TabuLLM: Feature Extraction and Interpretation of Text Columns in Tabular Data Using Large Language Models

Mansour T.A. Sharabiani[a,*], Alex Bottle[a], Alireza S. Mahani[b]

[a]*Imperial College London, London, UK*
[b]*New York Stock Exchange, New York, USA*

Abstract

TabuLLM is a Python package for extracting and interpreting LLM-based text features from datasets that combine free-text and structured columns. Motivated by a clinical study predicting postoperative acute kidney injury in paediatric cardiopulmonary bypass patients from electronic health records, the package integrates large language model (LLM) embeddings into `scikit-learn`-compatible pipelines for tabular datasets containing text columns alongside structured features. The package provides three core components: 1) `TextColumnTransformer` wraps the `LangChain` embedding ecosystem in a `scikit-learn` transformer interface with native multi-column handling; 2) `GMMFeatureExtractor` extends `scikit-learn`'s Gaussian mixture model with a `transform()` method that returns per-cluster log-joint features for dimensionality reduction; and 3) `ClusterExplainer` generates natural language cluster descriptions using LLMs with recursive summarization, optional outcome-based statistical testing, and narrative synthesis. On a fraud detection benchmark (4.8% fraud rate), incorporating text features via `TabuLLM` improves PR-AUC from 0.64 (structured features only) to 0.82 (text concatenation), with per-column stacking ensembles reaching 0.91. `TabuLLM` thus lowers the barrier to text–tabular integration: state-of-the-art embeddings become a drop-in addition to existing `scikit-learn` pipelines, with interpretability tools to understand what the text features capture. The package is available via `pip install tabullm`.

*Corresponding author

Email addresses: `mt5605@imperial.ac.uk` (Mansour T.A. Sharabiani), `robert.bottle@imperial.ac.uk` (Alex Bottle), `alireza.s.mahani@gmail.com` (Alireza S. Mahani)

Keywords: text feature extraction, large language models, tabular data, scikit-learn, Python

Metadata

Nr.	Code metadata description	Metadata
C1	Current code version	v1.3.0
C2	Permanent link to code/repository used for this code version	`https://github.com/asmahani/TabuLLM`; `https://pypi.org/project/tabullm/1.3.0/`
C3	Code Ocean compute capsule	N/A
C4	Legal Code License	MIT
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python
C7	Compilation requirements, operating environments & dependencies	Linux, macOS, Windows; Python $\geq$3.10; numpy, pandas, scikit-learn, scipy, pydantic, langchain-core
C8	If available, link to developer documentation/manual	`https://github.com/asmahani/TabuLLM#readme`
C9	Support email for questions	alireza.s.mahani@gmail.com

Table 1: Code metadata

1. Motivation and significance

Text columns frequently appear alongside numeric and categorical variables in tabular datasets, such as clinical records with notes, job postings with descriptions, or product listings with reviews. While large language models (LLMs) provide strong semantic representations of text [1, 2, 3], integrating those representations into conventional machine-learning pipelines alongside structured features remains cumbersome.

Existing tools address parts of this workflow but leave important gaps. Traditional vectorizers such as TF-IDF [4] are easy to use in `scikit-learn` but do not capture semantic similarity, whereas modern embedding stacks accessed through `LangChain` [5] provide strong text representations without `scikit-learn`-compatible interfaces for mixed-type tabular pipelines. `Scikit-LLM` [6] connects `scikit-learn` and LLMs for supervised tasks, but its text vectorizers assume one-dimensional string arrays, require manual concatenation for multi-column datasets, and do not provide dimensionality reduction, clustering, or interpretation tools. The `llm-feature-gen` package [7] generates interpretable categorical features through direct prompting

rather than dense embeddings and probabilistic dimensionality reduction. TabText [8] applies LLM embeddings to clinical EHR rows, but is domain-specific rather than a general-purpose set of composable `scikit-learn` components. `AutoGluon` Multimodal [9] jointly models text and tabular data, but as a black-box AutoML system rather than reusable pipeline primitives. On the interpretation side, `ClusterLLM` [10] improves clustering itself through LLM feedback, while `BERTopic` [11] produces keyword-based topic representations for pure text corpora rather than natural-language explanations of text columns within mixed-type tabular datasets. Meanwhile, `scikit-learn`'s `GaussianMixture` lacks a `transform()` method, leaving a gap for GMM-based feature extraction in pipelines.

