behavior. *Marketing Science*, 40(5):844–870, 2021.

- [JLL21] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.
- [JLLVR24] Hong Jun Jeon, Jason D Lee, Qi Lei, and Benjamin Van Roy. An information-theoretic analysis of in-context learning. *arXiv preprint arXiv:2401.15530*, 2024.
- [JLNZ23] Stefanus Jasin, Chengyi Lyu, Sajjad Najafi, and Huanan Zhang. Assortment optimization with multi-item basket purchase under multivariate mnl model. *Manufacturing & Service Operations Management*, 2023.
- [JSV20] Srikanth Jagabathula, Lakshminarayanan Subramanian, and Ashwin Venkataraman. A conditional gradient approach for nonparametric estimation of mixing distributions. *Management Science*, 66(8):3635–3656, 2020.
- [Kam03] Toshihiro Kamishima. Nantonac collaborative filtering: recommendation based on order responses. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 583–588, 2003.
- [KB06] Frank S Koppelman and Chandra Bhat. A self instructing course in mode choice modeling: multinomial and nested logit models. 2006.
- [KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [KMH⁺20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [KMN⁺16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

- [KZ14] N Bora Keskin and Assaf Zeevi. Dynamic pricing with an unknown demand model: Asymptotically optimal semi-myopic policies. *Operations research*, 62(5):1142–1167, 2014.
- [LBM23] Licong Lin, Yu Bai, and Song Mei. Transformers as decision makers: Provable in-context reinforcement learning via supervised pretraining. *arXiv preprint arXiv:2310.08566*, 2023.
- [LLK⁺19] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019.
- [LLW22] Hongyuan Lin, Xiaobo Li, and Lixia Wu. Multi-choice preferences learning and assortment recommendation in e-commerce. *Available at SSRN 4035033*, 2022.
- [LS20] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.
- [LWLQ22] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *AI Open*, 2022.
- [LWO⁺22] Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Hansen, Angelos Filos, Ethan Brooks, et al. In-context reinforcement learning with algorithm distillation. *arXiv preprint arXiv:2210.14215*, 2022.
- [LWW⁺24] Tao Li, Chenhao Wang, Yao Wang, Shaojie Tang, and Ningyuan Chen. Deep reinforcement learning for online assortment customization: A data-driven approach. *Available at SSRN 4870298*, 2024.
- [LWXX23] Shujie Luan, Ruxian Wang, Xiaolin Xu, and Weili Xue. Operations management under consumer choice models with multiple purchases. In *Operations Management under Consumer Choice Models with Multiple Purchases: Luan, Shujie— uWang, Ruxian— uXu, Xiaolin— uXue, Weili*. [SI]: SSRN, 2023.

- [LXP⁺24] Jonathan Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised pretraining can learn in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.
- [MAG99] Puneet Manchanda, Asim Ansari, and Sunil Gupta. The “shopping basket”: A model for multicategory purchase incidence decisions. *Marketing science*, 18(2):95–114, 1999.
- [Man77] Charles F Manski. The structure of random utility models. *Theory and decision*, 8(3):229, 1977.
- [Mau16] Andreas Maurer. A vector-contraction inequality for Rademacher complexities. In *International Conference on Algorithmic Learning Theory*, pages 3–17. Springer, 2016.
- [MCG24] Huiling Meng, Ningyuan Chen, and Xuefeng Gao. Reinforcement learning for intensity control: An application to choice-based network revenue management. *arXiv preprint arXiv:2406.05358*, 2024.
- [MDMBVZ14] Isabel Méndez-Díaz, Juan José Miranda-Bront, Gustavo Vulcano, and Paula Zabalá. A branch-and-cut algorithm for the latent-class logit assortment problem. *Discrete Applied Mathematics*, 164:246–263, 2014.
- [MDPZH20] Celestine Mendler-Dünner, Juan Perdomo, Tijana Zrnic, and Moritz Hardt. Stochastic optimization for performative prediction. *Advances in Neural Information Processing Systems*, 33:4929–4939, 2020.
- [MT00] Daniel McFadden and Kenneth Train. Mixed mnl models for discrete response. *Journal of applied Econometrics*, 15(5):447–470, 2000.
- [NDZ⁺19] Jeremy Nixon, Michael W Dusenberry, Linchuan Zhang, Ghassen Jerfel, and Dustin Tran. Measuring calibration in deep learning. In *CVPR Workshops*, volume 2, 2019.

- [NJUZ23] Sajjad Najafi, Stefanus Jasin, Joline Uichanco, and Jinglong Zhao. Assortment and price optimization under a multi-attribute (contextual) choice model. *Available at SSRN 4505644*, 2023.
- [NTS15] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pages 1376–1401. PMLR, 2015.
- [NZY21] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, 2021.
- [ONST22] Afshin Oroojlooyjadid, MohammadReza Nazari, Lawrence V Snyder, and Martin Takáč. A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing & Service Operations Management*, 24(1):285–304, 2022.
- [Ope23] OpenAI. Gpt-4 technical report, 2023.
- [PGHH22] Karlson Pfannschmidt, Pritha Gupta, Björn Haddendorst, and Eyke Hüllermeier. Learning context-dependent choice functions. *International Journal of Approximate Reasoning*, 140:116–155, 2022.
- [PRZ24] Zhenkang Peng, Ying Rong, and Tianning Zhu. Transformer-based choice model: A tool for assortment optimization evaluation. *Naval Research Logistics (NRL)*, 71(6):854–877, 2024.
- [PZMDH20] Juan Perdomo, Tijana Zrnic, Celestine Mendler-Dünner, and Moritz Hardt. Performative prediction. In *International Conference on Machine Learning*, pages 7599–7609. PMLR, 2020.
- [QB16] Sheng Qiang and Mohsen Bayati. Dynamic pricing with demand covariates. *arXiv preprint arXiv:1604.07463*, 2016.
- [QSQ⁺23] Meng Qi, Yuanyuan Shi, Yongzhi Qi, Chenxin Ma, Rong Yuan, Di Wu, and

- Zuo-Jun Shen. A practical end-to-end inventory management model with deep learning. *Management Science*, 69(2):759–773, 2023.
- [RAB20] Francisco J. R. Ruiz, Susan Athey, and David M. Blei. SHOPPER: A probabilistic model of consumer choice with substitutes and complements. *The Annals of Applied Statistics*, 14(1):1 – 27, 2020.
- [RNS⁺18] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.
- [ROS20] Nir Rosenfeld, Kojin Oshiba, and Yaron Singer. Predicting choice with set-dependent aggregation. In *International Conference on Machine Learning*, pages 8220–8229. PMLR, 2020.
- [RPG⁺21] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.
- [RSTT14] Paat Rusmevichientong, David Shmoys, Chaoxu Tong, and Huseyin Topaloglu. Assortment optimization under the multinomial logit model with random choice parameters. *Production and Operations Management*, 23(11):2023–2039, 2014.
- [RT12] Paat Rusmevichientong and Huseyin Topaloglu. Robust assortment optimization in revenue management under the multinomial logit choice model. *Operations research*, 60(4):865–882, 2012.
- [RVRK⁺18] Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.
- [SLA20] Brian Sifringer, Virginie Lurkin, and Alexandre Alahi. Enhancing discrete choice models with representation learning. *Transportation Research Part B: Methodological*, 140:236–261, 2020.

- [SPU19] Arjun Seshadri, Alex Peysakhovich, and Johan Ugander. Discovering context effects from raw choice data. In *International Conference on Machine Learning*, pages 5660–5669. PMLR, 2019.
- [SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
- [ŞT18] A Serdar Şimşek and Huseyin Topaloglu. An expectation-maximization algorithm to estimate the parameters of the markov chain choice model. *Operations Research*, 66(3):748–760, 2018.
- [TB21] Kiran Tomlinson and Austin R Benson. Learning interpretable feature context effects in discrete choice. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1582–1592, 2021.
- [TK85] Amos Tversky and Daniel Kahneman. The framing of decisions and the psychology of choice. In *Behavioral decision making*, pages 25–41. Springer, 1985.
- [TSKP23] Theja Tulabandhula, Deeksha Sinha, Saketh Reddy Karra, and Prasoon Patidar. Multi-purchase behavior: Modeling, estimation, and optimization. *Manufacturing & Service Operations Management*, 2023.
- [TVR04] Kalyan Talluri and Garrett Van Ryzin. Revenue management under a general discrete choice model of consumer behavior. *Management Science*, 50(1):15–33, 2004.
- [TVR06] Kalyan T Talluri and Garrett J Van Ryzin. *The theory and practice of revenue management*, volume 68. Springer Science & Business Media, 2006.
- [TVRVR04] Kalyan T Talluri, Garrett Van Ryzin, and Garrett Van Ryzin. *The theory and practice of revenue management*, volume 1. Springer, 2004.
- [Udw23] Rajan Udwani. Submodular order functions and assortment optimization. In *International Conference on Machine Learning*, pages 34584–34614. PMLR, 2023.

- [VCWV⁺22] Sander Van Cranenburgh, Shenhao Wang, Akshay Vij, Francisco Pereira, and Joan Walker. Choice modelling in the age of machine learning-discussion paper. *Journal of Choice Modelling*, 42:100340, 2022.
- [VdMH08] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.
- [Ver18] Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*, volume 47. Cambridge university press, 2018.
- [VHMN20] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. A review on the long short-term memory model. *Artificial Intelligence Review*, 53(8):5929–5955, 2020.
- [VSP⁺17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [Wai19] Martin J Wainwright. *High-dimensional statistics: A non-asymptotic viewpoint*, volume 48. Cambridge university press, 2019.
- [WCLT23] Hanzhao Wang, Zhongze Cai, Xiaocheng Li, and Kalyan Talluri. A neural network based choice model for assortment optimization. *arXiv preprint arXiv:2308.05617*, 2023.
- [WF21] Melvin Wong and Bilal Farooq. Reslogit: A residual neural network logit model for data-driven choice modelling. *Transportation Research Part C: Emerging Technologies*, 126:103050, 2021.
- [WFE⁺19] Edward Wagstaff, Fabian Fuchs, Martin Engelcke, Ingmar Posner, and Michael A Osborne. On the limitations of representing functions on sets. In *International Conference on Machine Learning*, pages 6487–6494. PMLR, 2019.
- [WLT23] Hanzhao Wang, Xiaocheng Li, and Kalyan Talluri. Transformer choice

- net: A transformer neural network for choice prediction. *arXiv preprint arXiv:2310.08716*, 2023.
- [WMZ20] Shenhao Wang, Baichuan Mo, and Jinhua Zhao. Deep neural networks for choice analysis: Architecture design with alternative-specific utility functions. *Transportation Research Part C: Emerging Technologies*, 112:234–251, 2020.
- [WTL21] Hanzhao Wang, Kalyan Talluri, and Xiaocheng Li. On dynamic pricing with covariates. *arXiv preprint arXiv:2112.13254*, 2021.
- [WWH24] Yueh-Hua Wu, Xiaolong Wang, and Masashi Hamaya. Elastic decision transformer. *Advances in Neural Information Processing Systems*, 36, 2024.
- [WZZ⁺13] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.
- [XRLM21] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.
- [YMCD⁺20] Reza Yousefi Maragheh, Xin Chen, James Davis, Jason Cho, Sushant Kumar, and Kannan Achan. Choice modeling and assortment optimization in the presence of context effects. *Available at SSRN 3747354*, 2020.
- [ZCS20] Huanan Zhang, Xiuli Chao, and Cong Shi. Closing the gap: A learning algorithm for lost-sales inventory systems with lead times. *Management Science*, 66(5):1962–1980, 2020.
- [ZKR⁺17] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.
- [ZR12] Morteza Zadimoghaddam and Aaron Roth. Efficiently learning from revealed

- preference. In *International Workshop on Internet and Network Economics*, pages 114–127. Springer, 2012.
- [ZZG22] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In *international conference on machine learning*, pages 27042–27059. PMLR, 2022.
- [ZZL⁺23] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.
- [ZZYW23] Yufeng Zhang, Fengzhuo Zhang, Zhuoran Yang, and Zhaoran Wang. What and how does in-context learning learn? bayesian model averaging, parameterization, and generalization. *arXiv preprint arXiv:2305.19420*, 2023.

Appendix A

Appendix for Chapter 2

A.1 Appendix for Numerical Experiments without Features

A.1.1 Model Details for Table 2.4

In this subsection, we describe the experiment setup of Table 2.4. For all neural networks trained in all experiments in the paper, we always set the batch size of training as 100 with learning rate as 0.0005 and training epoch as 100.

A.1.1.1 Data Generation.

The assortments (in samples) are generated by first deciding its size with a uniform distribution and then randomly select the items in the assortment. The parameters for different underlying choice models are generated in the following way:

- MNL: We sample the mean utility $u_i \in \mathbb{R}$ for each product $i = 1, \dots, n$ i.i.d. from the standard Normal distribution $\mathcal{N}(0, 1)$.

- MCCM: To distinguish the underlying true MCCM from an MNL model (which as we mentioned is a special case of MCCM), we consider the following generation mechanism:

We first set σ to control deviation ($\sigma = 2.5$ for dataset $n = 20$ and $\sigma = 4$ for dataset $n = 50$). We divide products into groups with a cluster number c_{num} ($c_{\text{num}} = 4$ for dataset $n = 20$ and $c_{\text{num}} = 10$ for dataset $n = 50$) to denote the number of groups. Then arriving probability λ is generated by:

$$\lambda_i = \frac{\exp(\mu_i)}{\sum_{j=1}^n \exp(\mu_j)},$$

where μ_i , $i = 1, \dots, n$ is identically and independently (i.i.d.) sampled from a Normal distribution $\mathcal{N}(0, \sigma^2)$. And the transition matrix ρ is generated row by row. For all the n rows, each row ρ_i is generated following:

$$\rho_{i,j} = \frac{\exp(\nu_{i,j})}{\sum_{j=1}^n \exp(\nu_{i,j})},$$

where $\nu_{i,j}$, $i, j = 1, \dots, n$ is independently sampled by Normal distribution $\mathcal{N}(\bar{\nu}_{i,j}, \sigma^2)$ and

$$\bar{\nu}_{i,j} = \begin{cases} 2\sigma & \forall (i, j) \in \{(i, j) | \exists k \in \{0, \dots, c_{\text{num}} - 1\}, kn/c_{\text{num}} < i, j \leq (k+1)n/c_{\text{num}}\}, \\ 0 & \text{otherwise.} \end{cases}$$

The intuition for the above is the following: We assume the set of all products can be divided evenly into several groups, and when a wanted product turns out to be unavailable, the customer is most likely to choose an alternative within the same group of this product.

- NP: The non-parametric (NP) choice model assumes that there is a fixed number of alternating schemes/types amongst customers. Each alternating scheme denotes a permutation of the total set of products, and following this scheme customers will check from the first product to the last until they find a product that is in the assortment. The formal definition is introduced in Section 2.3.1. Let n_{perm} denote the number of candidate permutations with positive probability, we set $n_{\text{perm}} = 10$ for dataset $n = 20$ and $n_{\text{perm}} = 20$ for dataset $n = 50$, and the parameters for the underlying model are the n_{perm} permutations of $\{1, \dots, n\}$ and their corresponding probabilities. These parameters

are randomly uniformly generated, and following the ground truth non-parametric model we generate synthetic samples.

- MMNL: We assume there are 5 customer types with equal sampled probability, i.e., $\alpha_c = 0.2$ for all $c = 1, \dots, 5$. For each type c , we set the mean utility $u_{c,i} \in \mathbb{R}$ for each product $i = 1, \dots, n$ as:

$$u_{c,i} \sim \begin{cases} \mathcal{N}(c + \frac{n}{5}, 1) & (c-1)n/5 + 1 \leq i \leq cn/5, \\ 0 & i = 0, \\ -50 & \text{otherwise.} \end{cases}$$

Thus, each customer type c will (almost) only consider a fixed set of products belonging to $(c-1)n/5 + 1 \leq i \leq cn/5$, and we set no-purchase's mean utility as 0 for all types.

A.1.2 Benchmark Algorithms

- MCCM-EM: We follow [ST18]'s EM algorithm to estimate the parameters of the MCCM. In summary, the algorithm starts with some random initial transition matrix ρ and arrival probabilities λ . For the expectation step, and based on the current transition matrix and arrival probabilities, the algorithm computes the expected probability of being interested in each product when a customer arrives and the number of times that a customer transitions from one product to another during the decision process for each sample. For the maximization step, it maximizes the likelihood to update the transition matrix and arrival probabilities by treating the previous two expected values as true values. The algorithm repeats the above two steps until it converges. In our experiments, the initial transition matrix and arrival probabilities are chosen by uniformly randomizing each entry and normalizing them to a stochastic matrix or distribution vector. We terminate the algorithm when the mean changes of updating the transition matrix and arrival probabilities are smaller than 0.01.
- MNL-MLE: We assume the underlying choice model is the MNL model and ignore all

observed features (if any). We estimate the unknown mean utilities of products by maximizing the likelihood (MLE).

A.2 Appendix for Numerical Experiments with Features

A.2.1 More Details for Table 2.5

A.2.1.1 Feature-based Choice Models.

Given an assortment $\mathcal{S} \subset \mathcal{N}$ and an input feature vector $\mathbf{z}_i \in \mathbb{R}^d$ for product $i \in \mathcal{N}$ (which can be product features along or the concatenation of customer features and product features):

- **Feature Based MNL Choice Model:** The mean utility $u_i(\mathbf{z}_i)$ for product $i \in \mathcal{N}$ is

$$u_i(\mathbf{z}_i) = \mathbf{z}_i^\top \boldsymbol{\beta},$$

where $\boldsymbol{\beta} \in \mathbb{R}^d$ is the coefficients of features.

- **Feature Based Markov Chain Choice Model:** This model is from [BB22]. For product $i \in \mathcal{N}$, the arriving probability λ_i

$$\lambda_i(\mathbf{z}_i) = \frac{\exp(\mathbf{z}_i^\top \boldsymbol{\beta})}{\sum_{j=1}^n \exp(\mathbf{z}_j^\top \boldsymbol{\beta})},$$

where $\boldsymbol{\beta} \in \mathbb{R}^d$. And the vector ρ_i in transition matrix

$$\rho_{i,j}(\mathbf{z}_i) = \frac{\exp(\mathbf{A}_j \mathbf{z}_i)}{\sum_{j=1}^n \exp(\mathbf{A}_j \mathbf{z}_i)},$$

where $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $\mathbf{A}_j$ is j -th row of $\mathbf{A}$. Then the choice probabilities can be computed as in MCCM based on λ 's and ρ 's.

A.2.1.2 Data Generation.

For the features, we generate the j -th feature $z_{i,j}$ of product i from i.i.d. standard Normal distribution $\mathcal{N}(0, 1)$. For parameters β in feature-based MNL and $(\mathbf{A}, \beta)$ in feature-based MCCM, we generate their entries also from i.i.d. standard Normal distribution $\mathcal{N}(0, 1)$. The assortments (in samples) are generated by first deciding their sizes with a uniform distribution and then randomly select the items in the assortment.

A.2.1.3 Architecture Tuning.

MCCM	Gated configuration	CE loss	Res configuration	CE loss
Benchmark	Oracle MNL-MLE(f)	1.932 2.212	Oracle MNL-MLE(f)	1.932 2.212
Assort-Net	[50,50]	2.010	[50,50]	1.988
	[50]	2.017	[50]	1.997
	[50,50,50]	2.015	[50,50,50]	2.000
	[100,50]	2.009	[100, 50]	2.008
Assort-Net(f)	[[1], [50,50]]	2.010	[[1], [100,50]]	1.984
	[[5,1], [50,50]]	2.010	[[5,1], [100,50]]	1.985
	[[5,5,1], [50,50]]	2.008	[[5,5,1], [100,50]]	1.985
	[[5,5,5,1], [50,50]]	2.007	[[5,5,5,1], [100,50]]	1.984
	[[10,1], [50,50]]	2.008	[[10,1], [100,50]]	1.984
	[[15,1], [50,50]]	2.009	[[15,1], [100,50]]	1.986
	[[20,1], [50,50]]	2.008	[[20,1], [100,50]]	1.983
	[[5,1], [50]]	2.017	[[5,1], [50]]	2.018
	[[5,1], [50,50,50]]	2.019	[[5,1], [100,100,50]]	1.987
	[[5,1], [100,50]]	2.010	[[5,1], [100,50]]	1.985

Table A.1: The data is generated by feature-based MCCM.

Table A.1 extends Table 2.5 with more neural network architectures. We find the model performance is quite stable with respect to architecture tuning.

We use $\text{Linear}(a, b)$ as an affine function with ReLu as the activation function, there the input dimension is a and the output dimension is b . For neural networks without features (under “Assort-Net”), the model configurations for Gated-Assort-Net, [50] denotes a one layer $\text{Linear}(50, 50)$, and [50, 50] denotes a two-layer network, $\text{Linear}(50, 50)$ followed by a $\text{Linear}(50, 50)$. [50, 50, 50] denotes a three-layer network, and [100, 50] denotes a two-layer, $\text{Linear}(50, 100)$ followed by a $\text{Linear}(100, 50)$. For the Res-Assort-Net part, [50] denotes one residual block, [50, 50]

denotes two residual blocks, $[50, 50, 50]$ three residual blocks, and $[100, 50]$ denotes a Res-Assort-Net with one residual block but contains a hidden layer of size 100 (so each block contains more than one layers, which is different and extended from Figure 2.1(b)).

For neural networks with features (under “Assort-Net(f)”), in the “Gated configuration” and “Res configuration” columns, the first list ($[1]$ or $[5, 1]$ or $[5, 5, 1]$ etc.) denotes the structure of the product encoder, and the second list ($[50]$ or $[50, 50]$ etc.) denotes the structure of the assortment network. The meanings of the configuration lists for the assortment network are generally the same as the feature-free cases, though some additional explanations need to be given for the Res-Assort-Net(f). According to the network structure, the assortment network takes as input the concatenated vector of latent utilities and assortments, leading to a vector of length 100. $[100, 50]$ indicates one residual block with the linear part $\text{Linear}(100, 100)$ and an ending layer $\text{Linear}(100, 50)$. $[50]$ indicates no residual block and just an ending layer $\text{Linear}(100, 50)$. $[100, 100, 50]$ indicates two residual blocks with linear part $\text{Linear}(100, 100)$ and an ending layer $\text{Linear}(100, 50)$. The first list gives the structure of the product encoder. Recall that the length of the product feature $d = 5$, and product features are encoded into latent utility values by several layers of the fully connected network. $[1]$ denotes a single layer $\text{Linear}(5, 1)$, $[5, 1]$ denotes a $\text{Linear}(5, 5)$ followed by a $\text{Linear}(5, 1)$. $[10, 1]$ denotes a $\text{Linear}(5, 10)$ followed by a $\text{Linear}(10, 1)$. The rest follow similar rules.

A.2.2 More Details on Real Data Experiments

A.2.2.1 Data Description.

- **SwissMetro.** The SwissMetro Data Set is a public survey data set collected to analyze preference between the Metro system against the other transport modes car and train. The description of the dataset is given in [Bie18]. For the convenience of network training, we clean the data by removing samples whose features are taking abnormal values. To be specific, we only keep samples whose “CHOICE” (final choice) is not 0 (which indicates unknown), “WHO” (who pays the ticket) value is not 0 (which indicates un-

known), “AGE” is not 6 (which indicates unknown), “INCOME” is not 4 (which indicates unknown), and “PURPOSE” is in $[1, 2, 3, 4]$ (1: Commuter, 2: Shopping, 3: Business, 4: Leisure). After cleaning, there are in total 9,135 samples and the number of alternatives in the assortment varies from 2 to 3 as only when owning the car the customer can have that choice. The sample sizes in the train, validate and test process are 7,000, 1,000 and 1,000 correspondingly. There are 8 customer features and 3 product features, shown in Table A.2. For product features. Note that “CAR” has no “HE” (Headaway, which means interval time between two consecutive train/metro/car arrivals) feature in the original dataset, so we set it to 0. This is reasonable since for private cars the waiting time is always 0. In this dataset, the product features are not static, as they vary among different types of customers.

- **Expedia Search.** The Expedia Data Set is a public data set from a Kaggle competition “Personalized Expedia Hotel Searches - ICDM2013”. It contains the ordered list of hotels according to the user’s search. We clean the data by dropping the searches with outlier prices ($> \$1,000$) and days of booking in advance (the gap between booking date and check-in date > 365) and also the features with missing values. After cleaning, there are in total 386,557 searches where the size of the offered hotel/searched results list (i.e., the number of alternatives in one assortment) is between 5 to 38, and 136,886 unique hotels. The sample sizes in the train, validate and test process are 200,000, 20,000, and 20,000 correspondingly. There are 6 customer features and 7 product features as shown in Table A.2.

For assortment vector $\mathbf{S}$, we encode the ranked positions in assortments (1 – 39 with 39 as a no-purchase option) and if some position has no shown product, we assign their product features as 0. Thus the assortment vector captures the ranking and assortment structure within the assortment. Notice that we only use the positions of products in offered assortments instead of the products themselves. There are several reasons: The number of unique products is too large (136,886 unique hotels in 386,557 offered assortments after cleaning) to be covered as assortment vector; Also in practice, hotels with different locations would never be included in the same assortment and the assortment effect

from such hotels should be ignored; Further, the neural choice model can utilize the product features (with customer features) to identify hotels' utilities and also explore the assortment effect from the different ranked positions thanks to the architecture.

Data	Customer Feature	Product Feature
SwissMetro	Gender Age Income First class or not Who pays (self, employer...) Purpose (commuter, shopping...) # luggages Owning annual ticket or not	Travel time Headaway Cost
Expedia	# nights stay # days booking in advance # adults # children # rooms Saturday included or not	Hotel star rating Hotel chain or not Location score (by Expedia) Historical prices Price Promotion or not Randomly ranked or not (by Expedia)

Table A.2: Features in SwissMetro and Expedia.

A.2.2.2 Tuning Architectures for Neural Networks and Performance.

The performances of different models on the SwissMetro and Expedia datasets are summarized in Table A.3, which is extended from Table 2.8. The configurations of both our assort-nets and the benchmark nets are given in the form of channel arrays. For our assort-nets, the first array denotes the structure of the customer encoder, the second array denotes the product encoder, and the third denotes the assort-net part of the whole network. For instance, for the gated-assort-net in the configuration list $[[30, 10], [10, 10], [30, 30, 3/39]]$, the first array $[30, 10]$ denotes the structure of customer encoder, having one hidden layer of size 30 and a final layer of size 10. The second array $[10, 10]$ denotes the structure of the product encoder, having one hidden layer of size 10 and a final layer of size 10. The size of the final layer of the customer encoder equals that of the product encoder. The third array $[30, 30, 3/39]$ denotes that the assort-net part has two hidden layers of size 30. The size of the final layer is dependent on the number of total products n , which is 3 for SwissMetro and 39 for Expedia. The Res-Assort-Net follows the same way to denote the configurations of the customer encoder and product

encoder. To denote the configuration of the Res-Assort-Net part, take the last configuration $[6/78, 60, 6/78] * 2$ for example, it denotes that there are 2 residual blocks, each containing a hidden layer of size 60 and the input and output of one block both have dimensions 6/78. Note there is a final (fully-connected) layer with input dimension 6/78 and output dimension 3/39

We defer the description of benchmark algorithms in Appendix A.2.3. The configurations $([100, 9]$ or $[100, 30])$ denote the corresponding network structure, having a hidden layer of size 100 and a final layer of size 9 or 30.

For Random Forest, the hyper-parameters including a number of trees and maximum depth are chosen by validation dataset performance. Specifically, the number of trees is tested among $\{10, 20, 50, 100, 200, 400\}$, and the maximum depth is tested among $\{4, 6, 8, 12, 16, 18, 20\}$. Other parameters are set as default values in the Python package scikit-learn's function `ensemble.RandomForestClassifier`.

We restrict all three parts of our network (product encoder, customer encoder, and assort-net) to a depth of no more than 2. The widths are tuned a little bit loosely (due to their relatively lesser effect on the model performance), but no more than 200. The architecture of TasteNet follows the original paper [HZPBA20], in which they used 110 hidden units. The architecture of DeepMNL is set to keep some degree of similarity with the architecture of TasteNet.

A.2.3 Benchmark Algorithms

- MNL(f)-MLE: We assume the underlying choice model is the feature-based MNL choice model introduced in Appendix A.2, where the i 's feature vector $\mathbf{z}_i = (\mathbf{f}^C, \mathbf{f}_i^P)$ is a concatenation of customer feature $\mathbf{f}^C$ (which may vary in different samples) and product feature $\mathbf{f}_i^P$. We estimate the unknown parameters by MLE.
- TasteNet: We follow the architecture introduced by [HZPBA20]. We train a neural network $g_\theta(\cdot)$ to encode the customer feature vector $\mathbf{f}^C \in \mathbb{R}^{d'}$ into a vector $(\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_n) := g_\theta(\mathbf{f}^C) \in \mathbb{R}^{nd}$, where $\boldsymbol{\alpha}_i \in \mathbb{R}^d$ is interpreted as the coefficient vector of product features

Model	Config	SwissMetr	Expedia
Gated-Assort-Net(f)	$[[10],[10],[3/39]]$	0.696	2.479
	$[[100,10],[10],[3/39]]$	0.638	2.468
	$[[10],[10,10],[3/39]]$	0.673	2.458
	$[[10],[10],[10,3/39]]$	0.708	2.463
	$[[100,10],[10,10],[10,3/39]]$	0.598	2.458
	$[[30,10],[10,10],[30,30,3/39]]$	0.623	2.403
Res-Assort-Net(f)	$[[100, 10], [10, 10], [6/78, 6/78]*1]$	0.607	2.468
	$[[100, 10], [10, 10], [6/78, 60, 6/78]*1]$	0.608	2.355
	$[[200, 20], [20, 20], [6/78, 60, 6/78]*2]$	0.610	2.325
MNL-MLE	–	0.883	2.827
MNL(f)-MLE	–	0.810	2.407
TasteNet	$[100, 9] + \text{MNL}$	0.691	–
	$[100, 9] + \text{MNL(f)}$	0.681	2.436
DeepMNL	$[100, 30] + \text{MNL}$	0.751	–
	$[100, 30] + \text{MNL(f)}$	0.741	2.374
Random Forest	–	0.633	2.458

Table A.3: Performance on the SwissMetro and Expedia (extended from Table 2.8).

$\mathbf{f}_i^P$. For product $i \in \{1, \dots, n\}$, its mean utility is encoded by:

$$u_i := \boldsymbol{\alpha}_i^\top \mathbf{f}_i^P.$$

This utility vector $\mathbf{u} \in \mathbb{R}^n$ is then passed to a softmax layer (i.e., an MNL operator), giving the predicted choice probability.

- DeepMNL: We modify the architecture introduced by [SLA20]. We train a neural network $g_\theta(\cdot)$ to encode the concatenated feature vector $(\mathbf{f}^C, \mathbf{f}_i^P) \in \mathbb{R}^{d+d'}$ of customer features $\mathbf{f}^C \in \mathbb{R}^{d'}$ and product features $\mathbf{f}_i^P \in \mathbb{R}^d$ into scalar $g_\theta((\mathbf{f}^C, \mathbf{f}_i^P)) \in \mathbb{R}$ as the mean utility of product i in MNL model. This utility vector is then passed to a softmax layer, giving the predicted choice probability.
- Random Forest: We follow the work [CGT21] to apply the random forest method to predict customer choice. Specifically, for each sample assortment, we concatenate the assortment vector $\mathbf{S}$, customer feature and all product features as its input (thus same as Gated-Assort-Net(f) and Res-Assort-Net(f)). For the products not offered in the assortment, we assign their product feature values as 0.

A.2.3.1 Dealing with Missing Products for TasteNet and DeepMNL.

