ScaleLLM: A Technique for Scalable LLM-augmented Data Systems

Ashwin Alaparthi*
University of Pennsylvania
Philadelphia, United States
aashwin@seas.upenn.edu

Paul Loh*
University of Pennsylvania
Philadelphia, United States
lohpaul9@seas.upenn.edu

Ryan Marcus
University of Pennsylvania
Philadelphia, United States
rcmarcus@seas.upenn.edu

Abstract

Large language models (LLMs) offer powerful semantic insights for data analytics, but row-by-row LLM calls quickly become prohibitively expensive in large datasets. We introduce ScaleLLM, a novel system that substantially reduces both latency and cost on text classification tasks. ScaleLLM couples LLM-generated labels on a small subset of data with a lightweight machine learning model for large-scale inference. This approach provides significant speedups—up to 37×—while maintaining accuracy close to that of a full LLM baseline, converging within 1% of its accuracy in several tasks. ScaleLLM also provides cost-accuracy trade-off projections, giving users fine-grained control over performance trade-offs. Our demonstration illustrates ScaleLLM's reusable embedding views, efficient inference architecture, and potential for integration with query optimization frameworks in LLM-augmented database systems.

CCS Concepts

• **Information systems → Clustering and classification**; **Document representation**; *Content analysis and feature selection*; *Semi-structured data.*

Keywords

large language models; cost optimization; embeddings

ACM Reference Format:
Ashwin Alaparthi, Paul Loh, and Ryan Marcus. 2025. ScaleLLM: A Technique for Scalable LLM-augmented Data Systems. In *Companion of the 2025 International Conference on Management of Data (SIGMOD-Companion '25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3722212.3725130

1 Introduction

Large Language Models (LLMs) have advanced rapidly, offering powerful capabilities for extracting semantic insights from large data systems. Several recent projects have showcased LLM integration into databases through semantic operators [5–7, 9], enabling row-level tasks such as question answering, summarization, and classification. However, these repeated LLM calls become prohibitively expensive and time-consuming at scale. For example, using OpenAI's GPT-4o-mini, it took about 30 minutes to process 60,000 rows, even with call parallelization during our experiments.

To address these scalability challenges, we introduce *ScaleLLM* — a technique that reduces LLM calls for classification tasks by pairing the LLM with a lightweight classical machine learning (ML) model and a pre-trained embedding model. Only a small subset of the data is processed by the LLM, and its outputs serve as training data for the smaller, student ML model. Augmented with generated embeddings, the smaller models then labels the remaining rows. Experimentally, we demonstrate a 37× speedup compared to calling the LLM on every row assuming reused embeddings, while maintaining accuracy nearly on par with a fully LLM-driven pipeline (i.e., calling the LLM for every row in the database). Notably, when accuracy is met, **LLM call count stays nearly constant with data size.**

Additionally, ScaleLLM provides cost-accuracy-latency trade-off projections by estimating model accuracy for different training set sizes. This enables fine-grained control over performance trade-offs that allows users to precisely and easily tune ScaleLLM for their needs. We believe that ScaleLLM represents a promising approach for future work in LLM-integrated data systems.

Through our demonstration, we hope to (1) allow participants to explore a scalable solution to reduce the overhead of row-level LLM calls, and (2) provide participants with a cost-accuracy-latency trade-off projection that provides them fine-grained control over system performance. Our demo video is available here: https://www.youtube.com/watch?v=eYKaojVkcsY

2 Related Work

Recent efforts have explored integrating large language models (LLMs) with database systems to enhance data analysis and semantic querying. Approaches include retrieving external knowledge via LLM prompts, augmenting schema information with generated text, or processing user queries that extend beyond standard relational operators. For example, SWAN [13] presents a framework for hybrid queries that combine relational data with LLM-generated context for more comprehensive results. Similarly, work by Wydmuch et al. [10] integrates LLMs by converting relational data into text documents and using prompts or fine-tuning for downstream tasks. Another recent system, GALOIS [8] converts SQL queries into sequences of LLM prompts to extract structured data. Recent work in LOTUS [7] and Palimpzest [5] have introduced declarative frameworks for making semantic queries over tables, which use a combination of embedding indexes and LLM calls to execute these queries. **While each of these system demonstrates the benefits**

*Both authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD, Berlin, Germany
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1564-8/2025/06
https://doi.org/10.1145/3722212.3725130

of extending data systems with LLM outputs, they rely on calling an LLM for every row. As a result, these systems have high computational overhead that makes them difficult to apply to large datasets in cost and latency sensitive environments.

