LLMs as Feature Engineers for Text-and-Tabular Prediction

Anonymous ACL submission

Abstract

We introduce an iterative framework that automates the extraction of interpretable, schema-bound categorical features from unstructured text for tabular prediction models. To navigate the feature space, a generator LLM proposes semantic definitions, a separate extractor LLM materializes the features, and a downstream tabular model evaluates their predictive performance. We optimize this search by translating explicit model errors, such as AUC ranking inversions, into natural-language feedback, steering the LLM to resolve specific predictive failures. Evaluated across three public datasets, this error-driven loop accelerates feature discovery by up to $3\times$ compared to unguided search. Empirically, the generated features demonstrate strong multi-view complementarity, strictly outperforming any subset when combined with TF-IDF and dense embeddings. Finally, the framework guarantees instance-level interpretability: the discovered features dominate SHAP importance rankings and provide a fully transparent, semantic audit trail for every prediction.

1 Introduction

Tabular models routinely combine engineered numerical and categorical features but fundamentally struggle to incorporate raw text (Shi et al., 2021), a ubiquitous challenge in recommendation engines and click-through rate (CTR) prediction, where structured user metadata must seamlessly interact with unstructured item descriptions. Current approaches force a strict dichotomy: dense sentence-transformer embeddings (Reimers and Gurevych, 2019) provide opaque predictive power without per-prediction explanations or human-readable structure, whereas hand-crafted features offer high interpretability but are computationally expensive to design and rarely transfer across tasks.

Large Language Models (LLMs) present a compelling alternative. Repurposed as zero-shot classifiers, LLMs can extract structured, high-level semantic attributes from raw text—mapping unstructured paragraphs into discrete categorical buckets such as a product's target demographic, a headline's emotional appeal, or an item's novelty (Wang et al., 2023). This allows for the generation of features that are simultaneously interpretable and highly predictive. However, discovering the optimal semantic dimensions for a specific task without laborious hand-curation remains an open challenge.

To address this, we introduce an iterative, agentic framework that automates the extraction of structured features from raw text. Within this loop, a *generator* LLM proposes categorical feature definitions (comprising a name, a discrete value set, and a description), an *extractor* LLM annotates the dataset by applying these definitions, and a downstream model evaluates the resulting features. This architecture is strictly task-agnostic and can optimize any target objective (e.g., classification AUC, regression MSE). It requires only two components: (1) a downstream tabular model to score candidates, and (2) a mechanism to translate concrete model errors into natural-language feedback to guide the generator LLM in the next iteration.

Our central methodological finding is that standard, unguided LLM proposals or simple scalar-score feedback loops fall short of discovering optimal features efficiently. Instead, the LLM must be presented with concrete, natural-language examples of where the current model fails—such as explicit ranking inversions for an AUC objective. By prompting the LLM to generate features that specifically separate these error cases, the iterative loop produces a highly compact feature set that is strictly complementary to the existing baseline model, driving direct improvements in downstream predictive performance.

Contributions

- **An iterative LLM-as-feature-engineer loop**

for text + tabular tasks. Prior LLM feature engineering work operates strictly on tabular columns. We extend this paradigm to extract discrete, semantic categoricals directly from raw unstructured text.

- **Multi-view complementarity.** We confirm empirically across three public datasets of varying text complexity (Kickstarter, Amazon Books, Stack Overflow) that LLM-generated features, dense sentence embeddings, and classical TF-IDF capture conditionally independent signals. Combining all three text representations strictly outperforms any subset.

- **Instance-level interpretability.** Unlike dense embeddings, our discovered LLM features dominate downstream SHAP importance rankings and provide a transparent, auditable semantic trace for every individual prediction.

- **An error-driven textual feedback framework.** We demonstrate that translating downstream model errors (e.g., misranked pairs) into natural-language constraints effectively steers LLM feature generation, significantly accelerating convergence compared to an unguided search.

2 Related Work

LLMs as feature generators Recent work leverages LLMs for feature generation, but prior methods operate exclusively on structured tabular inputs—producing indicator rules (Choi et al., 2024), Python transformations (Hollmann et al., 2023; Ko et al., 2025), decision-tree features (Nam et al., 2024), or evolutionary tabular search (Abhyankar et al., 2026). Crucially, none handle unstructured raw text. While Abhyankar et al. (2026) similarly observe that structured feedback outperforms scalar scores, their scope remains strictly tabular. Furthermore, while Summary Boosting (Manikandan et al., 2023) flattens structured tabular attributes into natural-language strings, it forces the LLM to act as the final estimator, obligating a generative model to evaluate complex numerical thresholds within a prompt. Conversely, our framework operates in the exact opposite direction: it leverages the LLM strictly at design time to map unstructured text into discrete, schema-bound categories. By converting text into tabular features rather than tabular features into text, we enable a traditional classifier to natively optimize continuous numerical boundaries, delegating mathematical evaluation to the model framework best suited for it.

LLMs as evolutionary optimizers A growing literature leverages LLMs as variation operators to iteratively optimize structured artifacts. For instance, *Evolution Through Large Models* (Lehman et al., 2023) and *FunSearch* (Romera-Paredes et al., 2024) introduced this paradigm for generating code and solving combinatorial problems. This approach has since been expanded to optimize prompts (Guo et al., 2024; Fernando et al., 2024) and discover reward functions (Ma et al., 2024). Closely related is *OPRO* (Yang et al., 2024), which formalizes how an LLM can propose next-step candidates by reviewing a history of past scores.

