ADL Final Project: StreamBench Challenge
Group 32

B10705013 陳彥廷
Department of Information Management, National Taiwan University
Taiwan

B10705044 余祐徵
Department of Information Management, National Taiwan University
Taiwan

R12725022 陳沛竹
Department of Information Management, National Taiwan University
Taiwan

R13922063 張哲語
Department of Computer Science and Information Engineering, National Taiwan University
Taiwan

R13922124 呂維晟
Department of Computer Science and Information Engineering, National Taiwan University
Taiwan

Abstract

The StreamBench benchmark ([6]) is a pioneering effort to measure the continuous improvement capabilities of language model agents (LLMs) in a feedback-driven environment. By evaluating LLMs in tasks like medical diagnosis and text-to-SQL generation, StreamBench introduces an input-feedback loop to enable adaptive learning. This report details our approach to enhancing LLMs via retrieval-augmented generation (RAG), dynamic parameter tuning, and creative algorithmic design. We assess our methods through extensive experiments, offering insights into effective mechanisms for improving LLMs in real-world scenarios.

1 Introduction

Task Definition. According to [6], the scenario of StreamBench is composed of three main components, which are agent(s), an environment, and a sequence of inputs. An agent is basically an LLM that generates outputs, and can be augmented with external modules to strengthen the quality of generation. For instance, building a retrieval-augmented generation (RAG) pipeline with a retriever and a document/vector store is a common setting. The environment will provide a feedback signal given the output of the agent(s). For the two tasks in this challenge, medical diagnosis and text-to-SQL generation, the feedback signal is binary, indicating the correctness of the output instead of the ground truth. With the agent and the environment prepared, a sequence of input stream will be fed into the agent step-by-step. At each time step, the agent takes an input and generates an output. The environment then sends a feedback signal based on the output before the next iteration begins, forming an input-feedback loop. Our goal is to implement the agent(s) and develop the algorithm that update the agent between each iteration, to maximize the accuracy of the whole input stream. There are two tasks in this challenge:

(1) **Medical Diagnosis:** The agent must provide a medical diagnosis from a set of 49 possible conditions, based on patient profiles that detail their symptoms.
(2) **Text-to-SQL Generation:** The agent must transform users' natural language queries into SQL code that fulfills their data requirements.

2 Related work

Prompt Engineering. Large Language Models (LLMs) have demonstrated increasingly sophisticated reasoning capabilities through different prompting techniques, as evidenced by recent research ([1], [3], [5]). Wei et al. ([5]) introduced the concept of chain-of-thought (CoT), which involves a series of intermediate reasoning steps, and compared model performance using CoT prompting versus standard prompting. Their research demonstrated that LLMs significantly benefit from CoT prompting, achieving substantial performance improvements across complex arithmetic, commonsense reasoning, and symbolic reasoning tasks. Kojima et al. ([3]) demonstrated that LLMs perform effectively as zero-shot reasoners when preceded by the prompt "Let's think step by step" before each response. This approach highlights how simple prompting can unlock high-level, multi-task cognitive abilities in these models. With powerful reasoning capabilities, LLMs can identify and extract relevant evidence from given contexts, and provide detailed explanations of their problem-solving processes. For example, Sun et al. ([4]) introduced CARP, which prompts LLMs to perform text classification tasks through progressive reasoning, achieving significant performance gains on widely-used benchmarks.

3 Approach

3.1 Overall Framework

Description of the StreamBench setup with input-feedback loops?
Baseline implementation using RAG for memory and retrieval?

3.2 Task-Specific Approaches

3.2.1 *Medical Diagnosis.*

Two-Stage Approach. An initial model selects up to four candidate diagnoses, and a second-stage model refines them to determine the final diagnosis.

3.2.2 Text-to-SQL Generation. This task requires generating SQL code from natural language queries. Improvements were made in two key areas:

Table Extraction. Given the extensive schema and the presence of irrelevant tables for addressing specific problems, we adopted

three methods to help the language model focus on the relevant tables and minimize noise:

(1) **Cosine Similarity-Based Filtering:** This method involves calculating the cosine similarity between the user query and the table schema. A predefined threshold is applied to determine which tables and columns are relevant for the given query.
(2) **Schema Refinement via LLM:** In this approach, the language model is prompted to identify and exclude irrelevant tables and columns, subsequently generating a Data Definition Language (DDL) script that represents the refined schema.
(3) **Table Selection via LLM Without Column Removal:** This method tasks the language model with determining which tables should be retained while preserving all their columns, ensuring no single column is removed during the filtering process.