For a broader overview, Fang et al. [12] survey LLM applications to tabular data, and related work has also focused on converting structured variables into text for LLM consumption or using LLMs directly for tabular prediction [13, 14, 15, 16]. `TabuLLM` addresses the complementary problem: extracting structured representations from *text* columns for integration into user-controlled `scikit-learn` workflows, while also providing LLM-based cluster explanation.

2. Software description

2.1. Software architecture

`TabuLLM` is organized into three modules corresponding to stages in a text feature extraction workflow (Figure 1):

1. **Embedding** (`tabullm.embed`): Converts text columns to dense numerical vectors via `LangChain`-supported models. Exports `TextColumnTransformer`.
2. **Clustering** (`tabullm.cluster`): Provides dimensionality reduction optimized for embeddings. Exports `GMMFeatureExtractor`.
3. **Explanation** (`tabullm.explain`): Generates natural language cluster interpretations using LLMs. Exports `ClusterExplainer`.

A `data` module provides `load_fraud()`, which on first call automatically downloads the fraud detection dataset from Zenodo [17] (no credentials required); the dataset originates from **(author?)** [18]. All transformer classes inherit from `scikit-learn`'s `BaseEstimator` and `TransformerMixin`, enabling integration with `Pipeline`, `ColumnTransformer`, and cross-validation utilities.

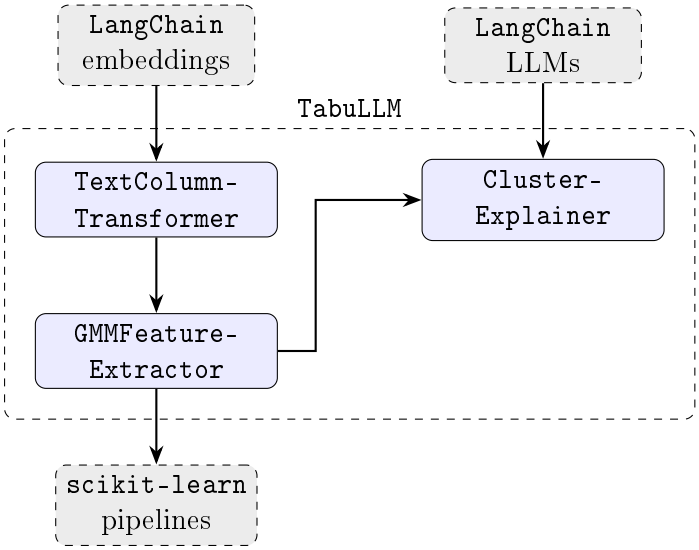


Figure 1: Architecture of `TabuLLM`. Solid boxes are package modules; dashed boxes are external dependencies. Arrows indicate data flow: text → embeddings → GMM features → downstream model, with cluster labels fed to the explainer.

2.2. Software functionalities

TextColumnTransformer.. Wraps any `LangChain`-supported embedding model (OpenAI, HuggingFace, Cohere, AWS Bedrock, etc.) in a `scikit-learn` `fit`/`transform` interface. The class handles multiple text columns natively: the `colnames` parameter maps DataFrame columns to descriptive labels in the embedding payload, and `colsep` controls the separator. An optional `normalize` parameter applies L2 normalization, making the geometric treatment of embeddings explicit regardless of provider. An `estimate_tokens()` method reports token counts and estimated API costs before embedding is invoked, complementing the `preview` mode in `ClusterExplainer` to provide cost transparency across both LLM-facing components.

GMMFeatureExtractor.. Extends `scikit-learn`'s `GaussianMixture` with a `transform()` method that returns per-cluster log-joint features – the log probability $\log p(\mathbf{x}, c_k)$ of each observation jointly with each Gaussian component. These features are aligned with the GMM's own hard assignment criterion ($k^* = \arg\max_k \log p(\mathbf{x}, c_k)$) and are strictly more informative than posterior probabilities (which are their softmax) or Mahalanobis distances (which ignore mixing weights and cluster volumes). The class reduces embedding dimensionality from hundreds or thousands to K features (typically 10–20), supports all four `scikit-learn` covariance types, and provides an

`assignment_confidence_stats()` method returning per-observation cluster quality diagnostics (maximum posterior, entropy, log-joint margin, log-density). An optional `include_log_density` flag appends the marginal log-density as an explicit outlier score.