Our iterative loop builds upon this foundational paradigm, adapting it for feature engineering through two specific design choices. First, we focus the search space on *schema-bound feature definitions* rather than free-form code. This ensures the outputs can be reliably applied as zero-shot classifiers while preserving SHAP-level interpretability. Second, instead of relying solely on scalar fitness scores, we provide the LLM with *structured, natural-language descriptions of model errors* (e.g., AUC ranking inversions). By explicitly highlighting *where* and *why* the current model is struggling, this error-driven guidance helps the LLM navigate the feature space more efficiently, significantly accelerating convergence over an unguided search. Table 1 summarizes these structural distinctions against prior optimization and feature engineering frameworks.

3 Methodology

3.1 High-Level View

Figure 1 illustrates our iterative agentic framework. At each step, a generator LLM proposes new feature definitions, an extractor materializes them across the dataset, a downstream tabular model evaluates their predictive utility, and the resulting errors are translated into natural-language feedback. Through this continuous loop, the LLM learns to propose increasingly effective features.

The system architecture is governed by three core design choices, each enforcing a critical property of the final feature set:

- **Schema-Bound Categorical Definitions.** Rather than generating dense embeddings or

Method	Input Modality	Search Space	Feedback Signal	Live Inference
CAAFE (Hollmann et al., 2023)	Tabular only	Python code transformations	Zero-shot (No iterative feedback loop)	Fast (LLM-free)
OPRO (Yang et al., 2024)	Unstructured text	Free-form text prompts	Scalar metrics (Past fitness scores)	Slow (LLM call)
Ours	Raw text → Tabular	Schema-bound categorical definitions	Textual gradients (error-based)	Fast (LLM-free)

Table 1: Conceptual comparison of LLM-driven optimization and feature engineering frameworks. Unlike prior work, our approach bridges unstructured text and tabular prediction by iteratively optimizing schema-bound categories using explicit model failures, all while maintaining LLM-free real-time inference.

free-form code, the LLM is constrained to output structured categorical definitions comprising a short name, a finite value set, and a semantic description (detailed in Section 3.2). This constraint provides three immediate benefits:

  - *Interpretability:* Features possess human-readable names and discrete states, enabling direct SHAP attribution and qualitative error analysis.
  - *Reliable Extraction:* Zero-shot classification into a small set of discrete buckets is a highly robust capability of current LLMs, avoiding the instability of continuous value prediction or arbitrary code execution.
  - *Decoupled Compute:* A feature definition is designed once by a frontier model (e.g., `GPT-5.4`) and subsequently applied across all rows by a faster, cost-effective batch model (e.g., `GPT-4.1-nano`), drastically amortizing inference costs.

- **Objective Evaluation via Downstream Models.** The LLM does not score its own proposals. Instead, every generated feature is materialized and rigorously evaluated by a fast tabular model (`HistGradientBoostingClassifier`). This grounds the search in a concrete task metric (e.g., AUC) rather than relying on the LLM's internal confidence, avoiding the well-documented pitfalls of LLM self-evaluation.

- **Feedback as the Optimization Lever.** The iterative architecture remains strictly task-agnostic; its optimization behavior is entirely dictated by the natural-language feedback presented to the generator LLM. As detailed in Section 3.4, swapping the feedback signal fundamentally shifts the system's output to serve distinct use cases.

3.2 Features as definitions

A feature definition is a structured JSON object:

```
{
  "name": "sem_review_depth",
  "values": [
    "very_detailed",
    "moderate",
    "surface",
    "barely_substantive"
  ],
  "description": "How much actual
      content the review contains.
      very_detailed = multiple
      paragraphs with specific examples;
       surface = a few sentences with
      general reactions; ..."
}
```

The LLM emits these in batches (one prompt → 90 definitions). Each definition is then applied row-by-row by a separate, cheaper enrichment LLM (`GPT-4.1-nano`) that acts as a zero-shot classifier: given the description, the row text, and the candidate values, it picks one value. Enrichment is amortised over the dataset and runs in parallel batches; for 80K rows this takes 30 minutes.

This separation matters: the generator LLM needs to reason about what dimensions of the text might matter for the task — a slow, complex job. The extractor LLM only needs to apply a single label given clear criteria — a fast, parallel job. Using the same model for both would be wasteful.

3.3 The Iterative Loop

Structurally, our agentic loop builds on the Optimization by Prompting (OPRO) framework (Yang et al., 2024). However, we extend OPRO by replacing its standard scalar-score feedback with rich, error-aligned textual constraints.