Due to the fixed size of input vectors when training the neural networks, the DeepMNL and TasteNet cannot be applied directly when the size of the assortment changes across different samples. For the SwissMetro dataset, TasteNet and DeepMNL are only trained based on the samples with full assortments where all products are offered. The number of such samples in the dataset is 7,839. We predict the full assortments choice probabilities by using trained TasteNet or DeepMNL, and use trained MNL-MLE or MNL(f)-MLE to predict the other assortments (with missing product) where the sample size is 1,296 in the dataset. We denote the corresponding prediction models as TasteNet(DeepMNL)+MNL and TasteNet(DeepMNL)+MNL(f); For Expedia dataset, since there are only 9 out of 386,557 samples are full assortments, we cannot use the same method as in SwissMetro due to the lacking data (otherwise we are almost reproducing the MNL-MLE or MNL(f)-MLE). Thus, we just assign 0 to the missing entries of product feature vectors, which is the same as Gated-Assort-Net(f) and Res-Assort-Net(f). Since these methods are recovering the missing feature vectors by 0 and the last layer is softmax, we also name them TasteNet(DeepMNL)+MNL(f).

A.3 Proof for Subsection 2.4.2

The idea of obtaining a generalization bound for neural networks via Rademacher complexity is not new [WZZ⁺13, NTS15, GRS18]. We customize the previous analysis to the case of GAsN—the key is the peeling argument (see [NTS15] for an example). We first introduce the definition of *empirical Rademacher complexity* which captures the complexity of a function class as a machine learning model. It is called as “empirical” because the complexity measure is contingent on the underlying dataset, and thus is a random variable itself.

Definition A.1 (Empirical Rademacher Complexity). *The empirical Rademacher complexity of function class $\mathcal{G}$ with respect to a dataset $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ is defined as*

$$\hat{\mathfrak{R}}_{\mathcal{D}}(\mathcal{G}) := \frac{1}{m} \mathbb{E} \left[\sup_{g \in \mathcal{G}} \sum_{k=1}^m \epsilon_k g(\mathbf{x}_k) \right]$$

where the expectation is taken with respect to ϵ_k 's – a sequence of independent Rademacher random variables with $\mathbb{P}(\epsilon_k = 1) = \mathbb{P}(\epsilon_k = -1) = \frac{1}{2}$.

The derivation of the generalization bound of the neural network reduces to the calculation of the corresponding Rademacher complexity.

Lemma 1. *Suppose $|r((i, \mathbf{S}); \boldsymbol{\theta})| \leq C$ holds for all $(i, \mathbf{S}) \in \mathcal{D}$ and $\boldsymbol{\theta} \in \Theta$ for some $C > 0$, and $(i_k, \mathbf{S}_k) \in \mathcal{D}$ is i.i.d. sampled from distribution $\mathcal{P}$. Then the following inequality holds with probability $1 - \delta$,*

$$R(\hat{\boldsymbol{\theta}}) \leq R(\boldsymbol{\theta}^*) + 2\hat{\mathfrak{R}}_{\mathcal{D}}(\{r(\cdot; \boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\}) + 5C\sqrt{\frac{2\log(8/\delta)}{m}}.$$

Proof. See Theorem 26.5 of [SSBD14]. □

The following lemma presents a basic property of the Rademacher complexity (with respect to the composition of a Lipschitz function).

Lemma 2. *Let $\mathcal{X}$ be an arbitrary set and $\mathcal{G}$ be a class of functions $\mathbf{g} = (g_1, \dots, g_n) : \mathcal{X} \rightarrow \mathbb{R}^n$. Suppose $h_k : \mathbb{R}^n \rightarrow \mathbb{R}$ has Lipschitz constant κ with respect to Euclidean norm for $k = 1, \dots, m$. Then, the following inequality holds for any fixed $(\mathbf{x}_1, \dots, \mathbf{x}_m) \in \mathcal{X}^m$,*

$$\mathbb{E} \left[\sup_{\mathbf{g} \in \mathcal{G}} \sum_{k=1}^m \epsilon_k h_k(\mathbf{g}(\mathbf{x}_k)) \right] \leq \sqrt{2}\kappa \mathbb{E} \left[\sup_{\mathbf{g} \in \mathcal{G}} \sum_{k=1}^m \sum_{i=1}^n \epsilon_{k,i} g_i(\mathbf{x}_k) \right],$$

where $\{\epsilon_k\}_{k=1, \dots, m}$ and $\{\epsilon_{k,i}\}_{k=1, \dots, m}$ for $i = 1, \dots, n$ are independent Rademacher sequences. Here the expectations on both sides are taken with respect to these Rademacher random variables.

Proof. See Corollary 4 of [Mau16]. □

For convenience, we rewrite the risk function $r((i, \mathbf{S}); \boldsymbol{\theta})$ as a function of L -th layer's output $\mathbf{z}_L$:

$$\phi((i, \mathbf{S}), \mathbf{z}_L) := -\log \frac{\exp(z_{L,i})}{\sum_{i' \in \mathcal{S}} \exp(z_{L,i'})} = r((i, \mathbf{S}); \boldsymbol{\theta})$$

Given a data sample $(i, \mathbf{S})$, the following lemma says that $\phi((i, \mathbf{S}), \mathbf{z}_L)$ is a Lipschitz function of $\mathbf{z}_L$.

Lemma 3. $\phi((i, \mathbf{S}), \mathbf{z}_L)$ has Lipschitz constant $\sqrt{2}$ with respect to $\mathbf{z}_L$ under the Euclidean norm.

Proof. For simplicity, we drop the subscript L . Then the j -th component of $\phi((i, \mathbf{S}), \mathbf{z})$'s gradient with respect to $\mathbf{z}$ is:

$$(\nabla \phi((i, \mathbf{S}), \mathbf{z}))_j = \begin{cases} \frac{\exp(z_j)}{\sum_{j' \in \mathcal{S}} \exp(z_{j'})} - 1, & j = i, \\ \frac{\exp(z_j)}{\sum_{j' \in \mathcal{S}} \exp(z_{j'})}, & j \in \mathcal{S}/i, \\ 0, & j \notin \mathcal{S}. \end{cases}$$

Thus

$$\begin{aligned} & \|\nabla \phi((i, \mathbf{S}), \mathbf{z})\|_2^2 \\ &= \left(\frac{\exp(z_i)}{\sum_{i' \in \mathcal{S}} \exp(z_{i'})} - 1 \right)^2 + \sum_{j \in \mathcal{S}/i} \left(\frac{\exp(z_j)}{\sum_{i' \in \mathcal{S}} \exp(z_{i'})} \right)^2 \\ &\leq \frac{\sum_{i' \in \mathcal{S}} \exp(z_{i'})^2}{\left(\sum_{i' \in \mathcal{S}} \exp(z_{i'}) \right)^2} + 1 \leq 2. \end{aligned}$$

By the mean value theorem, the proof is completed. $\square$

Note that for $l = 1, \dots, L$, the l -th layer's output $\mathbf{z}_l$ is a function of $\boldsymbol{\theta}$ and $\mathbf{S}$. We can define function classes as follows

$$\mathcal{G}_{l,i} := \{\mathbf{S} \in \{0, 1\}^n \rightarrow z_{l,i}(\mathbf{S}; \boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\}$$

for $l = 1, \dots, L$ and $i = 1, \dots, n_l$.

The following corollary relates the Rademacher complexities after and before the softmax operator in the last layer.

Corollary 4. *The following inequality holds,*

$$\widehat{\mathfrak{R}}_{\mathcal{D}}(\{r(\cdot; \boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\}) \leq 2 \sum_{i=1}^n \widehat{\mathfrak{R}}_{\mathcal{D}}(\mathcal{G}_{L,i}).$$

Proof. Denote

$$\mathcal{G}_L := \{\mathbf{S} \in \{0, 1\}^n \rightarrow \mathbf{z}_L(\mathbf{S}; \boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\}.$$

Let $\mathcal{G} = \mathcal{G}_L$, $\mathbf{g}(\cdot) = \mathbf{z}_L(\cdot; \boldsymbol{\theta})$ and $h_k(\cdot) = \phi((i_k, \mathbf{S}_k), \cdot)$ in Lemma 2. Together with Lemma 3, we have the following

$$\widehat{\mathfrak{R}}_{\mathcal{D}}(\{r(\cdot; \boldsymbol{\theta}) : \boldsymbol{\theta} \in \Theta\}) \leq \frac{2}{m} \mathbb{E} \left[\sup_{\boldsymbol{\theta} \in \Theta} \sum_{i=1}^n \sum_{k=1}^m \epsilon_{k,i} z_{L,i}(\mathbf{S}_k; \boldsymbol{\theta}) \right],$$

where expectations are with respect to $\boldsymbol{\epsilon}$. Then for any sequence $\{\epsilon_{k,i}\}$,

$$\sup_{\boldsymbol{\theta} \in \Theta} \sum_{i=1}^n \sum_{k=1}^m \epsilon_{k,i} z_{L,i}(\mathbf{S}_k; \boldsymbol{\theta}) \leq \sum_{i=1}^n \sup_{\boldsymbol{\theta} \in \Theta} \sum_{k=1}^m \epsilon_{k,i} z_{L,i}(\mathbf{S}_k; \boldsymbol{\theta}),$$

where $\epsilon_{k,i}$ are i.i.d. Rademacher random variables. Finally by definition of $\widehat{\mathfrak{R}}_{\mathcal{D}}(\mathcal{G}_{L,i})$ we complete the proof. $\square$

Then we can apply the standard analysis of fully connected networks [WZZ⁺13, NTS15, GRS18]:

Lemma 4. *For $i = 1, \dots, n$,*

$$\widehat{\mathfrak{R}}_{\mathcal{D}}(\mathcal{G}_{L,i}) \leq \frac{1}{\sqrt{m}} \left(\bar{b} \cdot \frac{(2\bar{W})^L - 1}{2\bar{W} - 1} + (2\bar{W})^L \sqrt{2 \log(2n)} \right).$$

Proof. All expectations below are with respect to ϵ . We begin with the last layer:

$$\begin{aligned}
\widehat{\mathfrak{R}}_{\mathcal{D}}(\mathcal{G}_{L,i}) &= \frac{1}{m} \mathbb{E} \left[\sup_{\theta \in \Theta} \sum_{k=1}^m \epsilon_k z_{L,i}(\mathbf{S}_k; \theta) \right] \\
&= \frac{1}{m} \mathbb{E} \left[\sup_{\theta \in \Theta} \sum_{k=1}^m \epsilon_k (\mathbf{w}_{L,i}^\top \mathbf{z}_{L-1}(\mathbf{S}_k; \theta) + b_{L,i})^+ \right] \\
&\leq \frac{1}{m} \mathbb{E} \left[\sup_{\theta \in \Theta} \sum_{k=1}^m \epsilon_k (\mathbf{w}_{L,i}^\top \mathbf{z}_{L-1}(\mathbf{S}_k; \theta) + b_{L,i}) \right] \\
&= \frac{1}{m} \mathbb{E} \left[\sup_{\theta \in \Theta} \sum_{k=1}^m \epsilon_k \mathbf{w}_{L,i}^\top \mathbf{z}_{L-1}(\mathbf{S}_k; \theta) \right] + \frac{1}{m} \mathbb{E} \left[\sup_{|b_{L,i}| \leq \bar{b}} \sum_{k=1}^m \epsilon_k b_{L,i} \right]. \tag{A.1}
\end{aligned}$$

Here the first inequality comes from the contraction property of Rademacher complexity and the fact that ReLU is 1-Lipschitz (see Lemma 26.9 of [SSBD14]).

For the first term of (A.1),

$$\begin{aligned}
\sup_{\theta \in \Theta} \sum_{k=1}^m \epsilon_k \mathbf{w}_{L,i}^\top \mathbf{z}_{L-1}(\mathbf{S}_k; \theta) &\leq \sup_{\theta \in \Theta} \|\mathbf{w}_{L,i}\|_1 \left\| \sum_{k=1}^m \epsilon_k \mathbf{z}_{L-1}(\mathbf{S}_k; \theta) \right\|_\infty \\
&\leq \bar{W} \sup_{\theta \in \Theta} \left\| \sum_{k=1}^m \epsilon_k \mathbf{z}_{L-1}(\mathbf{S}_k; \theta) \right\|_\infty \\
&= \bar{W} \sup_{i=1, \dots, n_{L-1}, \theta \in \Theta} \left| \sum_{k=1}^m \epsilon_k z_{L-1,i}(\mathbf{S}_k; \theta) \right| \\
&\leq \bar{W} \sup_{\theta \in \Theta} \left| \sum_{k=1}^m \epsilon_k z_{L-1,1}(\mathbf{S}_k; \theta) \right| \\
&\leq \bar{W} \sup_{\theta \in \Theta} \left[\sum_{k=1}^m \epsilon_k z_{L-1,1}(\mathbf{S}_k; \theta) \right] + \bar{W} \sup_{\theta \in \Theta} \left[- \sum_{k=1}^m \epsilon_k z_{L-1,1}(\mathbf{S}_k; \theta) \right],
\end{aligned}$$

where the first inequality follows the Cauchy–Schwarz inequality, the second inequality comes from $\|\mathbf{w}_{L,i}\|_1 \leq \bar{W}$, the third inequality comes from $\mathcal{G}_{L-1,1} = \mathcal{G}_{L-1,2} = \dots = \mathcal{G}_{L-1,n_{L-1}}$ and the last inequality is because $\mathcal{G}_{L-1,1}$ includes the zero function.

Then we note that ϵ and $-\epsilon$ have same distribution,

$$\begin{aligned} \mathbb{E} \left[\sup_{\theta \in \Theta} \sum_{k=1}^m \epsilon_k \mathbf{w}_{L,i}^\top \mathbf{z}_{L-1}(\mathbf{S}_k; \theta) \right] &\leq \bar{W} \mathbb{E} \left[\sup_{\theta \in \Theta} \sum_{k=1}^m \epsilon_k \mathbf{z}_{L-1,1}(\mathbf{S}_k; \theta) \right] + \bar{W} \mathbb{E} \left[\sup_{\theta \in \Theta} - \sum_{k=1}^m \epsilon_k \mathbf{z}_{L-1,1}(\mathbf{S}_k; \theta) \right] \\ &= 2\bar{W} \mathbb{E} \left[\sup_{\theta \in \Theta} \sum_{k=1}^m \epsilon_k \mathbf{z}_{L-1,1}(\mathbf{S}_k; \theta) \right] \\ &= 2\bar{W} m \hat{\mathfrak{R}}_{\mathcal{D}}(\mathcal{G}_{L-1,1}) \end{aligned}$$

For the second term of (A.1), we have

$$\begin{aligned} \mathbb{E} \left[\sup_{|b_{L,i}| \leq \bar{b}} \sum_{k=1}^m \epsilon_k b_{L,i} \right] &\leq \mathbb{E} \left[\sup_{|b_{L,i}| \leq \bar{b}} \left| \sum_{k=1}^m \epsilon_k \right| |b_{L,i}| \right] \\ &\leq \bar{b} \mathbb{E} \left[\left| \sum_{k=1}^m \epsilon_k \right| \right] = \bar{b} \mathbb{E} \left[\sqrt{\left(\sum_{k=1}^m \epsilon_k \right)^2} \right] \\ &\leq \bar{b} \sqrt{\mathbb{E} \left[\left(\sum_{k=1}^m \epsilon_k \right)^2 \right]} \leq \bar{b} \sqrt{m}, \end{aligned}$$

where the second inequality is from Jensen's inequality and the last inequality is the definition of Rademacher random variables.

Combining these two parts,, we have $\hat{\mathfrak{R}}_{\mathcal{D}}(\mathcal{G}_{L,i}) \leq 2\bar{W} \hat{\mathfrak{R}}_{\mathcal{D}}(\mathcal{G}_{L-1,1}) + \frac{\bar{b}}{\sqrt{m}}$ for $i = 1, \dots, n$. We can repeat this process $L - 1$ times. From Lemma 26.11 in [SSBD14] and by noting $\|\mathbf{S}\|_\infty \leq 1$,

$$\hat{\mathfrak{R}}_{\mathcal{D}}(\mathcal{G}_{1,1}) \leq 2\bar{W} \sqrt{\frac{2 \log(2n)}{m}} + \frac{\bar{b}}{\sqrt{m}}.$$

Thus we complete the proof. □

Proof for Theorem 2.1

Proof. Combining Lemma 1, Corollary 4, and Lemma 4, the proof is completed. □

A.4 Appendix for Assortment Optimization

For each underlying choice model with a fixed number of available products ($|\mathcal{N}| = 20, 40, 60$), we randomly generate 5 datasets/instances (with different parameters in the generating choice model) and for each dataset, we randomly generate 20 revenue vectors and constraints (if any). Thus, each underlying choice model has 100 randomly sampled optimization problems. Each method has two steps: the estimation step and the optimization step. The estimation step estimates the parameters of the choice model based on the training data and the optimization step treats such estimated choice model as the true one and (approximately) optimize it. The precise generation methods, estimation methods, and assortment optimization formulations with corresponding used parameters are introduced below. For the optimization step, we use Gurobi version 9.5.2 for solving all MIPs and LPs. Optimizing the assortment with trained Gated-Assort-Net can be slow since the relaxed problem is not convex. As discussed in Section 2.5, we tighten the upper bounds of decision variables (the big-M) following the methods from [FJ17], use exponential function's second order approximation $1 + x + x^2/2$, and set the maximal solving time limit as 300 seconds.

A.4.1 Data Generation

Each instance has a training dataset with 30,000 samples for estimation. The offered assortment $\mathcal{S}_k$ is generated by first randomly choosing assortment size $|\mathcal{S}_k|$ in $\{1, \dots, n\}$, and then randomly picking products from n and form the assortment $\mathcal{S}_k$. The choice i_k is generated by corresponding choice models including MNL, MCCM, NP, and MMNL, with the same generating process as in Appendix A.1 for their parameters.

For the revenue vector, the i -th product's revenue is generated uniformly in the interval $[10, 50]$ and we always set the revenue of the no-purchase option as 0.

For the capacity constraint (if applicable), the constraint is defined by $\mathbf{a}^T \mathbf{z}_0 \leq c$, where $\mathbf{a} \in \mathbb{R}^n$ has non-negative elements and c is a positive scalar. We call the i -th element a_i as the coefficient of i -th item and c as the budget. The i -th product's coefficient is generated uniformly in

[10, 50] and we always set the coefficient of the no-purchase option as 0. The budget c is generated uniformly among $\max\{\frac{\|\mathbf{a}\|_1}{n}, \|\mathbf{a}\|_\infty\}$ and $\max\{\frac{4\|\mathbf{a}\|_1}{n}, \|\mathbf{a}\|_\infty\}$, where $\|\mathbf{a}\|_1 = \sum_{i=1}^n a_i$ and $\|\mathbf{a}\|_\infty = \max_{i=1,\dots,n} a_i$. Note such a budget can make every product available to be offered in the assortment.

A.4.2 Estimation Step

The estimation step includes Neural Network (NN), MNL-MLE (MNL), and MCCM-EM (MC). For the neural network estimation method, we test one layer (NN(1)) and two layers (NN(2)) Gated-Assort-Nets with width in each layer equal to the number of products. For MNL-MLE (MNL), and MCCM-EM (MC), they apply the same process introduced in Appendix A.1.

A.4.3 Optimization Formulation

Below are the methods implemented to solve the (true) assortment optimization problems of each underlying choice model:

- MNL and MMNL: When there is no constraint, we use the revenue ordering method [TVR04] to get the optimal assortment for MNL and use the MILP formulation in [MDMBVZ14] for solving the assortment optimization problem for MMNL. We remark that MNL is a special case of MMNL and we can easily add the corresponding capacity constraint in the above MILP for both models.
- MCCM: When there is no constraint, the optimal assortment of MCCM can be computed by repeatedly using Bellman operators [BGG16] or by solving an LP [FT17]. The latter LP method can be extended to an MILP when there exists a cardinality constraint [DGSY20].
- NP: When there is no constraint, the assortment optimization problem can be formulated as a MILP:

Lemma 5. For an NP choice model with revenue vector $\boldsymbol{\mu}$, distribution vector $\boldsymbol{\lambda}$ on permutations $\text{Perm}_{\mathcal{N}}$, and permutation matrix $\mathbf{P}_j \in \{0, 1\}^{n \times n}$ for $j = 1, \dots, |\text{Perm}_{\mathcal{N}}|$, whose element $P_{i,i'} = 1$ if i' -th item is ranked in position i and otherwise 0 in j -th permutation, the assortment optimization problem can be formulated by following MILP with optimal assortment as solved $\mathbf{s}$:

$$\begin{aligned} \max_{\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\eta}} \quad & \sum_{j=1}^{|\text{Perm}_{\mathcal{N}}|} \lambda_j \boldsymbol{\eta}_j^\top \mathbf{P}_j \boldsymbol{\mu} \\ \text{s.t.} \quad & \mathbf{P}_j \mathbf{s} = \tilde{\mathbf{s}}_j, \quad \forall j = 1, \dots, |\text{Perm}_{\mathcal{N}}| \\ & \boldsymbol{\eta}_j \leq \tilde{\mathbf{s}}_j, \quad \forall j = 1, \dots, |\text{Perm}_{\mathcal{N}}| \end{aligned} \tag{A.2}$$

$$\eta_{j,i} \leq 1 - \tilde{s}_{j,i'}, \quad \forall j = 1, \dots, |\text{Perm}_{\mathcal{N}}|, \quad i = 1, \dots, n, \quad i' = 1, \dots, i-1 \tag{A.3}$$

$$\mathbf{s} \in \{0, 1\}^n, \quad \boldsymbol{\eta}_j = (\eta_{j,1}, \dots, \eta_{j,n}) \geq 0, \quad \forall j = 1, \dots, |\text{Perm}_{\mathcal{N}}|$$

Proof. Given an assortment vector $\mathbf{s} \in \{0, 1\}^n$ whose element equals 1 means the corresponding product is included in the assortment, equality constraint sets $\tilde{\mathbf{s}}_j$ to recover the available positions in permutation j : if i -th position's item is offered, then $\tilde{s}_{j,i} = 1$ otherwise 0. Constraints (A.2) and (A.3) will together push $\boldsymbol{\eta}_j$'s elements into 0 if the corresponding position is not available (by (A.2)) or if any of former positions is available (by (A.3)). Thus, since the objective function is maximizing a linear function of $\boldsymbol{\eta}_j$ with positive coefficients, $\boldsymbol{\eta}_j$ will be a $\{0, 1\}^n$ vector with only one element 1 whose position is the chosen position (the first available position) under assortment vector $\mathbf{s}$ under j -th permutation. By noting $\mathbf{P}_j \boldsymbol{\mu}$ will return the revenues of each position in j -th permutation, the objective function is the expected revenue given assortment $\mathbf{s}$. $\square$

Note that with additional capacity constraints, we can directly add it into the above formulation to get the optimal assortment.

A.5 Appendix for Section 2.6

A.5.1 Appendix for Subsection 2.6.1

A.5.1.1 Details for Figure 2.4.

Both the sample generation of the underlying MCCM and EM algorithm implementation are introduced in Appendix A.1. We compute the mean of absolute differences between estimations and true values as estimation errors, shown on the y-axis. We independently train 10 times based on different training data: the training data size is always set as 10,000 with total product number $n = 20$, but the assortment size (used in training) is varied and chosen from 2, 4, 6, 8, 10 as shown on the x-axis. The products in each sampled assortment are randomly chosen. The reported value in the figure is averaged over 10 independent trails.

A.5.1.2 Out-of-domain Performance under Assortment Distribution Shift.

In this experiment, we demonstrate whether a different distribution of generating assortments between training and testing will affect model performance, also known as the out-of-domain generalization. Specifically, we use one distribution to generate the assortment in the training data and test the performance with assortments generated from another distribution. Intuitively, this shows whether the model is capable of generalizing over the domain of assortments.

In the experiment, we generate four groups of training data:

- For the first dataset D-1, assortments are generated by first randomly choosing $s = |\mathcal{S}|$, and then randomly picking s products from $\mathcal{N}$ to form the assortment S .
- For the second dataset D-2, assortments are generated from a Bernoulli distribution. That is, for each product, with probability $1/2$ it appears in the assortment independent of others.

- For the third dataset D-3, the assortment is given by first randomly specifying either the first half (product 1 to product $n/2$, suppose $n/2$ is integer for convenience) or the second half (product $n/2 + 1$ to product n) of the total products will definitely not appear in the assortment. Then we generate the assortment within the other half following the same rule as in D-1.
- For the last dataset D-4, we require all the assortments to have a size of $n/3$ or $n/3 + 1$ (again assume they are integers for convenience) products, and then pick products randomly into the assortment. This design is inspired by the results of [GH20].

The total number of products $n = 30$ and each training dataset consists of $m = 100,000$ samples. The underlying true model for generating the data is the MCCM model.

	D-1	D-2	D-3	D-4
D-1	2.364	2.603	1.820	2.298
D-2	2.366	2.600	1.819	2.298
D-3	2.636	2.781	1.819	2.383
D-4	2.434	2.619	1.829	2.297
Mix	2.366	2.604	1.819	2.299
Oracle	2.358	2.595	1.817	2.293

Table A.4: Out-of-sample CE loss for out-of-domain performance of Gated-Assort-Net. Row headers denote the training dataset and column headers denote the testing dataset. The “Mix” header denotes a randomly mixed training set with equal proportion from the 4 datasets, and the “Oracle” header denotes the performance of the true model.

We train a neural network model with each of the training data and test its performance on the other datasets. In addition, we train a neural network model with a mixture of training data from the four datasets. Table A.4 summarizes the out-of-sample CE loss for each training-test pair. For illustration, we train a 1-layer Gated-Assort-Net. The row headers denote the training dataset, and the column headers denote the testing dataset.

The result shows that, except for the dataset D-3 (the third row), Gated-Assort-Net performs well in out-of-domain datasets. It is understandable that the datasets D-1 and D-2 are informative enough for a neural network model to catch the whole picture of the true underlying model. The dataset D-4 also enjoys this privilege, which is a bit surprising but also reinforces the findings in [GH20] (though not in an exact same spirit). The distribution of D-3 clearly fails to

capture the interplay between the first half and the second half of the total products, and thus cannot reach the performance of the oracle or generalize to different assortment distributions.

Details on the true model generation. The training data is generated from an MCCM. The arriving probability λ_i for product i is generated by:

$$\lambda_i = \frac{\exp(\mu_i)}{\sum_{j=1}^n \exp(\mu_j)},$$

where μ_i , $i = 1, \dots, n$ is identically and independently (i.i.d.) sampled by standard Normal distribution $\mathcal{N}(0, 1)$. And the transition matrix ρ is generated row by row. For all the n rows, each row ρ_i is generated following the same rule:

$$\rho_{i,j} = \frac{\exp(\nu_{i,j})}{\sum_{j=1}^n \exp(\nu_{i,j})},$$

where $\nu_{i,j}$, $i, j = 1, \dots, n$ is i.i.d. sampled by standard Normal distribution $\mathcal{N}(0, 1)$.

A.5.2 Appendix for Subsection 2.6.2

In this subsection, we present an experiment on the depth and width of the neural network. Arguably, a wider or deeper neural network gives a larger model capacity. However, for the task of choice modeling, we do not recommend a too wide or too deep neural network, mainly for several reasons. First, a complicated neural network architecture requires a larger number of samples to train, and there might be not enough amount of samples in the application contexts of choice modeling. Second, the customer choice behavior may not need a too complicated neural network to describe. In addition, the assortment optimization’s MIP prefers a smaller size one.

Figure A.1 presents an experiment on varying the width and depth of the four networks to fit an MCCM with product features. We do not observe a significant change in performance by further widening or deepening the architecture.

Experiment details of Figure A.1: The underlying model is an MCCM with product

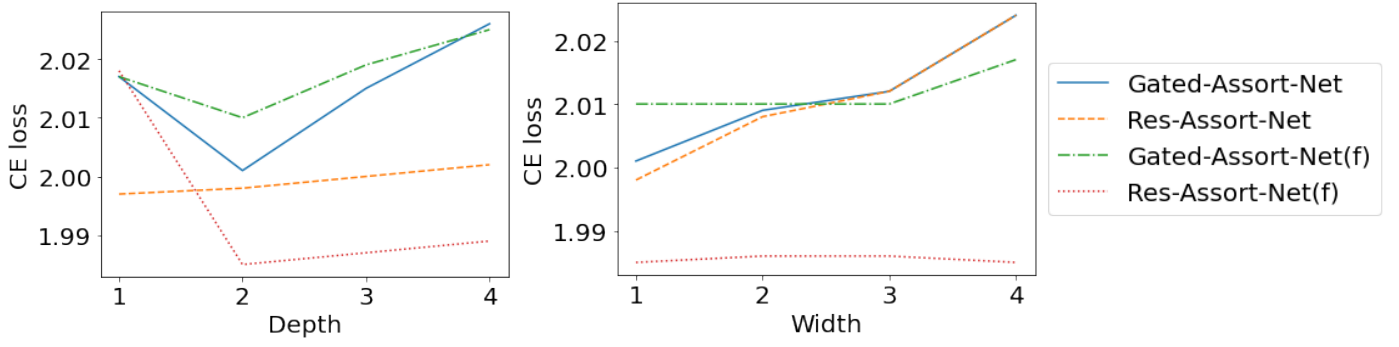


Figure A.1: Network structures and corresponding CE losses.

features. The sample generation method is the same as that of Table 2.5. We set the total number of products $n = 50$, and the length of product features $d = 5$. For choice networks with features, we use an encoder containing one middle layer to encode each product feature vector to its latent utility. The product encoder is (fixed as) a fully connected layer $\text{Linear}(5, 5)$ followed by another fully connected layer $\text{Linear}(5, 1)$. And for choice networks without features, we ignore product features and simply take assortments as input.

We carry out our training on 100,000 samples and the testing set has a size 10,000, and we tune the structure of each neural network model to compare their performances on the testing set.

- **Gated-Assort-Net:** Depth = 1 indicates a network with no hidden layer, just a fully connected layer $\text{Linear}(50, 50)$ that takes as input the assortment and outputs the final layer. Depth = 2 indicates a network with one hidden layer, which is a $\text{Linear}(50, 50)$ followed by another $\text{Linear}(50, 50)$, and Depth = 3 indicates two hidden layers. For the experiment that tunes the width, we fix Depth = 2, which is one hidden layer. And we tune the size of the hidden layer. Width = 2 indicates a $\text{Linear}(50, 100)$ followed by a $\text{Linear}(100, 50)$, and Width = 3 indicates a $\text{Linear}(50, 150)$ followed by a $\text{Linear}(150, 50)$.
- **Res-Assort-Net:** Depth is exactly the number of residual blocks. For the experiment that tunes the width, we fix that there is one residual block, though this block contains one hidden layer. Width = 2 indicates that the fully connected part of the residual block is a $\text{Linear}(50, 100)$ followed by a $\text{Linear}(100, 50)$, and Width = 3 indicates a $\text{Linear}(50, 150)$ followed by a $\text{Linear}(150, 50)$.

- Gated-Assort-Net(f): Following the same rule introduced above, product features are encoded into a vector of latent utilities. The meanings of depth and width are exactly the same as in Gated-Assort-Net.
- Res-Assort-Net(f): According to the structure of the Res-Assort-Net with feature, latent utilities and assortments are concatenated into a vector of length 100, as the input of the residual blocks. Set aside the ending fully connected layer $\text{Linear}(100, 50)$, the rest parts of the residual blocks are exactly the same as the ones in Res-Assort-Net. So depth is the number of residual blocks, and width indicates the size of the hidden layer. Width = 2 indicates that the fully connected part of the residual block is a $\text{Linear}(100, 200)$ followed by a $\text{Linear}(200, 100)$, and Width = 3 indicates a $\text{Linear}(100, 300)$ followed by a $\text{Linear}(300, 100)$.

A.5.3 Appendix for Subsection 2.6.3

We compare the training procedure of the following two plans:

- Cold start: We ignore the previous network of $\mathcal{N}$, and train a new network from scratch for the new set $\mathcal{N} \cup \mathcal{N}'$.
- Warm start: We initialize the weights of the new network according to the previously well-trained network of $\mathcal{N}$.