Log-joint feature construction.. Given a fitted GMM with K components, the log-joint feature for observation $\mathbf{x}$ and component k is:

$$\ell_k(\mathbf{x}) = \log \pi_k - \tfrac{1}{2} \log |\boldsymbol{\Sigma}_k| - \tfrac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) + \text{const} \tag{1}$$

where π_k are mixing weights, $\boldsymbol{\mu}_k$ are component means, and $\boldsymbol{\Sigma}_k$ are covariance matrices. This is the quantity maximized by the GMM's own hard assignment rule ($k^* = \arg\max_k \ell_k(\mathbf{x})$), ensuring that the feature representation is *fully aligned* with the model's structure.

The log-joint decomposes into three terms: the Mahalanobis distance (shape of the cluster), the log-determinant of the covariance (cluster volume), and the log mixing weight (cluster prevalence). Mahalanobis distances ignore the latter two, producing features misaligned with the assignment criterion whenever clusters differ in size or prevalence. Membership probabilities, by contrast, are the softmax of the log-joints – the normalization $p(c_k \mid \mathbf{x}) = \text{softmax}_k(\ell(\mathbf{x}))$ discards the overall magnitude $\log p(\mathbf{x})$, collapsing information about how well any cluster fits. Concretely, an outlier equidistant from all cluster centers yields identical posteriors to a typical central observation (e.g., both return $[0.5, 0.5]$ for $K = 2$), whereas their log-joints differ substantially. `GMMFeatureExtractor` thus returns $\mathbf{L} \in \mathbb{R}^{n \times K}$ where $L_{ik} = \ell_k(\mathbf{x}_i)$, preserving both relative cluster membership and absolute typicality in K unconstrained features.

ClusterExplainer.. Generates natural language descriptions of text clusters using LLMs. Key capabilities include:

- *Automatic recursive summarization*: handles arbitrarily large clusters by hierarchically combining batch-level summaries within LLM context limits;
- *Outcome-based statistical testing*: optional one-versus-rest tests (Fisher's exact for binary, Welch's t-test for continuous outcomes) with multiple testing correction (Bonferroni, Holm, or Benjamini–Hochberg);
- *User-supplied covariates*: the `observation_stats` parameter accepts arbitrary per-observation scalars (e.g., cluster diagnostics from `assignment_confidence_stats()`) for additional statistical analysis;

- *Narrative synthesis*: a final LLM pass interprets cluster descriptions and statistical results as a coherent narrative;
- *Blind labeling*: cluster descriptions are generated without knowledge of the outcome variable, ensuring honest descriptions reusable across analytical questions;
- *Cost transparency*: `preview()` mode reports token counts and estimated costs before LLM invocation;
- *Customizable prompts*: all five internal prompts can be replaced for domain-specific tuning via the `custom_prompts` dictionary parameter (see Appendix A).

2.3. Design philosophy

Three principles guide `TabuLLM`'s architecture.

scikit-learn compatibility.. Both transformer classes inherit from `BaseEstimator` and `TransformerMixin`, adhering strictly to the `fit`/`transform` interface. This enables direct integration with `Pipeline`, `ColumnTransformer`, `GridSearchCV`, and any custom meta-estimator in the `scikit-learn` ecosystem – without any adapter code.

Vendor-agnostic LLM integration.. By depending on `LangChain` abstractions rather than any specific provider, `TabuLLM` supports a wide range of LLMs for both embedding (`TextColumnTransformer`: OpenAI, Cohere, HuggingFace, Voyage, and more) and text generation (`ClusterExplainer`: OpenAI, Anthropic, Google, AWS Bedrock, Azure, local models, and more) through a unified interface. Users can select models based on cost, performance, and data-privacy requirements without modifying application code.

Modularity and composability.. Each class is independently usable: `TextColumnTransformer` can be used without clustering or explanation; `GMMFeatureExtractor` accepts embeddings from any source; and `ClusterExplainer` accepts cluster labels from any algorithm. This allows both incremental adoption and novel compositions – for example, the stacking ensemble in Section 3 embeds column groups independently through separate `TextColumnTransformer`/`GMMFeatureExtractor` pipelines and combines their predictions via a meta-learner, a pattern enabled by the modular design.

3. Illustrative examples

We demonstrate `TabuLLM` on a fraud detection dataset consisting of 17,880 job postings (4.8% fraudulent). The dataset has seven text columns (title, location, department, company profile, description, requirements, benefits) and eight structured features (binary and categorical). Replication code for this section is in `examples/01_fraud_detection_walkthrough.ipynb`; reproducing it requires: `pip install tabullm[examples]`.