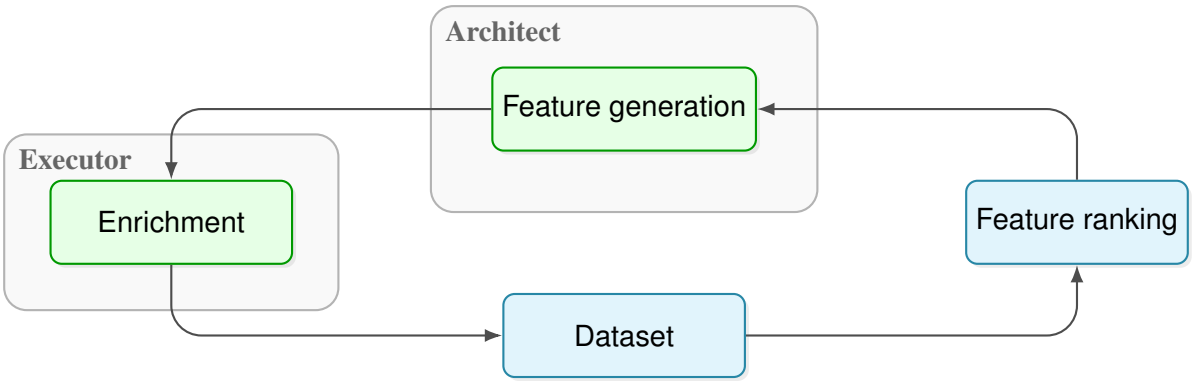

Figure 1: Agentic recommender workflow. An architect proposes features, an executor evaluates them, and feedback drives the search loop.

More precisely, at each iteration t, the framework executes the following sequence:

1. **Generate:** The *generator* LLM proposes $\sim$90 new categorical feature definitions. This generation is strictly conditioned on: (a) a "hall of fame" containing the current best-performing features (names, value sets, and descriptions), (b) a running memory of all previously evaluated features to prevent duplicates, and (c) the current feedback signal (Section 3.4).

2. **Enrich:** The *extractor* LLM acts as a zero-shot classifier, applying each newly proposed definition to the dataset to produce discrete categorical columns.

3. **Score:** Each new column is individually evaluated against the downstream task metric. The entire historical pool of generated features remains available for the subsequent selection phase.

4. **Select & Evaluate:** A greedy forward selection algorithm over the full candidate pool identifies the top-K feature set. This optimized set drives the current iteration's headline metric and updates the hall of fame for the subsequent generation step.

5. **Compute Feedback:** Based on the current model's predictions, we compute the target-specific feedback signal (detailed in Section 3.4) and formulate it as natural-language guidance for the next iteration.

The batch size of $\sim$90 definitions per iteration strikes a deliberate balance: it provides the greedy selector with sufficient diversity while remaining small enough that each round of targeted feedback tightly steers the subsequent generation step.

3.4 The Feedback Signal

The feedback signal dictates the framework's optimization behavior. To drive strict predictive improvements, we must surface concrete instances of model failure. By translating the mathematical concept of "where the model is wrong" into structured natural language, we explicitly steer the LLM's feature proposals toward complementing the current model.

For a binary classification task optimized for AUC, an "error" is defined as a ranking inversion: a true positive ranked below a true negative. We sample these misranked pairs from the current model's cross-validated predictions and present them explicitly to the generator LLM as contrastive text:

> **SUCCEEDED** (but model predicted 35%): *"Smart Herb Garden—grow fresh herbs"*
> **FAILED** (but model predicted 62%): *"Revolutionary concept changes everything"*
> $\rightarrow$ **Task:** Design a feature that explicitly separates these.

To prevent the LLM from destroying existing predictive signal, we also present well-ranked pairs as negative constraints ("the model already gets these right—do not duplicate this signal"). This loss-aligned feedback perfectly targets the downstream objective: every inversion the LLM successfully separates directly improves the target metric.

This error-driven formulation generalizes across tasks. For instance for Mean Squared Error (MSE) regression, the system surfaces items with the highest squared residuals; for ranking (e.g., NDCG), it provides misordered list positions. The underlying framework remains strictly task-agnostic; only the natural-language presentation of the error changes.

Finally, to ensure the LLM remains calibrated to the task scale, the prompt includes the current baseline metric, the total count of residual errors, and the marginal value of fixing a single instance.

3.5 Feature Selection

While the LLM acts as the generator of categorical definitions, the downstream model serves as the definitive evaluator. After the enrichment phase, we isolate the most complementary features from the generated pool using a greedy forward selection algorithm (Guyon and Elisseeff, 2003).

The procedure initializes with a baseline model trained exclusively on the existing tabular features. At each step, every candidate feature in the accumulated pool is temporarily added to the active feature set, and the model is re-evaluated on the downstream target metric. The single candidate that yields the highest performance gain is permanently absorbed into the selected set. This incremental process repeats until no remaining candidate improves the metric, or until a predefined feature budget is exhausted.

4 Experiments

4.1 Datasets and tasks

We evaluate our framework on three public text-and-tabular prediction tasks, deliberately selected to span a spectrum of unstructured text complexity—from short, dense titles to long, structured documents. To ensure consistent evaluation, all datasets are subsampled to a comparable scale (60K–80K rows) and split into stratified 5-fold cross-validation sets using a fixed random seed. Complete details regarding raw dataset sourcing, data filtering pipelines, and target variable binarization are provided in Appendix A.

Kickstarter (Short Text). 80K crowdfunding projects sourced from Kaggle, filtered to completed projects (successful or failed). The unstructured text is the project name (averaging 3–12 words). Tabular features include main category, subcategory, currency, country, `goal_usd`, `duration_days`, `launch_month`, `launch_year`, and launch hour. The target is binary project success (baseline rate: $\sim 40\%$). This represents the short-text regime, where the semantic signal is real but highly concentrated.