Retry Algorithm. After generating the SQL code, we connected to an empty SQLite database using the filtered schema and the generated SQL code. If syntax errors were encountered, the error messages were appended to the prompt, and the process was retried up to five times. If errors occurred three times consecutively, the model was switched to the alternative one.

3.3 Algorithmic Enhancements

Few-Shot Filtering. We enhanced the original top-k approach by introducing a cosine similarity threshold to filter the RAG-retrieved data, ensuring more relevant and accurate selections.

prompt engineering. In the medical diagnosis task, We enhance the prompt used by the Self-StreamICL agent by integrating few-shot learning with RAG-retrieved answers from prior inputs. Additionally, we incorporate Chain-of-Thought (CoT) reasoning to improve response quality. When predictions are correct, the entire response, including reasoning steps, is stored in the RAG, enabling more meaningful incremental learning. This approach has demonstrated positive outcomes.

Multi-Agentic-Memory StreamICL. To enhance performance, we utilized two models: Qwen/Qwen2.5-7B-Instruct and meta-llama/Llama-3.1-8B-Instruct. The models were alternated every 10 rounds.

Parameter fine-tuning strategies. In the text-to-SQL task, we experimented with an approach that utilized the original Self-StreamICL agent as usual while collecting correct input-output pairs. We then applied QLoRA to train an adapter and used the updated model to continue the task. Unfortunately, this approach yielded very poor performance.

4 Medical Diagnosis Experiments

4.1 Tuning the Two-Stage Approach

The two-stage approach allows the model to select a subset of labels it considers promising for further evaluation. We experimented with different limits on the number of options the model could advance to the second stage, using the Qwen model in all cases. The pink line represents the baseline model (no second stage), which achieved an accuracy of 0.51. The green line, where at most three options could proceed to the second stage, reached an accuracy of 0.62. The orange line, where at most two options were allowed, achieved the highest accuracy of 0.67.

These results suggest that limiting the number of options in the second stage can enhance performance, likely because it reduces ambiguity and prevents the model from overanalyzing less relevant labels. Allowing too many options may introduce unnecessary complexity and lead to confusion, ultimately harming the model's ability to make accurate predictions.

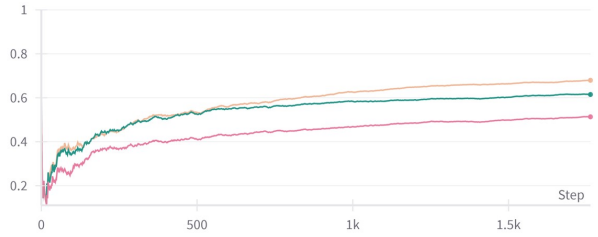

Figure 1: Impact of Different Numbers of Options in the Second Stage

4.2 Model comparison

We conducted experiments using multiple models and combined two of them to implement the Multi-Agentic-Memory (MAM) approach in StreamICL. All three experiments followed a two-stage approach with identical settings. The figure shows the rolling accuracy of the Qwen model (green line), the MAM approach combining Qwen and Llama (yellow line), and the Llama model (purple line). The Qwen model achieves the highest accuracy of 0.67, followed by MAM with 0.61, while the Llama model shows the weakest performance at 0.49.

The results highlight the importance of model selection in this task. Although the MAM approach can enhance performance, its effectiveness is strongly influenced by the choice of underlying models.

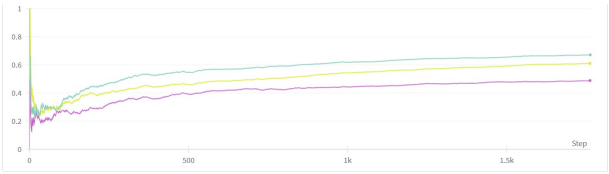

Figure 2: Performance Comparison of Different Models

4.3 Prompting Approaches

The figure below shows the performance of different prompting designs. The orange line represents the Chain of Thought (COT) technique, achieving the highest accuracy of 0.69. The two-stage approach (green line) is close, reaching an accuracy of 0.67, while the baseline model (pink line) has the lowest accuracy of 0.52.

Although COT achieves the highest accuracy, it requires significantly longer training time than the other approaches. This is

because COT generates detailed intermediate reasoning steps for each prediction, resulting in longer sequences and increased computational overhead during training. This highlights the trade-off between training time and performance.