Figure 2.5 presents the validation losses under the two training schemes for the two models with the old set $|\mathcal{N}| = 20$ and the new set $|\mathcal{N}'| = 5$. The experiment shows that when there is an abundant amount of data for augmented models, warm start training will converge to the optimal solution faster in the starting stages as the right panel. More importantly, when there is not enough data, warm start training will not only be faster but also produce a better solution. The second observation can be practically useful, as the retailers may carry out small-scale experiments to test if it is profitable to introduce more products to their shops. The

experiment shows that the neural networks can be trained with a small amount of new data if most of its parameters are inherited from a well-trained model.

Experiment details of Figure 2.5: We use MCCM to generate the synthetic data for Figure 2.5. We first generate the arriving probability λ and the transition matrix ρ for the underlying MCCM. We consider 25 products and so $\lambda \in \mathbb{R}^{26}, \rho \in \mathbb{R}^{26 \times 26}$, as product 0 (the no-purchase option) needs to be included in addition. Following this MCCM we generate 200,000 pieces of samples, with each sample containing an assortment and the corresponding choice probability and the one-hot encoded final choice vector. We call this synthetic dataset D-augment. Then, based on the λ and ρ for the D-augment, we can do some modifications to classify product 21 to 25 into product 0. We get $\lambda' \in \mathbb{R}^{21}$ such that $\lambda'_i = \lambda_i$ for $i = 1, \dots, 20$ and $\lambda'_0 = \lambda_0 + \sum_{i=21}^{25} \lambda_i$. In similar manners, we get $\rho' \in \mathbb{R}^{21 \times 21}$, corresponding to a shrunk version of the original MCCM, i.e., a new MCCM with a total number of products $n = 20$ plus one no-purchase option. For this new model, we also generate 200,000 pieces of samples, and we call this dataset D-shrink.

We first train models on dataset D-shrink, the Gated-Assort-Net has one hidden layer of size 50, that is a Linear(20, 50) followed by a Linear(50, 20), and the Res-Assort-Net has one residual block, also with one hidden layer of size 50. We train the Gated-Assort-Net and Res-Assort-Net on 100,000 training samples. After that, we consider the augmented dataset D-augment, and we train Gated-Assort-Net and Res-Assort-Net with one hidden layer of size 50. However, we consider two methods for training the augmented model. One is the cold start method, in which we randomly initialize the model parameters and train the model on D-augment; the other is the warm start method, in which we initialize parameters at corresponding positions with the models trained on D-shrink and then train the model on D-augment. The results are given in Figure 2.5, on which we plot how validation loss varies with training epochs. The left panel is the training on D-augment with training sample size $m = 2,000$, and the right panel is the training with sample size $m = 100,000$.

A.5.4 Appendix for Subsection 2.6.4: Hotel Data Example

We apply Algorithm 1 to a public hotel dataset. The data is from [BFG09] and it is originally collected from five U.S. properties of a major hotel chain. For each customer, the offered products in the assortment are the different room types, such as king room smoking and 2 double beds room non-smoking. We preprocess the data as in [ST18], removing purchases without matched room type in the assortment, room types with few purchases (less than 10), and adding an auxiliary no-purchase option (create four no-purchase records for each purchase record). The summary statistics of the dataset are given in Table A.5.

	Hotel 1	Hotel 2	Hotel3	Hotel 4	Hotel5
# Samples	8935	870	6180	1330	1065
# Products	12	7	17	8	7
# Train Samples	6000	600	4000	900	800
# Validate Samples	1000	100	1000	100	100
# Test Samples	1000	100	1000	200	100

Table A.5: Description of Hotel data ([BFG09]).

Table A.6 demonstrates the performance of two benchmark methods and the two neural network models. The experiments are repeated over 10 times varying the training and test sets randomly and the averaged results reported: each time the train/validation/test sets are independently and randomly separated. In this experiment, we train our Gated-Assort-Net and Res-Assort-Net for each of the hotels using only samples for the hotel. Both networks are taking the simplest configuration: Gated-Assort-Net has no middle layer and Res-Assort-Net has one residual block. Compared to the benchmark MCCM-EM algorithm, we observe that our models have better performance on the dataset of Hotel 1 and Hotel 3. This is due to the fact that these two datasets have a much larger number of samples than the other 3 hotels. This coincides with our previous observations that, a moderately large number of samples is necessary to fully show the advantage of a neural network model. When the number of samples is small, or possibly, the customer choice pattern is simple, the more analytical model of MNL-MLE or MCCM-EM is preferable. To improve the performances of the neural choice model, we can apply Algorithm 1: we generate abundant synthetic data from the well-trained MCCM, which has the lowest losses in validation data, to feed our neural choice model. Specifically, for

each hotel in Hotel 2, Hotel 4, and Hotel 5, we generate 150,000 samples as training dataset $\tilde{\mathcal{D}}_{\text{train}} = \{(i_k, \mathcal{S}_k), k = 1 \dots, 150,000\}$. The offered assortment $\mathcal{S}_k$ is generated by first randomly choosing assortment size $|\mathcal{S}|$ in $\{1, \dots, |\mathcal{N}|\}$, and then randomly picking products from $\mathcal{N}$ and form the assortment $\mathcal{S}$ and the choice i_k is generated by the corresponding fitted MCCM for each hotel. We re-train a new neural network on the synthetic data $\tilde{\mathcal{D}}_{\text{train}}$ and fine-tune it on $\mathcal{D}_{\text{train}}$. We report the performances on Table A.6: by Algorithm 1, using the neural choice model as a meta choice model can outperform or at least as good as the benchmarks.

Model	Dataset				
	Hotel 1	Hotel 2	Hotel3	Hotel 4	Hotel5
MNL-MLE	0.775	0.902	0.980	0.661	0.660
MCCM-EM	0.757	0.856	0.945	0.656	0.655
Gated-Assort-Net	0.748	0.869	0.934	0.678	0.680
Res-Assort-Net	0.748	0.879	0.928	0.688	0.679
Gated-Assort-Net(Meta)	-	0.856	-	0.647	0.648
Res-Assort-Net(Meta)	-	0.876	-	0.648	0.648

Table A.6: Performance on the Hotel data. The Meta means using neural networks as meta choice models by Algorithm 1 and the reported number is the out-of-sample CE loss.

A.6 More Extensions

A.6.1 Assortment Effect Processed by Fully Connected Layers

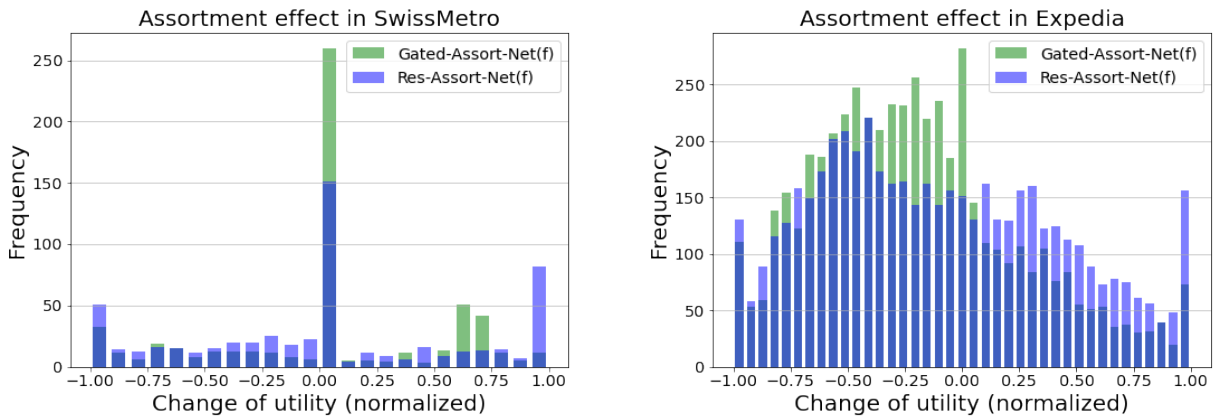


Figure A.2: Effects of fully connected layers in SiwssMetro and Expedia. A larger absolute value means a larger assortment effect from fully connected layers.

In practice, we want to understand how the neural networks leverage assortment effect into pur-

chase prediction. Recall the feature-based neural choice models in Figure 2.3, the latent utilities are processed by fully connected layers with assortment information propagated through logic gates in Gated-Assort-Net(f) or directly as input features in Res-Assort-Net(f). To interpret such processing, we compute a normalized difference between the input latent utility and output utility of each network. Such a difference reflects the assortment effect in the network. We denote such difference as Δu and plot the histograms of Gated-Assort-Net and Res-Assort-Net in Figure A.2.

From Figure A.2, we find that effects from fully connected layers will vary due to different dataset structures and networks. For the SwissMetro data set, the majority of changes in normalized utilities Δu are located in 0 for both networks. One plausible reason is that there are only three products and only a small number of samples (13.5%) do not offer all the products. Thus, the effect of assortments through the fully connected layer is very small. However, since the Expedia data has more complicated assortment patterns, the Δu 's do not concentrate at 0 anymore and have more variance. This also explains why the Res-Assort-Net(f) which has a stronger involvement of the assortment vector, shows an advantage on the Expedia dataset.

Further, by comparing Gated-Assort-Net(f) and Res-Assort-Net(f) in both data sets, there are two interesting observations: (1) There are more ± 1 's in Res-Assort-Net(f), which means more dramatic changes in normalized utilities after passing the fully connected layers; (2) The histograms of Res-Assort-Net(f) are flatter than those of Gated-Assort-Net(f), which means the influences of fully connected layers have more variations. Both of the above observations can be partially explained by the architectures: the Gated-Assort-Net(f) only uses assortments as logical gates but Res-Assort-Net(f) also concatenates it as input features and can squeeze more information to process the input latent utilities, which leads to potentially larger degree of influence with more variations.

Experiment details of Figure A.2: For each offered product $i \in \mathcal{S}$, we first normalize its input and output utilities of the fully connected layers (from Gated-Assort-Net(f) or Res-Assort-Net(f)) by

$$\tilde{u}_i^{I/O} = \frac{u_i^{I/O} - \min_{j \in \mathcal{S}} u_j^{I/O}}{\max_{j \in \mathcal{S}} u_j^{I/O} - \min_{j \in \mathcal{S}} u_j^{I/O}},$$

where u^I denotes input utilities and u^O denotes outputs of the fully connected layers. Note that the normalization is within assortment: different assortments may have different scales on utilities and what matters is the relative value.

Then we compute the difference:

$$\Delta u_i := \tilde{u}_i^O - \tilde{u}_i^I$$

for each $i \in \mathcal{S}$ of the fully connected layers. $\Delta u_i = 0$ means the normalized utilities are the same before and after the processing of fully connected layers. If Δu_i is -1 or 1 , product i is dramatically influenced: before processing it has the largest utility (smallest utility), i.e. $\tilde{u}_i^I = 1$ ($\tilde{u}_i^I = 0$), but after processing it becomes the smallest (largest).

We randomly choose 200 samples from the test data to plot Figure A.2. The configurations of Gated-Assort-Net(f) and Res-Assort-Net(f) are chosen as the ones with the best performance shown in Table A.3.

A.6.2 Model Calibration

Another important aspect of overall performance is the matter of calibration. Ideally, we hope the predicted probability matches the empirical/true probability. For deep learning models, a high accuracy may often come with overconfidence in predictions (overestimation of the true probabilities) [GPSW17], which is called *miscalibration*. Figure A.3 is the calibration plot of predicting Metro for the SwissMetro dataset, and we observe that all the neural network models stay mostly below the 45-degree reference line, i.e., suffer from an over-confidence in predicting the probability.

Table A.7 summarizes the calibration performances of different models averaged for all choices. In general, both Gated-Assort-Net(f) and Res-Assort-Net(f) have good performances in calibration, which complements the low out-of-sample CE losses shown in Table A.3 in the sense of reliability.

Metric calculation for Table A.7: We use the Adaptive Expected Calibration Error (ACE)

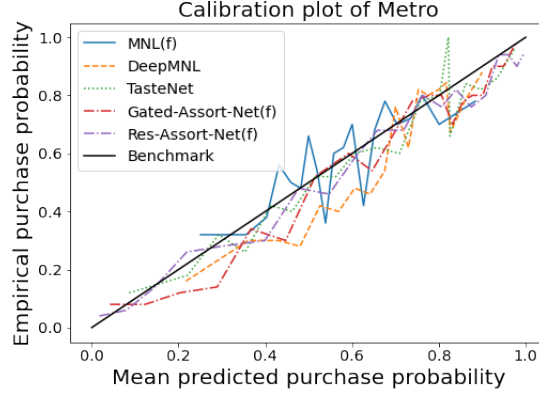


Figure A.3: Overconfidence of predicting Metro. The predicted probabilities of the test samples are binned into 20 bins with the same number of samples. For each bin, we compute the mean predicted prob. as the x-axis and the mean empirical prob. as the y-axis. The benchmark is the black solid line where $x = y$.

Model	Datasets	
	SwissMetro	Expedia
MNL-MLE	4.56	0.79
MNL(f)-MLE	6.22	0.61
TasteNet+MNL	5.50	—
TasteNet+MNL(f)	4.66	0.60
DeepMNL+MNL	8.19	—
DeepMNL+MNL(f)	7.26	0.59
Random Forest	7.56	0.41
Gated-Assort-Net(f)	4.67	0.39
Res-Assort-Net(f)	4.65	0.42

Table A.7: Calibration errors on SwissMetro and Expedia by ACE ($\times 10^{-2}$).

as our metric to measure calibration performance [NDZ⁺19]:

$$\text{ACE} = \frac{1}{mB} \sum_{i=1}^n n_i \sum_{b=1}^B |\text{acc}(b, i) - \text{conf}(b, i)|,$$

where m, B, n are the number of samples (assortments), bins, and products. Here n_i is the number of assortment samples where product i is included, $\text{acc}(b, i)$, $\text{conf}(b, i)$ are the mean of empirical purchase probabilities (confidence) and mean of predictions (accuracy) in b -th bin for product i . Further, for each product i , the bins are chosen by equal mass, i.e., each bin will contain the same number of samples.

For Table A.7, we choose the number of bins $B = 25$. The configurations of models are chosen as the one with the lowest CE losses in Table A.3. The training and testing data are the same

as in Appendix A.2.2.

Appendix B

Appendix for Chapter 3

B.1 Proof for Theorem 3.2

Proof. For items $\mathcal{N} = \{1, 2, \dots, n\}$, assume the underlying sequential choice model is represented by parameters $\{u_i^{\mathcal{C}, \mathcal{S}}\}$ for (3.8). Then it is sufficient to prove the output virtual utilities of some TCNet (with specific parameters) $\tilde{u}_i^{\mathcal{C}, \mathcal{S}} = u_i^{\mathcal{C}, \mathcal{S}}$ for all $i, \mathcal{C}, \mathcal{S}$. Since the items are finite, without loss of generality, we assume the input feature of each item i is the one-hot vector with dimension n where only i -th element is 1 and all others are 0's.

We first focus on the assortment encoder part. We construct a one-layer assortment encoder without the embedding sub-layer. For the self-attention sub-layer, we set its query weight matrix W_Q as zero matrix, value weight matrix W_V as identity matrix, activation function $\phi(x) = 1 + \text{Relu}(x)$ and without any residual connection. Under the above setting, the self-attention sub-layer just sums all rows in the input matrix $X^{\mathcal{S}}$ for each item: for any item $i \in \mathcal{S}$, its output is a vector with dimension n : $\sum_{j \in \mathcal{S}} x_j$. Then for the following feed-forward sub-layer, we set all parameters as 0 to "mute" this sub-layer and set the residual connection to make sure the final outputs of the assortment encoder $\tilde{X}^{\mathcal{S}}$ has each row $\tilde{x}_i^{\mathcal{S}} = \sum_{j \in \mathcal{S}} x_j$. Indeed, this output uses 0 and 1 to encode the assortment information: the positions with value 1 indicate the items in the assortment $\mathcal{S}$, and all other positions are the items not in the assortment.

Then for the candidates encoder part, we also construct it with only one layer and without the embedding sub-layer. For its self-attention sub-layer, we use the same structure as in the assortment encoder's self-attention except that we add the residual connection. Thus, with residual connection, the self-attention's output for item i is $x_i + \sum_{j \in \mathcal{C}} x_j$. For the following assortment-candidates attention sub-layer, we again set its query weight matrix W_Q as zero matrix, value weight matrix W_V as identity matrix, with residual connection, but the activation function is now Softmax. Thus, this attention itself (without residual connection) will output the mean of all rows in $\tilde{X}^{\mathcal{S}}$, i.e., $\sum_{j \in \mathcal{S}} x_j$, for each item $i \in \mathcal{C}$. With residual connection, the final output for each item i of this sub-layer is just $x_i + \sum_{j \in \mathcal{C}} x_j + \sum_{j \in \mathcal{S}} x_j$. We should notice this output vector uses four values to encode all input information: the position with value 3 indicates the item i we focus on, value 2's indicate the items as candidates by noting $\mathcal{C} \subseteq \mathcal{S}$, value 1's indicate the items are not candidates but were from the assortment (i.e., items in the basket), and 0's indicate the items not in the assortment. Finally, as in the assortment encoder part, for the following feed-forward network, we set all parameters as 0 to "mute" this sub-layer and set the residual connection to make sure the final outputs of the candidates encoder are still $\tilde{x}_i^{\mathcal{C}, \mathcal{S}} = x_i + \sum_{j \in \mathcal{C}} x_j + \sum_{j \in \mathcal{S}} x_j$.

Now we only need to show that the utility decoder can perfectly decode the input information $\tilde{x}_i^{\mathcal{C}, \mathcal{S}}$ to approximate $u_i^{\mathcal{C}, \mathcal{S}}$. We utilize the universal approximation theorem [Hor91] to prove it.

We set the utility decoder with one layer:

$$\text{decoder}(x) = \sum_{\mathcal{S} \subseteq \mathcal{N}, \mathcal{C} \subseteq \mathcal{S}, i \in \mathcal{C}} u_i^{\mathcal{C}, \mathcal{S}} \left(\text{Relu}(w^\top(x - \tilde{x}_i^{\mathcal{C}, \mathcal{S}}) - 1) + \text{Relu}(w^\top(x - \tilde{x}_i^{\mathcal{C}, \mathcal{S}}) + 1) - 2 \times \text{Relu}(w^\top(x - \tilde{x}_i^{\mathcal{C}, \mathcal{S}})) \right),$$

where $w = (4^1, 4^2, \dots, 4^n) \in \mathbb{R}^n$. Thus, the linear operation $w^\top \tilde{x}_i^{\mathcal{C}, \mathcal{S}}$ maps the input information $(i, \mathcal{C}, \mathcal{S})$ uniquely to an integer. And by noticing the operation

$$\text{Relu}(w^\top(x - \tilde{x}_i^{\mathcal{C}, \mathcal{S}}) - 1) + \text{Relu}(w^\top(x - \tilde{x}_i^{\mathcal{C}, \mathcal{S}}) + 1) - 2 \times \text{Relu}(w^\top(x - \tilde{x}_i^{\mathcal{C}, \mathcal{S}}))$$

equals to 1 when $x = \tilde{x}_i^{\mathcal{C}, \mathcal{S}}$ and 0 for all $x = \tilde{x}_{i'}^{\mathcal{C}', \mathcal{S}'}$ where $(i', \mathcal{C}', \mathcal{S}') \neq (i, \mathcal{C}, \mathcal{S})$, we can conclude that $\tilde{u}_i^{\mathcal{C}, \mathcal{S}} = \text{decoder}(\tilde{x}_i^{\mathcal{C}, \mathcal{S}}) = u_i^{\mathcal{C}, \mathcal{S}}$. $\square$

B.2 Appendix for Section 3.6.1

This appendix provides detailed information about the datasets and their preprocessing steps, model implementations, training processes, and hyperparameter tuning methods referenced in §3.6.1.

B.2.1 Dataset

	Single Choice								Multiple Choice	
	Car	Sushi	Expedia(book)	Hotel	SFwork	SFShop	Flight	Retail	Bakery	Expedia(click)
# Samples	4,652	5,000	40,000	9,330	5,029	3,157	40,000	17,200	20,000	8,041
# Features	21	7	13	0	0	0	0	0	4	13
Largest Assortment Size	6	10	39	17	6	8	26	21	50	38
Mean Assortment Size	6	10	25.90	10.99	4.38	7.20	10.58	17.20	50	26.85
Largest Chosen Subset Size	1	1	1	1	1	1	1	1	8	10
Mean Chosen Subset Size	1	1	1	1	1	1	1	1	3.55	4.95

Table B.1: Summary statistics of datasets.

Table B.1 summarizes the datasets used in the experiments and the details are as follows:

- Car: The data is available at <http://qed.econ.queensu.ca/jae/2000-v15.5/mcfadden-train/>. Each observation contains the single choice from six cars with varying features. We use 21 item features: *Prices divided by log(income), Range, Acceleration, Top speed, Pollution, Size, "Big enough", Luggage space, Operating cost, Station availability, Sports utility vehicle (SUV), Sports car, Station wagon, Truck, Van, Electric Vehicle (EV), Commute j 5 times for EV, College times EV, Constant for Compressed Natural Gas (CNG), Constant for methanol, College times methanol*. The details of the features can be found in [MT00].
- Sushi: The data is available at <https://www.kamishima.net/sushi/>. We use the dataset *sushi3b.5000.10.order*, which shows the top-10 ranked sushis for each customer. We use these top-10 sushis as the assortment and the top-1 as the final choice. We use 7 item

features *Style*, *Major-group*, *Minor-group*, *Heaviness*, *Frequency-eaten*, *Normalized-price*, *Frequency-sold*, and 6 customer features *Gender*, *Age*, *Time to fill the survey*, *Most Long-Lived East/West ID until Age 15*, *Current East/West ID*, *Match of Long-Lived and Current Region*. The details of the features can be found in [Kam03].

- **Hotel:** The data is available at <https://pubsonline.informs.org/doi/abs/10.1287/msom.1080.0231> and from [BFG09]. It is originally collected from five U.S. properties of a major hotel chain. Each observation contains different room types to be chosen and the final choice. We use the Hotel 1's dataset. We preprocess the data as in [ST18], removing purchases without matched room type in the assortment, room types with few purchases (less than 10), and adding an auxiliary no-purchase option (create four no-purchase records for each purchase record).
- **SFwork and SFshop:** The raw data is available at <https://github.com/tfresource/modechoice> and originally from [KB06]. We use the cleaned version from <https://github.com/arjunsesh/cdm-icml>. The datasets contain the choices of transportation to commute to shop or work around the San Francisco. Dataset features are not used in our study.
- **Flight and Retail:** These are two private real datasets without features. The airline data contains the offered choices of offered bundles, i.e., bundle of seat, food etc. The offered assortments are decided by both the selling strategy and remaining seats. The retailing data is collected from a fashion-retailing company, where the items are clothes (from a same category and thus most customers purchase at most one item) and the assortments are decided by the inventories. Both datasets contain records of purchase transactions only, so we add no-purchases following the same way as Hotel dataset.
- **Bakery:** the data is available at <http://users.csc.calpoly.edu/~dekhtyar/466-Spring2018/labs/lab01.html>. All observations share the same offered assortment containing 50 items from a bakery and the final (multiple) chosen items. We also encode each item with a unique one-hot encoder as the feature. We use the 20000 folder's dataset with 4 item features: *Flavor*, *Food Type*, *Price*, *Food or Drink*. To construct the sequential choice

dataset, we randomly sample one item from the chosen subset as the final choice, and add other unchosen items as the candidates (and the assortment are still all 50 items).

- Expedia: the data is available at <https://www.kaggle.com/c/expedia-hotel-recommendations/overview>. It contains the ordered list of hotels according to the user’s search, and the clicked and final booked (if any) hotel of each search session. We clean the data by dropping the searches with outlier prices ($> \$1,000$) and days of booking in advance (gap between booking date and check-in date > 365) and also the features with missing values. We use 7 item features *Hotel star rating*, *Hotel chain or not*, *Location score (by Expedia)*, *Historical prices*, *Price*, *Promotion or not*, *Randomly ranked or not (by Expedia)* and 6 customer features *# nights stay*, *# days booking in advance*, *# adults*, *# children*, *# rooms*, *Saturday included or not*. We also use the rankings (in the website) of each item as a feature. To construct the single choice dataset, we choose the booked hotel as the final single choice, and add the no-purchase option if there is no booking. For the multiple choice dataset, we use the sessions with total clicks in the range of 2 to 10, and use clicked hotels as the final choices. For the sequential choice dataset we use the same construction method as in the Bakery dataset.

In addition, due to the computation resource limitation, we truncate all datasets (Expedia and Flight) with more than 40000 samples to 40000 samples.

B.2.2 Models, Hyperparameters, and Training

- TCNet: Our implementation of the Transformer Choice Net contains a single-layer candidates encoder and an assortment encoder with one embedding sublayer. The utility decoder also contains a single layer but without an activation function. We employ multi-head attention, layer normalization, residual connection, and dropout as described in [VSP⁺17], with the number of heads being a tunable hyperparameter and dropout rate 0.1. When inferring multi-choice model, we add a virtual stopping item during the training as discussed in §3.3.2.

- DeepMNL: We modify the architecture by [WMZ20, SLA20]. We use a neural network g to model the (mean) utility of each item i as a function of its features x_i : $u_i^{\mathcal{S}} = u_i = g(x_i)$ for all $i \in \mathcal{N}$ and $\mathcal{S} \subseteq \mathcal{N}$. The architecture we implement has two layers for the NN. This design is specific to scenarios with observed features; otherwise, it reduces to the MNL model.
- AssortNet: We implement the AssortNet architecture by [WCLT23]. In the absence of features, AssortNet uses a neural network g as an assortment encoder, which directly maps the assortment $\mathcal{S}$ into the assortment-specific utilities. Specifically, for any assortment $\mathcal{S}$, the output $g(\mathcal{S}) \in \mathbb{R}^n$ models $u_i^{\mathcal{S}} = g_i(\mathcal{S})$ for $i \in \mathcal{S}$ and $u_i^{\mathcal{S}} = -\infty$ for $i \notin \mathcal{S}$. When features are present, feature encoders are employed to embed these features. The architectural implementation comprises a single layer for the assortment encoder and two layers for the feature encoders (if needed).
- SDANet: We implement the Set-Dependent Aggregation Net (SDANet) by [ROS20], which uses two neural networks w and ϕ to model the assortment-specific utilities: $u_i^{\mathcal{S}} = w(\mathcal{S}) \cdot \phi(x_i, \mathcal{S})$. The exact implemented architecture follows the default setting of the posted code. https://drive.google.com/file/d/1KZVbqfVR6QNIpv38y4e8ptzVdbi_GQH4/view.
- FATENet: We implement the First Aggregate Then Evaluate Net (FATENet) by [PGHH22], which uses two neural networks U' and ϕ to model the assortment-specific utilities: $u_i^{\mathcal{S}} = U' \left(x_i, \frac{1}{S} \sum_{j \in \mathcal{S}} \phi(x_j) \right)$. The exact implemented architecture follows the default setting of their posted code. <https://github.com/kiudee/cs-ranking>.
- Mixed-MNL: We train a featured mixed-MNL model [MT00] by setting the number of customer types the same as the number of item features, and each customer type's utility on item i is a linear function of its features. This follows the setting in [TB21].
- DLCL: We implement decomposed linear context logit (DLCL) by [TB21], which is an extension of Mixed-MNL such that each customer type's utility on item i is a linear function of both the item i itself and the averaged features over all items in the assortment.

The number of customer types is still set by the number of item features, which follows the setting in [TB21].

B.2.3 Training Process and Hyperparameters Tuning

We train all models using the Adam optimizer [KB14], targeting the minimization of training loss specified in §3.6.1. The learning rate decays exponentially at a rate of 0.95 every 10 epochs. The initial learning rate is tuned from the values 0.001, 0.0005, and 0.0001. All trainable parameters are initialized using the uniform Xavier method [GB10]. Training has a total of 100 epochs without early stopping, utilizing a mini-batch size of 256. The model with the lowest validation error over the 100 epochs is chosen for testing. Additionally, for models DeepMNL, AssortNet, SDANet, FATENet, and TCNet, we tune the width of the hidden layers from 32, 128, and 256. For the number of heads in TCNet, we tune it from 4, 8, 16, and 32. For the DeepMNL, AssortNet, SDANet, FATENet, and TCNet in multiple choices prediction task, we tune the threshold defined in (3.9) from 0.1, 0.3, 0.5, 0.7, 0.9.

B.3 Appendix for Section 3.6.2

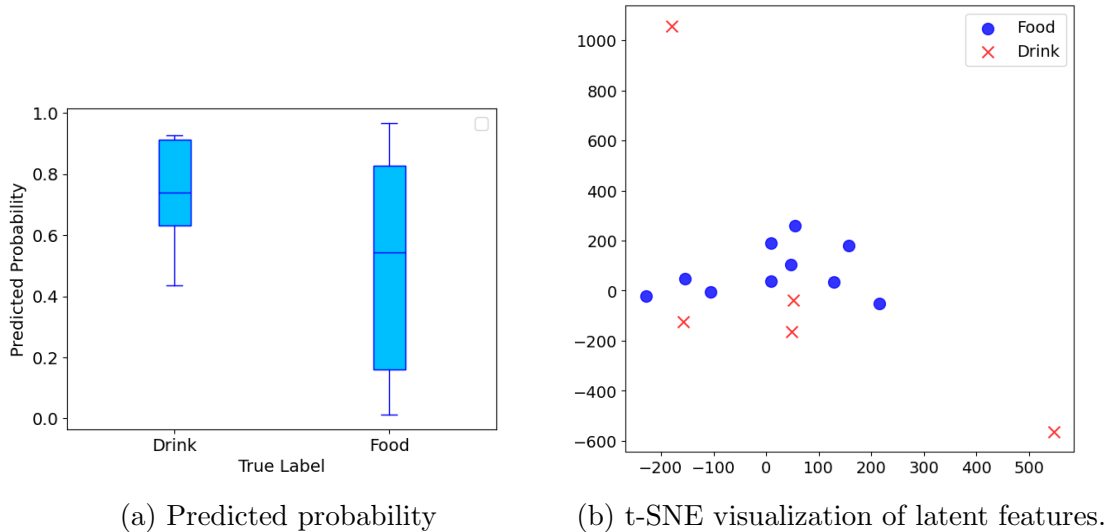


Figure B.1: Probing the latent representation to identify drink from food.

Experiment details. We train the TCNet with 4 heads on all attention sub-layers, and

we choose the initial learning rate as 0.0005 and the hidden dimension as 64. All other hyperparameters, training process, and dataset details are the same as in Appendix B.2. For training the linear logistic model in Figure 3.2, we select the cakes Strawberry Cake, Truffle Cake, Napoleon Cake, and Apple Pie (not cake but still more expensive and larger compared to snacks), and the snacks Ganache Cookie, Gongolais Cookie, Raspberry Cookie, and Lemon Cookie as training set. The testing dataset for both Figures 3.2a and 3.2b are shown in Figure 3.2b. For training the logistic model in Figure B.1, we select the items with indices from 35 to 44 as the training set, where the first 5 items are food and the others are drinks. The testing dataset for Figure B.1a contains all other samples, which includes 35 types of food and 5 drinks, and testing dataset for Figure B.1b contains the other 5 drinks and the food items with index from 0 to 10 (items information can be found in <http://users.csc.calpoly.edu/~dekhtyar/466-Spring2018/labs/lab01.html>).

B.4 Appendix for Section 3.6.3

B.4.1 Experiment Details

We train the TCNet with a single head on all attention sub-layers, and we choose the initial learning rate as 0.001 and the hidden dimension as 32. All other hyperparameters, training process, and dataset details are the same as in Appendix B.2.

B.4.2 More Visualization Examples

We use two more examples to showcase how the attention scores in each attention sub-layer help TCNet encode important items.