Amazon Books Reviews (Medium Text). 80K book reviews sampled from the Kaggle Amazon dataset, restricted to reviews with at least 5 total helpfulness votes. The text is the review body itself (median length ~ 680 characters). Tabular features include star rating, item price, review timestamp, and total helpfulness votes. The target is binary `helpful`, defined as a helpful-vote ratio of ≥ 0.7 (baseline rate: $\sim 60\%$). This represents the medium-text regime, featuring substantive, opinionated, but bounded paragraphs.

Stack Overflow Questions (Long, Structured Text). 60K programming questions from the public Kaggle dataset assessing question quality. Originally human-labeled into three tiers, we binarize the target to High Quality (HQ) versus not-HQ (baseline rate: $\sim 33\%$). The text concatenates the question title and body (median length ~ 780 characters), inherently including complex structures like code blocks, error messages, and stack traces. Tabular features include the primary tag, number of tags, body length, title length, and temporal creation features (year, month, day, hour). This represents the text-richest regime, offering the largest information capacity per row.

The Text-Complexity Spectrum. Together, these three datasets capture distinct regimes of text informativeness. Kickstarter relies on ultra-short hooks, Amazon Books provides conversational paragraphs, and Stack Overflow introduces long, multi-format technical questions. This progression explicitly allows us to evaluate how the LLM-as-feature-engineer framework scales as the unstructured text becomes increasingly central to the prediction task.

4.2 Implementation Details

Models and Architecture. Feature generation is handled by a frontier LLM (in our case `GPT-5.4`), which is invoked purely at design time to propose batches of approximately 90 new feature definitions per iteration. The subsequent row-by-row extraction is performed by a more cost-effective model (`GPT-4.1-nano`), acting as a zero-shot classifier over the text. For the downstream evaluator, we utilize a `HistGradientBoostingClassifier` from `scikit-learn` (Pedregosa et al., 2011), as it natively and efficiently handles discrete categorical variables without the need for high-dimensional one-hot encoding. For full reproducibility, the exact instruction templates used for both the generator LLM (incorporating the loss-aligned feedback) and

the extractor LLM are reproduced in Appendix B and Appendix C, respectively.

Compute and Hyperparameters. Evaluating the entire historical pool of generated features during greedy selection is computationally intensive but trivially parallelizable. We distribute candidate evaluation across 32 workers (via `joblib.Parallel`). Under this configuration, evaluating a pool of 1,500 candidate features takes approximately 5 minutes per greedy step. Across all experiments, we cap the greedy forward selection algorithm at a maximum feature budget of $K = 10$.

4.3 Prediction with LLM features

We instantiate the loss-aligned variant of our framework (Section 3.4) and evaluate our LLM-generated categorical features against two standard text representations: classical TF-IDF (top-5000 unigrams and bigrams, reduced to 50 components via TruncatedSVD) and dense sentence embeddings (`all-MiniLM-L6-v2`, reduced to 50 components via PCA). All text representations are concatenated to the tabular baseline. We report 5-fold cross-validated AUC, with significance tested via paired t-tests across folds.

Three-View Complementarity. Table 2 details the predictive performance across all feature combinations. Among the single-view text representations (top block), LLM features are the strongest standalone addition for Kickstarter and Amazon Books. On Stack Overflow, dense embeddings edge out LLM features—a shift consistent with the dataset's dense, code-heavy text structure.

Crucially, however, the three representations exhibit *conditional independence*. Across all three datasets, adding LLM features to TF-IDF yields a strictly larger improvement than either baseline alone. Adding LLM features to dense embeddings provides an even greater boost, and combining all three views (lexical + distributional + semantic) achieves the highest overall AUC. This aligns perfectly with multi-view learning theory (Blum and Mitchell, 1998; Sridharan and Kakade, 2008), which posits that conditionally independent views of the same input jointly improve performance. Much like established paradigms combining TF-IDF with dense word embeddings in NLP, our LLM categorical features extract explicit semantic dimensions that remain invisible to both traditional token-based and dense vector representations.

Error-driven feedback dramatically accelerates convergence. To isolate the value of error-aligned feedback, we ran a rigorous no-feedback ablation ("random + memory") utilizing the same generator LLM, the same 90 features per iteration, and the same 15-iteration budget. In this baseline, the LLM only sees a hall of fame of past best features and a list of features not to duplicate, but receives no error-driven guidance. Table 3 reports the final ΔAUC after 15 iterations for both variants, along with the specific iteration where our feedback-guided method eclipses the baseline's final ceiling.

While a brute-force random search with memory can eventually stumble upon useful features given enough iterations, our error-aligned feedback loop navigates the feature space with remarkable efficiency. Table 3 shows that our method strictly dominates the 15-iteration baseline across all three datasets, securing particularly notable absolute gains on Amazon Books (+0.0052 ΔAUC). However, the most profound advantage of our approach lies in its convergence speed. By explicitly targeting model failures, the feedback-guided variant achieves the baseline's absolute peak performance in a fraction of the time—requiring only 3 iterations on Amazon Books (a 5x reduction in LLM inference costs) and 4 iterations on Kickstarter. In industrial applications where LLM API budgets and design-time compute are primary bottlenecks, this ability to rapidly and intelligently converge on high-signal features makes error-driven guidance indispensable.