3.1. Embedding text columns

`TextColumnTransformer` converts all seven text columns to dense embedding vectors in a few lines of standard `scikit-learn`-style code:

```
from tabullm import TextColumnTransformer, load_fraud
from langchain_huggingface import HuggingFaceEmbeddings

X, y, metadata = load_fraud()

text_cols = ['title', 'location', 'department', '
    company_profile',
             'description', 'requirements', 'benefits']

embedding_model = HuggingFaceEmbeddings(
    model_name='sentence-transformers/all-MiniLM-L6-v2'
)
text_transformer = TextColumnTransformer(model=
    embedding_model)
X_emb = text_transformer.fit_transform(X[text_cols])
# X_emb.shape: (17880, 384)
```

Listing 1: Embedding seven text columns with TextColumnTransformer.

The result is a 17,880 $\times$ 384 matrix of semantic vectors. While rich in information, 384 dimensions are high-dimensional and difficult to interpret directly – motivating the next step.

3.2. Dimensionality reduction

`GMMFeatureExtractor` reduces the embedding matrix to K log-joint features, one per GMM component:

```
from tabullm import GMMFeatureExtractor

gmm = GMMFeatureExtractor(n_components=10, random_state
    =42)
```

```
X_features = gmm.fit_transform(X_emb)
# X_features.shape: (17880, 10)

cluster_labels = gmm.labels_
obs_stats = gmm.assignment_confidence_stats(X_emb)
```

Listing 2: GMM-based dimensionality reduction with GMMFeatureExtractor.

The 384-dimensional embedding is reduced to 10 features, one per cluster (11 with `include_log_density=True`). The `assignment_confidence_stats()` call returns per-observation diagnostics – maximum posterior, entropy, log-joint margin, and log-density – that characterize how confidently each observation belongs to its assigned cluster. These are passed to `ClusterExplainer` in the next step for outcome-based analysis.

3.3. Cluster explanation

`ClusterExplainer` generates natural language descriptions of each cluster and tests their association with the outcome:

```
from tabullm import ClusterExplainer
from langchain_openai import ChatOpenAI

explainer = ClusterExplainer(
    llm=ChatOpenAI(model='gpt-4o-mini'),
    text_transformer=text_transformer,
    observations='job postings',
    text_fields='title, location, department, company
       profile, '
                'description, requirements, and benefits
                   '
)

result_df, global_results, stat_assoc_df, synthesis = \
    explainer.explain(
        X, cluster_labels,
        y=y,
        y_label="fraudulent job posting (1=fraud, 0=
           legitimate)",
        observation_stats=obs_stats,
        synthesize=True,
        max_members_per_cluster=100,
        random_state=42
    )
```

Listing 3: Cluster interpretation with outcome analysis and narrative synthesis.

Cluster descriptions are generated without knowledge of `y` (blind labeling), ensuring labels reflect text content rather than fraud signal. The `observation_stats` argument passes the GMM diagnostics for outcome association testing alongside the cluster-level results. Table 2 shows a representative subset of `result_df`.

Cluster label	**Description**	**Size**	**OR**	**p-value**
Diverse Multi-Sector Workforce	Broad range of positions across oil/-gas, manufacturing, and IT; emphasizes staffing agency placement and immediate hiring needs.	2,670	6.95	< 0.001***
Financial Services & Admin Support	Customer-facing and administrative roles in financial services and collections; emphasizes bilingual capabilities and physical demands.	1,449	2.67	< 0.001***
Healthcare & Caregiving	Entry-level home health aide and CNA roles; emphasizes flexibility, compassion, and urgent staffing needs across care facilities.	1,260	1.09	n.s.
Early-Stage Growth Companies	Series A/B startups in sales and marketing; highlights rapid scaling, equity compensation, and founder mentality.	2,775	0.52	< 0.001***
Tech & Creative Professionals	Mid-to-senior engineering, design, and UX roles in global tech hubs; focuses on modern stacks, equity, and mission-driven work.	2,372	0.04	< 0.001***

Table 2: Representative cluster explanations with fraud associations generated by `ClusterExplainer` ($K = 10$, 5 of 10 clusters shown). OR: odds ratio; p-values from Fisher's exact test (one-vs-rest) with Bonferroni correction; *** $p < 0.001$, n.s. not significant.

Fraud associations span from strongly enriched (OR = 6.95 for loosely described multi-sector postings) to strongly protective (OR = 0.04 for specific tech roles). Notably, the per-observation diagnostics from `assignment_confidence_stats()` also reveal a fraud signal: the `synthesis` narrative reports that fraudulent postings exhibit significantly lower `log_joint_margin` (mean 252 vs. 316, $p < 0.001$) and `log_density` (mean 995 vs. 1054, $p < 0.001$), indicating that fraudulent content is atypical and poorly differentiated – fraudsters produce generic postings that fail to cohere around any recognizable occupational archetype.