3 System Design

This section describes ScaleLLM's embedding generation, analysis, and answer completion workflow.

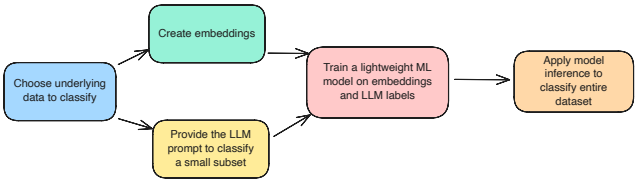

Figure 1: High level overview of ScaleLLMs workflow

As we can see from Figure 1, ScaleLLM works through a multi-stage process, with minimal user input. The user initially provides a query that selects the data they would like to classify, a prompt for the LLM to qualify a small subset, and the number of rows that make up the label subset. User data is used to create embeddings, and label a small subset using an LLM. The embeddings and labels are used to train a lightweight ML model. Finally, this model is used to classify all the rows in the database.

3.1 Embeddings

The objective of generating and storing embeddings separately is to ensure their high reusability. During the subsequent model training phase, the specific task, such as classification, and the classification questions may vary, resulting in different labels; however, the embeddings themselves remain consistent. Hence, for a given view, the embeddings can be reused to train new models, representing a one-time investment in cost and time. This provides a cheaper alternative to repeated LLM calls, which reprocess the same data with each new question.

We have two options for generating embeddings: (1) utilizing a remote embedding API, such as OpenAI's Ada-2 Text Embedding Model, or (2) executing models locally. We evaluated both approaches to assess their speed, accuracy, and overall feasibility. Embeddings are batched and generated, and stored in a new table with a link to the original user data.

3.2 Model Analysis

Based on a user's specification for a classification task, we analyze the model's expected performance given the provided data. We randomly sample up to N rows from the user query view to label using the LLM, which serves as training labels for the smaller model. Finally, we fit an accuracy curve to various splits of the labeled data to determine the ideal number of rows to label while maintaining a balance between accuracy and cost to query the LLM.

To predict model accuracy as a function of training size, we use k-fold cross-validation. Smaller models are trained on varying percentages of the labeled dataset, with accuracy and standard deviation computed across folds for each subset. We assume a normal distribution and apply maximum likelihood estimation (MLE) to fit a power-law saturation model. This allows us to extrapolate accuracy trends beyond the observed data, offering insights into the anticipated performance improvements associated with increased training sizes. We also incorporate standard deviations of observed accuracies to account for uncertainties. This step helps assess model scalability by identifying diminishing returns from adding more LLM-labeled training data.

We train four models—Random Forest, Linear SVM, RBF SVM, and Ridge Classifier—and evaluate their accuracies to identify the best-performing classifier.

3.3 Populating Answers

Once users have been provided with insights into the accuracy trends and the recommended number of rows to populate with LLM completions to achieve the desired model accuracy, the final step involves populating all rows with the model predictions. If users request additional LLM-completion rows beyond those generated in the previous stage, new labels are produced, and a new model is selected from the pool of classifiers based on the highest accuracy and trained on the expanded dataset.

Using the finalized model, we populate the new column for all rows within the user view. **In our experiments, we observed that the model completes the task in a matter of seconds for tens of thousands of rows.**

3.4 Update Maintenance

LLMScale's update system efficiently manages changes to base tables within a view, ensuring consistency in embeddings and answers. Currently, it supports only full-view updates, similar to PostgreSQL's materialized view refreshes; however, single-row updates are achievable through Incremental View Maintenance.

Initially, each view row is hashed using an algorithm robust to changes in nested SELECT statement ordering, provided semantic information remains unchanged. This hash is stored as metadata in the embeddings table. During updates, hashes are recalculated by re-executing the query; rows with altered or absent hashes trigger embedding insertions or updates. Consequently, the answers view is updated by re-invoking the model only on these modified rows.