4.4 Interpretability: SHAP analysis of selected features

A core advantage of our framework is the semantic interpretability of LLM-generated features—a property dense sentence embeddings inherently lack. To substantiate this, we conduct a SHAP (Lundberg and Lee, 2017) analysis to explicitly quantify each feature's marginal contribution to the predictions.

For each dataset, we train the downstream model using the tabular baseline augmented with the top 10 selected LLM features. Computing the mean absolute SHAP values across the test set reveals a striking result: the generated semantic features do not merely contribute positively; they *dominate* global feature importance. Across all three domains, multiple LLM features rank strictly above the strongest hand-engineered tabular baselines.

Method	Kickstarter	Amazon Books	Stack Overflow
Tabular only	0.7444	0.7611	0.8544
+ TF-IDF SVD-50	0.7528	0.8248	0.9468
+ MiniLM PCA-50	0.7572	0.8201	**0.9510**
+ 10 LLM features (ours)	**0.7639**	**0.8366**	0.9320
+ LLM + TF-IDF	0.7662	0.8445	0.9599
+ LLM + Embeddings	0.7680	0.8481	0.9612
+ LLM + TF-IDF + Emb	**0.7692**	**0.8510**	**0.9700**

Table 2: Multi-view comparison: 5-fold cross-validated AUC for each combination of representations on each dataset. Bold values mark the best single-view text representation (top group) and the best overall combination (bottom row). LLM features are the strongest single view on Kickstarter and Amazon, while MiniLM is strongest on Stack Overflow — yet adding LLM features always improves on top of either alternative, and all three combined is strictly best across datasets.

Dataset	Method ΔAUC	Baseline ΔAUC	Reaches at iter
Kickstarter	$+\mathbf{0.0225}$	$+0.0212$	4
Amazon Books	$+\mathbf{0.0731}$	$+0.0679$	3
Stack Overflow	$+\mathbf{0.0761}$	$+0.0752$	9

Table 3: Method vs. no-feedback baseline. Both variants run for 15 iterations with identical generator/extractor models and per-iteration budgets. Our method strictly outperforms the baseline's final ΔAUC across all three datasets. More importantly, the "Reaches at iter" column demonstrates that loss-aligned feedback achieves the baseline's absolute peak performance in a fraction of the time, delivering up to a 5x reduction in LLM inference costs for the same downstream performance.

- **Kickstarter.** The most influential feature overall is the LLM-generated `sem_specificity_of_core_noun` (mean |SHAP| = 2.25), which classifies the semantic clarity of the project title. It is nearly 10× stronger than the second feature, `sem_target_audience` (0.24, also LLM-generated). By contrast, the dominant hand-engineered tabular feature, `goal_usd`, ranks only sixth (0.09). In total, 7 of the top 15 features are LLM-generated, including the top two slots.
- **Amazon Books.** The same pattern holds. The top feature is the LLM-generated `sem_content_materiality_focus` (0.59), which categorizes the conceptual versus physical nature of the review, followed by `sem_negative_target_location` (0.45). The strongest tabular feature, `review_time`, ranks third (0.35), and the user-provided `review_score` is seventh (0.16). Overall, LLM-generated features occupy 10 of the top 14 slots.
- **Stack Overflow.** The pattern reaches its strongest form on the text-richest dataset. The single most important feature is the LLM-generated `struct_title_orthography_quality` (1.10), which assesses capitalization and spelling, easily beating the strongest tabular feature, `primary_tag` (0.67). Another LLM feature, `tone_formality_vs_fragmentation` (0.40), ranks fourth. In total, 5 of the top 10 overall most-influential features are LLM-generated.

Table 4 reports the top-10 features by mean |SHAP|. As indicated by the shaded cells, LLM-generated features occupy the top slot on every dataset and completely dominate the overall rankings. For qualitative assessment, the exact JSON definitions of the top three LLM features per dataset—including their complete value sets and natural-language descriptions—are provided in Appendix D.

Instance-Level Interpretability. Beyond global trends, our framework guarantees per-prediction interpretability. Because LLM-generated features map to human-readable concepts, SHAP can decompose individual decisions into transparent audit trails. For example, a confident "helpful" Amazon review prediction traces to exact semantic

Kickstarter		Amazon Books		Stack Overflow	
sem_specificity_of_core_noun	2.25	sem_content_materiality_focus	0.59	struct_title_orthography_quality	1.10
sem_target_audience	0.24	sem_negative_target_location	0.45	primary_tag	0.67
country	0.17	review_time	0.35	creation_year	0.62
launch_year	0.15	sem_inspection_depth	0.30	tone_formality_vs_fragmentation	0.40
category	0.15	sem_review_depth	0.24	n_tags	0.37
goal_usd	0.09	sem_bias_or_agenda_signal	0.19	struct_body_formatting_stability	0.32
sem_title_community_reference_qualit	0.08	review_score	0.16	creation_hour	0.31
sem_release_context_specificity_refi	0.07	sem_textual_anchor_type	0.12	sem_environment_anchor_specificity	0.27
main_category	0.05	sem_review_subject_domain	0.10	creation_month	0.19
duration_days	0.04	struct_revision_or_editing_signal	0.10	sem_framework_magic_dependency	0.18

Table 4: Top-10 features by mean |SHAP| across the test set for each dataset. LLM-generated features are highlighted (shaded cells). The top feature on every dataset is LLM-generated, and the LLM-feature dominance strengthens as the text content of the task gets richer.

factors: engaging with concepts (`sem_content_materiality_focus=ideas_arguments`, $+0.36$), deep analysis (`sem_inspection_depth=deep`, $+0.18$), and a 5-star rating ($+0.09$). This granular transparency is fundamentally impossible with dense embeddings, where SHAP only attributes importance to opaque vector dimensions (e.g., "Dimension 14"). By making every prediction fully auditable, our approach is ideal for deployments requiring algorithmic verification by domain experts.