3.4. Predictive pipeline

The same components compose directly into a predictive `scikit-learn` pipeline combining text and structured features:

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

binary_cols = metadata['binary_columns']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42)

full_pipeline = Pipeline([
    ('features', ColumnTransformer([
        ('text', Pipeline([
            ('embed', TextColumnTransformer(model=
               embedding_model)),
            ('reduce', GMMFeatureExtractor(n_components
               =10,
                                           random_state
                                              =42))
        ]), text_cols),
        ('binary', 'passthrough', binary_cols),
        # ... other non-text preprocessing
    ])),
    ('classifier', RandomForestClassifier(n_estimators
       =1000))
])

full_pipeline.fit(X_train, y_train)
```

Listing 4: Full predictive pipeline combining text and structured features.

`TextColumnTransformer` and `GMMFeatureExtractor` slot naturally into `scikit-learn`'s `Pipeline` and `ColumnTransformer` abstractions with no adapter code. Section 4 evaluates this pipeline systematically under repeated cross-validation, comparing embedding representations, column configurations, and a per-column stacking variant.

4. Experimental evaluation

All experiments use Repeated Stratified k-Fold cross-validation (RSKF; 10 repeats $\times$ 10 folds, $n = 100$ total), reporting mean $\pm$ standard error across

folds. Full replication code is in the `paper_experiments/` directory of the repository. Given the 4.8% fraud rate, we report PR-AUC as the primary metric: unlike ROC-AUC, it focuses on the minority class and avoids ceiling effects that compress inter-model differences at high performance levels [19]. Log-loss and Brier Skill Score (BSS, where 0 = no better than always predicting the marginal fraud rate and 1 = perfect) are reported as calibration-oriented complements. A structured-features-only baseline (no text) yields PR-AUC = 0.642, log-loss = 0.142, BSS = 0.424.

4.1. Diminishing returns in column concatenation

Figure 2 traces PR-AUC, log-loss, and BSS as text columns are progressively concatenated before embedding ($K = 10$), in both forward (title → benefits) and backward (benefits → title) order. The forward sweep peaks at four columns (PR-AUC = 0.824, BSS = 0.580) then plateaus; the backward sweep converges near the same level. Including all seven columns yields no improvement over the four-column subsets in either direction (PR-AUC = 0.815 at all seven). This motivates embedding columns independently so that each column's signal is captured without dilution from noisier columns. The same pattern was observed in our earlier clinical application of `TabuLLM` to predicting acute kidney injury, where per-column stacking outperformed a single model on concatenated text [3].

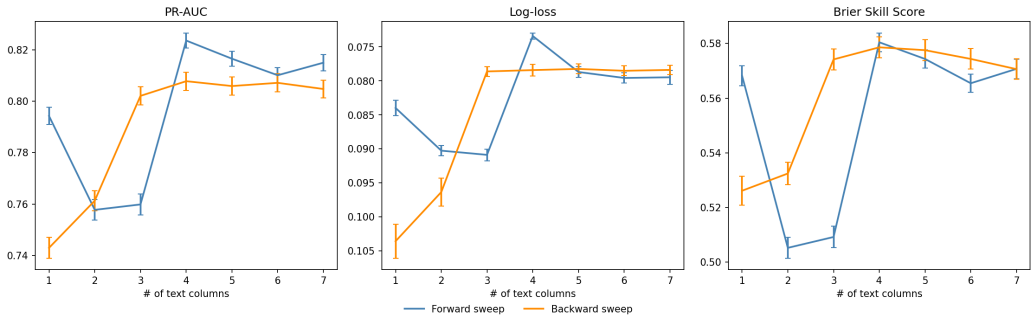


Figure 2: PR-AUC, log-loss, and Brier Skill Score (mean ± 1 SE, 100 RSKF folds) as a function of the number of text columns concatenated before embedding ($K = 10$), in forward (title → benefits) and backward (benefits → title) order.

4.2. From concatenation to stacking

`TabuLLM`'s composable design supports a natural extension: embed each column independently, reduce each to K GMM features, train one base classifier per column, and combine predictions via a logistic-regression meta-learner:

```python
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

# Each text column embedded and reduced independently
estimators = [
    (col, Pipeline([
        ('features', ColumnTransformer([
            ('gmm', GMMFeatureExtractor(n_components
               =10), emb_cols[col]),
            ('binary', 'passthrough', binary_cols),
            # ... other structured feature
               preprocessing
        ])),
        ('clf', RandomForestClassifier(n_estimators
           =100))
    ]))
    for col in text_cols
]
stacking = StackingClassifier(
    estimators=estimators,
    final_estimator=LogisticRegression(),
    cv=5
)
```

Listing 5: Per-column stacking ensemble.

With MiniLM at $K = 10$, per-column stacking achieves PR-AUC $= 0.907$, log-loss $= 0.050$, BSS $= 0.730$, compared with PR-AUC $= 0.815$, log-loss $= 0.080$, BSS $= 0.571$ for concatenation of all seven columns at the same K.

4.3. Sensitivity to representation and GMM complexity

Figure 3 compares five representations – TF-IDF with truncated SVD (384 dimensions), MiniLM, BGE-Large, OpenAI-small, and OpenAI-large – across $K \in \{5, 10, 20, 50, 100\}$ GMM components, all evaluated as per-column stacking ensembles. For all Section 4 sweeps, we use `GMMFeatureExtractor(covariance_type='spherical')` rather than the package default `full`, to keep covariance parameter growth manageable as both embedding dimensionality and K increase, especially for high-dimensional OpenAI-large embeddings. The 384-dimension SVD setting for TF-IDF was validated by additional runs at 1024, 1536, and 3072 dimensions, none of which improved performance (PR-AUC at $K = 100$: 0.928, 0.928, 0.926, 0.919 respectively), confirming that the choice does not

artificially constrain TF-IDF. All representations substantially outperform the no-text baseline (PR-AUC = 0.642). Performance rises steeply from $K = 5$ to $K = 20$ and flattens thereafter. Three of the four neural embedding representations (MiniLM, OpenAI-small, OpenAI-large) consistently outperform TF-IDF across all K and metrics; BGE-Large is roughly on par. At $K = 100$, TF-IDF reaches PR-AUC = 0.928 (BSS = 0.774) while OpenAI-large reaches PR-AUC = 0.942 (BSS = 0.806) – gaps of 1.4 pp in PR-AUC and 3.2 pp in BSS. OpenAI-small and OpenAI-large emerge as the top performers, consistent with our earlier findings in the AKI application where frontier embedding models led individual-model rankings [3].

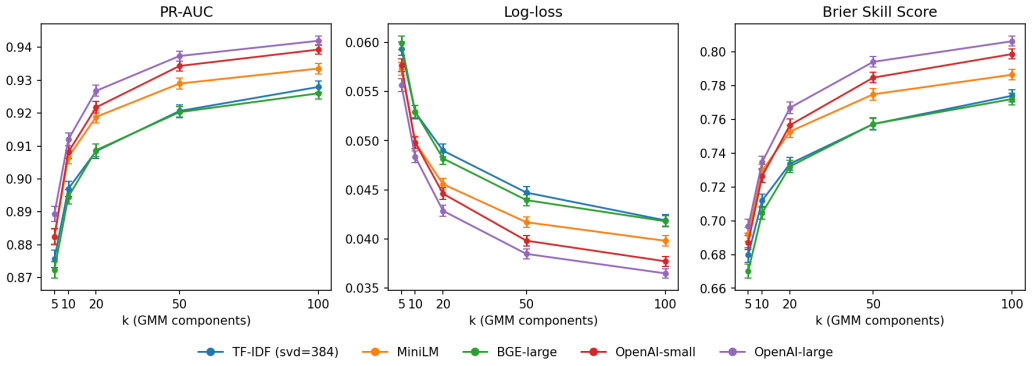


Figure 3: PR-AUC, log-loss, and Brier Skill Score (mean ± 1 SE, 100 RSKF folds) across GMM components K and five embedding representations.

5. Impact

`TabuLLM` addresses a practical gap in the ML ecosystem: although powerful text embedding models and tabular ML tools already exist, combining them still requires non-trivial integration work. By providing `scikit-learn`-compatible wrappers over `LangChain` embeddings and LLMs, the package makes multi-column text features a drop-in addition to existing tabular workflows while preserving freedom to choose models based on cost, performance, and privacy requirements. This design has already proved useful both in a clinical study of postoperative acute kidney injury [3] and on the included fraud benchmark, where adding text improves PR-AUC by 27 percentage points over the structured-features-only baseline.

Filling a gap in scikit-learn's GMM implementation.. Unlike `scikit-learn`'s `GaussianMixture`, `GMMFeatureExtractor` provides a `transform()` method, returning log-joint features aligned with the model's own assignment criterion.

This enables GMM-based dimensionality reduction in pipelines and unlocks interpretability through cluster-based analysis.

Automated, scalable interpretation.. `ClusterExplainer` provides a structured path from unsupervised discovery to interpretation through recursive summarization, blind labeling, statistical testing, and narrative synthesis. The dominant wall-clock cost is K per-cluster LLM calls, typically tens of seconds to a few minutes for $K = 10$–20 at modern provider latencies. API rate limits constrain naïve parallelism, but the per-cluster step and sibling recursion branches are structurally independent. Because the internal call graph already uses `LangChain` LCEL composition, native async `aexplain()` support with a configurable concurrency cap is targeted for `TabuLLM` v1.4.0.

Dependency stability.. `LangChain`'s rapid release cycle is managed through a layered policy: the runtime dependency is restricted to `langchain-core` only, with provider packages isolated in optional extras; version ranges are tightened to tested windows in `pyproject.toml`; Dependabot triggers weekly dependency reviews so upstream releases become reviewable pull requests before reaching users; and the CI matrix covers two `langchain-core` minor lines simultaneously to surface compatibility regressions early.

Limitations.. `TabuLLM` inherits several constraints of LLM-based pipelines. Transmitting text to cloud-hosted APIs may be unsuitable for privacy-sensitive records, motivating locally hosted models in those settings. Most embedding and generation models are optimized for English, so performance on low-resource languages or non-Latin scripts may be substantially lower. API costs scale with corpus size and cluster count, which can make large-scale use of `ClusterExplainer` expensive. Finally, hosted models such as `gpt-4o-mini` may change without a version bump; reproducible workflows should therefore pin versioned model identifiers and persist generated outputs rather than re-running LLM calls.