We construct a sequential choice model for four items $\mathcal{N} = \{A, A', B, L\}$, where B indicates a benchmark item and L indicates the 'leave' option. We assume items A' , B and L have constant utility 1 independent of $\mathcal{C}$ and $\mathcal{S}$. However, the utility of A can be boosted as follows:

In the first example, the utility of A equals to 100 if $A' \in \mathcal{C}$ and otherwise equals 1. This indicates an (extreme) case that the appearance of A' as a candidate can boost the utility of A . A similar choice behavior can be found in the experiment by [AJ08].

In second example, the utility of A equals to 100 if $A' \in \mathcal{S} \setminus \mathcal{C}$ (and otherwise equals to 1). This indicates an (extreme) case that the utility of A will be boosted when A' has been chosen. This describes the “buy one (A') and get another (A) free” situation.

We train two TCNets based on these two choice models. Figure B.2 and B.3 output the attention scores of two examples respectively.

Example 1: Figure B.2 illustrates the (normalized) attention scores derived from the trained TCNet for two inputs: $(\mathcal{S}_1, \mathcal{C}_1)$ and $(\mathcal{S}_2, \mathcal{C}_2)$, where $\mathcal{S}_1 = \mathcal{S}_2 = \{A, A', B, L\}$, $\mathcal{C}_1 = \{A, B, L\}$, $\mathcal{C}_2 = \{A, A', B, L\}$. Recall that the attention scores can basically quantify the influence of one item on another. Figure B.2a presents the candidates encoder’s self-attention score matrix, displaying relatively uniform attention allocated to all three candidates in $\mathcal{C}_1$; In contrast, Figure B.2b exhibits a significant increase in attention towards A for both B and L when A' is included among the candidates. This shift can reveal $A' \in \mathcal{C}_2$ significantly alters the choice probabilities, unlike in the first scenario where the absence of A' results in fairly balanced attention among the remaining candidates. Figures B.2c and B.2d represent the candidates encoder’s cross attention score matrix. There is a noticeable rise in weights on A for all three items A', B, L as depicted in Figure B.2d, compared to Figure B.2c. This elevation in weights demonstrates the substantial impact A has on the choice probability, attributed to its high utility value.

Example 2: Figure B.3 shows the attention scores derived from the TCNet again for two inputs same as in Example 1: $(\mathcal{S}_1, \mathcal{C}_1)$ and $(\mathcal{S}_2, \mathcal{C}_2)$. Figure B.3a and Figure B.3b display the assortment encoder’s self-attention score matrix for two inputs, which are identical since $\mathcal{S}_1 = \mathcal{S}_2$. One observation is the large attention on A' for A , which reveals the potentially large influence of A' since it appears in the assortment (while the assortment encoder is currently unknown whether A' is already chosen or not). Figures B.3c and B.3d represent the candidates encoder’s cross-attention score matrix. There are extreme weights on A for B, L as depicted in Figure B.3c,

compared to Figure B.3d. This indicates that the large influence of A on the choice probability, due to the fact that $A' \in \mathcal{S}_1 \setminus \mathcal{C}_1$, is encoded in B.3c; While when $A' \notin \mathcal{S}_1 \setminus \mathcal{C}_1$, because the (true) choice probabilities are equal among all candidates in Figure B.3d, the weights are more balanced.

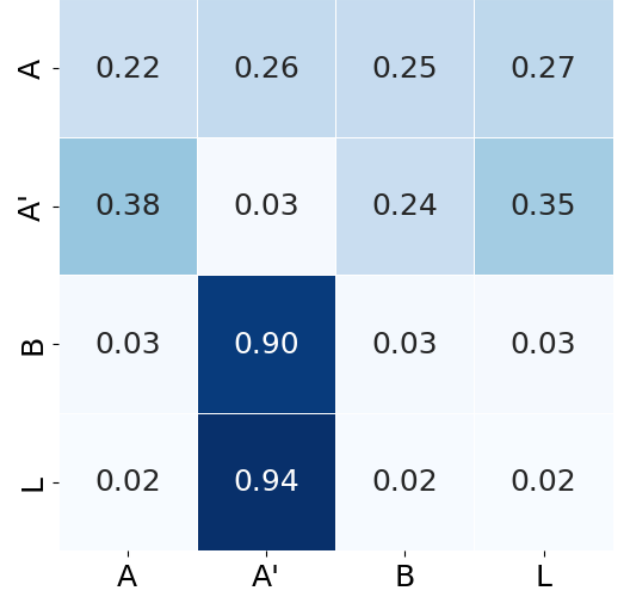
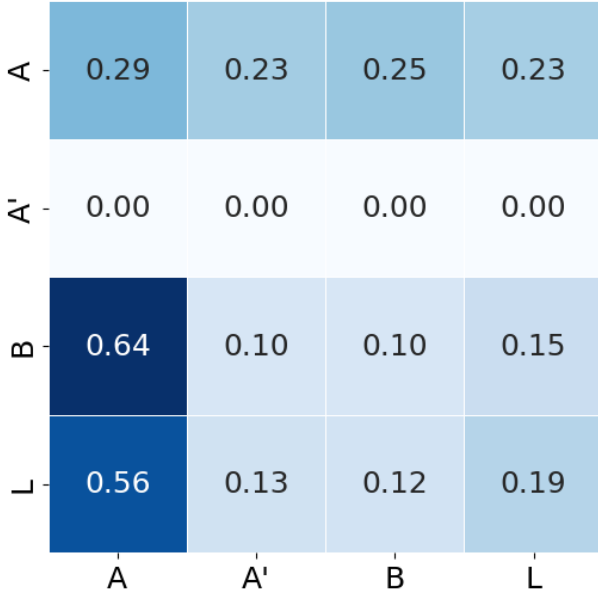
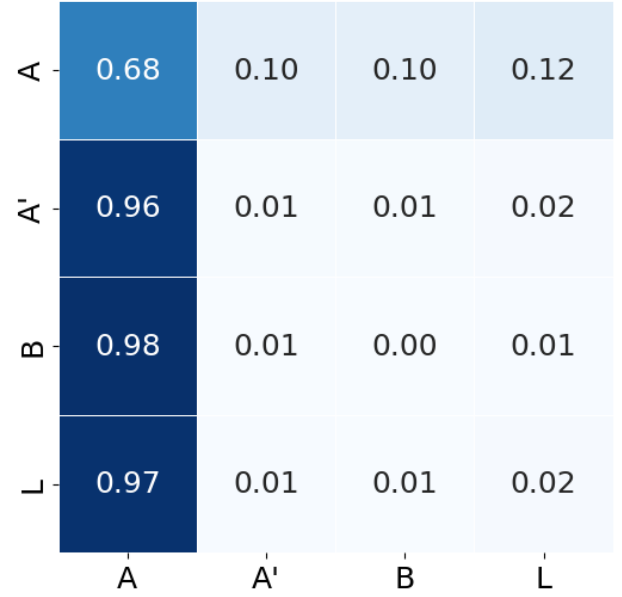
(a) Candidates attention of $(\mathcal{S}_1, \mathcal{C}_1)$.(b) Candidates attention of $(\mathcal{S}_2, \mathcal{C}_2)$.(c) Cross attention of $(\mathcal{S}_1, \mathcal{C}_1)$.(d) Cross attention of $(\mathcal{S}_2, \mathcal{C}_2)$.

Figure B.2: First example: candidates encoder's (self) attention and cross attention scores for two inputs $(\mathcal{S}_1, \mathcal{C}_1)$ and $(\mathcal{S}_2, \mathcal{C}_2)$, where $\mathcal{S}_1 = \mathcal{S}_2 = \{A, A', B, L\}$, $\mathcal{C}_1 = \{A, B, L\}$, $\mathcal{C}_2 = \{A, A', B, L\}$. Each row indicates the normalized attention scores for the labeled item. If the row's corresponding item is not available, we set its attention scores as 0.

Below are the experiment details:

$\mathcal{A}$	0.03	0.89	0.04	0.03
$\mathcal{A}'$	0.30	0.07	0.40	0.23
$\mathcal{B}$	0.33	0.10	0.37	0.19
$\mathcal{L}$	0.37	0.12	0.15	0.37
	A	A'	B	L

(a) Assortment attention of $(\mathcal{S}_1, \mathcal{C}_1)$.

$\mathcal{A}$	0.03	0.89	0.04	0.03
$\mathcal{A}'$	0.30	0.07	0.40	0.23
$\mathcal{B}$	0.33	0.10	0.37	0.19
$\mathcal{L}$	0.37	0.12	0.15	0.37
	A	A'	B	L

(b) Assortment attention of $(\mathcal{S}_2, \mathcal{C}_2)$.

$\mathcal{A}$	0.55	0.20	0.11	0.15
$\mathcal{A}'$	0.00	0.00	0.00	0.00
$\mathcal{B}$	0.95	0.02	0.01	0.02
$\mathcal{L}$	0.95	0.02	0.01	0.02
	A	A'	B	L

(c) Cross attention of $(\mathcal{S}_1, \mathcal{C}_1)$.

$\mathcal{A}$	0.16	0.36	0.19	0.29
$\mathcal{A}'$	0.20	0.33	0.17	0.30
$\mathcal{B}$	0.57	0.16	0.08	0.19
$\mathcal{L}$	0.32	0.25	0.14	0.29
	A	A'	B	L

(d) Cross attention of $(\mathcal{S}_2, \mathcal{C}_2)$.

Figure B.3: Second example: assortment encoder's self-attention and candidates encoder's cross attention scores for two inputs $(\mathcal{S}_1, \mathcal{C}_1)$ and $(\mathcal{S}_2, \mathcal{C}_2)$, where $\mathcal{S}_1 = \mathcal{S}_2 = \{A, A', B, L\}$, $\mathcal{C}_1 = \{A, B, L\}$, $\mathcal{C}_2 = \{A, A', B, L\}$. Each row indicates the normalized attention scores for the labeled item. If the row's corresponding item is not available, we set its attention scores as 0.

Training data generation: We generate 24,000 samples for both training datasets, where all samples' assortments are independently and identically sampled by including each item A, A', B with probability 0.5 and always including L . The candidates are similarly sampled by including the items in assortments with probability 0.5 while always including L . Conditional on the sampled assortment and candidates, the final choice is sampled by the underlying two choice

models respectively. TCNet architecture and training process: Both trained TCNets contain a single-layer candidates encoder and an assortment encoder with one embedding sub-layer. The utility decoder also contains a single layer but without an activation function. We employ single-head attention, layer normalization, residual connection, and dropout (with dropout rate 0.1) as described in [VSP⁺17]. The hidden dimensions of all sub-layers (if available) are set as 6. The training process is the same as in Appendix B.2.3 except there are only 20 training epochs and we apply 0.002 initial learning rate and 0.0005 weight decay on the optimizer.

Appendix C

Appendix for Chapter 4

C.1 Model Architecture and Alg^{*}

C.1.1 Model Architecture for OMGPT

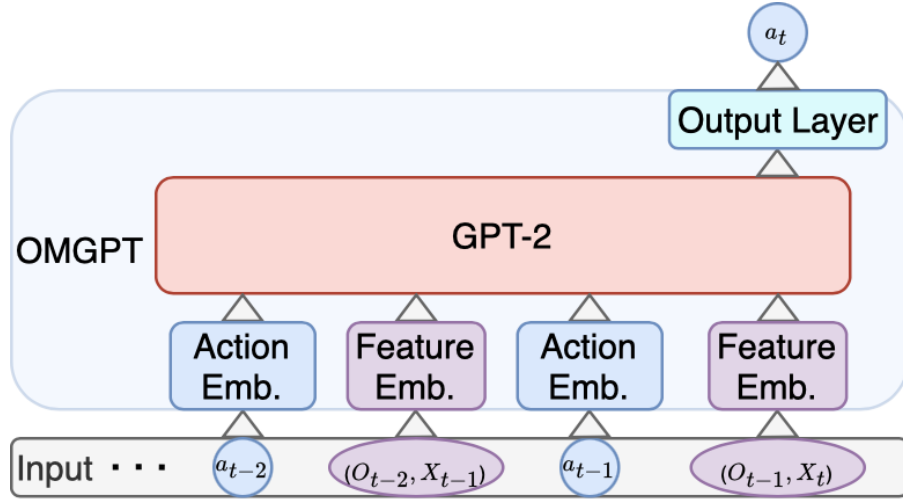


Figure C.1: The architecture of OMGPT, where all actions a_τ and the combined “feature” elements $(O_{\tau-1}, X_\tau)$ in H_t pass through the same action embedding layer and feature embedding layer, respectively. The resulting embedding vectors are then processed by a (trainable) GPT-2 model, which summarizes them into a single embedding vector. Finally, this embedding vector passes through the output layer, transforming it into the predicted action a_t .

We adopt the transformer architecture for regression tasks from [GTLV22] to solve sequential decision making tasks, with modifications tailored to our specific setting. The architecture is

summarized in Figure C.1.

Input Sequence Structure. Since we have three types of elements (contexts, actions, and observations), and the actions are the “label type” we want to predict sequentially, we first combine the contexts and the observations into “features”, similar to a supervised learning task. Specifically, instead of directly using H_t as the input sequence at time t , we construct two types of elements derived from H_t :

- **“Feature” elements** combine the context $X_\tau \in \mathbb{R}^{d_x}$ at each time $\tau \leq t$ with the observation $O_{\tau-1} \in \mathbb{R}^{d_o}$ from $\tau - 1$ (with O_0 set as a zero vector in our experiments). Formally, the feature elements are $\{(O_{\tau-1}, X_\tau)\}_{\tau=1}^t \subset \mathbb{R}^{d_o+d_x}$, and they serve as the input “features” for prediction.
- **“Label”/action elements** are the actions $\{a_\tau\}_{\tau=1}^{t-1}$ from H_t that were predicted in the testing phase (or generated in the pre-training phase).

Thus, the input sequence becomes $\{(O_0, X_1), a_1, \dots, (O_{t-2}, X_{t-1}), a_{t-1}, (O_{t-1}, X_t)\}$, which contains $2t - 1$ elements in total to predict a_t^* .

Architecture. The transformer is based on the GPT-2 family [RNS⁺18]. It uses two learnable linear transformations to map each “feature” and “label” element into vectors in the embedding space respectively. These transformations are referred to as the feature embedding layer and the action embedding layer. The output embedding vectors, which share the same dimension, are processed through the GPT-2’s attention mechanism, resulting in a single vector that encapsulates relevant contextual information. This vector then undergoes another learnable linear transformation, referred to as the output layer, moving from the embedding space to the action $\mathcal{A}$, ultimately resulting in the prediction of a_t^* . In our experiments, the GPT-2 model has 12 layers, 16 attention heads, and a 256-dimensional embedding space. The experiments are conducted on 2 A100 GPUs with `DistributedDataParallel` method of Pytorch.

C.1.2 Difference from [LXP⁺24]

While [LXP⁺24] also propose a supervised learning approach using transformers for general reinforcement learning/Markov decision process problems (e.g., game playing), there are four key differences between their work and ours:

- Pre-training data generation:** In pre-training, we develop a novel data generation method specifically tailored to these sequential decision making processes, as detailed in Appendix C.2. In addition, [LXP⁺24] propose constructing input sequences for predicting a_t^* using a set of tuples $(X_\tau, a_\tau, O_\tau, X_{\tau+1})$, which are sampled from a trajectory and are not contiguous or in chronological order. Intuitively, the goal is for the pre-trained transformer to implicitly learn the transitions and environment dynamics represented by the progression from (X_τ, a_τ, O_τ) to $X_{\tau+1}$, thereby “informing” its decision making process. In contrast, our approach generates contiguous sequences, i.e., $H_t = (X_1, a_1, O_1, \dots, X_{t-1}, a_{t-1}, O_{t-1}, X_t)$ to predict a_t^* . This approach offers two benefits: (i) It aligns with the testing phase, where decisions are made in chronological order. Such alignment is often necessary in the classic machine learning framework [SSBD14]. (ii) The chronological order itself can contain important information in operational decision making. For example, in a market where the demand model changes abruptly at some point t' (e.g., a lockdown during a pandemic), the decision maker (and OMGPT) should focus more on the histories after t' to learn the new market conditions.
- Underlying problems:** We focus on sequential decision making problems that do not have a transition probability matrix, unlike Markov decision processes. The advantage here is that the optimal action is either available in closed form or can be efficiently solved. This allows us to use the optimal action as the prediction target in the training data, making the supervised learning paradigm more practical for real-world operational applications. In addition, [LXP⁺24] focuses primarily on problems with discrete action spaces, whereas we conduct experiments and provide analyses for both discrete and continuous action spaces.

- **GPT model architecture:** Due to the different data generation methods and the special structure of the underlying problems, we employ a different GPT architecture.
- **Analysis:** The distinctions outlined above align our proposed paradigm more closely with sequential decision making in operations management and in-context learning literature [DLD⁺22]. This alignment grants us deeper analytical insights into the underlying mechanisms and performance of our OMGPT model. More details are elaborated in Section 4.5.

C.1.3 More Details on Alg^{*}

In this subsection, we study the behavior of $\text{TF}_{\hat{\theta}}$ by comparing regret performances on $\text{TF}_{\hat{\theta}}$ and the algorithms utilizing the posterior distribution $\mathcal{P}(\gamma|H)$ of environments. Besides Bayes-optimal decision functions as defined in Section 4.5.1, we can also name the later algorithms as the *oracle posterior algorithms* as they are assumed to know the posterior distribution $\mathcal{P}(\gamma|H)$ (or equally, the prior distribution $\mathcal{P}_{\gamma}$).

We first present a general framework for these oracle posterior algorithms in Algorithm 2: posterior averaging, sampling, and median. For simplicity of notation, we omit the subscript t and only consider a finite space Γ of environments, while the algorithms can be easily extended to the infinite case. For the posterior median algorithm, we assume the action space $\mathcal{A} \subseteq \mathbb{R}$.

Figure C.2 presents the testing regret and action suboptimality for $\text{TF}_{\hat{\theta}}$ compared to oracle posterior algorithms. Generally, all algorithms exhibit good regret performance. However, we observe that $\text{TF}_{\hat{\theta}}$ is (i) superior to posterior sampling in multi-armed bandits, as shown in (a), and (ii) outperforms posterior averaging in linear bandits, as shown in (c).

Specifically, (b) shows that posterior sampling has higher action suboptimality during the initial time steps compared to $\text{TF}_{\hat{\theta}}$. This suggests that the exploration inherent in the sampling step of posterior sampling can introduce additional regret, which might be unnecessary for simpler problems. Conversely, (d) indicates that the decisions from posterior averaging do not converge to the optimal action a_t^* as quickly as those from $\text{TF}_{\hat{\theta}}$ and posterior sampling. This suggests

Algorithm 2 Posterior averaging/sampling/median at time t

Require: Posterior probability $\mathcal{P}(\gamma|H)$ and optimal action a_γ^* for each environment $\gamma \in \Gamma$, algorithm $\text{Alg} \in \{\text{posterior averaging, posterior sampling, posterior median}\}$

```

  %% Posterior averaging
  1: if Alg=posterior averaging then
  2:    $a = \sum_{\gamma \in \Gamma} \mathcal{P}(\gamma|H) \cdot a_\gamma^*$ 
  %% Posterior sampling
  3: else if Alg=posterior sampling then
  4:    $a = a_{\tilde{\gamma}}^*$ , where  $\tilde{\gamma} \sim \mathcal{P}(\cdot|H)$ 
  %% Posterior median
  5: else if Alg=posterior median then
  6:   Sort and index the environments as  $\{\gamma_i\}_{i=1}^{|\Gamma|}$  such that the corresponding optimal actions
      are in ascending order:  $a_{\gamma_1}^* \leq a_{\gamma_2}^* \leq \dots \leq a_{\gamma_{|\Gamma|}}^*$ .
  7:   Choose  $a = \min \left\{ a_{\gamma_i}^* \mid \sum_{i'=1}^i \mathcal{P}(\gamma_{i'}|H) \geq 0.5 \right\}$ 
  8: end if
  9: Return:  $a$ 

```

that posterior averaging may be too greedy, thus failing to sufficiently explore the environment. This second observation also numerically supports Proposition 3.

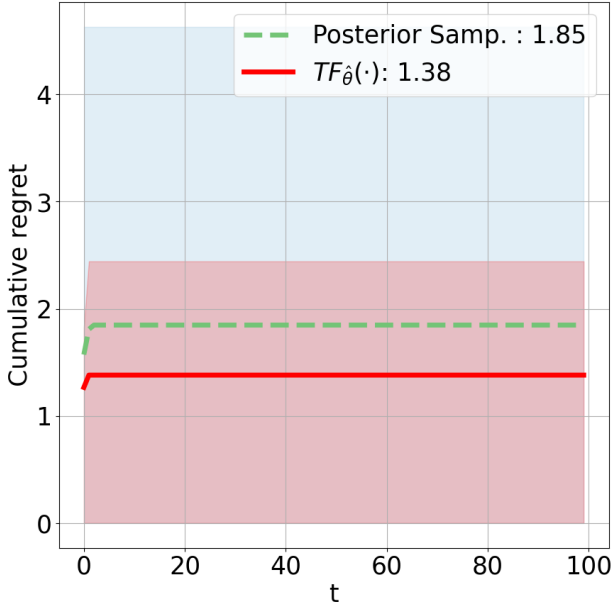
These observations illustrate that $\text{TF}_{\hat{\theta}}$ can outperform oracle posterior algorithms in certain scenarios. This further demonstrates that the advantage of $\text{TF}_{\hat{\theta}}$ over benchmark algorithms is not solely due to its use of prior knowledge about the task. Rather, $\text{TF}_{\hat{\theta}}$ discovers a new decision rule that achieves better short-term regret than oracle posterior algorithms. It can be more greedy than posterior sampling while exploring more than posterior averaging.

Setup. We consider a multi-armed bandits task (with 20 arms) and linear bandits task (with 2-dimensional actions) with 4 environments. The results are based on 100 runs. The details of tasks with their specific oracle posterior algorithms are provided in Appendix C.4.

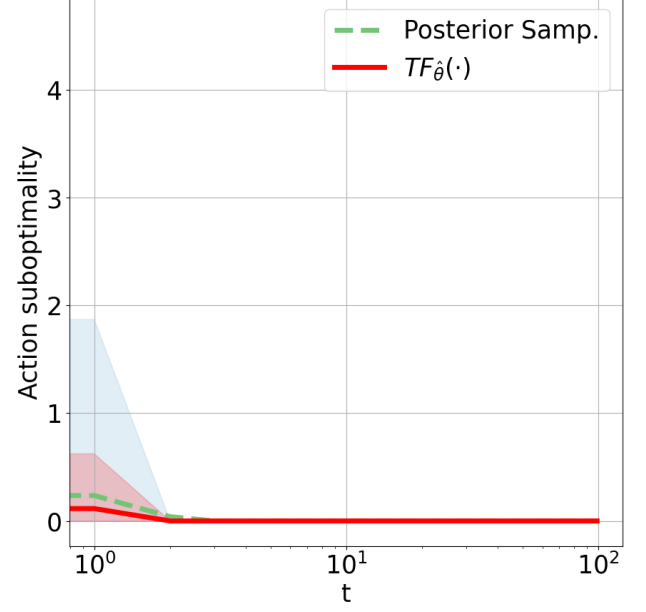
C.2 More Details on Pre-training

C.2.1 Generalization Issue in Pre-training

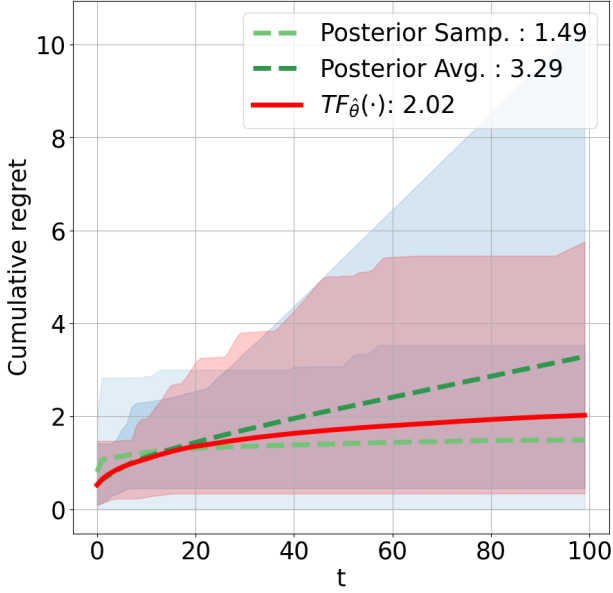
We first point out a subtle point of the supervised pre-training setting presented in Section 4.4.3. Unlike (traditional) supervised learning, the expected loss (4.5) may not be equal to the



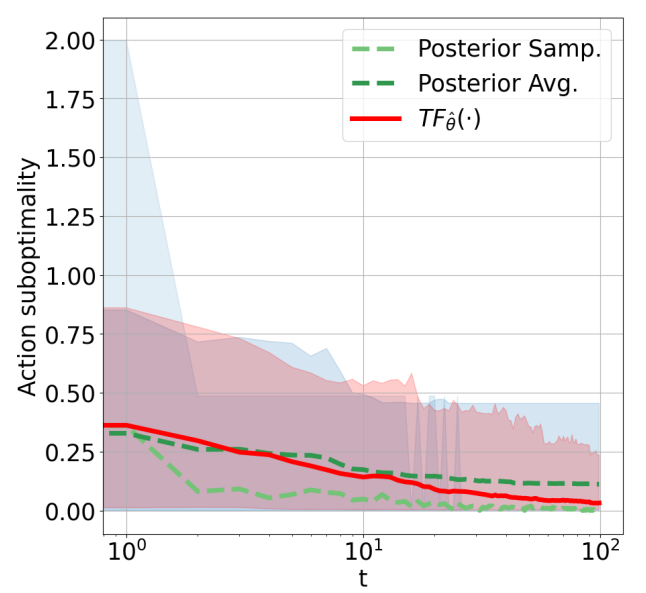
(a) Regret for multi-armed bandits



(b) Action suboptimality for multi-armed bandits



(c) Regret for linear bandits



(d) Action suboptimality for linear bandits

Figure C.2: Performances of $\text{TF}_{\hat{\theta}}$ and oracle posterior algorithms. The numbers in the legend bar are the final regret at $t = 100$ and the shaded areas indicate the 90% (empirical) confidence intervals.

expectation of the empirical loss (4.11). As noted earlier, this is because in the sequence H_t , the action $a_t = \text{TF}_{\hat{\theta}}(H_t)$ as in (4.12) during the test phase as opposed to that $a_t = \tilde{f}(H_t)$ as in (4.10) for the pre-training data $\mathcal{D}_{\text{PT}}$. Taking expectation with respect to γ in (4.5), we can define the expected loss as

$$L(\text{TF}_{\theta}) := \mathbb{E}_{\gamma \sim \mathcal{P}_{\gamma}} [L(\text{TF}_{\theta}; \gamma)].$$

However, the expectation of the empirical loss (4.11) is

$$L_{\tilde{f}}(\text{TF}_\theta) := \mathbb{E}_{\gamma \sim \mathcal{P}_\gamma} [L_{\tilde{f}}(\text{TF}_\theta; \gamma)] \quad (\text{C.1})$$

where

$$L_{\tilde{f}}(\text{TF}_\theta; \gamma) := \mathbb{E}_{(H_t, a_t^*) \sim \mathcal{P}_{\gamma, \tilde{f}}} \left[\sum_{t=1}^T l(\text{TF}_\theta(H_t), a_t^*) \right].$$

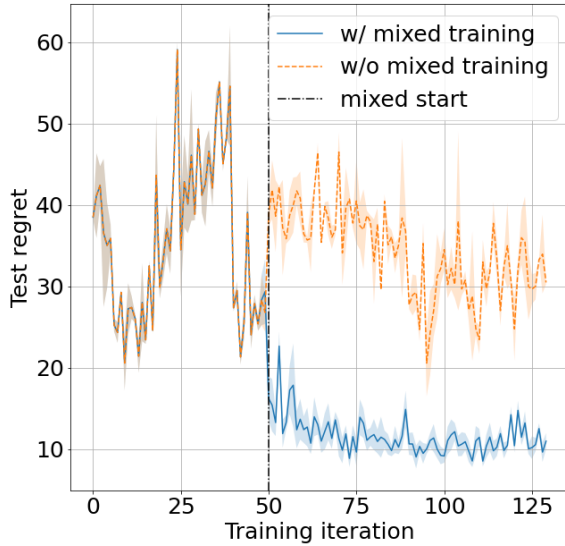
Here $\tilde{f}$ is the pre-specified decision function used in generating the pre-training data $\mathcal{D}_{\text{PT}}$. We note that the discrepancy between $L(\text{TF}_\theta)$ and $L_{\tilde{f}}(\text{TF}_\theta)$ is caused by the difference in the generations of the samples (H_t, a_t^*) , namely, $\mathcal{P}_{\gamma, \text{TF}_\theta}$ versus $\mathcal{P}_{\gamma, \tilde{f}}$; which essentially reduces to how the actions a_t 's are generated in H_t . In the training phase, this is generated based on $\tilde{f}$, while measuring the test loss requires that the actions to be generated are based on the transformer TF_θ . This discrepancy is somewhat inevitable because when generating the training data, there is no way we can know the final parameter $\hat{\theta}$. Consequently, there is no direct guarantee on the loss for $L(\text{TF}_{\hat{\theta}}; \gamma)$ or $L(\text{TF}_{\hat{\theta}})$ with $\hat{\theta}$ being the final learned parameter. This out-of-distribution (OOD) issue motivates the design of our algorithm.

Algorithm 3 Supervised pre-training transformers for sequential decision making

Require: Iterations M , early training phase $M_0 \in [1, M]$, number of training sequences per iteration n , ratio $\kappa \in [0, 1]$, pre-training decision function $\tilde{f}$, prior distribution $\mathcal{P}_\gamma$, initial parameter θ_0

- 1: Initialize $\theta_1 = \theta_0$
- 2: **for** $m = 1, \dots, M_0$ **do**
 `%% Early training phase`
- 3: Sample $\gamma_1, \gamma_2, \dots, \gamma_n$ from $\mathcal{P}_\gamma$ and $S_i = \left\{ \left(H_1^{(i)}, a_1^{(i)*} \right), \dots, \left(H_T^{(i)}, a_T^{(i)*} \right) \right\}$ from $\mathcal{P}_{\gamma_i, \tilde{f}}$
- 4: Optimize over the generated dataset $\mathcal{D}_m = \{S_i\}_{i=1}^n$ and obtain the updated parameter θ_{m+1}
- 5: **end for**
- 6: **for** $m = M_0 + 1, \dots, M$ **do**
 `%% Mixed training phase`
- 7: Sample $\gamma_1, \gamma_2, \dots, \gamma_n$ from $\mathcal{P}_\gamma$
- 8: For $i = 1, \dots, \kappa n$, sample $S_i = \left\{ \left(H_1^{(i)}, a_1^{(i)*} \right), \dots, \left(H_T^{(i)}, a_T^{(i)*} \right) \right\}$ from $\mathcal{P}_{\gamma_i, \tilde{f}}$
- 9: For $i = \kappa n, \dots, n$, sample $S_i = \left\{ \left(H_1^{(i)}, a_1^{(i)*} \right), \dots, \left(H_T^{(i)}, a_T^{(i)*} \right) \right\}$ from $\mathcal{P}_{\gamma_i, \text{TF}_{\theta_m}}$
- 10: Optimize over the generated dataset $\mathcal{D}_m = \{S_i\}_{i=1}^n$ and obtain the updated parameter θ_{m+1}
- 11: **end for**
- 12: **Return:** $\hat{\theta} = \theta_{M+1}$

Algorithm 3 describes our algorithm to resolve the above-mentioned OOD issue between training and testing. It consists of two phases, an early training phase and a mixed training phase. For the early training phase, the data are generated from the distribution $\mathcal{P}_{\gamma_i, \tilde{f}}$, while for the mixed training phase, the data are generated from both $\mathcal{P}_{\gamma_i, \tilde{f}}$ and $\mathcal{P}_{\gamma_i, \text{TF}_{\theta_t}}$ with a ratio controlled by κ . The decision function $\tilde{f}$ can be chosen as a uniform distribution over the action space or other more complicated options. Intuitively, the benefits of involving the data generated from $\mathcal{P}_{\gamma_i, \text{TF}_{\theta_t}}$ is to make the sequences $H_t^{(i)}$'s in the training data closer to the ones generated from the transformer. For the mixed training phase, a proportion of the data is still generated from the original $\mathcal{P}_{\gamma_i, \tilde{f}}$ because the generation from $\mathcal{P}_{\gamma_i, \text{TF}_{\theta_t}}$ costs relatively more time. For our numerical experiments, we choose $\kappa = 1/3$.