5 Industrial Deployment and Performance Trade-offs

To validate the real-world utility of agentic feature discovery, the framework was deployed across two live Click-Through Rate (CTR) prediction models in a high-throughput recommendation system (McMahan et al., 2013). Offline evaluation demonstrated modest but consistent predictive improvements, yielding relative Information Gain (RIG) lifts of approximately +0.1% to +0.26%. However, online A/B testing revealed that these offline metrics understated the true business impact once the models interacted with live bidding dynamics and cold-start traffic. In live traffic experiments, the LLM-augmented models achieved substantial financial gains. One traffic deployment yielded a +6.6% increase in Gross Revenue (GR) and a +6.3% lift in a monitored margin proxy. A second deployment drove a +6% margin proxy lift and a +5% improvement in Conversion-Value Utilization (CVU), supported by cold-start AUC wins in 12 out of 13 early-life hour buckets.

Crucially, these financial and predictive gains required material computational trade-offs. The inclusion of these LLM-generated categorical features and their resulting feature crosses substantially increased serving complexity. Across the two deployments, inference latency rose by 30% to 50%, and CPU time per ad increased by 25% to 30%. In one traffic segment, this latency overhead resulted in a +6.5% increase in critical timeout losses. Consequently, the final production deployments necessitated strict real-time monitoring of client error rates, underscoring that future agentic loops must explicitly optimize for strict computational budgets alongside predictive relevance.

6 Conclusion

We introduced an agentic framework that extracts schema-bound categorical features from unstructured text to enhance tabular models. We demonstrated that translating downstream ranking inversions into natural-language feedback explicitly guides the LLM through the feature space, accelerating convergence over unguided search.

Significantly, these LLM-extracted features and traditional dense embeddings capture orthogonal predictive signals. While embeddings map continuous distributional semantics, our schema-bound features extract discrete conceptual dimensions. Combining these paradigms strictly outperforms either approach in isolation. Beyond predictive gains, these discrete features resolve the interpretability bottleneck of dense embeddings, enabling SHAP to generate transparent semantic audit trails for every individual prediction. This orthogonal complementarity suggests agentic feature discovery is a highly promising direction for unlocking the full value of unstructured text. Future work will extend this framework to multimodal inputs, continuous targets, and open-weights models.

Limitations

While our framework effectively bridges unstructured text and tabular prediction, it faces three primary limitations. First, relying on proprietary models (e.g., `GPT-5.4`) introduces API dependencies and limits exact reproducibility. Future work should evaluate whether open-weights alternatives (e.g., Llama 3) can reliably execute this error-driven search. Second, our evaluation is restricted to English datasets. It remains unclear how well the generator's semantic priors transfer to low-resource languages or specialized, jargon-heavy domains without fine-tuning. Finally, although LLM computation occurs offline, adding new categorical features still impacts live serving. In our industrial deployment, expanded feature vectors increased inference latency by 30% to 50%. In latency-critical environments, practitioners must balance predictive gains against computational costs, potentially by restricting the maximum feature budget.

Ethical Considerations

While our framework improves predictive performance and interpretability, its deployment introduces three key ethical risks:

- **Propagation of Bias:** Generator and extractor LLMs can introduce historical, cultural, or linguistic biases from their training data into the engineered semantic features, which the downstream model may subsequently amplify.
- **Metric Over-Optimization:** Optimizing loops strictly for downstream performance or commercial metrics, such as our observed $+6.6\%$ Gross Revenue lift, can inadvertently incentivize polarizing content. We actively mitigate this risk through instance-level SHAP interpretability, which provides a transparent semantic audit trail enabling human-in-the-loop verification and moderation.
- **Data Privacy:** Processing raw, user-generated text fields that may contain personally identifiable information (PII) introduces privacy vulnerabilities during API-based extraction, necessitating strict data-masking and anonymization preprocessing.

AI Assistant Acknowledgment

During the preparation of this work, the authors utilized Large Language Models (LLMs) to assist with brainstorming experimental designs, generating and debugging code for the empirical pipeline, and refining the prose of the manuscript. All AI-assisted outputs were rigorously reviewed, edited, and verified by the human authors, who assume full responsibility for the final contents and claims of this paper.

References

Nikhil Abhyankar, Parshin Shojaee, and Chandan K. Reddy. 2026. LLM-FE: Automated feature engineering for tabular data with LLMs as evolutionary optimizers. *Transactions on Machine Learning Research*.

Avrim Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of COLT*, pages 92–100.