6. Conclusions

`TabuLLM` provides an end-to-end, `scikit-learn`-compatible workflow for extracting, reducing, and interpreting features from text columns in tabular datasets. Its three core components – embedding via `LangChain`, dimensionality reduction via GMM log-joint features, and LLM-based cluster interpretation – are modular and can be used independently or composed into full pipelines. The package is freely available on PyPI (`pip install tabullm`) with source code on GitHub.

Future development will focus on a convenience wrapper combining embedding and clustering into a single pipeline step, inclusion of a traditional text transformer for benchmarking, and expanded documentation with domain-specific examples.

CRediT authorship contribution statement

Mansour T.A. Sharabiani: Conceptualization, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing. **Alireza S. Mahani:** Conceptualization, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing. **Alex Bottle:** Supervision, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used Claude Code (Anthropic) in order to assist with drafting and editing manuscript text. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

References

[1] N. Muennighoff, N. Tazi, L. Magne, N. Reimers, MTEB: Massive text embedding benchmark (2022). arXiv:2210.07316.

[2] C. Alba, B. Xue, J. Abraham, T. Kannampallil, C. Lu, The foundational capabilities of large language models in predicting postoperative risks using clinical notes (2024). arXiv:2402.17493.

[3] M. T. A. Sharabiani, A. S. Mahani, A. Bottle, Y. Srinivasan, R. Issitt, S. Stoica, Genai exceeds clinical experts in predicting acute kidney injury following paediatric cardiopulmonary bypass, Scientific Reports 15 (1) (2025) 20847. doi:10.1038/s41598-025-04651-8.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[5] H. Chase, Langchain: Building applications with LLMs through composability, gitHub repository, available at https://github.com/langchain-ai/langchain (2023).
URL https://github.com/langchain-ai/langchain

[6] I. Kondrashchenko, et al., Scikit-llm: Sklearn meets large language models, gitHub repository, available at https://github.com/BeastByteAI/scikit-llm (2023).
URL https://github.com/BeastByteAI/scikit-llm

[7] J. Jeršova, llm-feature-gen: Discover and generate interpretable features using large language models, PyPI package, available at https://pypi.org/project/llm-feature-gen/ (2026).

[8] K. Villalobos Carballo, L. Na, Y. Ma, L. Boussioux, C. Zeng, L. R. Soenksen, D. Bertsimas, Tabtext: Language-based representations of tabular health data for predictive modelling (2022). arXiv:2206.10381.

[9] X. Shi, J. Mueller, N. Erickson, M. Li, A. J. Smola, Benchmarking multimodal AutoML for tabular data with text fields, in: NeurIPS 2021 Datasets and Benchmarks Track, 2021. arXiv:2111.02705.

[10] Y. Zhang, Z. Wang, J. Shang, Clusterllm: Large language models as a guide for text clustering, Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (2023) 13903–13920doi:10.18653/v1/2023.emnlp-main.858.

[11] M. Grootendorst, BERTopic: Neural topic modeling with a class-based TF-IDF procedure (2022). arXiv:2203.05794.

[12] X. Fang, W. Xu, F. A. Tan, J. Zhang, Z. Hu, Y. Qi, S. Nickleach, D. Socolinsky, S. Sengamedu, C. Faloutsos, Large language models (LLMs) on tabular data: Prediction, generation, and understanding – A survey, Transactions on Machine Learning Research (2024).
URL https://openreview.net/forum?id=IZnrCGF9WI

[13] S. Hegselmann, A. Buendia, H. Lang, M. Agrawal, X. Jiang, D. Sontag, Tabllm: Few-shot classification of tabular data with large language models, arXiv preprint arXiv:2210.10723 (2022).

[14] S. Han, J. Yoon, S. O. Arik, T. Pfister, Large language models can automatically engineer features for few-shot tabular learning, arXiv preprint arXiv:2404.09491 (2024). doi:10.48550/arXiv.2404.09491.

[15] B. Koloski, A. Margeloiu, X. Jiang, B. Škrlj, N. Simidjievski, M. Jamnik, Llm embeddings for deep learning on tabular data, arXiv preprint arXiv:2502.11596 (2025).

[16] N. Hollmann, S. Müller, L. Purucker, A. Krishnakumar, M. Feurer, F. Hutter, Accurate predictions on small data with a tabular foundation model, Nature 637 (2025) 319–326. doi:10.1038/s41586-024-08328-6.

[17] A. S. Mahani, M. T. A. Sharabiani, Real or fake job posting prediction dataset — zenodo archive, concept DOI; resolves to latest version. https://doi.org/10.5281/zenodo.18884001 (2024). doi:10.5281/zenodo.18884001.

[18] S. Vidros, C. Kolias, G. Kambourakis, L. Akoglu, Automatic detection of online recruitment frauds: Characteristics, methods, and a public dataset, Future Internet 9 (1) (2017) 6. doi:10.3390/fi9010006.

[19] T. Saito, M. Rehmsmeier, The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets, PLOS ONE 10 (3) (2015) e0118432. doi:10.1371/journal.pone.0118432.

Appendix A. Default `ClusterExplainer` Prompt Templates

`ClusterExplainer` uses five internal `LangChain PromptTemplate` instances, each overridable at instantiation time via the `custom_prompts` dictionary parameter. Placeholders in braces are filled at runtime from the observation data and class parameters; `{format_instructions}` and `{payload}` are populated internally. The source is `tabullm/explain.py`.