(a) Training dynamics

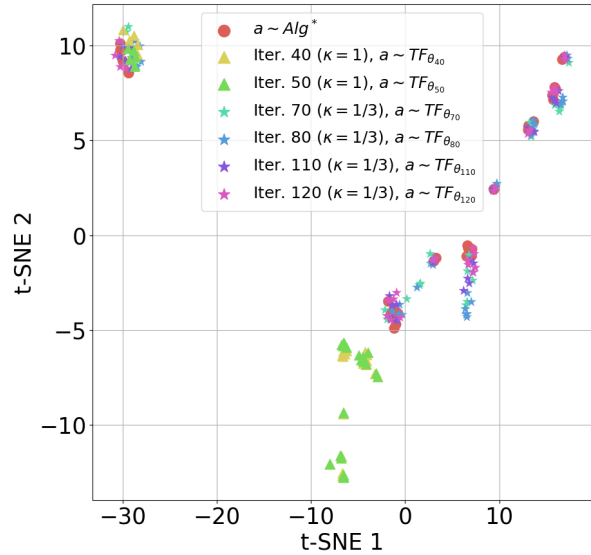
(b) t-SNE of the generated H_{30} 's

Figure C.3: (a) Training dynamics. Orange: $M_0 = M = 130$. Blue: $M_0 = 50$ and $M = 130$. It shows the effectiveness of injecting/mixing the transformer-generated sequence into the training procedure. (b) A visualization of the H_t with a_τ 's in H_t generated from various TF_{θ_m} . For each TF_{θ_m} , we generate 30 sequences. The decision function Alg^* is defined in the next section. We observe (i) there is a shift over time in terms of the transformer-generated action sequence, and thus the training should adaptively focus more on the recently generated sequence like the design in Algorithm 3; (ii) the action sequence gradually gets closer to the optimal decision function Alg^* . The experiment details and setups are deferred to the end of this subsection.

Algorithm 3 from an optimization perspective can be viewed as an iterative procedure to

optimize

$$\mathbb{E}_{\gamma \sim \mathcal{P}_\gamma} \left[\mathbb{E}_{(H_t, a_t^*) \sim \kappa \mathcal{P}_{\gamma, \tilde{f}} + (1-\kappa) \mathcal{P}_{\gamma, \text{TF}_\theta}} \left[\sum_{t=1}^T l(\text{TF}_\theta(H_t), a_t^*) \right] \right]$$

where both the input data pair (H_t, a_t^*) and the prediction function involve the parameter θ . This falls into the paradigm of performative prediction [PZMDH20, IYZ21, MDPZH20] where the prediction model may affect the data generated to be predicted. In the literature of performative prediction, a critical matter is the instability issue which may cause the parameter θ_m to oscillate and not converge. [PZMDH20] shows that the matter can be solved with strong conditions such as smoothness and strong convexity on the objective function. However, we do not encounter this instability in our numerical experiment, and we make an argument as the following claim. That is, when the underlying function class, such as $\{\text{TF}_\theta : \theta \in \Theta\}$ is rich enough, one does not need to worry about such instability.

Claim 1. *The instability of a performative prediction algorithm and the resultant non-convergence does not happen when the underlying prediction function is rich enough.*

To see this claim, consider the following setup where a feature-target pair (X, Y) is generated from some distribution $\mathcal{P}_{X,Y}(\theta)$. Then we consider the minimization of the following loss

$$\min_{\theta} \mathbb{E}_{\mathcal{P}_{X,Y}(\theta)} [l(f_\theta(X), Y)]$$

for some loss function $l : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$.

Under this formulation, both the data generation distribution $\mathcal{P}_{X,Y}(\theta)$ and the prediction function f_θ are parameterized by θ . This describes the *performative prediction* problem [PZMDH20].

In addition, we note that in our pre-trained transformer setting, the joint distribution can be factorized in

$$\mathcal{P}_{X,Y}(\theta) = \mathcal{P}_X(\theta) \cdot \mathcal{P}_{Y|X}.$$

That is, the parameter θ only induces a covariate shift by affecting the marginal distribution of X , but the conditional distribution $\mathcal{P}_{Y|X}$ remains the same for all the θ . This exactly matches our setting of pretrained transformer; to see this, different decision functions, f or TF_{θ_m} , only

differ in terms of the generation of the actions a_τ 's in H_t (for $\tau = 1, \dots, t-1$) which is the features X in this formulation but the optimal action a_t^* will only be affected by X_t in H_t .

Also, under this factorization, we define the Bayes-optimal estimator as

$$f^*(X) = \min_y \mathbb{E}_{\mathcal{P}_{Y|X}}[l(y, Y)|X].$$

Then it is easy to note that when $f^* = f_{\theta^*}$ for some θ^* , and have

$$\theta^* \in \arg \min_{\theta} \mathbb{E}_{\mathcal{P}_{X,Y}(\theta')} [l(f_{\theta}(X), Y)]$$

for any θ' .

In this light, the oscillating behavior of the optimization algorithms in [PZMDH20] will not happen. Because for all the data generation distribution $\mathcal{P}_{X,Y}(\theta')$, they all point to one optimal θ^* .

Back to the context of the pre-trained transformer, such a nice property is contingent on two factors: (i) the transformer function class is rich enough to cover f^* ; (ii) there are infinitely many samples, or we can use the expected loss. Also, the above argument is connected to the argument of Proposition 1.

C.2.1.1 Details for Figure C.3

Figure C.3 (a). In Figure C.3 (a), we independently implement Algorithm 3 to train two OMGPTs on a dynamic pricing task, which has 6-dimensional contexts (more details can be found in Appendix C.4.2). For both OMGPTs, the training parameters are identical except for the M_0 in Algorithm 3: the first curve (blue, thick line) uses $M_0 = 50$, while the second curve (orange, dashed line) uses $M_0 = 130$, meaning no transformer-generated data is utilized during training. All other parameters follow the configuration detailed in Appendix C.2.2. The figure shows the mean testing regret at each training iteration across 128 randomly sampled environments, with the shaded area representing the standard deviation.

Figure C.3 (b). In Figure C.3 (b), we consider a dynamic pricing task with a pool of 8 linear demand functions and has 6-dimensional contexts (more details can be found in Appendix C.4.2). We generate 30 sequences from TF_{θ_m} across different training iterations for this task: before using the transformer-generated data in the training ($m = 40, 50$) and after using such data ($m = 70, 80, 110, 120$). All generated sequences share the same context sequence $\{X_t\}_{t=1}^T$ to control the effect of contexts on the chosen actions and the resulting observations. We stack the first 20 actions and observations into a single sample point and use t-SNE method to visualize these high-dimension points as shown in Figure C.3 (b). We further apply the same method to generate points from Alg^* .

We can observe that: (i) compared to the points from TF which have not been trained on self-generated data (i.e., $\text{TF}_{\theta_{40}}, \text{TF}_{\theta_{50}}$), the points from the trained ones are closer to the Alg^* 's, i.e., the expected decision rule of a well-trained TF; (ii) When being trained with more self-generated data, the points from TF_{θ_m} get closer to the Alg^* 's. These observations suggest that utilizing the self-generated data (like in Algorithm 3) can help mitigate the OOD issue as discussed above: the data from TF_{θ_m} can get closer to the target samples from Alg^* during the training and thus the pre-training data is closer to the testing data.

C.2.2 Pre-training Implementation

During pre-training, we use the AdamW optimizer with a learning rate of 10^{-4} and a weight decay of 10^{-4} . The dropout rate is set to 0.05. For implementing Algorithm 3 in our experiments, we set the total iterations $M = 130$ with an early training phase of $M_0 = 50$. The number of training sequences per iteration is $n = 1500 \times 64$, which are randomly split into 1500 batches with a batch size of 64. The transformer's parameter θ is optimized to minimize the averaged loss of each batch. As suggested in Proposition 2, we select the cross-entropy loss for the multi-armed bandits, squared loss for the dynamic pricing, and absolute loss for the linear bandits and newsvendor problem. Further, we select the squared loss for the queuing control and the revenue management.

To reduce computation costs from sampling histories during the early training phase, instead of

sampling a new dataset D_m in each iteration (as described in Algorithm 3), we initially sample 10^6 data samples before the pre-training phase as an approximation of $\mathcal{P}_{\gamma, \tilde{f}}$. Each sample in the batch is uniformly sampled from these 10^6 data samples.

To reduce transformer inference costs during the mixed training phase, we reset the number of training sequences to $n = 15 \times 64$ per iteration and set the ratio $\kappa = 1/3$, meaning 10×64 samples are from $\mathcal{P}_{\gamma, \text{TF}_{\theta_m}}$ and 5×64 samples are from $\mathcal{P}_{\gamma, \tilde{f}}$ (sampled from the pool of pre-generated 10^6 samples as before). Consequently, we adjust the number of batches from 1500 to 50 to fit the smaller size of training samples, keeping the batch size at 64.

Decision Function $\tilde{f}$ in Pre-training. To mitigate the OOD issue mentioned in Section C.2, we aim for the decision function $\tilde{f}$ to approximate Alg^* or $\text{TF}_{\hat{\theta}}$. However, due to the high computation cost of Alg^* and the unavailability of $\text{TF}_{\hat{\theta}}$, we set

$$\tilde{f}(H_t) = a_t^* + \epsilon'_t \quad (\text{C.2})$$

where ϵ'_t is random noise simulating the suboptimality of Alg^* or $\text{TF}_{\hat{\theta}}$ and independent of H_t . As we expect such suboptimality to decrease across t , we also reduce the influence of ϵ'_t across t . Thus, in our experiments, ϵ'_t is defined as:

$$\epsilon'_t \sim \begin{cases} 0 & \text{w.p. } \max\{0, 1 - \frac{2}{\sqrt{t}}\}, \\ \text{Unif}[-1, 1] & \text{w.p. } \min\{1, \frac{2}{\sqrt{t}}\}, \end{cases}$$

where for discrete action spaces we replace $\text{Unif}[-1, 1]$ by $\text{Unif}\{-2, -1, 1, 2\}$ and for continuous discrete action spaces with dimension $d > 1$ we apply the uniform distribution of the set $[-1, -1]^d$. We further project $\tilde{f}(H_t)$ into $\mathcal{A}$ when $\tilde{f}(H_t) \notin \mathcal{A}$.

Curriculum. Inspired by [GTLV22] in the regression task, we apply curriculum training to potentially speed up Algorithm 3. This technique uses “simple” task data initially and gradually increases task complexity during the training. Specifically, we train the transformer on samples with a smaller horizon $\tilde{T} = 20$ (generated by truncating samples from $\mathcal{P}_{\gamma, \tilde{f}}$ or $\mathcal{P}_{\gamma, \text{TF}_{\theta_m}}$) at the

beginning and gradually increase the sample horizon to the target $T = 100$. We apply this curriculum in both the early training phase ($m \leq 50$) and the mixed training phase ($m > 50$). The last 30 iterations focus on training with $\tilde{T} = 100$ using non-truncated samples from $\mathcal{P}_{\gamma, \text{TF}_\theta}$ to let the transformer fit more on the non-truncated samples. The exact setting of $\tilde{T}$ is as follows:

$$\tilde{T} = \begin{cases} 20 \times (m \% 10 + 1) & \text{when } m = 1, \dots, 50, \\ 20 \times (m \% 10 - 4) & \text{when } m = 51, \dots, 100, \\ 100 & \text{when } m = 100, \dots, 130. \end{cases}$$

C.2.3 More Experiments on Pre-training

In this section, we present ablation studies to explore the impact of two key factors: the decision function $\tilde{f}$ used for generating pre-training data, and the mix ratio κ applied in Algorithm 3. Figure C.4 summarizes the results, where we pre-train and test models in multi-armed bandits problems defined in Appendix C.4.2.

- **Effect of $\tilde{f}$:** We evaluate the influence of different decision functions $\tilde{f}$ on the performance of OMGPT in a multi-armed bandit task. Three types of decision functions are considered: the one we designed, as defined in (C.2), the UCB algorithm for multi-armed bandits [LS20], and a 50/50 mixture of both. The training dynamics for each function are shown in Figure C.4a. We observe that all three approaches achieve similar final performance once training ends. While it is possible that some decision functions could lead to faster convergence or better performance after pre-training, we did not encounter any training failures due to the choice of $\tilde{f}$.
- **Effect of κ :** We also conduct an experiment to numerically investigate the impact of different mix ratios κ . The results are presented in Figure C.4b. We test four values of κ (10%, 33%, 66%, 90%), keeping all other hyperparameters constant for a multi-armed bandit task. The findings show that OMGPT’s performance remains robust across different values of κ , except when $\kappa = 0.9$, where performance declines. This supports our earlier

discussion in Algorithm 3, highlighting the importance of mixing a non-trivial amount of self-generated data (i.e., non-trivial value for $1 - \kappa$).

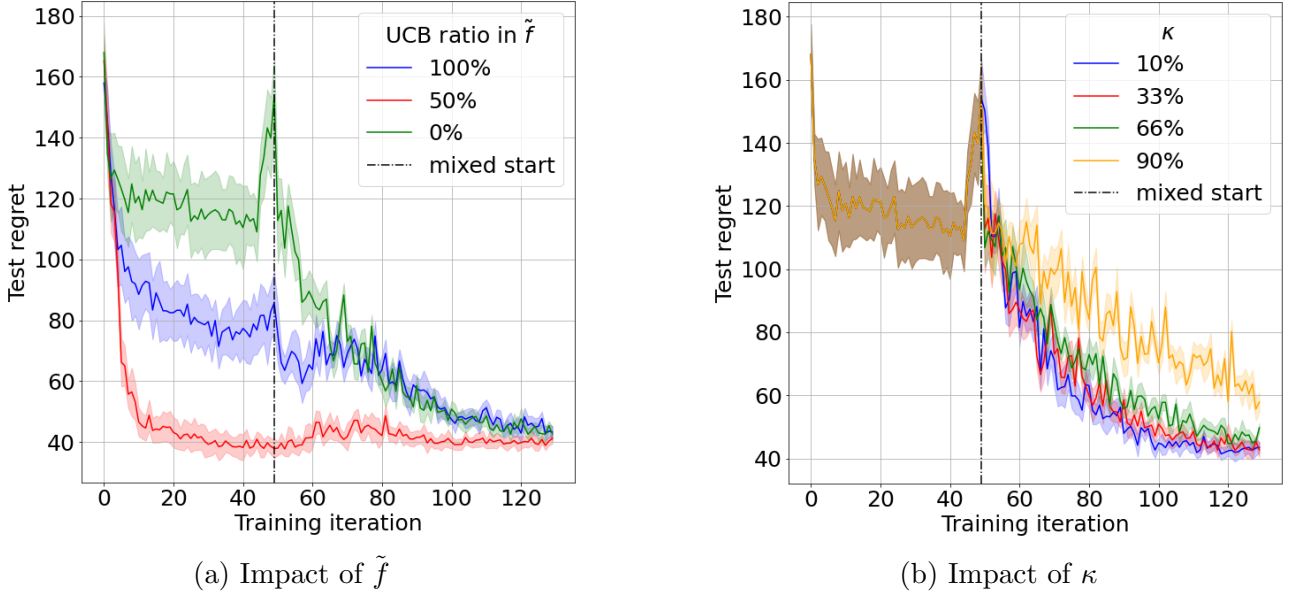


Figure C.4: (a) The effect of $\tilde{f}$ (decision function for generating the pre-training data), where we mix the UCB algorithm [LS20] and decision function (C.2) with different ratios to create the pre-training data. (b) The effect of κ (ratio of samples generated by $\tilde{f}$ in the mixed training phase).

C.3 Proofs for Section 4.5

C.3.1 Proofs for Subsection 4.5.1

Proof of Proposition 1

Proof. The proof can be done with the definition of $\mathbf{Alg}^*$. Specifically, note

$$\mathbb{E} \left[\sum_{t=1}^T l(f(H_t), a_t^*) \right] = \mathbb{E} \left[\sum_{t=1}^T \mathbb{E} [l(f(H_t), a_t^*) | H_t] \right] \geq \mathbb{E} \left[\sum_{t=1}^T \mathbb{E} [l(\mathbf{Alg}^*(H_t), a_t^*) | H_t] \right]$$

for any $f \in \mathcal{F}$, we finish the proof. $\square$

Proof of Corollary 3

Proof. Since Alg^* is a function of any possible history H_t , we omit the subscript t for the optimal actions for notation simplicity, and use $a^{(\gamma)*}$ as the optimal action of γ .

- **Posterior sampling under the cross-entropy loss.** We assume that $\mathcal{A} = \{a_1, \dots, a_J\}$ and the output of Alg^* is a probability vector $(p_1, \dots, p_J)$ such that $\sum_{j=1}^J p_j = 1$, meaning the probability of choosing each action. Then

$$\text{Alg}^*(H) = \arg \min_{\sum_{j=1}^J p_j = 1} \int_{\gamma} -\log(p_{a^{(\gamma)*}}) \mathbb{P}(H|\gamma) d\mathcal{P}_{\gamma} = \arg \min_{\sum_{j=1}^J p_j = 1} \sum_{j=1}^J (-\log(p_j)) \int_{\{\gamma: a^{(\gamma)*}=j\}} \mathbb{P}(H|\gamma) d\mathcal{P}_{\gamma}.$$

By noting $\int_{\{\gamma: a^{(\gamma)*}=j\}} \mathbb{P}(H|\gamma) d\mathcal{P}_{\gamma}$ is the probability such that action j is the optimal action conditional on H (i.e., the posterior distribution of the optimal action), we are indeed minimizing the cross-entropy of the posterior distribution of the optimal action relative to the decision variables p_j . Thus, the optimal solution $p_j^* = \int_{\{\gamma: a^{(\gamma)*}=j\}} \mathbb{P}(H|\gamma) d\mathcal{P}_{\gamma}$ for each j and $\text{Alg}^*(H_t)$ behaves as the posterior sampling.

- **Posterior averaging under the squared loss.** By definition,

$$\begin{aligned} \text{Alg}^*(H) &= \arg \min_{a \in \mathcal{A}} \int_{\gamma} \|a - a^{(\gamma)*}\|_2^2 \mathbb{P}(H|\gamma) d\mathcal{P}_{\gamma} \\ &= \frac{\int_{\gamma} a^{(\gamma)*} \mathbb{P}(H|\gamma) d\mathcal{P}_{\gamma}}{\int_{\gamma} \mathbb{P}(H|\gamma) d\mathcal{P}_{\gamma}} \\ &= \mathbb{E}_{\gamma}[a^{(\gamma)*} | H] \end{aligned}$$

where the second line is by the first order condition.

- **Posterior median under the absolute loss.** By definition,

$$\text{Alg}^*(H) = \arg \min_{a \in \mathcal{A}} \int_{\gamma} |a - a^{(\gamma)*}| \mathbb{P}(H|\gamma) d\mathcal{P}_{\gamma}.$$

Then by zero-subgradient condition, we can conclude that $\text{Alg}^*(H)$ satisfies

$$\int_{\{\gamma: \text{Alg}^*(H) \leq a^{(\gamma)*}\}} \mathbb{P}(H|\gamma) d\mathcal{P}_{\gamma} = \int_{\{\gamma: \text{Alg}^*(H) \geq a^{(\gamma)*}\}} \mathbb{P}(H|\gamma) d\mathcal{P}_{\gamma}.$$

Divide both sides of this equation by $\int_{\gamma} \mathbb{P}(H|\gamma) d\mathcal{P}_{\gamma}$, we can conclude that

$$\int_{\{\gamma: \text{Alg}^*(H) \leq a^{(\gamma)*}\}} d\mathcal{P}(\gamma|H) = \int_{\{\gamma: \text{Alg}^*(H) \geq a^{(\gamma)*}\}} d\mathcal{P}(\gamma|H)$$

where $\mathcal{P}(\gamma|H_t)$ is the posterior distribution of γ . Hence $\text{Alg}^*(H)$ is the posterior median.

□

C.3.2 Proof of Proposition 2

To prove Proposition 2, we assume the following mild conditions. First, for all these problems, the noise random variable ϵ_t has mean 0, is i.i.d. across time, and is independent of the action a_t . Second, for the linear bandits problem, the L_{∞} norm of the unknown parameter w is bounded by D . Third, for the dynamic pricing problem, we assume the demand function is differentiable with respect to the action coordinate, and both the absolute value of the first- and second-order derivatives is bounded by D . Further details regarding the problem formulations can be found in Appendix C.4.2.

Proof. Since both the regret $\text{Regret}(f; \gamma)$ and the prediction error $L(f; \gamma)$ are the sums of single-step regret and prediction error over the horizon, it is sufficient to prove the single-step surrogate property of the loss function. Therefore, in the following proof, we will focus on proving the single-step surrogate property and omit the subscript t for simplicity.

- For the multi-arm bandits problem, without loss of generality, we assume $\mathcal{A} = \{a_1, \dots, a_J\}$ and the optimal arm is a_1 . For $1 \leq i \leq J$, let μ_j be the mean of the reward distribution for arm j , and define $\Delta_j = \mu_1 - \mu_j$ as the reward gap for arm j . Define $\Delta_{\max} = \max\{\Delta_1, \dots, \Delta_J\}$ as the maximum action gap. Under cross-entropy loss, we denote the output distribution of decision function f is $(p_1, \dots, p_J)$. Then the single-step regret can

be bounded by:

$$\mathbb{E}[r(X, a^*) - r(X, f(H)) | H] = \mu_1 - \sum_{j=1}^J p_j \mu_j = \sum_{j=2}^J p_j \Delta_j \leq \Delta_{\max}(1 - p_1) \leq -\Delta_{\max} \cdot \log(p_1),$$

where the last step is by the inequality $1 - x \leq -\log(x)$ for $0 < x \leq 1$. Then by the definition of cross-entropy loss, we finish the proof.

- For the dynamic pricing problem, we have

$$r(X, a^*) - r(X, f(H)) = \nabla_a r(X, a^*)(a^* - f(H)) + \nabla_a^2 r(X, \tilde{a})|f(H) - a^*|^2 \leq D|f(H) - a^*|^2$$

where $\tilde{a}$ is in the line of $f(H)$ and a^* . Here, the first step is by the Taylor expansion, and the second step is by the first order condition of a^* and the assumption. Then by noting $|f(H) - a^*|^2$ is the square loss, we finish the proof.

- For the linear bandits problem, we have

$$r(X, a^*) - r(X, f(H)) = w^\top a^* - w^\top f(H) \leq \|w\|_\infty \|f(H) - a^*\|_1 \leq D\|f(H) - a^*\|_1,$$

where the second step is by the Holder's inequality and the last step is by the assumption. Then by noting $\|f(H) - a^*\|_1$ is the absolute loss, we finish the proof.

- For the newsvendor problem, we have

$$\begin{aligned} r(X, a^*) - r(X, f(H)) &\leq \max\{h, l\} \mathbb{E}_X [|a^* - d(X) - f(H) + d(X)|] \\ &= \max\{h, l\} |a^* - f(H)| \end{aligned}$$

where the first line is because the (random) reward function is $\max\{h, l\}$ -Lipschitz in $a - d(X)$. Then by noting $|f(H) - a^*|$ is the absolute loss, we finish the proof.

□

C.3.3 Proof of Proposition 3

Proof. **For the linear bandits problem**, we consider the following example. Suppose we have two environments, γ_1, γ_2 , each with standard normal distributed noise and unknown parameters $w_{\gamma_1} = (1, 0)$ and $w_{\gamma_2} = (0, 1)$, respectively. Let the action space be $\mathcal{A} = [-1, 1] \times [-1, 1]$. The optimal actions for the two environments are $a^{(\gamma_1)*} = (1, 0)$, $a^{(\gamma_2)*} = (0, 1)$. We assume the prior distribution $\mathcal{P}_\gamma$ is given by $\mathcal{P}_\gamma(\gamma_1) = \mathcal{P}_\gamma(\gamma_2) = \frac{1}{2}$.

Then if Alg^* is the posterior averaging, we have

$$a_t = \text{Alg}^*(H_t) = \mathcal{P}_\gamma(\gamma_1|H_t) \cdot a^{(\gamma_1)*} + \mathcal{P}_\gamma(\gamma_2|H_t) \cdot a^{(\gamma_2)*},$$

where the posterior distribution $\mathcal{P}_\gamma(\gamma_i|H_t)$ is given by

$$\mathcal{P}_\gamma(\gamma_i|H_t) = \frac{\prod_{\tau=1}^{t-1} \mathbb{P}_{\gamma_i}(O_\tau|a_\tau)}{\sum_{i'=1}^2 \prod_{\tau=1}^{t-1} \mathbb{P}_{\gamma_{i'}}(O_\tau|a_\tau)}, \quad i = 1, 2.$$

We now use induction to show $\mathcal{P}_\gamma(\gamma_i|H_t) = \frac{1}{2}$ for any $t \geq 1$ and $i = 1, 2$, where H_t can be generated by either γ_1 or γ_2 . This results in $a_t = \frac{1}{2}a^{(\gamma_1)*} + \frac{1}{2}a^{(\gamma_2)*} = (\frac{1}{2}, \frac{1}{2})$, which does not change with respect to t and will cause regret linear in T .

Step 1. Since $\mathcal{P}_\gamma(\gamma_i|H_0) = \mathcal{P}_\gamma(\gamma_i) = \frac{1}{2}$, we have $a_1 = \frac{1}{2}a^{(\gamma_1)*} + \frac{1}{2}a^{(\gamma_2)*} = (\frac{1}{2}, \frac{1}{2})$, and the conclusion holds for $t = 1$.

Step 2. Now assume the conclusion holds for t , i.e., $\mathcal{P}_\gamma(\gamma_i|H_t) = \frac{1}{2}$, and $a_t = (\frac{1}{2}, \frac{1}{2})$. Since $w_{\gamma_1}^T a_t = w_{\gamma_2}^T a_t = \frac{1}{2}$, we have $\mathbb{P}_{\gamma_i}(O_t|a_t) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(O_t - \frac{1}{2})^2}{2}\right)$ for $i = 1, 2$. Observe that

$$\mathcal{P}_\gamma(\gamma_i|H_{t+1}) = \frac{\mathbb{P}_{\gamma_i}(O_{t+1}|a_{t+1})\mathcal{P}_\gamma(\gamma_i|H_t)}{\sum_{i'=1}^2 \mathbb{P}_{\gamma_{i'}}(O_{t+1}|a_{t+1})\mathcal{P}_\gamma(\gamma_{i'}|H_t)} = \frac{1}{2},$$

which implies $a_{t+1} = (\frac{1}{2}, \frac{1}{2})$, and the conclusion holds for $t + 1$.

Thus, the conclusion holds for all $t \geq 1$. Then the regret is

$$\text{Regret}(\mathbf{Alg}^*; \gamma_i) = \mathbb{E} \left[\sum_{t=1}^T r(X_t, a_t^*) - r(X_t, a_t) \right] = \frac{1}{2}T$$

for $i = 1, 2$.

For the dynamic pricing problem, we can construct a similar example. Suppose we have two environments without context X_t , denoted by γ_1, γ_2 . The demands O_t of them are set to be $O_t = 2 - a_t + \epsilon_t$ and $O_t = \frac{4}{5} - \frac{1}{5} \cdot a_t + \epsilon_t$ respectively, where $\epsilon_t \stackrel{i.i.d.}{\sim} \mathcal{N}(0, 1)$ and a_t is the price. Accordingly, the optimal actions are then $a^{(\gamma_1)*} = 1$ and $a^{(\gamma_2)*} = 2$.

Then if $\mathbf{Alg}^*$ is the posterior averaging, we have

$$a_t = \mathbf{Alg}^*(H_t) = \mathcal{P}_\gamma(\gamma_1|H_t) \cdot a^{(\gamma_1)*} + \mathcal{P}_\gamma(\gamma_2|H_t) \cdot a^{(\gamma_2)*},$$

where the posterior distribution $\mathcal{P}_\gamma(\gamma_i|H_t)$ is given by

$$\mathcal{P}_\gamma(\gamma_i|H_t) = \frac{\prod_{\tau=1}^{t-1} \mathbb{P}_{\gamma_i}(O_\tau|a_\tau)}{\sum_{i'=1}^2 \prod_{\tau=1}^{t-1} \mathbb{P}_{\gamma_{i'}}(O_\tau|a_\tau)}, \quad i = 1, 2.$$

with $\mathbb{P}_{\gamma_1}(O_\tau|a_\tau)$ as the normal distribution $\mathcal{N}(2 - a_\tau, 1)$ and $\mathbb{P}_{\gamma_2}(O_\tau|a_\tau)$ as $\mathcal{N}(\frac{4-a_\tau}{5}, 1)$.

Observe that $a_1 = \frac{1}{2}a^{(\gamma_1)*} + \frac{1}{2}a^{(\gamma_2)*} = \frac{3}{2}$ satisfies $2 - a_1 = \frac{4-a_1}{5}$. Following a similar analysis as in the linear bandits example, we can conclude that $\mathcal{P}_\gamma(\gamma_i|H_t) = \frac{1}{2}$ for any t and $i = 1, 2$, where H_t can be generated by either γ_1 or γ_2 . Therefore, $a_t = \frac{3}{2}$ for all $t \geq 1$ and the regret is

$$\text{Regret}(\mathbf{Alg}^*; \gamma_i) = \mathbb{E} \left[\sum_{t=1}^T r(X_t, a_t^*) - r(X_t, a_t) \right] = \frac{1}{4}T \cdot \mathbb{I}\{\gamma_i = \gamma_1\} + \frac{1}{20}T \cdot \mathbb{I}\{\gamma_i = \gamma_2\},$$

for $i = 1, 2$.

□

C.3.4 Proof and Discussions of Theorem 4.2

We first introduce a few key lemmas for proving Theorem 4.2. We denote $\mathcal{P}_t(\cdot) = \mathcal{P}_\gamma(\cdot|H_t)$, i.e., the posterior distribution of the (testing) environment at time t , which generates H_t through (4.12), conditional on the history H_t . And let $\mathbb{E}_t[\cdot]$ denote the corresponding conditional expectation. Then under Assumption 1, 2, 3, we have the following lemmas. Their proofs can be found in the end of this subsection.

Lemma 6 (Concentration Inequality). *Recall $\lambda_t = \lambda_{\min}(\sum_{\tau=1}^t \Delta(X_\tau, a_\tau))$ is the minimum eigenvalue of the matrix $\sum_{\tau=1}^t \Delta(X_\tau, a_\tau)$, and $\gamma \in \Gamma$ is the environment to generate O_t and X_t through (4.1). Then given $t > 0$, $\gamma' \in \Gamma$ and $1 > \delta > 0$, with probability $1 - \delta$,*

$$\begin{aligned} \sum_{\tau=1}^t \log \left(\frac{\mathbb{P}_\gamma(O_\tau|X_\tau, a_\tau)}{\mathbb{P}_{\gamma'}(O_\tau|X_\tau, a_\tau)} \right) &\geq \frac{1}{2} \sum_{\tau=1}^t \|\gamma - \gamma'\|_{\Delta(X_\tau, a_\tau)}^2 - 4C_{\sigma^2} \log(1/\delta) \\ &\geq \frac{1}{2} \lambda_t \|\gamma - \gamma'\|_2^2 - 4C_{\sigma^2} \log(1/\delta), \end{aligned}$$

for any possible $a'_\tau \in \mathcal{A}$.