Sungwon Choi, Hyeonsu Jeong, Sangdoo Yun, and 1 others. 2024. Large language models can automatically engineer features for few-shot tabular learning. In *Proceedings of ICML*.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2024. Promptbreeder: Self-referential self-improvement via prompt evolution. *Proceedings of ICML*.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024. Evoprompt: Connecting LLMs with evolutionary algorithms yields powerful prompt optimizers. In *Proceedings of ICLR*.

Isabelle Guyon and André Elisseeff. 2003. An introduction to variable and feature selection. *Journal of Machine Learning research*, 3(Mar):1157–1182.

Noah Hollmann, Samuel Müller, and Frank Hutter. 2023. Large language models for automated data science: Introducing caafe for context-aware automated feature engineering. In *Proceedings of NeurIPS*.

Jeonghyun Ko and 1 others. 2025. Ferg-llm: Feature engineering by reason generation large language models. In *Proceedings of NAACL*.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. 2023. Evolution through large models. In *Handbook of evolutionary machine learning*, pages 331–366. Springer.

Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. *Proceedings of NeurIPS*.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Eureka: Human-level reward design via coding large language models. In *Proceedings of ICLR*.

Hariharan Manikandan, Yiding Jiang, and J. Zico Kolter. 2023. Language models are weak learners. *Proceedings of NeurIPS*.

H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, and 1 others. 2013. Ad click prediction: a view from the trenches. In *Proceedings of KDD*, pages 1222–1230.

Jaehyun Nam, Kyuyoung Kim, Seoyeon Oh, Jihoon Tack, Jihun Kim, and Jinwoo Shin. 2024. Optimized feature generation for tabular data via llms with decision tree reasoning. In *Proceedings of NeurIPS*.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, and 1 others. 2011. Scikit-learn: Machine learning in python. *Journal of machine Learning research*, 12:2825–2830.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of EMNLP*, pages 3982–3992.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475.

Xingjian Shi, Jonas Mueller, Nick Erickson, Mu Li, and Alexander J. Smola. 2021. Benchmarking multimodal automl for tabular data with text fields. *Preprint*, arXiv:2111.02705.

Karthik Sridharan and Sham M. Kakade. 2008. An information theoretic framework for multi-view learning. *Proceedings of COLT*.

Zhiqiang Wang, Yiran Pang, and Yanbin Lin. 2023. Large language models are zero-shot text classifiers. *arXiv*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *Proceedings of ICLR*.

A Dataset construction

All three datasets are derived from public Kaggle releases and processed into a uniform schema: a single text column, a small tabular metadata table, and a binary target. Subsampling and stratification use a fixed random seed (42) for full reproducibility.

A.1 Kickstarter

Source: `kemical/kickstarter-projects` on Kaggle (`ks-projects-201801.csv`, 378K rows). **Filtering:** we keep only projects whose final state is `successful` or `failed` (dropping `live`, `canceled`, `suspended`, and `undefined`), then uniformly downsample to 80K rows, preserving the original class balance ($\sim$40% successful). **Text column:** `name` (the project title, 3–12 words). **Tabular columns:** `main_category`, `category`, `currency`, `country`, `goal` (converted to USD as `goal_usd`), `deadline` and `launched` converted to `duration_days`, `launch_year`, `launch_month`, `launch_dayofweek`, `launch_hour`. **Label:** `success` $= 1$ if final state is `successful`, else 0.

A.2 Amazon Books reviews

Source: the Amazon Books Reviews dataset on Kaggle (`Books_rating.csv`, 3M reviews). **Filtering:** we restrict to reviews with at least 5 total helpfulness votes (`helpful_total` ≥ 5) to avoid noisy single-vote signals, then uniformly downsample to 80K rows. **Text column:** `review/text` (median length $\sim$682 characters). **Tabular columns:** `review/score` (1–5 stars), `Price`, `review/time` (Unix timestamp, converted to years since 2000), `helpful_total` (total votes). **Label:** `helpful` $= 1$ if helpfulness ratio `helpful_positive / helpful_total` ≥ 0.7, else 0 (baseline rate $\sim$60%).

A.3 Stack Overflow questions

Source: **Source:** `stackoverflow/60k-stack-overflow-questions-with-quality-rate` on Kaggle (60K rows, three quality tiers). **Label binarization:** we collapse the three original labels (`HQ`, `LQ_EDIT`, `LQ_CLOSE`) into a binary target: `high_quality` $= 1$ if `HQ`, else 0 (baseline rate $\sim$33%). **Text column:** `text` formed by concatenating `Title` and `Body` with `"|||"` as separator (median length $\sim$780 characters, containing HTML, code blocks, error messages, and stack traces). **Tabular columns:** `primary_tag` (first tag in the `Tags` field), `n_tags` (number of tags), `body_length` and `title_length` (character counts), and temporal features `creation_year`, `creation_month`, `creation_dayofweek`, `creation_hour` parsed from `CreationDate`.