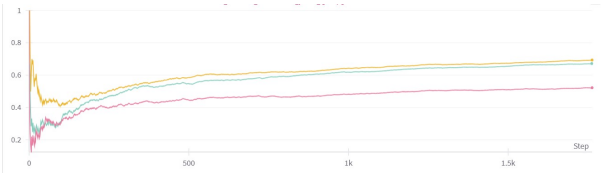

Figure 3: Performance Comparison of Different Prompt Approaches

In our subsequent experiment, we adjusted the hyperparameters and the Chain of Thought (CoT) prompt, achieving an accuracy of 0.70521. This result was submitted, with the prompt and hyperparameter details included in Appendix A.

4.4 Exploring Alternative Strategies

By analyzing the results of incorrect predictions, we observed that the model tends to favor one label over another when faced with two similar labels, often overlooking the less-predicted option. This behavior may stem from the model being heavily influenced by the information stored in RAG.

To address this issue, we experimented with modifying the two-stage approach by adding additional information to the prompt. Specifically, when the model seemed indecisive between two options, we provided feedback on how well and how frequently it had predicted each label. However, this adjustment led to a slight drop in performance. The accuracy decreased from 0.67 to 0.65, suggesting that this modification did not effectively mitigate the issue and may have introduced unintended complexities.

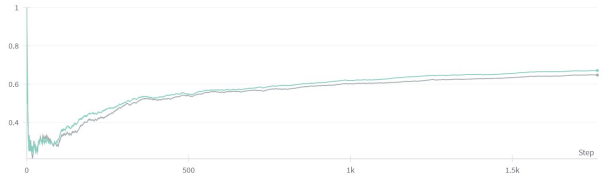

Figure 4: Performance Comparison: With and Without Additional Information in Prompts

Another approach we explored to address this problem was limiting RAG usage in the early stages while continuing to store correct predictions and their related information in RAG. Specifically, we set a 50% probability of accessing RAG during the first 40 rounds, gradually increasing this probability by 10% every 40 rounds until it reached 100%. This approach aimed to prevent the model from being overly influenced by RAG early on, which could lead to completely ignoring similar labels.

As shown in the figure, this method(green line) led to a slight improvement in accuracy during the first 250 stages. However, as time went on, it couldn't keep up with the performance of the two-stage model, suggesting that while limiting early RAG usage reduced initial bias, it wasn't enough to fully address the issue.

4.5 Results in Medical Diagnosis

The results of the experiment are tested with Qwen/Qwen2.5-7B-Instruct.

Approach	Accuracy
Self-StreamICL + CoT	**0.70521**
Self-StreamICL + CoT + Few-Shot Filtering	0.68820
TODO	?
Baseline: Self-StreamICL	0.53061

Table 1: Performance of Methods on Medical Diagnosis

5 Text-to-SQL Generation Experiments

5.1 Improvement on Retrieval

Since text-to-SQL is a relatively complex task that requires LLM to understand the input question and then generate well-structured SQL code, we have an intuition that the retrieved demonstrations (under the Self-StreamICL baseline setting) play a crucial role in augmenting the prompt. We do observe that decreasing the top_k argument in RAG leads to a slightly better performance. It's probably because including too many questions and code in the prompt may confuse the model and cause the "lost in the middle" problem. Hence we first try resampling from the retrieved demonstration set (top_k is set to 20). By applying Softmax to the negative distance of the demonstration to the retrieval query, we get a probability distribution. Then we sort these demonstrations in descending order and decide a cutoff point when the accumulated probability reaches a threshold p. The demonstrations after the cutoff index are discarded and will not be included in the prompt. Through experiments we obtain the best $p = 0.7$ with accuracy 0.311 on the public dataset.

Building on this insight, we further try improving on the aspect of retrieval. Inspired by the **F**orward-**L**ooking **A**ctive **RE**trieval augmented generation (**FLARE**) framework by [2], we implement this adaptive retrieval method. For a given question, the LLM first generates a temporary response under a zero-shot setting. If the probabilities of all tokens in the response exceed a threshold t, then this response is the final answer. Otherwise, if the probability of any token in the response is lower than t, then the retrieval is triggered and the model will regenerate the final answer under the few-shot setting. Combined with the resampling method mentioned above, we find the best $t = 0.8$ through experiments with accuracy 0.314 on the public dataset. Unfortunately, although the above methods