This lemma provides a concentration inequality for the log-likelihood ratio between the true environment and any other environment, in terms of the “exploration intensity” λ_t and the distance between the two environments. The proof is based on the standard martingale concentration inequality, along with some algebraic manipulations.

Lemma 7. *For any $\gamma_1, \gamma_2, \gamma' \in \Gamma$ such that $\gamma_1 + \gamma', \gamma_2 + \gamma' \in \Gamma$, we have*

$$\left| \log \left(\frac{\mathbb{P}_{\gamma_1+\gamma'}(O|X, a)}{\mathbb{P}_{\gamma_1}(O|X, a)} \right) - \log \left(\frac{\mathbb{P}_{\gamma_2+\gamma'}(O|X, a)}{\mathbb{P}_{\gamma_2}(O|X, a)} \right) \right| \leq C_\Gamma \|\gamma'\|_2 (\|\gamma'\|_2 + \|\gamma_1 - \gamma_2\|_2),$$

for any $X \in \mathcal{X}$, $a \in \mathcal{A}$ and $O \in \mathcal{O}$.

This lemma bounds the difference in log-likelihood changes between two environments, γ_1 and γ_2 , when both are shifted by γ' . On the one hand, this difference is affected by the shift γ' , where a smaller shift results in a smaller difference. On the other hand, it is also influenced by

the distance between the two environments. The proof is based on the smoothness assumed in Assumption 2.

C.3.4.1 Proof of Theorem 4.2

Proof. Recall $\mathbb{E}_t[\cdot] = \mathbb{E}_{\gamma'}[\cdot|H_t]$ is the expectation with respect to the posterior distribution over the possible (testing) environment γ' conditional on H_t . The regret can be decomposed as the following

$$\begin{aligned}
\text{Regret}(\text{TF}_{\hat{\theta}}; \gamma) &= \sum_{t=1}^T \mathbb{E}[r(X_t, \text{Alg}^*(H_t)) - r(X_t, \text{TF}_{\hat{\theta}}(H_t))] + \sum_{t=1}^T \mathbb{E}[r(X_t, a_t^*) - r(X_t, \text{Alg}^*(H_t))] \\
&\leq \Delta_{\text{Exploit}}T + \sum_{t=1}^T \mathbb{E}[r(X_t, a_t^*) - r(X_t, \text{Alg}^*(H_t))] \\
&\leq \Delta_{\text{Exploit}}T + \bar{r}\mathbb{E}[t_0 - 1] + \mathbb{E}\left[\sum_{t=t_0}^T C_r \mathbb{E}_t[\|\gamma - \gamma'\|_2^2]\right] \\
&= \Delta_{\text{Exploit}}T + \bar{r}\mathbb{E}[t_0 - 1] \\
&\quad + \mathbb{E}\left[\sum_{t=t_0}^T C_r \mathbb{E}_t\left[\|\gamma - \gamma'\|_2^2 \mathbb{1}_{\{\|\gamma - \gamma'\|_2^2 < \epsilon_t\}}\right] + \sum_{t=t_0}^T C_r \mathbb{E}_t\left[\|\gamma - \gamma'\|_2^2 \mathbb{1}_{\{\|\gamma - \gamma'\|_2^2 \geq \epsilon_t\}}\right]\right] \\
&\leq \Delta_{\text{Exploit}}T + \bar{r}\mathbb{E}[t_0 - 1] + C_r \mathbb{E}\left[\sum_{t=t_0}^T \epsilon_t\right] + \mathbb{E}\left[\sum_{t=t_0}^T C_r \mathbb{E}_t[\|\gamma - \gamma'\|_2^2 \mathbb{1}_{\{\|\gamma - \gamma'\|_2^2 \geq \epsilon_t\}}]\right]
\end{aligned}$$

where in the last two lines we use a sequence $\{\epsilon_t\}_{t=t_0}^T$ (which is defined in Theorem 4.2) to decompose $\|\gamma - \gamma'\|_2^2$ into $\|\gamma - \gamma'\|_2^2 \mathbb{1}_{\{\|\gamma - \gamma'\|_2^2 < \epsilon_t\}}$ and $\|\gamma - \gamma'\|_2^2 \mathbb{1}_{\{\|\gamma - \gamma'\|_2^2 \geq \epsilon_t\}}$ at each $t \geq t_0$. The first term is automatically bounded by ϵ_t . Thanks to the concentration inequality in Lemma 6, for the second term, the tail event $\|\gamma - \gamma'\|_2^2 \geq \epsilon$ will have a probability exponentially decreasing in $\epsilon \geq \epsilon_t$, which is shown in Lemma 8 below. And the second term, as its expectation, can be accordingly bounded by Corollary 5.

Lemma 8. *Conditional on $t \geq t_0$, for $\epsilon \geq \epsilon_t$ and $1 > \delta > 0$, with probability $1 - \delta$,*

$$\mathcal{P}_t(\|\gamma - \gamma'\|_2^2 \geq \epsilon) \leq \frac{K_t}{K_t + C_t(\epsilon, \delta)}$$

where

$$C_t(\epsilon, \delta) = \exp \left(-4C_{\sigma^2} \log(K_t/\delta) + \frac{1}{16} \lambda_{t-1} \epsilon \right),$$

and K_t is defined in Theorem 4.2 and is an upper bound of the covering number of Γ (with a radius as a function of ϵ_t and λ_{t-1} and L_2 norm).

Corollary 5. Conditional on $t \geq t_0$, for $1 > \delta > 0$,

$$\mathbb{E} \left[\mathbb{E}_t \left[\|\gamma - \gamma'\|_2^2 \mathbb{1}_{\{\|\gamma - \gamma'\|_2^2 \geq \epsilon_t\}} \right] \right] \leq \mathbb{E} \left[\frac{16}{\lambda_{t-1}} \log(2K_t(K_t/\delta)^{4C_{\sigma^2}}) \right] + 4\bar{\gamma}^2 \delta,$$

where the expectation on the right-hand-side is with respect to the potential randomness in λ_{t-1} and K_t is defined in Theorem 4.2.

By combining all together and choosing $\delta = 1/T$, we have

$$\text{Regret}(\text{TF}_{\hat{\theta}}; \gamma) \leq \Delta_{\text{Exploit}} T + \bar{r} \mathbb{E}[t_0 - 1] + C_r \mathbb{E} \left[\sum_{t=t_0}^T \epsilon_t \right] + 16C_r(4C_{\sigma^2} + 1) \mathbb{E} \left[\sum_{t=t_0}^T \frac{\log(2K_t T)}{\lambda_{t-1}} \right] + 4C_r \bar{\gamma}^2.$$

□

Discussions on Technique Contributions The key challenge in the proof is to uniformly upper bound the likelihood ratio $\frac{p_t(\gamma')}{p_t(\gamma)}$ for all $\gamma' \in \Gamma$. However, the commonly used covering number technique from the machine learning literature, which creates covering balls to discretize and approximate continuous spaces by their centers, encounters several issues when directly applied here. Thus, we developed new techniques for the proof of Theorem 4.2. Specifically:

- **Dependence of the covering set on γ :** In traditional covering techniques, the covering set (or sets, as in Dudley's theorem, which will be discussed later) is constructed independently of γ . In this case, γ' and γ could fall within the same ball, and approximating both with the same center could yield an upper bound for $\frac{p_t(\gamma')}{p_t(\gamma)}$ of 1, which is meaningless. The failure arises because traditional covering techniques focus on a single point (e.g., γ), while we are interested in two points, γ' and γ . In our proof, we construct covering sets that depend on the underlying environment γ , allowing us to isolate it from other balls.

- **Larger ball for γ :** In traditional covering techniques, each ball has the same radius, and this radius is used to uniformly bound the approximation error across all balls when using their centers to approximate any point within them. However, in our case, due to the γ -dependent covering set, the ball centered at γ may overlap with other balls (e.g., $B(\tilde{\gamma}, \epsilon)$), which could lead to large approximation errors. This is because points in $B(\tilde{\gamma}, \epsilon)$ could be arbitrarily close to γ , resulting in a likelihood ratio of 1, which could be far from the approximation $\frac{p_t(\tilde{\gamma})}{p_t(\gamma)}$. To address this, we adjust the boundaries of γ 's ball, ensuring that points from other balls are sufficiently distant from γ .
- **Our “chaining trick” differs from Dudley’s theorem:** Dudley’s theorem (also known as chaining, see Lemma 27.4 of [SSBD14]) is typically used to sharpen bounds, such as the uniform bound of $\frac{p_t(\gamma')}{p_t(\gamma)}$, by constructing a sequence of covering sets with decreasing radii. However, as mentioned earlier, Dudley’s covering sets are independent of γ , whereas our approach requires the dependence. Our strategy is to reformulate the desired result as $\mathbb{E}_t [\|\gamma - \gamma'\|_2^2] = \int_{\epsilon > 0} \mathcal{P}_t(\|\gamma - \gamma'\|_2^2 \geq \epsilon) d\epsilon$ and construct a sequence of covering sets related to ϵ . This technique is inspired by [KZ14], which bounds the estimation error of an estimated optimal action rather than the likelihood ratio as we do.

C.3.4.2 Proof of Example 1

Proof. We first assume there exists constants $\bar{\gamma}, \bar{X}, \underline{a}, \bar{a} > 0$ such that $\Gamma = \{\gamma \in \mathbb{R}^{2d} \mid \|\gamma\|_2 \leq \bar{\gamma}\}$, $\mathcal{X} = \{X \in \mathbb{R}^d \mid \|X\|_2 \leq \bar{X}\}$, $\mathcal{A} = [\underline{a}, \bar{a}]$, and $\mathcal{P}_\gamma$ is a uniform distribution in Γ . We further assume the minimum eigenvalue of the context’s covariance matrix is lower bounded by $\underline{\lambda} > 0$. Then we verify the assumptions in Theorem 4.2 one by one. For notation simplicity, we denote $Z = (X, a \cdot X)^\top$.

Assumption 1. This is automatically satisfied by the above assumptions.

Assumption 2. We define $\Delta(X, a) = \frac{1}{2}ZZ^\top$. Then we have

$$\begin{aligned} \log \left(\frac{\mathbb{P}_\gamma(O_t|X, a)}{\mathbb{P}_{\gamma'}(O_t|X, a)} \right) &= \frac{-(D_t - \alpha^\top X - \beta^\top X \cdot a)^2 + (D_t - \alpha'^\top X - \beta'^\top X \cdot a)^2}{2} \\ &= \|\gamma - \gamma'\|_{\Delta(X, a)}^2 - (\gamma - \gamma')^\top Z \cdot \epsilon_t, \end{aligned}$$

where $\gamma' = (\alpha', \beta')$. Then

- Since ϵ_t has 0 mean and is independent from Z , we have

$$\text{KL}(\mathbb{P}_\gamma(\cdot|X, a)|\mathbb{P}_{\gamma'}(\cdot|X, a)) = \mathbb{E}_{\epsilon_t} [\|\gamma - \gamma'\|_{\Delta(X, a)}^2 - (\gamma - \gamma')^\top Z \cdot \epsilon_t] = \|\gamma - \gamma'\|_{\Delta(X, a)}^2$$

- For the term $(\gamma - \gamma')^\top Z \cdot \epsilon_t$, it follows a Normal distribution with variance $((\gamma - \gamma')^\top Z)^2 = \|\gamma - \gamma'\|_{ZZ^\top}^2 = 2\|\gamma - \gamma'\|_{\Delta(X, a)}^2$. Thus, we have $C_{\sigma^2} = \sqrt{2}$.
- $\log \mathbb{P}_{\gamma'}(O|X, a) = \frac{-(D_t - \alpha'^\top X - \beta'^\top X \cdot a)^2}{2}$ which is concave in $\gamma' \in \Gamma$. Further, we have

$$\|\nabla_{\gamma'}^2 \log \mathbb{P}_{\gamma'}(O|X, a)\|_2^2 = \|ZZ^\top\|_2^2 = \|Z\|_2^4 \leq (1 + \bar{a}^2)^2 \bar{X}^4$$

due to the boundedness assumptions on X and a . Thus, we can assign $C_\Gamma = (1 + \bar{a}^2)^2 \bar{X}^4$.

Assumption 3. For notation simplicity, we denote $\text{Alg}^*(H_t) = \tilde{a}_t^*$

- By assumptions,

$$\begin{aligned} &\mathbb{E} [r(X_t, \text{Alg}^*(H_t)) - r(X_t, \text{TF}_{\hat{\theta}}(H_t)|H_t)] \\ &= \mathbb{E} [\tilde{a}_t^*(\alpha^\top X_t + \beta^\top X_t \cdot \tilde{a}_t^*) - (\tilde{a}_t^* + \Delta_t)(\alpha^\top X_t + \beta^\top X_t \cdot (\tilde{a}_t^* + \Delta_t)|H_t)] \\ &= \mathbb{E} [-\Delta_t(\alpha^\top X_t + 2\beta^\top X_t \cdot \tilde{a}_t^*) - \beta^\top X_t \Delta_t^2|H_t] \\ &= -\mathbb{E} [\beta^\top X_t \Delta_t^2|H_t] \\ &\leq \frac{\bar{X}\bar{\gamma}C^2}{\sqrt{T}} \end{aligned}$$

where the second last line is because Δ_t is zero mean and is independent of $H_t, \tilde{a}_t^*$ and γ , and the last line is because of the boundedness assumptions. Thus, we can assign $\Delta_{\text{Exploit}} = \frac{\bar{X}\bar{\gamma}C^2}{\sqrt{T}}$.

- By the proof of Theorem 2 in [BK21] and Jensen's inequality, there exists a constant $C_0 > 0$, which depends on the boundedness assumptions, such that

$$\begin{aligned}\mathbb{E}[r(X_t, a_t^*) - r(X_t, \mathbf{Alg}^*(H_t)) | H_t] &= -\beta^\top X_t \mathbb{E}[(a_t^* - \tilde{a}_t^*)^2 | H_t] \\ &\leq C_0 \bar{\gamma} \bar{X} \mathbb{E}_{\gamma'}[\|\gamma - \gamma'\|_2^2 | H_t]\end{aligned}$$

where the last two lines are because of the boundedness assumption. Thus, we can assign $C_r = C_0 \bar{\gamma} \bar{X}$.

Therefore, all assumptions in Theorem 4.2 are satisfied. We now need to compute $\mathbb{E}[t_0 + \sum_{t=t_0}^T \frac{1}{\lambda_{t-1}}]$. We first bound the value of $\sum_{\tau=2}^t (a_\tau - \bar{a}_{\tau-1})^2$, where $\bar{a}_t = \frac{\sum_{\tau=1}^t a_\tau}{t}$ is the mean of actions/prices. This will later be shown to serve as a lower bound for λ_t with high probability.

Since $a_\tau = \mathbf{Alg}^*(H_\tau) + \Delta_\tau$, we have

$$\begin{aligned}\sum_{\tau=2}^t (a_\tau - \bar{a}_{\tau-1})^2 &= \sum_{\tau=2}^t (\mathbf{Alg}^*(H_\tau) + \Delta_\tau - \bar{a}_{\tau-1})^2 \\ &= \sum_{\tau=2}^t \Delta_\tau^2 + 2\Delta_\tau(\mathbf{Alg}^*(H_\tau) - \bar{a}_{\tau-1}) + (\mathbf{Alg}^*(H_\tau) - \bar{a}_{\tau-1})^2\end{aligned}$$

By assumptions, the sequence $\{\Delta_\tau(\mathbf{Alg}^*(H_\tau) - \bar{a}_{\tau-1})\}_{\tau=2}^T$ is a martingale difference sequence with respect to $\mathcal{F}_{\tau-1} = (\mathbf{Alg}^*(H_1), \Delta_1, \dots, \mathbf{Alg}^*(H_{\tau-1}), \Delta_{\tau-1}, \mathbf{Alg}^*(H_\tau))$. Thus, by Theorem 2.19 in [Wai19] and the union bound, we have that, with probability $1 - 1/T$, for all $t = 2, \dots, T$,

$$\sum_{\tau=2}^t \Delta_\tau(\mathbf{Alg}^*(H_\tau) - \bar{a}_{\tau-1}) \geq -\frac{4(\bar{a} + C) \log T \sqrt{t}}{T^{1/4}}.$$

Thus, with probability $1 - 1/T$, we have for all $t = 2, \dots, T$,

$$\begin{aligned} \sum_{\tau=2}^t (a_\tau - \bar{a}_{\tau-1})^2 &\geq \sum_{\tau=2}^t \Delta_\tau^2 + 2\Delta_\tau(\mathbf{Alg}^*(H_\tau) - \bar{a}_{\tau-1}) \\ &\geq \frac{t}{2C^2\sqrt{T}} - \frac{8(\bar{a} + C)\log T\sqrt{t}}{T^{1/4}}. \end{aligned}$$

And when $t \geq 512(\bar{a} + C)^2 C^4 \log^2 T \sqrt{T}$, we have $\sum_{\tau=2}^t (a_\tau - \bar{a}_{\tau-1})^2 \geq \frac{t}{2C^2\sqrt{T}} - \frac{8(\bar{a}+C)\log T\sqrt{t}}{T^{1/4}} \geq \frac{t}{4C^2\sqrt{T}}$. Thus, with probability $1 - 1/T$, we have $t_0 \leq 512(\bar{a} + C)^2 C^4 \log^2 T \sqrt{T}$.

Further, by Lemma 1 of [BK21] and the analysis in Example 1 of [KZ14], for each t , there exist constants $C_1, C_2 > 0$, related to the boundedness of the parameters, such that with probability at least $1 - d \exp(-C_1 \sum_{\tau=2}^t (a_\tau - \bar{a}_{\tau-1})^2)$,

$$\lambda_t \geq C_2 \sum_{\tau=2}^t (a_\tau - \bar{a}_{\tau-1})^2.$$

Thus, by the union bound, for all $t \geq \max\{512(\bar{a} + C)^2 C^4 \log^2 T \sqrt{T}, \frac{4C^2 \log(dT^2)\sqrt{T}}{C_1}\}$, we have $\lambda_t = \Omega(t/\sqrt{T})$ for all $t = 1, \dots, T$ with probability at least $1 - \frac{2}{T}$. With $\Delta_{\text{Exploit}} = O(1/\sqrt{T})$, we can complete the result by Theorem 4.2. $\square$

C.3.4.3 Proof of Theorem 4.3

Proof. For both of the two regret bounds, we first decompose the regret:

$$\begin{aligned} \text{Regret}(\mathbf{TF}_{\hat{\theta}}; \gamma) &= \sum_{t=1}^T \mathbb{E}[r(X_t, \mathbf{Alg}^*(H_t)) - r(X_t, \mathbf{TF}_{\hat{\theta}}(H_t))] + \sum_{t=1}^T \mathbb{E}[r(X_t, a_t^*) - r(X_t, \mathbf{Alg}^*(H_t))] \\ &\leq \Delta_{\text{Exploit}} T + C_r \mathbb{E} \left[\sum_{t=1}^T \sum_{\gamma_i \neq \gamma} \mathcal{P}(\gamma_i | H_t) \|\gamma_i - \gamma\|_2^2 \right], \end{aligned}$$

where the last line is by Assumption 3.

Now we apply the union bound and Lemma 6 to γ_i for $i = 1, \dots, n$ and for $t = 1, \dots, T$: with

probability $1 - 1/T$, for all $i = 1, \dots, n$, and $t = 1, \dots, T$

$$\mathcal{P}(\gamma_i|H_t) \leq \mathcal{P}(\gamma|H_t) \cdot \exp\left(-\frac{1}{2} \sum_{\tau=1}^{t-1} \|\gamma_i - \gamma\|_{\Delta(X_\tau, a_\tau)}^2 + 4C_{\sigma^2} \log(nT^2)\right).$$

Since $\mathcal{P}(\gamma_i|H_t) + \mathcal{P}(\gamma|H_t) \leq 1$, we have with probability $1 - 1/T$, for all $i = 1, \dots, n$, and $t = 1, \dots, T$

$$\mathcal{P}(\gamma_i|H_t) \leq \frac{1}{1 + \exp\left(\frac{1}{2} \sum_{\tau=1}^{t-1} \|\gamma_i - \gamma\|_{\Delta(X_\tau, a_\tau)}^2 - 4C_{\sigma^2} \log(nT^2)\right)}.$$

For the first regret bound, recall $t_i = \min\{t = 2, \dots, T \mid \sum_{\tau=1}^{t-1} \|\gamma_i - \gamma\|_{\Delta(X_\tau, a_\tau)}^2 \geq 16C_{\sigma^2} \log(nT^2)\}$, and thus for $i = 1, \dots, n$,

$$\begin{aligned} \mathbb{E}\left[\sum_{t=1}^T \mathcal{P}(\gamma_i|H_t) \|\gamma_i - \gamma\|_2^2\right] &\leq \bar{r} \sum_{\gamma_i \neq \gamma} \mathbb{E}[t_i - 1] + \sum_{\gamma_i \neq \gamma} \mathbb{E}\left[\sum_{t=t_i}^T \mathcal{P}(\gamma_i|H_t) \|\gamma_i - \gamma\|_2^2\right] \\ &\leq 4\bar{\gamma}^2 + \bar{r} \sum_{\gamma_i \neq \gamma} \mathbb{E}[t_i - 1] + \sum_{\gamma_i \neq \gamma} \mathbb{E}\left[\sum_{t=t_i}^T \frac{\|\gamma_i - \gamma\|_2^2}{1 + \exp\left(\frac{1}{4} \sum_{\tau=1}^{t-1} \|\gamma_i - \gamma\|_{\Delta(X_\tau, a_\tau)}^2\right)}\right] \\ &< 4\bar{\gamma}^2 + \bar{r} \sum_{\gamma_i \neq \gamma} \mathbb{E}[t_i - 1] + \sum_{\gamma_i \neq \gamma} \mathbb{E}\left[\sum_{t=t_i}^T \frac{4\|\gamma_i - \gamma\|_2^2}{\sum_{\tau=1}^{t-1} \|\gamma_i - \gamma\|_{\Delta(X_\tau, a_\tau)}^2}\right] \end{aligned}$$

where the second inequality is because $\|\gamma - \gamma_i\|_2^2$ is bounded by $4\bar{\gamma}^2$, and the last inequality is because $x < \exp(x) + 1$ for $x > 0$. Thus, we can conclude the result.

Now for the second regret bound, we think about two situations for each $i = 1, \dots, n$ and $t = t_0, \dots, T$:

- If $\lambda_{t-1} \|\gamma_i - \gamma\|_2^2 < 16C_{\sigma^2} \log(nT^2)$, then since $\mathcal{P}(\gamma_i|H_t) < 1$, we have

$$\mathcal{P}(\gamma_i|H_t) \|\gamma_i - \gamma\|_2^2 < \frac{16C_{\sigma^2} \log(nT^2)}{\lambda_{t-1}}.$$

- If $\lambda_{t-1}\|\gamma_i - \gamma\|_2^2 \geq 16C_{\sigma^2} \log(nT^2)$. Since $\exp(x) + 1 > x$ for $x > 0$, we have

$$\mathcal{P}(\gamma_i|H_t)\|\gamma_i - \gamma\|_2^2 < \frac{1}{1 + \exp\left(\frac{1}{4}\lambda_{t-1}\|\gamma_i - \gamma\|_2^2\right)}\|\gamma_i - \gamma\|_2^2 < \frac{4}{\lambda_{t-1}}.$$

Thus, we can conclude that

$$\mathbb{E} \left[\sum_{t=1}^T \sum_{\gamma_i \neq \gamma} \mathcal{P}(\gamma_i|H_t)\|\gamma_i - \gamma\|_2^2 \right] < 4\bar{\gamma}^2 + \bar{r}\mathbb{E}[t_0 - 1] + n\mathbb{E} \left[\sum_{t=t_0}^T \frac{16C_{\sigma^2} \log(nT^2) + 4}{\lambda_{t-1}} \right]$$

and finish the second regret bound.

□

C.3.4.4 Proof of Example 2

Proof. We first assume there exists constants $\underline{a}, \bar{a} > 0$ such that $\mathcal{A} = [\underline{a}, \bar{a}]$ and the potential optimal actions $\frac{1}{2}, \frac{1}{4} \in \mathcal{A}$, and $\mathcal{P}_\gamma$ is a uniform distribution in Γ . Then by a similar analysis as in the proof of Example 1, all the assumptions in Theorem 4.3 can be verified, where $\Delta_{\text{Exploit}} = 0$, $C_{\sigma^2} = \sqrt{2}$, $\bar{\gamma} = \sqrt{5}$, $\bar{r} = \frac{1}{4}$, and $\Delta(X, a) = \begin{bmatrix} 1 & a \\ a & a^2 \end{bmatrix}$.

Without loss of generality, we assume $\gamma = \gamma_1 = (1, -1)$. Then, we have

$$\|\gamma_2 - \gamma\|_{\Delta(X, a)}^2 = a^2 \geq \underline{a}^2$$

for all a . Thus, we have $t_2 \leq \frac{16\sqrt{2}\log(2T^2)}{\underline{a}^2}$, and for all $t = 2, \dots, T$,

$$\frac{4\|\gamma_2 - \gamma\|_2^2}{\sum_{\tau=1}^{t-1} \|\gamma_2 - \gamma\|_{\Delta(X_\tau, a_\tau)}^2} \leq \frac{4}{(t-1)\underline{a}^2}.$$

Therefore, by (4.16), we have

$$\begin{aligned}
\text{Regret}(\text{TF}_{\hat{\theta}}; \gamma) &< 4C_r \bar{\gamma}^2 + \Delta_{\text{Exploit}} T + \bar{r} C_r \sum_{\gamma_i \neq \gamma} \mathbb{E}[t_i - 1] + C_r \sum_{\gamma_i \neq \gamma} \mathbb{E} \left[\sum_{t=t_i}^T \frac{4\|\gamma_i - \gamma\|_2^2}{\sum_{\tau=1}^{t-1} \|\gamma_i - \gamma\|_{\Delta(X_\tau, a_\tau)}^2} \right] \\
&< 4\bar{\gamma}^2 + \frac{16\sqrt{2}C_r \bar{r} \log(2T^2)}{\underline{a}^2} + \sum_{t=2}^T \frac{4C_r}{(t-1)\underline{a}^2} \\
&= O(\log T)
\end{aligned}$$

which completes the proof. $\square$

C.3.4.5 Proofs of Lemmas

Proof for Lemma 6

Proof. By the Bernstein-type concentration bound for a martingale difference sequence (Theorem 2.19 in [Wai19]), under Assumption 2, we have for any $\gamma' \in \Gamma$, $t > 0$ and $1 > \delta > 0$ with probability $1 - \delta$,

$$\sum_{\tau=1}^t \log \left(\frac{\mathbb{P}_\gamma(O_\tau | X_\tau, a_\tau)}{\mathbb{P}_{\gamma'}(O_\tau | X_\tau, a_\tau)} \right) - \mathbb{E} \left[\log \left(\frac{\mathbb{P}_\gamma(O_\tau | X_\tau, a_\tau)}{\mathbb{P}_{\gamma'}(O_\tau | X_\tau, a_\tau)} \right) \right] \geq -\sqrt{2C_{\sigma^2} \sum_{\tau=1}^t \|\gamma - \gamma'\|_{\Delta(X_\tau, a_\tau)}^2 \log(1/\delta)},$$

where

$$\sum_{\tau=1}^t \mathbb{E} \left[\log \left(\frac{\mathbb{P}_\gamma(O_\tau | X_\tau, a_\tau)}{\mathbb{P}_{\gamma'}(O_\tau | X_\tau, a_\tau)} \right) \right] = \sum_{\tau=1}^t \text{KL}(\mathbb{P}_\gamma(\cdot | X_\tau, a_\tau) \| \mathbb{P}_{\gamma'}(\cdot | X_\tau, a_\tau)) \geq \sum_{\tau=1}^t \|\gamma - \gamma'\|_{\Delta(X_\tau, a_\tau)}^2.$$

We think about two situations:

- If $\sqrt{\sum_{\tau=1}^t \|\gamma - \gamma'\|_{\Delta(X_\tau, a_\tau)}^2} \geq 2\sqrt{2C_{\sigma^2} \log(1/\delta)}$, then

$$\begin{aligned}
\sum_{\tau=1}^t \log \left(\frac{\mathbb{P}_\gamma(O_\tau | X_\tau, a_\tau)}{\mathbb{P}_{\gamma'}(O_\tau | X_\tau, a_\tau)} \right) &\geq \sum_{\tau=1}^t \|\gamma - \gamma'\|_{\Delta(X_\tau, a_\tau)}^2 - \frac{1}{2} \sum_{\tau=1}^t \|\gamma - \gamma'\|_{\Delta(X_\tau, a_\tau)}^2 \\
&= \frac{1}{2} \sum_{\tau=1}^t \|\gamma - \gamma'\|_{\Delta(X_\tau, a_\tau)}^2 \\
&> \frac{1}{2} \sum_{\tau=1}^t \|\gamma - \gamma'\|_{\Delta(X_\tau, a_\tau)}^2 - 4C_{\sigma^2} \log(1/\delta).
\end{aligned}$$

- If $\sqrt{\sum_{\tau=1}^t \|\gamma - \gamma'\|_{\Delta(X_\tau, a_\tau)}^2} < 2\sqrt{2C_{\sigma^2} \log(1/\delta)}$, then

$$\begin{aligned}
\sum_{\tau=1}^t \log \left(\frac{\mathbb{P}_\gamma(O_\tau | X_\tau, a_\tau)}{\mathbb{P}_{\gamma'}(O_\tau | X_\tau, a_\tau)} \right) &> \sum_{\tau=1}^t \|\gamma - \gamma'\|_{\Delta(X_\tau, a_\tau)}^2 - 4C_{\sigma^2} \log(1/\delta) \\
&\geq \frac{1}{2} \sum_{\tau=1}^t \|\gamma - \gamma'\|_{\Delta(X_\tau, a_\tau)}^2 - 4C_{\sigma^2} \log(1/\delta).
\end{aligned}$$

Thus we get the first inequality. Since $\sum_{\tau=1}^t \|\gamma - \gamma'\|_{\Delta(X_\tau, a_\tau)}^2 = (\gamma - \gamma')^\top \left(\sum_{\tau=1}^t \Delta(X_\tau, a_\tau) \right) (\gamma - \gamma') \geq \lambda_t \|\gamma - \gamma'\|_2^2$, we can get the second inequality. $\square$

Proof for Lemma 7

Proof.

$$\begin{aligned}
&\left| \log \left(\frac{\mathbb{P}_{\gamma_1 + \gamma'}(O | X, a)}{\mathbb{P}_{\gamma_1}(O | X, a)} \right) - \log \left(\frac{\mathbb{P}_{\gamma_2 + \gamma'}(O | X, a)}{\mathbb{P}_{\gamma_2}(O | X, a)} \right) \right| \\
&\leq \left| \nabla_\gamma \log(\mathbb{P}_{\gamma_1}(O | X, a))^T \gamma' - \nabla_\gamma \log(\mathbb{P}_{\gamma_2 + \gamma'}(O | X, a))^T \gamma' \right| \\
&= \left| \gamma'^\top \nabla_\gamma^2 \log(\mathbb{P}_{\tilde{\gamma}}(O | X, a)) (\gamma_1 - \gamma_2 - \gamma') \right| \\
&\leq C_\Gamma \|\gamma'\|_2 (\|\gamma'\|_2 + \|\gamma_1 - \gamma_2\|_2)
\end{aligned}$$

where the first line is by the concavity under Assumption 2, the second line is by the mean value theorem for vector value functions (see Section 12.11 of [Apo74]) with $\tilde{\gamma}$ in the line between γ_1

and $\gamma_2 + \gamma'$, and the last line is by the definition of C_Γ in Assumption 2. $\square$

Proof of Lemma 8

Proof. With little abusing of notations, we use $p_t(\gamma')$ as the density function of the posterior distribution of possible (testing) environment γ' given H_t . Recall H_t is generated through (4.12) with environment γ .

$$\mathcal{P}_t(\|\gamma - \gamma'\|_2^2 \geq \epsilon) = \int_{\|\gamma - \gamma'\|_2^2 \geq \epsilon} p_t(\gamma') d\gamma'.$$

Let $\tilde{\Gamma} = \{\gamma_1, \dots, \gamma_k\}$ as the largest packing set of Γ including γ with $\|\cdot\|_2$ norm and a radius ϵ' (i.e., all other packing sets including γ have a size no larger than k). The choice of ϵ' will be specified later. Without loss of generality, we suppose $\gamma_1 = \gamma$.