4 Initial Results

In our experiments, we compare our system against a baseline in which a Large Language Model (LLM) processes every row. We aim to address two key questions: (1) Does our approach yield a substantial speed-up relative to the baseline? and (2) How accurate is our system compared to the baseline when evaluated against a ground-truth label set? We conducted experiments on three distinct workloads: classifying Yahoo Questions and Answers (60k rows) [12], classifying user utterances into domains (MTOP, 22k rows) [4], and identifying offensive tweets (11k rows) [11].

Figure 2 demonstrates that **ScaleLLM achieves ground-truth accuracy comparable to calling the LLM on every row**. Specifically, ScaleLLM attains within 1% of the LLM-only baseline for `5.0%`, `5.8%` and `0.5%` of the Yahoo, MTOP and Offensive Tweet datasets respectively. Notably, this high accuracy is maintained even though

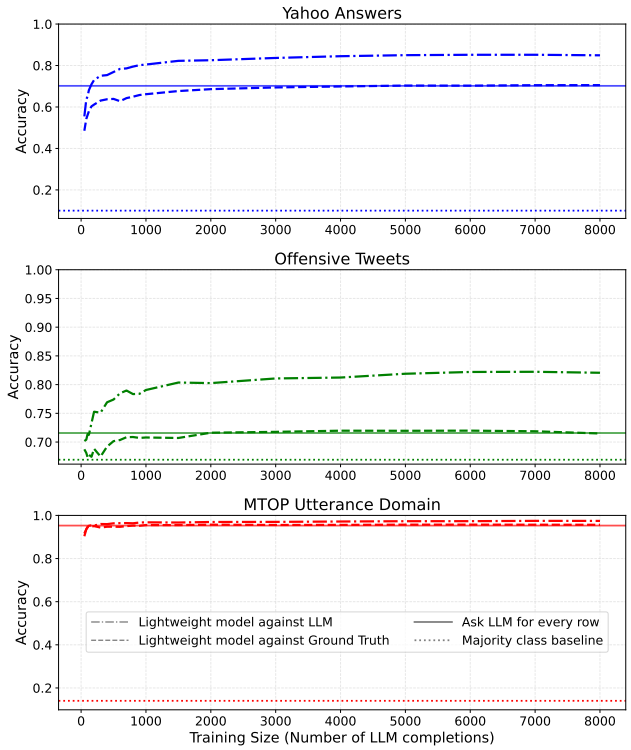

Figure 2: Accuracy of ScaleLLM on a held-out validation set of LLM completions and a predetermined ground truth. The dashed line represents the ground-truth accuracy achieved when the LLM is applied to every row. ScaleLLM converges quickly to this accuracy line while requiring LLM completions for only a small subset of rows.

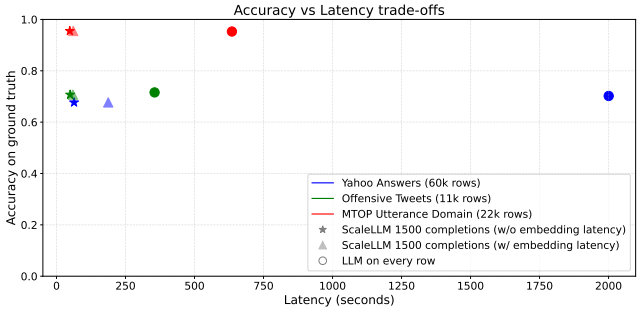

Figure 3: ScaleLLM's accuracy-latency trade-off. On the Yahoo Answers dataset, ScaleLLM attains a 37× speed-up with only a 3% reduction in accuracy against the ground truth.

ScaleLLM does not always achieve perfect accuracy with the LLM completions in its validation set.

In Figure 3 we observable the **favorable trade-off between accuracy and latency for ScaleLLM**. For our largest dataset Yahoo Answer (60k rows), ScaleLLM achieves a 37× speed-up excluding embedding latency (11× including embeddings), at the cost of a 3-point reduction in ground-truth accuracy compared to calling the LLM on every row.