```
# label_direct
# Stage: direct labelling when all observations fit in
    one LLM call.
"Below are {nobs} {observations} organised into {
    ngroups} groups. "
```

```
"Each entry contains {text_fields}. "
"Assign each group a concise label and a description of
    no more than "
"{nwords} words that captures what makes that group
   distinctive. "
"Labels should be mutually distinct across groups.\n"
"{format_instructions}\n\n{payload}"

# label_from_summaries
# Stage: labelling from pre-built summaries (stage 2 of
    recursive flow).
"Below are summaries of {ngroups} groups of {
   observations}, "
"characterised by their {text_fields}. "
"Assign each group a concise label and a description of
    no more than "
"{nwords} words that captures what makes that group
   distinctive. "
"Labels should be mutually distinct across groups.\n"
"{format_instructions}\n\n{payload}"

# summarize_observations
# Stage: recursive summarization, depth 0 (raw
   observation batch).
"Below is a sample of {observations}. Each entry
   contains {text_fields}. "
"Write a summary of no more than {nwords} words that
   captures the common "
"themes and distinguishing characteristics of this
   group.\n\n{payload}"

# combine_summaries
# Stage: recursive summarization, depth > 0 (merging
   partial summaries).
"The following are partial summaries of the same group
   of {observations}. "
"Write a single coherent summary of no more than {
   nwords} words that "
"captures the common themes and distinguishing
   characteristics of "
"the group.\n\n{payload}"

# synthesize
```

```
# Stage: final narrative synthesis pass (used when
    synthesize=True).
"{task}\n\n"
"--- Cluster descriptions ---\n"
"{cluster_section}\n"
"{global_section}"
"{stat_section}"
```

Listing 6: Default prompt templates in `ClusterExplainer`. Comments indicate the processing stage in which each template is used.