Then $\tilde{\Gamma}$ is also a covering set of Γ (with the same norm and radius). The proof follows a standard analysis based on definitions: if $\exists \gamma' \in \Gamma$ such that $\|\gamma' - \gamma_i\|_2 > \epsilon'$ for all $i = 1, \dots, k$, then we can add γ' into $\tilde{\Gamma}$ to construct a new packing set containing γ , which contradicts $\tilde{\Gamma}$ is the largest.

Thus by the definition of the covering set, for all $\gamma' \in \Gamma$, there exists $\gamma_i \in \tilde{\Gamma}$ such that $\gamma' \in B(\gamma_i, \epsilon')$, where $B(\gamma_i, \epsilon') = \{\gamma' \in \Gamma : \|\gamma' - \gamma_i\|_2 \leq \epsilon'\}$, i.e., the ball with center γ_i and radius ϵ' .

Since the interested set $\{\gamma' \in \Gamma : \|\gamma - \gamma'\|_2^2 \geq \epsilon\} \subset \Gamma$, we can find $\tilde{\Gamma}' = \{\gamma'_1, \dots, \gamma'_{k'}\} \subset \tilde{\Gamma}$ which is the smallest subset of $\tilde{\Gamma}$ such that $\{\gamma' \in \Gamma : \|\gamma - \gamma'\|_2^2 \geq \epsilon\} \subseteq \bigcup_{i=1}^{k'} B(\gamma'_i, \epsilon')$.

Now we apply the union bound and Lemma 6 to the density ratios of $\gamma'_i, i = 1, \dots, k'$. Recall under Assumption 1, the prior is a uniform distribution. Then with probability $1 - \delta$,

$$\frac{p_t(\gamma'_i)}{p_t(\gamma)} = \prod_{\tau=1}^{t-1} \left(\frac{\mathbb{P}_{\gamma'_i}(O_\tau | X_\tau, a_\tau)}{\mathbb{P}_\gamma(O_\tau | X_\tau, a_\tau)} \right) \leq \exp \left(-\frac{1}{2} \lambda_{t-1} \|\gamma'_i - \gamma\|_2^2 + 4C_{\sigma^2} \log(k'/\delta) \right) \quad \forall i = 1, \dots, k'.$$

Then we can bound the interested probability by applying the above result: with probability

$1 - \delta$,

$$\begin{aligned}
\int_{\|\gamma - \gamma'\|_2^2 \geq \epsilon} p_t(\gamma') d\gamma' &\leq \sum_{i=1}^{k'} \int_{\|\gamma' - \gamma'_i\|_2 \leq \epsilon'} p_t(\gamma') d\gamma' \\
&= \sum_{i=1}^{k'} \int_{\|\gamma'\|_2 \leq \epsilon'} p_t(\gamma'_i + \gamma') d\gamma' \\
&= \sum_{i=1}^{k'} \frac{p_t(\gamma'_i)}{p_t(\gamma)} \cdot \int_{\|\gamma'\|_2 \leq \epsilon'} p_t(\gamma + \gamma') \cdot \frac{p_t(\gamma'_i + \gamma)}{p_t(\gamma'_i)} \cdot \frac{p_t(\gamma)}{p_t(\gamma + \gamma')} d\gamma' \\
&\leq \sum_{i=1}^{k'} \exp(C_\Gamma \epsilon' (t-1) (\|\gamma - \gamma'_i\|_2 + \epsilon')) \frac{p_t(\gamma'_i)}{p_t(\gamma)} \int_{\|\gamma'\|_2 \leq \epsilon'} p_t(\gamma + \gamma') d\gamma' \\
&\leq \sum_{i=1}^{k'} \exp \left(4C_{\sigma^2} \log(k'/\delta) + C_\Gamma \epsilon' (t-1) (\|\gamma - \gamma'_i\|_2 + \epsilon') - \frac{1}{2} \lambda_{t-1} \|\gamma - \gamma'_i\|_2^2 \right) \int_{\|\gamma - \gamma'\|_2 \leq \epsilon'} p
\end{aligned}$$

where the second last step is by Lemma 7 with $\|\gamma'\|_2 \leq \epsilon'$.

Now by choosing $\epsilon' = \frac{\lambda_{t-1} \min\{\sqrt{\epsilon}, 1\}}{8C_\Gamma(t-1) + 2\lambda_{t-1}}$, we have

- $\epsilon' < \frac{\lambda_{t-1} \min\{\sqrt{\epsilon}, 1\}}{2\lambda_{t-1}} = \frac{\min\{\sqrt{\epsilon}, 1\}}{2}$
- $2C_\Gamma \epsilon' (t-1) < \frac{\lambda_{t-1}}{4} \|\gamma - \gamma'_i\|_2$ for all $i = 1, \dots, k'$. To prove this, we first note that for all $i = 1, \dots, k'$

$$\|\gamma - \gamma'_i\|_2 \geq \sqrt{\epsilon} - \epsilon' \geq \epsilon'.$$

Otherwise, if there exists γ'_i such that $\|\gamma - \gamma'_i\|_2 < \sqrt{\epsilon} - \epsilon'$, then $B(\gamma'_i, \epsilon') \cap \{\gamma' \in \Gamma : \|\gamma - \gamma'\|_2^2 \geq \epsilon\} = \emptyset$ and we can drop γ'_i in $\tilde{\Gamma}'$ but still keep the correspondingly constructed union of balls covering $\{\gamma' \in \Gamma : \|\gamma - \gamma'\|_2^2 \geq \epsilon\}$, which contradicts the fact that $\tilde{\Gamma}'$ is the smallest subset. Thus, we have

$$\begin{aligned}
\frac{\lambda_{t-1}}{4} \|\gamma - \gamma'_i\|_2 - 2C_\Gamma \epsilon' (t-1) &\geq \frac{\lambda_{t-1}}{4} (\sqrt{\epsilon} - \epsilon') - 2C_\Gamma \epsilon' (t-1) \\
&> \frac{1}{4} (\lambda_{t-1} \sqrt{\epsilon} - (8C_\Gamma(t-1) + 2\lambda_{t-1}) \epsilon') \\
&= \frac{1}{4} (\lambda_{t-1} \sqrt{\epsilon} - \lambda_{t-1} \min\{\sqrt{\epsilon}, 1\}) \\
&\geq 0.
\end{aligned}$$

Thus, with probability $1 - \delta$

$$\begin{aligned}
& \int_{\|\gamma - \gamma'\|_2^2 \geq \epsilon} p_t(\gamma') d\gamma' \\
& \leq \sum_{i=1}^{k'} \exp \left(4C_{\sigma^2} \log(k'/\delta) + 2C_\Gamma \epsilon'(t-1) \|\gamma - \gamma'_i\|_2 - \frac{1}{2} \lambda_{t-1} \|\gamma - \gamma'_i\|_2^2 \right) \int_{\|\gamma - \gamma'\|_2 \leq \epsilon'} p_t(\gamma') d\gamma' \\
& \leq \sum_{i=1}^{k'} \exp \left(4C_{\sigma^2} \log(k'/\delta) - \frac{1}{4} \lambda_{t-1} \|\gamma - \gamma'_i\|_2^2 \right) \int_{\|\gamma - \gamma'\|_2 \leq \epsilon'} p_t(\gamma') d\gamma' \\
& < k \exp \left(4C_{\sigma^2} \log(k/\delta) - \frac{1}{16} \lambda_{t-1} \epsilon \right) \int_{\|\gamma - \gamma'\|_2^2 < \epsilon} p_t(\gamma') d\gamma'
\end{aligned}$$

where the last inequality is because $k' \leq k$ and $\|\gamma - \gamma'_i\|_2 \geq \sqrt{\epsilon} - \epsilon'$ with $\epsilon' \leq \frac{\sqrt{\epsilon}}{2}$.

In the following, we bound k which is the size of the constructed packing set $\tilde{\Gamma}$. We use $N_P(\tilde{\epsilon})$ and $N_C(\tilde{\epsilon})$ to denote the packing number and covering number of Γ with $\|\cdot\|_2$ norm and radius $\tilde{\epsilon}$ respectively.

$$\begin{aligned}
k & \leq N_P(\epsilon') \\
& \leq N_C\left(\frac{\epsilon'}{2}\right) \\
& = N_C\left(\frac{\lambda_{t-1} \min\{\sqrt{\epsilon}, 1\}}{16C_\Gamma(t-1) + 4\lambda_{t-1}}\right) \\
& \leq N_C\left(\frac{\lambda_{t-1} \min\{\sqrt{\epsilon_t}, 1\}}{16C_\Gamma(t-1) + 4\lambda_{t-1}}\right) \\
& \leq \max \left\{ 1, \left(\frac{32\bar{\gamma}\sqrt{d}(C_\Gamma(t-1) + \lambda_{t-1})}{\lambda_{t-1} \min\{\sqrt{\epsilon_t}, 1\}} \right)^d \right\}
\end{aligned}$$

where the first line is from the definition of the packing number, the second line comes from the relationship between covering and packing numbers (Lemma 4.2.8 in [Ver18]), the third line is from the definition of ϵ' , the fourth line is by $\epsilon_t \leq \epsilon$, and the last line is from Example 27.1 in [SSBD14] for upper bounds of covering numbers in Euclidean (sub)space.

Recall the notations

$$K_t = \max \left\{ 1, \left(\frac{32\bar{\gamma}\sqrt{d}(C_\Gamma(t-1) + \lambda_{t-1})}{\lambda_{t-1} \min\{\sqrt{\epsilon_t}, 1\}} \right)^d \right\},$$

$$C_t(\epsilon, \delta) = \exp \left(-4C_{\sigma^2} \log(K_t/\delta) + \frac{1}{16} \lambda_{t-1} \epsilon \right).$$

Then we have with probability $1 - \delta$,

$$\int_{\|\gamma - \gamma'\|_2^2 \geq \epsilon} p_t(\gamma') d\gamma' \leq \frac{K_t}{C_t(\epsilon, \delta)} \int_{\|\gamma - \gamma'\|_2^2 < \epsilon} p_t(\gamma') d\gamma'.$$

Since

$$\int_{\|\gamma - \gamma'\|_2^2 \geq \epsilon} p_t(\gamma') d\gamma' + \int_{\|\gamma - \gamma'\|_2^2 < \epsilon} p_t(\gamma') d\gamma' = 1,$$

by rearranging the term we can conclude the result. $\square$

Proof of Corollary 5

Proof. For any given $\epsilon \geq \epsilon_t$ and $1 > \delta > 0$, we denote the event $\mathcal{E} = \left\{ \mathcal{P}_t(\|\gamma - \gamma'\|_2^2 \geq \epsilon) \leq \frac{K_t}{K_t + C_t(\epsilon, \delta)} \right\}$, which holds with probability $1 - \delta$ by Lemma 8, and its complementary event as $\bar{\mathcal{E}}$. Then

$$\begin{aligned} \mathbb{E} \left[\mathbb{E}_t \left[\|\gamma - \gamma'\|_2^2 \mathbb{1}_{\{\|\gamma - \gamma'\|_2^2 \geq \epsilon_t\}} \right] \right] &= \mathbb{E} \left[\int_{\epsilon \geq \epsilon_t} \mathcal{P}_t(\|\gamma - \gamma'\|_2^2 \geq \epsilon) d\epsilon \right] \\ &= \mathbb{E} \left[\int_{\epsilon \geq \epsilon_t} \mathcal{P}_t(\|\gamma - \gamma'\|_2^2 \geq \epsilon) (\mathbb{1}_{\mathcal{E}} + \mathbb{1}_{\bar{\mathcal{E}}}) d\epsilon \right] \\ &\leq \mathbb{E} \left[\int_{\epsilon \geq \epsilon_t} \frac{K_t}{K_t + C_t(\epsilon, \delta)} d\epsilon \right] + \int_{0 \leq \epsilon \leq 4\bar{\gamma}^2} \delta d\epsilon \\ &= \mathbb{E} \left[\int_{\epsilon \geq \epsilon_t} \frac{K_t}{K_t + C_t(\epsilon, \delta)} d\epsilon \right] + 4\bar{\gamma}^2 \delta, \end{aligned}$$

where the third line is because $\mathcal{P}_t(\|\gamma - \gamma'\|_2^2 \geq \epsilon) \leq 1$ and $\mathcal{P}_t(\|\gamma - \gamma'\|_2^2 \geq \epsilon) = 0$ for $\epsilon > 4\bar{\gamma}^2$, with the probability of $\bar{\mathcal{E}}$ being δ . The expectation in the last line is with respect to the potential randomness in λ_{t-1} .

We now bound the first term. Note that for any $C_1, C_2 > 0$, we have

$$\int_{\epsilon \geq \epsilon_t} \frac{C_1}{C_1 + \exp(C_2 \epsilon)} d\epsilon = \frac{\log(C_1 + \exp(C_2 \epsilon_t))}{C_2} - \epsilon_t < \frac{\log(C_1 + \exp(C_2 \epsilon_t))}{C_2}.$$

By plugging $C_t(\epsilon, \delta)$, we have $\mathbb{E} \left[\int_{\epsilon \geq \epsilon_t} \frac{K_t}{K_t + C_t(\epsilon, \delta)} d\epsilon \right] \leq \mathbb{E} \left[\frac{16}{\lambda_{t-1}} \log \left(K_t \left(\frac{K_t}{\delta} \right)^{4C_{\sigma^2}} + \exp\left(\frac{\lambda_{t-1} \epsilon_t}{16}\right) \right) \right].$

For $\epsilon_t = \frac{16d}{\lambda_{t-1}} \log \left(\frac{32\bar{\gamma}\sqrt{d}(C_\Gamma(t-1) + \lambda_{t-1})}{\lambda_{t-1}} \right)$, we have for all $t \geq t_0$

$$\begin{aligned} \frac{\lambda_{t-1}\epsilon_t}{16} &= d \log \left(\frac{32\bar{\gamma}\sqrt{d}(C_\Gamma(t-1) + \lambda_{t-1})}{\lambda_{t-1}} \right) \\ &\leq d \log \left(\frac{32\bar{\gamma}\sqrt{d}(C_\Gamma(t-1) + \lambda_{t-1})}{\lambda_{t-1} \min\{\sqrt{\epsilon_t}, 1\}} \right) \\ &= \log K_t \\ &< \log \left(K_t (K_t/\delta)^{4C_\sigma^2} \right) \end{aligned}$$

where the last inequality is because $K_t \geq 1$ and $\delta < 1$. Then we have

$$\mathbb{E} \left[\int_{\epsilon \geq \epsilon_t} \frac{K_t}{K_t + C_t(\epsilon, \delta) d\epsilon} \right] \leq \mathbb{E} \left[\frac{16}{\lambda_{t-1}} \log (2K_t (K_t/\delta)^{4C_\sigma^2}) \right].$$

□

C.4 Appendix for Numerical Experiments

C.4.1 More Numerical Experiments

C.4.1.1 Matchings of $\text{TF}_{\hat{\theta}}$ and Alg^*

Here we provide more results regarding the matching of $\text{TF}_{\hat{\theta}}$ and Alg^* across different tasks: Figure C.5 provides more examples comparing decisions made by $\text{TF}_{\hat{\theta}}$ and Alg^* at sample path levels, while Figure C.6 compares them at population levels.

We plot the actions generated from $\text{TF}_{\hat{\theta}}$ and Alg^* on the same environment with the same contexts in Figure C.5 across multi-armed bandits, dynamic pricing and newsvendor tasks. Each row has a different number of possible environments with positive probability by $\mathcal{P}_\gamma$ (See Appendix C.4.2 for the definition of such a finite pool of environments). The population-level differences $\{\text{TF}_{\hat{\theta}}(H_t) - \text{Alg}^*(H_t)\}_{t=1}^T$ are presented in Figure C.6. We observe that across

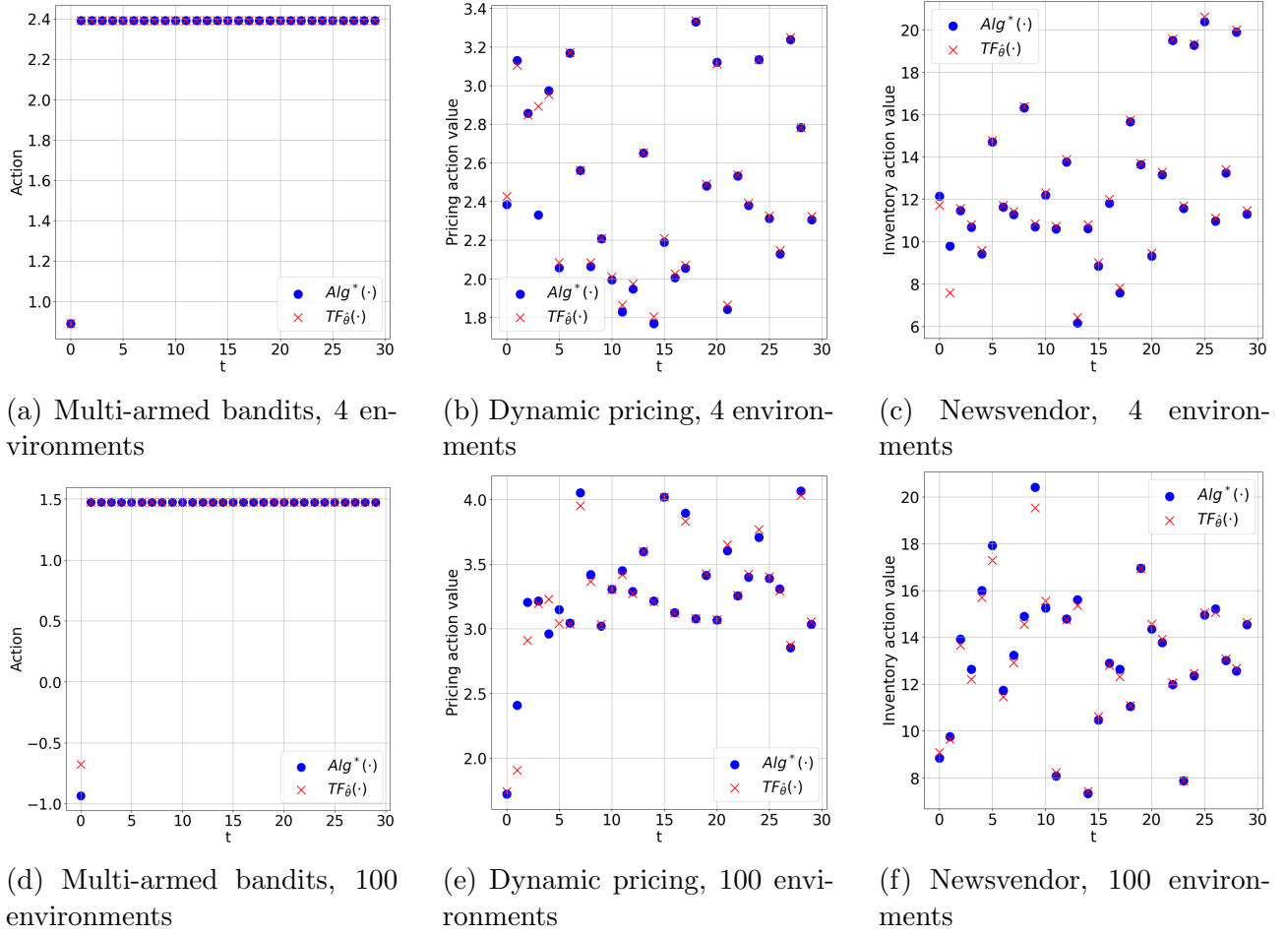


Figure C.5: Examples to compare the actions from the transformer TF_θ and the optimal decision function Alg^* .

different tasks: (1) TF_θ nearly matches Alg^* , but (2) the matchings are not perfect. The actions from TF_θ and Alg^* are not exactly the same, and their differences increase as the underlying problems become more complex, as seen by comparing the first and second rows in Figure C.6.

Setup. For Figure C.5, each subfigure is based on a sampled environment with a sampled sequence of contexts $\{X_t\}_{t=1}^{30}$ from the corresponding task. The data generation process follows the description detailed in Appendix C.4.2. Both the pre-training and testing samples are drawn from $\mathcal{P}_\gamma$, i.e., from these fixed 16/100 environments. The remaining setups for these tasks follow the methods provided in Appendix C.4.2.

For each task shown in Figure C.6, we generate 100 sequences of $\{\text{TF}_\theta(H_t) - \text{Alg}^*(H_t)\}_{t=1}^{100}$, i.e., each subfigure is based on 10000 samples of $\text{TF}_\theta(H_t) - \text{Alg}^*(H_t)$. For dynamic pricing and newsvendor, actions are scalars and thus $\text{TF}_\theta(H_t) - \text{Alg}^*(H_t)$ is also a scalar; For multi-armed

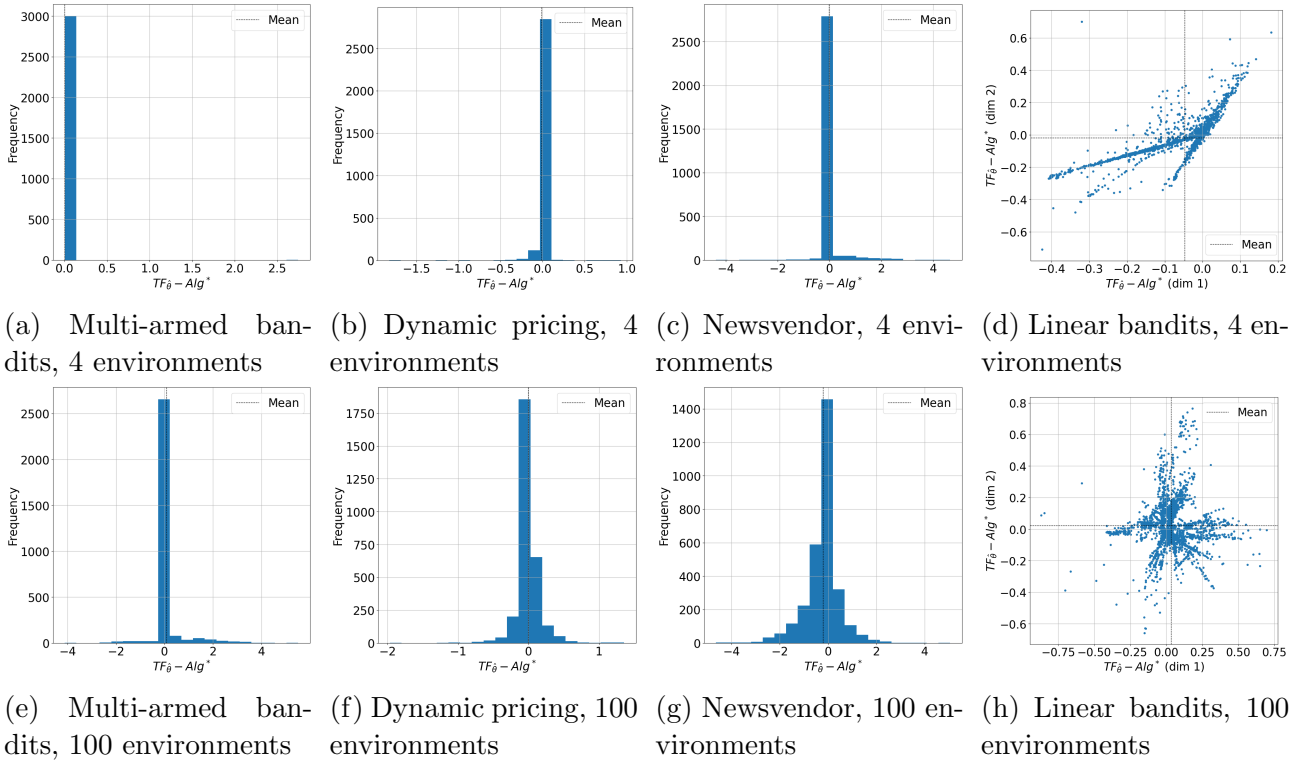


Figure C.6: $\text{TF}_{\hat{\theta}}(H_t) - \text{Alg}^*(H_t)$ across different tasks with various numbers of possible environments. More possible environments lead to a harder decision making problem.

bandits, the action space is discrete and we use the reward difference associated with the chosen actions (by $\text{TF}_{\hat{\theta}}(H_t)$ or $\text{Alg}^*(H_t)$) as the value of $\text{TF}_{\hat{\theta}}(H_t) - \text{Alg}^*(H_t)$. We use histograms to summarize the samples from them. For linear bandits, the actions are 2-dimensional and thus we use the scatters to show $\text{TF}_{\hat{\theta}}(H_t) - \text{Alg}^*(H_t)$ in Figure C.6 (d) or (h).

C.4.1.2 Benefit of Utilizing Prior Knowledge

Here we present further results on the performance of $\text{TF}_{\hat{\theta}}$ to demonstrate the benefit of utilizing prior knowledge $\mathcal{P}_{\gamma}$. Specifically, we compare $\text{TF}_{\hat{\theta}}$ with benchmark algorithms on (i) simple tasks, which include only 4 possible environments in both pretraining and testing samples (Figure C.7), and (ii) more complex tasks, which involve either 100 possible environments (Figure C.8 (a), (b)) or 16 environments with two possible types of demand functions (Figure C.8 (c), (d)). See Appendix C.4.2 for the definition of finite environments used here.

Figures C.7 and C.8 illustrate the consistently and significantly superior performance of $\text{TF}_{\hat{\theta}}$ and Alg^* across all tasks compared to the benchmark algorithms. These results underscore the

advantage of leveraging prior knowledge about the tested environments.

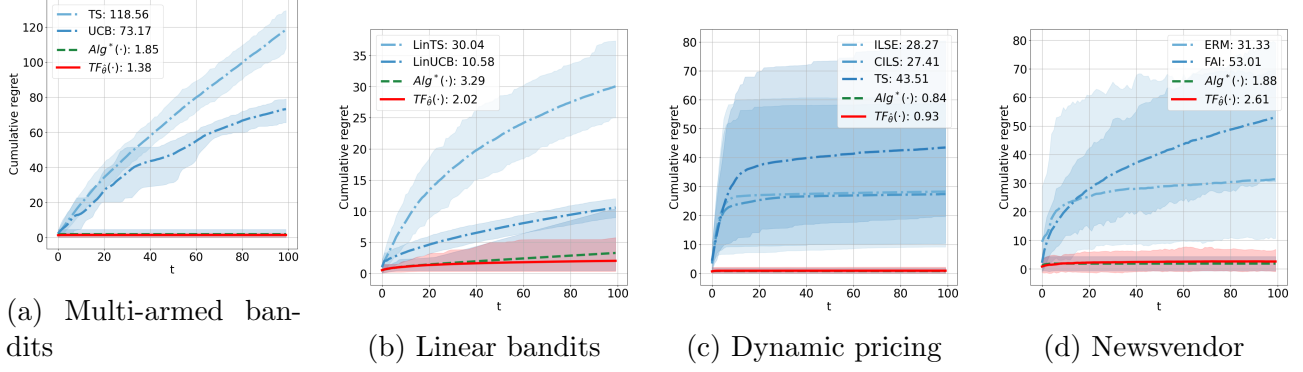


Figure C.7: The average out-of-sample regret on tasks with simpler environments. Each task only has 4 possible environments. The numbers in the legend bar are the final regret at $t = 100$ and the shaded areas indicate the 90% (empirical) confidence intervals. The details of benchmarks can be found in Appendix C.4.3.

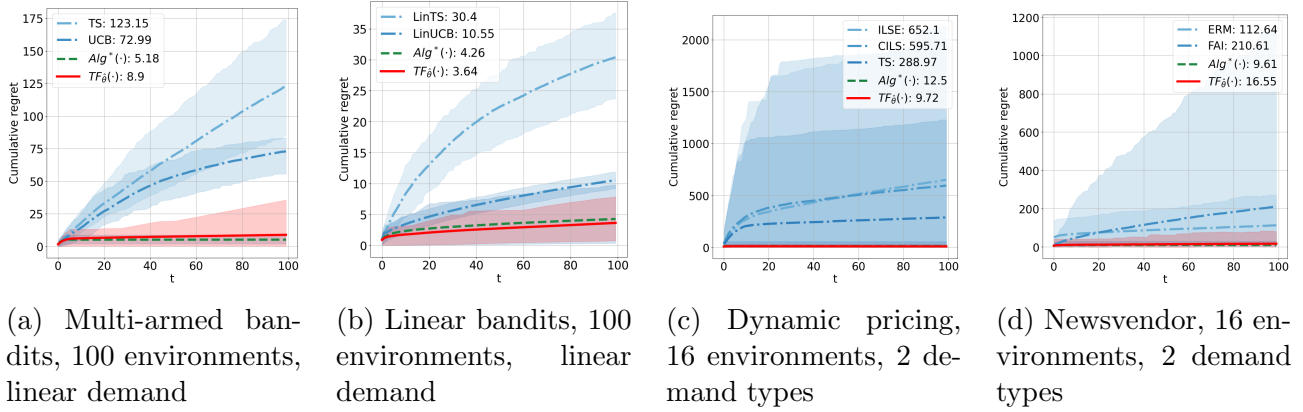


Figure C.8: The average out-of-sample regret on harder tasks. With 100 environments in (a) multi-armed bandits and (b) linear bandits; or with 16 environments, 2 demand types in (c) dynamic pricing and (d) newsvendor. The numbers in the legend bar are the final regret at $t = 100$ and the shaded areas indicate the 90% (empirical) confidence intervals. The details of benchmarks can be found in Appendix C.4.3

Setup. All figures are based on 100 runs. We consider four tasks with 4 environments (i.e., the support of $\mathcal{P}_{\gamma}$ contains only 4 environments) in Figure C.7 and two tasks with 100 environments in Figure C.8 (a), (b). The setup for these tasks follows the generation methods provided in Appendix C.4.2.

For the tasks in Figure C.8 (c), (d), there are 16 environments included in the support of $\mathcal{P}_{\gamma}$, where 8 environments have demand functions of the linear type and the other 8 have demand functions of the square type (see Appendix C.4.2 for the definitions of these two types). Both the pretraining and testing samples are drawn from $\mathcal{P}_{\gamma}$, i.e., from these 16 environments. The

remaining setups for these tasks follow the methods provided in Appendix C.4.2.

C.4.1.3 Impact of Network Architecture

Here we investigate the effect of different network architectures on performance. Specifically, we replace the transformer/GPT-2 architecture used in OMGPT with Long Short-Term Memory (LSTM) [Hoc97], a popular neural network architecture for sequence modeling before the rise of transformers [VHMN20].

We maintain the overall architecture of OMGPT, as described in Appendix C.1.1 except for replacing the transformer/GPT-2 module with LSTM. We evaluate two LSTM variants: a 5-layer LSTM and a 12-layer LSTM (for comparison, the tested transformer also has 12 layers). Typically, LSTM architectures should be shallower than transformers in practice, which is why we also include the 5-layer version. Other hyperparameters like embedding space dimension are kept the same across all models. We pre-train and test these models on dynamic pricing problems (see Appendix C.4.2 for the experimental setup), and the pre-training procedure is identical to the one outlined in Appendix C.2.