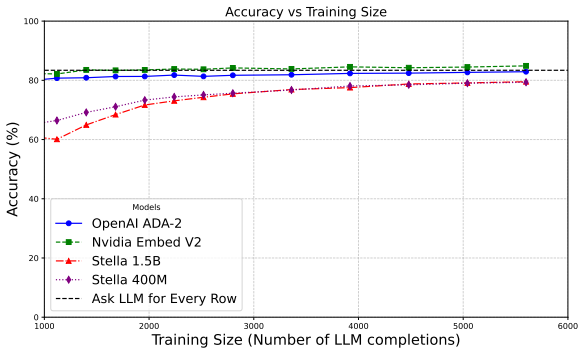

Figure 4: Accuracy of various embedding models against baseline LLM completions

To evaluate embedding models, we examined several leading embedding models tailored for classification tasks, as indicated by the Massive Text Embedding Benchmark (MTEB) rankings. We selected nvidia/NV-Embed-v2 and NovaSearch/stella_en_1.5B_v5 for their high MTEB score, and we included NovaSearch/stella_en_400M_v5 for its high performance and low GPU memory requirements, reducing the cost barrier for on-device embedding models. We also evaluated against OpenAIs Ada-2 as a remote embedding alternative. As illustrated in Figure 4, when provided with a sufficient number of training examples, **the accuracy of all evaluated embedding models converges to a level comparable to that of the LLM**. With all datasets tested, local models achieve significantly faster processing times by leveraging efficient batching and eliminating network latency, resulting in up to a 25× reduction in embedding time in certain cases.

5 Demonstration

ScaleLLM is presented through a web-based interface powered by PostgreSQL (Figure 5) that enables the audience to balance accuracy and latency when working with LLMs. The demonstration proceeds through the following stages:

Stage **1** *(Embedding Creation):* The participant initiates embedding creation by providing a query, a list of primary keys, and a prompt. For each row in the query results, ScaleLLM generates reusable embeddings based on the user-defined prompt. The prompt leverages a straightforward template syntax, enabling the insertion of column values using curly bracket notation.

Stage **2** *(Answer View Creation):* Next, the participant derives an *answer view* with data originating from a selected embedding view. In this step, the participant specifies the classification classes, a template prompt for the LLM, and the initial number of LLM completions to be used for accuracy assessment.

Stage **2a** *(Accuracy Analysis):* ScaleLLM compares the performance of a lightweight ML model against the larger LLM's outputs for the user's review. The participant may opt to collect more LLM completions to improve model performance based on the accuracy analysis. **This allows the audience to observe a projection of the accuracy-latency-cost trade-offs of the system.**

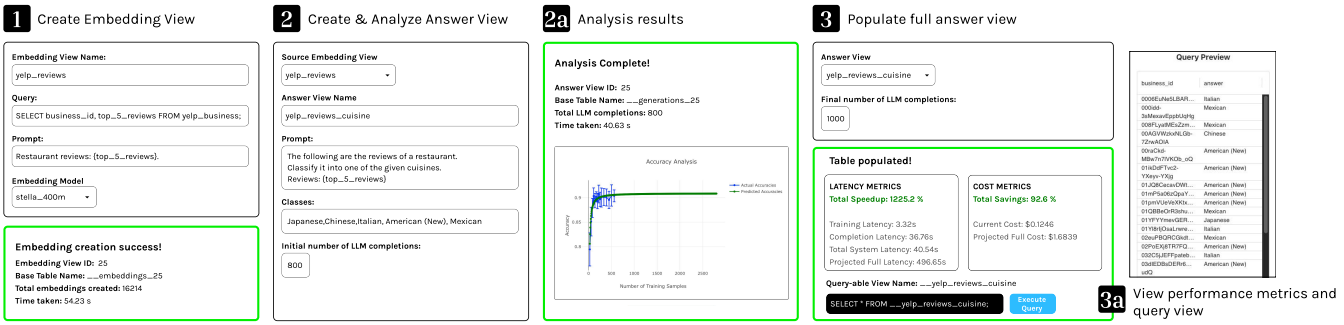

Figure 5: Demonstration flow of ScaleLLM

Stage 3 (Populating Answer View / Model Deployment): The participant then refines the target number of LLM completions based off the previous analysis. ScaleLLM gathers the required data from the LLM, selects and trains an optimal lightweight ML model, and applies this model to the remaining rows of the dataset.

Stage 3a (Performance Metrics and Queryable View): Upon completion, the system reports speed and cost statistics. **This demonstrates to the audience the scalability benefits of ScaleLLM over a fully LLM-based pipeline**. Finally, ScaleLLM produces a queryable view containing all rows from the original query, augmented with an additional `answer` column. This view can be explored and previewed in the ScaleLLM interface.