B Generator prompt

The generator LLM receives a single prompt assembling six sections in order: a task instruction, a diversity requirement, the current performance baseline, available strategies, the hall of fame, the feedback signal, and the output format. We give the full template below with ⟨angled placeholders⟩ that are filled in per iteration.

```
You are a feature engineering expert. Your ONLY
job: output a JSON object with feature definitions.
Do NOT explain your reasoning. Do NOT output
anything except valid JSON.

Generate <N> features extracted from text column
<TEXT_COL> that improve the model's ability to
rank items correctly (predict which will succeed
vs fail).

## Current model performance

  Baseline AUC (tabular only): <BASE_AUC>
  Total misranked pairs to fix: <N_MISRANKED>

Each feature you propose will be scored by how
much it improves this metric.

## Where the model ranks WRONG

These pairs are ranked incorrectly --- the model
thinks the negative is more likely positive than
the positive:

  SUCCEEDED (pred=<P_succ>%): "<text_succ>"
  FAILED    (pred=<P_fail>%): "<text_fail>"
  -> Design a feature that separates these.

[ ... 5 misranked pairs in total ... ]

## Where the model ranks CORRECTLY (do not
     duplicate)

  SUCCEEDED (pred=<P_succ>%): "<text_succ>"
  FAILED    (pred=<P_fail>%): "<text_fail>"

[ ... 3 well-ranked pairs ... ]

## Current best features
<TOP_K HoF features, each with name, type, values,
description, and ScuiAUC contribution>

## Already tested (do NOT re-propose)
<comma-separated list of all past feature names>

## Output format

Output ONLY a valid JSON object. No markdown, no
explanation. Each feature must have: name
(snake_case with domain prefix like sem_ or
struct_), type ("single"), values (list of 3-10
strings), description (string).
```

C Extractor prompt

The extractor LLM receives one prompt per (row, feature) pair via the batch API. The prompt is generated automatically from the feature definition by

the `llm-enrichment-lib` library; we reproduce its structure here.

```
You are a structured-data extractor. Given the
input text, classify it into one of the listed
values for each feature. Output ONLY a valid JSON
object mapping feature name to "value;confidence"
where confidence is in [0, 1].

## Input text
<text_column_value>

## Features to extract

### <feature_name>
<feature_description>
Values: <comma-separated list of allowed values>

[ ... one block per feature in the current batch
     ... ]

## Output format

A single JSON object:
{
  "<feature_name_1>": "<value>;<confidence>",
  "<feature_name_2>": "<value>;<confidence>",
  ...
}
```

In post-processing, the confidence suffix is stripped from the value (`"clear;0.9"` → `"clear"`), and the resulting categorical column is appended to the working CSV.

D Top discovered features by dataset

For each dataset we list the three LLM-generated features with the highest mean |SHAP| (from Table 4), reproducing the exact name, type, value set, and natural-language description as proposed by the generator LLM. These are the actual features used by the final downstream model, unedited.

D.1 Kickstarter

1. `sem_specificity_of_core_noun` *(single)*
Values: *highly_specific_core_noun*, *moderately_specific_core_noun*, *broad_category_noun*, *vague_abstract_core_noun*, *no_clear_core_noun*, *unclear*.
Description: "Specificity of the main noun anchoring the title, such as dictionary, guide, album, ball, economy, or art."

2. `sem_target_audience` *(single)*
Values: *mass_market*, *niche_hobby*, *tech_enthusiasts*, *artists_creators*, *families*, *activists*, *unclear*.
Description: "Who the project targets. *mass_market* = broad appeal; *niche_hobby* = specific interest group; *tech_enthusiasts* = gadget/software people; *artists_creators* = creative community; *families* = children/parents; *activists* = social/political cause; *unclear* = no clear target."

3. `sem_title_community_reference_quality` *(single)*
Values: *community_with_specific_project*, *community_as_beneficiary*, *community_as_identity_signal*, *community_without_project_clarity*, *no_community_reference*, *unclear*.
Description: "How community references are used and whether they clarify the project."

D.2 Amazon Books

1. `sem_content_materiality_focus` *(single)*
Values: *abstract_impressions*, *ideas_arguments*, *narrative_events*, *examples_exercises*, *material_object_quality*, *mixed_materiality*.
Description: "What kind of material the reviewer is chiefly engaging: abstract impressions, ideas, narrative events, examples/exercises, or physical object quality."

2. `sem_negative_target_location` *(single)*
Values: *none*, *book_content*, *author_claims*, *edition_production*, *publisher_packaging*, *other_reviewers_or_public*, *mixed_targets*.
Description: "Where criticism is aimed: the book's content, the author's claims, the edition/production, packaging/publisher framing, other reviewers/public discourse, or mixed."

3. `sem_inspection_depth` *(single)*
Values: *glance_level*, *sampled_portions*, *substantial_portions*, *close_inspection*, *systematic_inspection*.
Description: "Apparent depth of inspection of the book or edition reflected in the review."

D.3 Stack Overflow

1. `struct_title_orthography_quality` *(single)*
Values: *professional*, *minor_issues*, *casual_lowercase*, *noisy_typos*, *chaotic_stylized*.
Description: "Overall orthographic quality of the title, capturing capitalization, spelling, and stylized noisy writing."

2. `tone_formality_vs_fragmentation` *(single)*
Values: *formal_complete*, *neutral_complete*, *informal_complete*, *telegraphic_fragmented*, *chaotic_fragmented*.
Description: "Joint view of tone formality and sentence completeness."

3. `struct_body_formatting_stability` *(single)*
Values: *stable_and_readable*, *minor_breaks*, *noticeably_irregular*, *unstable_or_messy*.
Description: "Visual stability of body formatting including paragraphs, line breaks, and transitions."