Figure C.9 presents the testing regret for the three models. The results indicate no significant difference between the 5-layer and 12-layer LSTM models. However, the transformer consistently performs better than both LSTM variants. This outcome aligns with the transformer's superior performance in other domains, such as natural language processing, and suggests that transformers are more effective for sequential decision making tasks as well.

C.4.1.4 More Results for Figure 4.13

Figure C.10 presents the complete results corresponding to Figure 4.13.

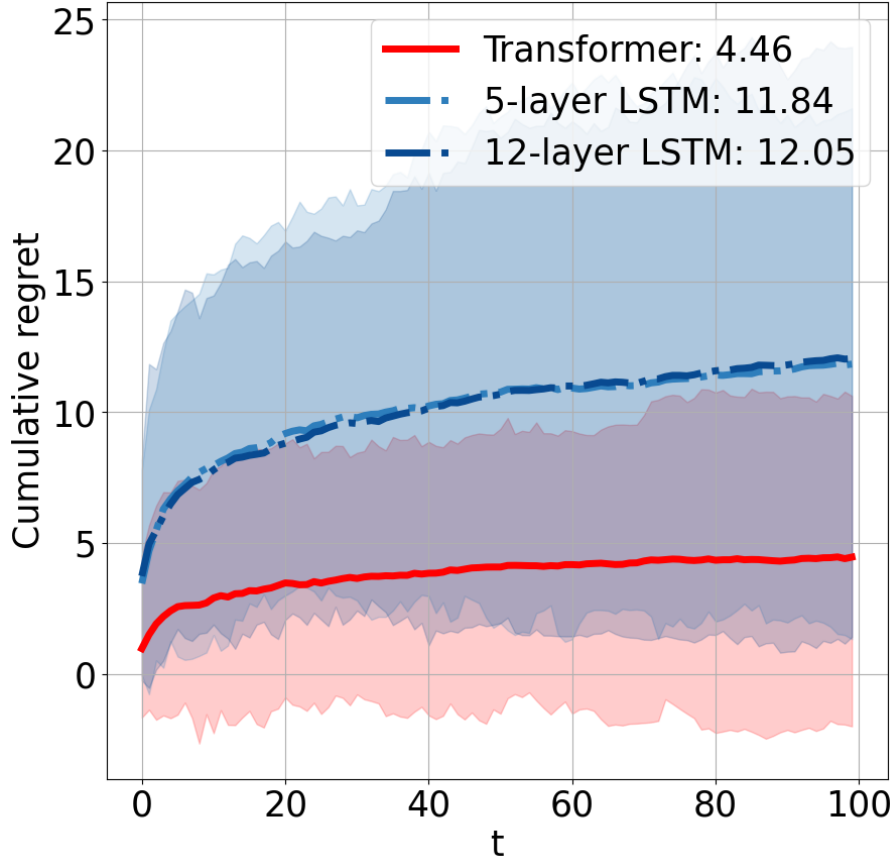


Figure C.9: Comparison of the average out-of-sample regret between using the Transformer and LSTM architectures in OMGPT.

C.4.2 Sequential Decision Making Problems and Environment Generation

We provide the formulations of the sequential decision making problems studied in this paper, along with their generation methods in experiments. We set the time horizon $T = 100$ for all tasks except Figure 4.6 and the revenue management task, where we set $T = 50$ to reduce the computational cost of running the benchmark algorithm and generating the pre-training data.

Stochastic multi-armed bandit There is no context, i.e., $X_t = \text{null}$ for all t . The action $a_t \in \mathcal{A} = \{1, 2, \dots, k\}$ denotes the index of the arm played at time t . In experiments, We consider the number of actions/arms $|\mathcal{A}| = 20$. The environment parameter γ encapsulates the expected reward r_a of each arm $a = 1, \dots, 20$, where the reward of arm a is sampled from $\mathcal{N}(r_a, 0.2)$. Thus, the environment distribution $\mathcal{P}_\gamma$ is defined by the (joint) distribution of

$(r_1, \dots, r_{20})$. We set $r_a \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1)$ for each action a . The optimal arm is $a^* = \arg \max_a r_a$.

Linear bandits There is no context, i.e., $X_t = \text{null}$ for all t . The action $a_t \in \mathcal{A} \subset \mathbb{R}^d$ is selected from some pre-specified domain $\mathcal{A}$. The random reward $R(X_t, a_t) = w^\top a_t + \epsilon_t$ where ϵ_t is some zero-mean noise random variable and $w \in \mathbb{R}^d$ is a vector unknown to the decision maker. The observation $O_t = R(X_t, a_t)$ and the expected reward $r(X_t, a_t) = \mathbb{E}[R(X_t, a_t)|X_t, a_t] = w^\top a_t$. In experiments, we consider the dimension of actions/arms $d = 2$ and $\mathcal{A} = \mathcal{B}(0, 1)$, i.e., a unit ball centered at the origin (with respect to the Euclidean norm). The environment parameter γ encapsulates the (linear) reward function's parameter $w \in \{\tilde{w} \in \mathbb{R}^2 : \|\tilde{w}\|_2 = 1\}$, where the reward of arm a is sampled from $\mathcal{N}(w^\top a, 0.2)$. Thus, the environment distribution $\mathcal{P}_\gamma$ is defined by the distribution of w . We set w to be uniformly sampled from the unit sphere, and the corresponding optimal arm being $a^* = w$.

Dynamic pricing In experiments, we set the noise $\epsilon_t \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(0, 0.2)$ and the context X_t to be sampled i.i.d. and uniformly from $[0, 5/2]^d$, with $d = 6$ as the dimension of contexts (except Figure 4.5, where we set $d = 4, 10, 20$ and accordingly change the dimension of the following introduced parameters), and $\mathcal{A} = [0, 30]$. We consider the linear demand function family $D_t = \alpha^\top X_t - \beta^\top X_t \cdot a_t + \epsilon_t$, where $\alpha, \beta \in \mathbb{R}^d$ are the demand parameters. Thus, the environment distribution $\mathcal{P}_\gamma$ is defined by the (joint) distribution of (α, β) . We set α to be uniformly sampled from $[1/2, 3/2]^6$ and β to be uniformly sampled from $[1/20, 21/20]^6$, independent of α . The optimal action $a_t^* = \frac{\alpha^\top X_t}{2 \cdot \beta^\top X_t}$. We also consider a square demand function family, which will be specified later, to test the OMGPT's performance on a mixture of different demand-type tasks.

Newsvendor problem In experiments, we set the context $\tilde{X}_t$ to be sampled i.i.d. and uniformly from $[0, 3]^d$ with $d = 4$, and $\mathcal{A} = [0, 30]$. We consider the linear demand function family $D_t = w^\top \tilde{X}_t + \epsilon_t$, where $w \in \mathbb{R}^4$. Under uncensored demand setting, the observation $O_t = D_t$, while under the censored demand setting, the observation $O_t = \min\{D_t, a_t\}$. The environment parameter γ encapsulates (i) the upper bound $\bar{\epsilon}$ of the noise ϵ_t , where $\epsilon_t \stackrel{\text{i.i.d.}}{\sim}$

$\text{Unif}(0, \bar{\epsilon})$; (ii) the left-over cost h (with the lost-sale cost l being 1); and (iii) the demand parameter w . Accordingly, we set (i) $\bar{\epsilon} \sim \text{Unif}[1, 10]$; (ii) $h \sim \text{Unif}[1/2, 2]$; and (iii) w uniformly sampled from $[0, 3]^4$. The optimal action can be computed by $a_t^* = w^\top \tilde{X}_t + \frac{\bar{\epsilon}}{1+h}$ [DHR24], which is indeed the $\frac{1}{1+h}$ quantile of the random variable $w^\top \tilde{X}_t + \epsilon_t$. When implementing OMGPTs, we add the information of h into the context (since it is known), i.e., $X_t = (h, \tilde{X}_t)$. We also consider a square demand function family, which will be specified later, to test the OMGPT's performance on a mixture of different demand-type tasks.

In the non-perishable setting applied in Section 4.6.2.2, any unsold products can be carried over to the next period, becoming part of the inventory. Specifically, let I_t denote the inventory at the beginning of each period t , with $I_1 = 0$. This leads to several adjustments: (i) the action a_t must satisfy $a_t \geq I_t$ for each t ; (ii) the context now includes I_t , which should be considered when making decisions, i.e., $X_t = (I_t, h, \tilde{X}_t)$; (iii) the optimal decision is now $a_t^* = \max\{0, w^\top \tilde{X}_t + \frac{\bar{\epsilon}}{1+h} - I_t\}$. (iv) $I_{t+1} = \max\{0, a_t - D_t\}$ (under uncensored cases). In the experiment, we follow the same generation method but allow the action space $\mathcal{A}$ to vary over time as $[I_t, 30 + I_t]$.

Queuing control We consider a service rate controlling task for a single queue, which is adopted by an example of [DFVR03] (Section 6.1). We assume the queue begins with the empty state at $t = 1$. At the beginning of each time period t , the decision maker can first observe the length of queue, and then decide the service rate $a_t \in \{0, 0.2, \dots, 1\}$ to process the customers in the queue. Then right before the end of the time period, one customer will arrive with an unknown arrival rate λ . We assume the queue has a maximum length 4 and set the observation O_t by the length of the queue at the end of the time period t , then we have

$\mathcal{O} = \{0, 1, 2, 3, 4\}$. Given the service rate a_t , the transition matrix of the length of the queue is

$$\begin{array}{c} \begin{array}{ccccc} & 0 & 1 & 2 & 3 & 4 \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} & \left(\begin{array}{ccccc} 1 - \lambda & \lambda & 0 & 0 & 0 \\ (1 - \lambda)a_t & (1 - \lambda)(1 - a_t) + \lambda a_t & \lambda(1 - a_t) & 0 & 0 \\ 0 & (1 - \lambda)a_t & (1 - \lambda)(1 - a_t) + \lambda a_t & \lambda(1 - a_t) & 0 \\ 0 & 0 & (1 - \lambda)a_t & (1 - \lambda)(1 - a_t) + \lambda a_t & \lambda(1 - a_t) \\ 0 & 0 & 0 & (1 - \lambda)a_t & 1 - a_t + \lambda a_t \end{array} \right) \end{array} \end{array}.$$

The reward function is a minus cost function with a known cost parameter c :

$$r(X_t, a_t) = -(O_{t-1} + c \cdot a_t^2),$$

where the context $X_t = (O_{t-1}, c)$ feeding the information of the cost parameter to the decision marker. In our experiments, the parameters are uniformly sampled from

$$\lambda \sim \text{Unif}(\{0.1, 0.26, 0.42, 0.58, 0.74, 0.9\}),$$

$$c \sim \text{Unif}(\{5, 16, 27, 38, 49, 60\}).$$

Thus there are in total 36 possible environments.

Revenue management We consider a revenue management problem [TVR06, Jas15] with horizon T and budget constraints on J types of resources. In each time period t , a customer arrives, with their type i.i.d. sampled from a pool of K possible types according to an unknown distribution, and requests a certain amount of each resource with a reward $\tilde{r}_t$. The decision is whether to accept or reject the customer's order. Specifically, the problem is modeled using

the following linear program:

$$\begin{aligned}
& \max \quad \sum_{t=1}^T \tilde{r}_t y_t \\
& \text{s.t.} \quad \sum_{t=1}^T \tilde{\mathbf{A}}_t y_t \leq T \cdot \mathbf{1} \\
& \quad y_t \in \{0, 1\}, \quad t = 1, \dots, T
\end{aligned} \tag{C.3}$$

where $\mathbf{A}_t = (A_{1,t}, \dots, A_{J,t})^\top \in \mathbb{R}^J$ represents the resource consumption and the decision variables are $\mathbf{y} = (y_1, \dots, y_T)^\top \in \{0, 1\}^T$. There are J constraints and T decision variables.

In this decision problem, the action $a_t \in [0, 1]$ represents the probability of accepting an order, and X_t represents the revealed customer type, i.e., $X_t = (\tilde{r}_t, \tilde{\mathbf{A}}_t) \in \mathcal{X}$, where $\mathcal{X}$ contains K distinct elements. The reward is defined as $r(X_t, a_t) = \tilde{r}_t \cdot a_t$, and the observation at time t is $O_t = X_t$.

In our experiments, we set $J = 3$ (the number of resource types) and $K = 3$ (the number of potential customer types). Each vector element in the space $\mathcal{X}$ is independently and identically sampled from a uniform distribution in the range $[1, 2]^4$. Additionally, the unknown distribution for the arrival probability of each customer type is generated as $(\frac{\mu_1}{\mu_1 + \mu_2 + \mu_3}, \frac{\mu_2}{\mu_1 + \mu_2 + \mu_3}, \frac{\mu_3}{\mu_1 + \mu_2 + \mu_3})$, where μ_1, μ_2, μ_3 are independently sampled from a uniform distribution over the range $[0, 1]$.

Other settings

- **Finite Pool of Environments.** To study the behavior and test the performance of the OMGPT on finite possible environments (e.g., to see if and how $\text{TF}_{\hat{\theta}}$ converges to Alg^*), we also consider a finite pool of environments as the candidates of γ . Specifically, for some tasks we first sample finite environments (e.g., $\gamma_1, \dots, \gamma_4$) i.i.d. following the sampling rules previously described. We then set the environment distribution $\mathcal{P}_\gamma$ as a uniform distribution over the pool of sampled environments (e.g., $\{\gamma_1, \dots, \gamma_4\}$). We should notice this pool of environments does not restrict the context generation (if any): in both the pre-training and testing, contexts are generated following the rules previously described

and are independent of the given finite pool.

- **Tasks with Two Demand Types.** To test the OMGPT's performance on tasks with possibly different demand types, we also consider the square demand function family for the newsvendor tasks. Specifically, besides the linear demand function family, we also consider $d^{\text{sq}}(X_t, a_t) = (w^\top X_t)^2$ in the newsvendor problem. Thus, with a slight abuse of notation, we can augment γ such that it also parameterizes the type of demand (linear or square) and the type is unknown to the decision maker. In experiments related to multi-type demands, the probability of each type being sampled is $1/2$, while the distributions over other parameters of the environment remain the same as in the linear demand case.
- **Non-stationary environment.** To evaluate OMGPT's performance in non-stationary environments, we modify the demand function in the newsvendor problem, where $d^t(X_t, a_t) = w_t^\top X_t$ varies over time due to changes in the unknown parameter w_t . Specifically, we assume there exists a random time $\tau \leq T$, uniformly sampled from $[1, T]$, such that $w_1 = w_2 = \dots = w_\tau$ and $w_{\tau+1} = w_{\tau+2} = \dots = w_T$. The parameters w_1 and $w_{\tau+1}$ are independently sampled using the generation method described earlier for the newsvendor problem.

C.4.3 Benchmark Algorithms

C.4.3.1 Multi-armed bandits

- **Upper Confidence Bound (UCB)** [LS20]: Given H_t , the action a_t is chosen by $a_t = \arg \max_{a \in \mathcal{A}} \hat{r}_a + \frac{\sqrt{2 \log T}}{\min\{1, n_a\}}$, where n_a is the number of pulling times of arm a before time t and $\hat{r}_a$ is the empirical mean reward of a based on H_t .
- **Thompson sampling (TS)** [RVRK⁺18]: Given H_t , the action a_t is chosen by $a_t = \arg \max_{a \in \mathcal{A}} \tilde{r}_a$, where each $\tilde{r}_a \sim \mathcal{N}\left(\hat{r}_a, \frac{\sqrt{2 \log T}}{\min\{1, n_a\}}\right)$ is the sampled reward of arm a and $\hat{r}_a, n_a$ follow the same definitions as in UCB.
- **Alg***: Given H_t from a task with a pool of $|\Gamma|$ environments $\{\gamma_1, \dots, \gamma_{|\Gamma|}\}$, the action a_t is

chosen by the posterior sampling defined in Algorithm 2. The posterior distribution can be computed by

$$\mathcal{P}(\gamma_i|H_t) = \frac{\exp(-\frac{1}{\sigma^2} \sum_{\tau=1}^{t-1} (O_\tau - r_{a_\tau}^i)^2)}{\sum_{i'=1}^{|\Gamma|} \exp(-\frac{1}{\sigma^2} \sum_{\tau=1}^{t-1} (O_\tau - r_{a_\tau}^{i'})^2)},$$

where r_a^i is the expected reward of a in environment γ_i and σ^2 is the variance of the noise (which equals to 0.2 in our experiments).

C.4.3.2 Linear bandits

- **LinUCB** [CLRS11]: Given H_t , we define $\Sigma_t = \sum_{\tau=1}^{t-1} a_\tau a_\tau^\top + \sigma^2 I_d$, where σ^2 is the variance of the reward noise. The action a_t is chosen by $a_t \in \arg \max_{a \in \mathcal{A}} \hat{w}_t^\top a + \sqrt{2 \log T} \|a\|_{\Sigma_t^{-1}}$, where $\hat{w}_t = \Sigma_t^{-1} (\sum_{\tau=1}^{t-1} O_\tau \cdot a_\tau)$ is the current estimation of w from H_t .
- **LinTS** [AG13]: Given H_t , the action a_t is chosen by $a_t = \arg \max_{a \in \mathcal{A}} \tilde{w}_t^\top a$, where $\tilde{w}_t \sim \mathcal{N}(\hat{w}_t, \sqrt{2 \log T} \Sigma_t^{-1})$ as the sampled version of w and $\hat{w}_t, \Sigma_t$ follow the same definitions in LinUCB.
- **Alg***: Given H_t from a task with a pool of $|\Gamma|$ environments $\{\gamma_1, \dots, \gamma_{|\Gamma|}\}$, the action a_t is chosen by the posterior median defined in Algorithm 2. The posterior distribution can be computed by

$$\mathcal{P}(\gamma_i|H_t) = \frac{\exp(-\frac{1}{\sigma^2} \sum_{\tau=1}^{t-1} (O_\tau - w_i^\top a_\tau)^2)}{\sum_{i'=1}^{|\Gamma|} \exp(-\frac{1}{\sigma^2} \sum_{\tau=1}^{t-1} (O_\tau - w_{i'}^\top a_\tau)^2)},$$

where w_i is the reward function parameter in environment γ_i and σ^2 is the variance of the noise (which equals to 0.2 in our experiments).

C.4.3.3 Dynamic pricing

All the benchmark algorithms presented below assume the demand model belongs to the linear demand function family as defined in Appendix C.4.2 (which can be mis-specified when we deal with dynamic pricing problems with two demand types).

- Iterative least square estimation (ILSE) [QB16]: At each t , it first estimates the unknown demand parameters (α, β) by applying a ridge regression based on H_t , and chooses a_t as the optimal action according to the estimated parameters and the context X_t . Specifically, we denote $(\hat{\alpha}_t, -\hat{\beta}_t) = \Sigma_t^{-1}(\sum_{\tau=1}^{t-1} O_\tau \cdot z_\tau)$ as the estimation of $(\alpha, -\beta)$ through a ridge regression, where $z_\tau = (X_\tau, a_\tau \cdot X_\tau)$ serves as the “feature vector” and $\Sigma_t = \sum_{\tau=1}^{t-1} z_\tau z_\tau^\top + \sigma^2 I_d$ (σ^2 is the variance of the demand noise). Then the action is chosen as the optimal one by treating $(\hat{\alpha}_t, \hat{\beta}_t)$ as the true parameter: $a_t = \frac{\hat{\alpha}_t^\top X_t}{2\hat{\beta}_t^\top X_t}$.
- Constrained iterated least squares (CILS) [KZ14]: It is similar to the ILSE algorithm except for the potential explorations when pricing. Specifically, we follow the notations in ILSE and denote $\hat{a}_t = \frac{\hat{\alpha}_t^\top X_t}{2\hat{\beta}_t^\top X_t}$ as the optimal action by treating $(\hat{\alpha}_t, \hat{\beta}_t)$ as the true parameter (the chosen action of ILSE), and denote $\bar{a}_{t-1}$ as the empirical average of the chosen actions so far. Then the CILS chooses

$$a_t = \begin{cases} \bar{a}_{t-1} + \text{sgn}(\delta_t) \frac{t^{-\frac{1}{4}}}{10}, & \text{if } |\delta_t| < \frac{1}{10} t^{-\frac{1}{4}}, \\ \hat{a}_t, & \text{otherwise,} \end{cases}$$

where $\delta_t = \hat{a}_t - \bar{a}_{t-1}$. The intuition is that if the tentative price $\hat{a}_t$ stays too close to the history average, it will introduce a small perturbation around the average as price experimentation to encourage parameter learning.

- Thompson sampling for pricing (TS) [WTL21]: Like the ILSE algorithm, it also runs a regression to estimate (α, β) based on the history data, while the chosen action is the optimal action of a sampled version of parameters. Specifically, we follow the notations in ILSE and then TS chooses

$$a_t = \frac{\tilde{\alpha}}{2 \cdot \tilde{\beta}},$$

where $(\tilde{\alpha}, \tilde{\beta}) \sim \mathcal{N}((\hat{\alpha}_t^\top X_t, \hat{\beta}_t^\top X_t), \tilde{\Sigma}_t^{-1}) \in \mathbb{R}^2$ are the sampled parameters of the “intercept” and “price coefficient” in the linear demand function and

$$\tilde{\Sigma}_t = \left(\begin{pmatrix} X_t & \mathbf{0} \\ \mathbf{0} & X_t \end{pmatrix}^\top \Sigma_t^{-1} \begin{pmatrix} X_t & \mathbf{0} \\ \mathbf{0} & X_t \end{pmatrix} \right)^{-1}$$

is the empirical covariance matrix given X_t .

- **Alg***: Given H_t from a task with a pool of $|\Gamma|$ environments $\{\gamma_1, \dots, \gamma_{|\Gamma|}\}$, the action a_t is chosen by the posterior averaging defined in Algorithm 2. To compute the posterior distribution, we follow the notations in ILSE and denote $w = (\alpha, \beta)$ as the stacked vector of parameters, then the posterior distribution is

$$\mathcal{P}(\gamma_i | H_t) = \frac{\exp(-\frac{1}{\sigma^2} \sum_{\tau=1}^{t-1} (O_\tau - w_i^\top z_\tau)^2)}{\sum_{i'=1}^{|\Gamma|} \exp(-\frac{1}{\sigma^2} \sum_{\tau=1}^{t-1} (O_\tau - w_{i'}^\top z_\tau)^2)},$$

where w_i is the demand function parameter in environment γ_i and σ^2 is the variance of the noise (which equals to 0.2 in our experiments).

C.4.3.4 Newsvendor

All the benchmark algorithms presented below assume the demand model belongs to the linear demand function family as defined in Appendix C.4.2 (which can be mis-specified when we deal with newsvendor problems with two demand types).

- Empirical risk minimization (ERM) [BR19]: Since the optimal action a_t^* is the $\frac{1}{1+h}$ quantile of the random variable $w^\top \tilde{X}_t + \epsilon_t$ [DHR24], ERM conducts a linear quantile regression based on the observed contexts and demands $\{(\tilde{X}_\tau, O_\tau)\}_{\tau=1}^{t-1}$ to predict the $\frac{1}{1+h}$ quantile on $\tilde{X}_t$.
- Feature-based adaptive inventory algorithm (FAI) [DHR24]: FAI is an online gradient descent style algorithm aiming to minimize the cost $\sum_{t=1}^T h \cdot (a_t - O_t)^+ + l \cdot (O_t - a_t)^+$ (we set $l = 1$ in our experiments). Specifically, it chooses $a_t = \tilde{w}_t^\top \tilde{X}_t$, where

$$\tilde{w}_t = \begin{cases} \tilde{w}_{t-1} - \frac{h}{\sqrt{t}} \cdot \tilde{X}_{t-1}, & \text{if } O_{t-1} < a_{t-1}, \\ \tilde{w}_{t-1} + \frac{l}{\sqrt{t}} \cdot \tilde{X}_{t-1}, & \text{otherwise,} \end{cases}$$

is the online gradient descent step and $\tilde{w}_0$ can be randomly sampled in $[0, 1]^d$.

- **Alg***: Given H_t from a task with a pool of $|\Gamma|$ environments $\{\gamma_1, \dots, \gamma_{|\Gamma|}\}$, the action a_t is chosen by the posterior median as shown in Algorithm 2. To compute the posterior distribution, we denote $\bar{\epsilon}_\gamma$ and β_γ as the noise upper bound and demand function parameter of γ at $\tau \leq t-1$, and define the event $\mathcal{E}_{\gamma,\tau} = \left\{0 \leq O_\tau - \beta_\gamma^\top \tilde{X}_\tau \leq \bar{\epsilon}\right\}$ to indicate the feasibility of environment γ from $(\tilde{X}_\tau, O_\tau)$, and denote $\bar{\mathcal{E}}_{\gamma,t} = \bigcap_{\tau=1}^{t-1} \mathcal{E}_{\gamma,\tau}$ to indicate the feasibility at t . Then the posterior distribution of the underlying environment is

$$\mathcal{P}(\gamma_i | H_t) = \frac{\mathbb{1}_{\bar{\mathcal{E}}_{\gamma_i,t}} \cdot \bar{\epsilon}_{\gamma_i}^{1-t}}{\sum_{i'=1}^{|\Gamma|} \mathbb{1}_{\bar{\mathcal{E}}_{\gamma_{i'},t}} \cdot \bar{\epsilon}_{\gamma_{i'}}^{1-t}}.$$

C.4.3.5 Queuing Control

- **Random**: The algorithm samples an action a_t uniformly from the action space $\mathcal{A}$ at each time step t .

C.4.3.6 Revenue Management

- **Adaptive Allocation Algorithm (Ada)** [CLY24]: This is a re-solving-based algorithm that recalculates a linear optimization problem at each time step, using the sample counts as estimates for the unknown arrival probabilities, as outlined in Algorithm 4.

Algorithm 4 Adaptive Allocation Algorithm[CLY24]

- 1: Input: $T, \mathcal{X} = \{(\tilde{r}^k, \tilde{\mathbf{A}}^k)\}_{k=1}^K$
- 2: Initialize $\mathbf{B}_1 = T \cdot \mathbf{1}, \mathbf{b}_1 = \mathbf{B}_1/T$
- 3: Set $y_1 = 1$
- 4: **for** $t = 2, \dots, T$ **do**
- 5: Compute $\mathbf{B}_t = \mathbf{B}_{t-1} - \tilde{\mathbf{A}}_{t-1} y_{t-1}$
- 6: Compute $\mathbf{b}_t = \mathbf{B}_t/(T - t + 1)$
- 7: Solve the following linear program where the decision variables are $(y'_1, \dots, y'_K)$:

$$\begin{aligned}
 \max \quad & \sum_{k=1}^K \frac{n_{t-1}(k)}{t-1} \tilde{r}^k y'_k \\
 \text{s.t.} \quad & \sum_{k=1}^K \frac{n_{t-1}(k)}{t-1} \tilde{\mathbf{A}}^k \cdot \mathbf{y}'_k \leq \mathbf{b}_t \\
 & 0 \leq y'_k \leq 1, \quad k = 1, \dots, K
 \end{aligned} \tag{C.4}$$

- 8: Denote the optimal solution as $\mathbf{y}'_t = (y'_{1,t}, \dots, y'_{K,t})$
- 9: Observe $(\tilde{r}_t, \tilde{\mathbf{A}}_t)$ and identify $(\tilde{r}_t, \tilde{\mathbf{A}}_t) = (\tilde{r}^k, \tilde{\mathbf{A}}^k)$ for some k
- 10: Set

$$y_t = \begin{cases} 1, & \text{with probability } y'_{k,t} \\ 0, & \text{with probability } 1 - y'_{k,t} \end{cases}$$

when the constraint permits; otherwise set $y_t = 0$.

- 11: Update the counts

$$n_t(k) = \begin{cases} n_{t-1}(k) + 1, & \text{if } (\tilde{r}_t, \tilde{\mathbf{A}}_t) = (\tilde{r}^k, \tilde{\mathbf{A}}^k) \\ n_{t-1}(k), & \text{otherwise} \end{cases}$$

- 12: **end for**
 - 13: Output: $\mathbf{y} = (y_1, \dots, y_T)$
-

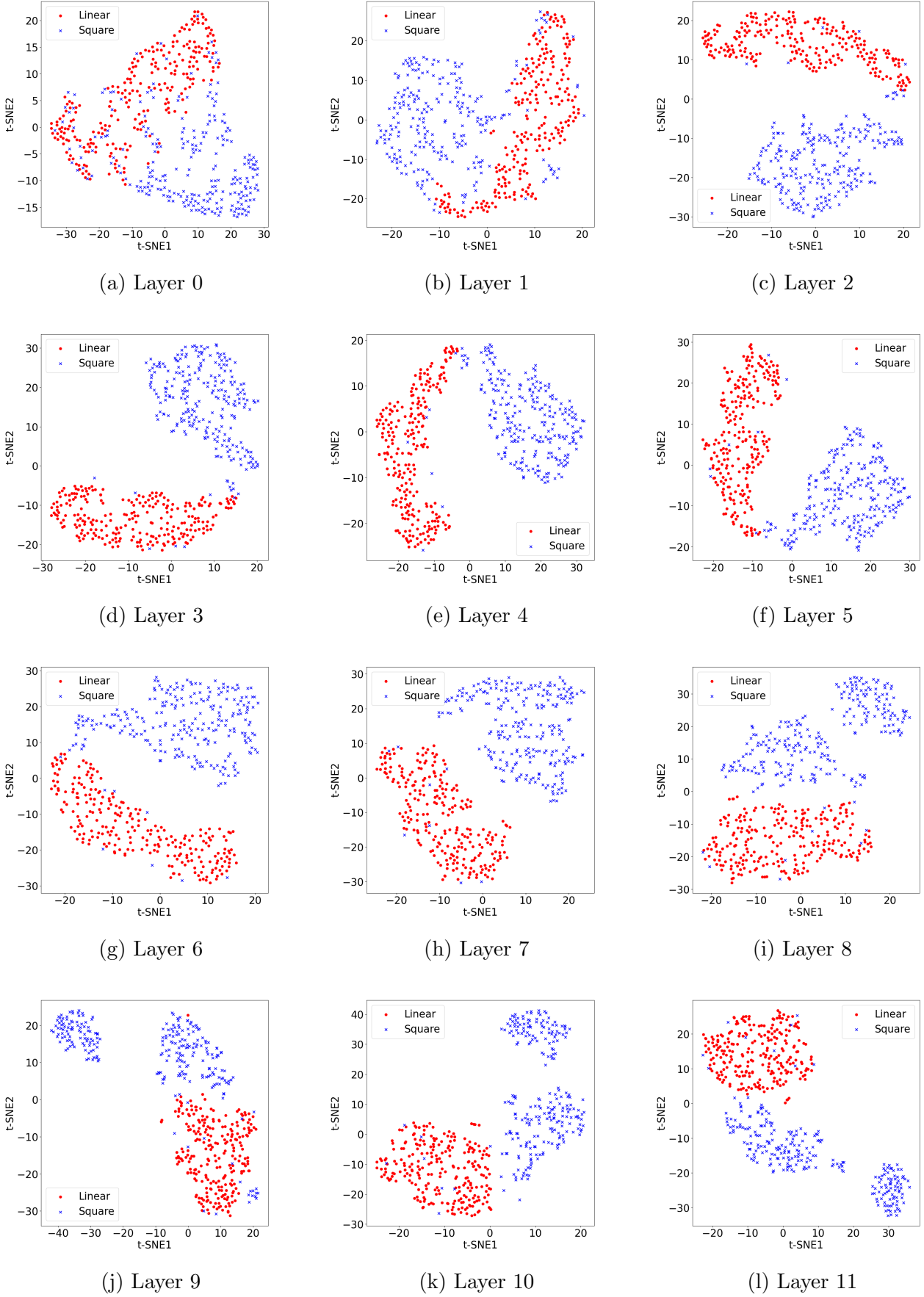


Figure C.10: Visualization of output embedding vectors from different layers. The vectors corresponding to different demand function types become clearly separated starting from layer 2, indicating that environment inference occurs during the processing. We omit the layer 12 here.