6 Future Work

As a proof of concept, ScaleLLM illustrates new avenues in scaling LLM-augmented data systems. The following outlines potential directions for further research:

Improvements to ScaleLLM Although the current web interface demonstrates the system's capabilities, ScaleLLM could be more effectively deployed as a SQL-native tool or plug-in, or incorporated into broader ecosystems of semantic data operators [5, 7]. In addition, it would be useful to automatically choose the number of training rows generated based on user preferences. This could extend approaches seen in other LLM-augmented data systems such as Palimpzest [5], which chooses among semantic execution plans along a latency-quality Pareto frontier. Further work can also be done on techniques to detect and address domain shift in underlying data to know when to re-train the lightweight model.

Technique Selection for Text Classification. Beyond harnessing LLMs, techniques like `fastText` [3] use their own bag-of-words embedding models for text classification. A future system could intelligently select among these methods, optimizing for cost, latency, or accuracy as dictated by the problem requirements.

Improving System Accuracy. Since ScaleLLM's performance converges with the quality of its training data, training data quality now becomes of renewed importance. ScaleLLM could be integrated with a human-in-the-loop system like CrowdDB to provide reliable training data [1]. Alternatively, LLM-only techniques to enhance training data quality such as through expensive ensemble approaches can also improve completion accuracy [2].

7 Conclusion

This work demonstrates how ScaleLLM avoids expensive row-level LLM calls for classification tasks by leveraging a lightweight ML model trained on a small subset of labeled rows. Experimental results show that this approach significantly improves runtime performance and lowers cost while incurring a marginal decrease in accuracy. We hope our demonstration will inspire future efforts and provide to the SIGMOD audience valuable insights on techniques and trade-offs for scaling LLM-augmented DB systems.

References

[1] Michael J. Franklin et al. 2011. CrowdDB: answering queries with crowdsourcing. In *Proceedings of the 29th ACM SIGMOD International Conference on Management of Data*. ACM, 61–72. doi:10.1145/1989323.1989331
[2] Dongfu Jiang et al. 2023. LLM-Blender: Ensembling Large Language Models with Pairwise Ranking and Generative Fusion. arXiv:2306.02561 [cs.CL] https://arxiv.org/abs/2306.02561
[3] Armand Joulin et al. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, 427–431.
[4] Haoran Li et al. 2021. MTOP: A Comprehensive Multilingual Task-Oriented Semantic Parsing Benchmark. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty (Eds.). Association for Computational Linguistics, Online, 2950–2962. doi:10.18653/v1/2021.eacl-main.257
[5] Chunwei Liu et al. 2024. A Declarative System for Optimizing AI Workloads. arXiv:2405.14696 [cs.CL]
[6] Shicheng Liu et al. 2024. SUQL: Conversational Search over Structured and Unstructured Data with Large Language Models. arXiv:2311.09818 [cs.CL] https://arxiv.org/abs/2311.09818
[7] Liana Patel et al. 2024. Semantic Operators: A Declarative Model for Rich, AI-based Analytics Over Text Data. arXiv:2407.11418 [cs.DB] https://arxiv.org/abs/2407.11418
[8] Mohammed Saeed et al. 2023. Querying Large Language Models with SQL. arXiv:2304.00472 [cs.DB] https://arxiv.org/abs/2304.00472
[9] Abirami Sukumaran. 2023. SQL-only LLM for text generation using Vertex AI model in BigQuery. https://cloud.google.com/blog/products/ai-machine-learning/llm-with-vertex-ai-only-using-sql-queries-in-bigquery. Accessed: January 25, 2025.
[10] Marek Wydmuch et al. 2024. Tackling prediction tasks in relational databases with LLMs. arXiv:2411.11829 [cs.LG] https://arxiv.org/abs/2411.11829
[11] Marcos Zampieri et al. 2019. SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media (OffensEval). In *Proceedings of the 13th International Workshop on Semantic Evaluation*. 75–86.
[12] Xiang Zhang et al. 2015. Character-level Convolutional Networks for Text Classification. *CoRR* abs/1509.01626 (2015). arXiv:1509.01626 http://arxiv.org/abs/1509.01626
[13] Fuheng Zhao et al. 2024. Hybrid Querying Over Relational Databases and Large Language Models. arXiv:2408.00884 [cs.DB] https://arxiv.org/abs/2408.00884