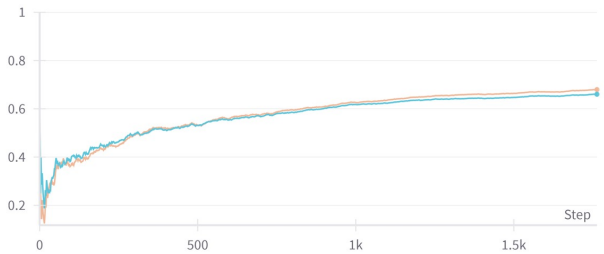

Figure 5: Impact of Limiting the Usage of RAG in Early Stages

lead to slightly better performance than the Self-StreamICL baseline in our local experiments, they don't result in better scores than the Self-StreamICL baseline on the Kaggle leaderboard. As a result, these methods are not included in the final code.

5.2 Exploring Table Extraction Strategies

A comparison of the three methods reveals their respective strengths and weaknesses. The primary drawback of the first method lies in the difficulty of determining an appropriate threshold for cosine similarity, which often results in inconsistent table filtering quality. The second method frequently generates excessive DDL code, leading to slower processing and occasional incomplete table outputs. In contrast, the third method, despite foregoing column removal, provides relatively complete and reliable data. As a result, we ultimately chose the third approach for its superior balance of effectiveness and completeness.

5.3 Parameter fine-tuning strategies

Since approaches like prompt engineering and table extraction do not directly improve the model over time, they primarily enhance the model's accuracy on individual inputs. At most, they contribute to incremental improvement by collecting more correct input-output pairs for the RAG.

Therefore, we attempted to use QLoRA to train an adapter that could directly adapt the model to this specific task.

First, we used the original Self-StreamICL to collect 100 correct input-output pairs. Then, we paused the process to train an adapter using QLoRA and resumed the task with the updated model. The resulting accuracy was 0.29726, which is lower than the baseline, indicating minimal improvement in the model's performance.

The reasons for this result are likely twofold:

(1) **Insufficient training data:** With an accuracy rate of approximately 30%, the total amount of usable training data is less than 500 examples.
(2) **Resource limitations:** Due to the 12 GB GPU memory available on our machine, training the QLoRA was constrained. We set `LoRA_r` to 32 and the batch size to 2, which may not be sufficient for effective training.

5.4 Results in Text-to-SQL Generation

Approach	Rolling Accuracy
Self-StreamICL + QLoRA	0.29726
Self-StreamICL + Demo resampling	0.31095
Self-StreamICL + LLM Table extraction	0.31226
Self-StreamICL + Adaptive RAG	0.31355
TODO	?
TODO	?
Baseline: Self-StreamICL	0.31095

Table 2: Performance of Methods on Text-to-SQL Generation

6 Discussion

THESE ARE OUTLINE GENERATED BY CHAPTGPT

Effectiveness of RAG?
Evaluation of RAG as a baseline and its limitations?
Challenges in Continuous Learning?
Issues with feedback sparsity and noisy inputs?
Task-Specific Insights?
Comparison of adaptive learning requirements between the two tasks?
Future Directions?
Suggestions for enhancing the benchmark and methodology?

7 Conclusion

Summary of findings: StreamBench demonstrates the feasibility of feedback-driven LLM improvement?
Contributions to the field: A novel benchmark and methods for continuous learning?
Call to action for further research in adaptive LLMs?

8 Work Distribution

References

[1] Jie Huang and Kevin Chen-Chuan Chang. 2023. Towards Reasoning in Large Language Models: A Survey. In *Findings of the Association for Computational Linguistics: ACL 2023*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 1049–1065. https://doi.org/10.18653/v1/2023.findings-acl.67

[2] Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active Retrieval Augmented Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Singapore, 7969–7992. https://doi.org/10.18653/v1/2023.emnlp-main.495

[3] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2024. Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, 22199–22213.

[4] Xiaofei Sun, Xiaoya Li, Jiwei Li, Fei Wu, Shangwei Guo, Tianwei Zhang, and Guoyin Wang. 2023. Text Classification via Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 8990–9005. https://doi.org/10.18653/v1/2023.findings-emnlp.603

[5] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2024. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, 24824–24837.

[6] Cheng-Kuang Wu, Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Chen, and Hung-yi Lee. 2024. StreamBench: Towards Benchmarking Continuous Improvement of Language Agents. https://doi.org/10.48550/arXiv.2406.08747 arXiv:2406.08747 [cs].

9 Appendix A

9.1 The CoT prompt in the final submission

The final CoT prompt in the medical diagnosis task.

, ,

You are an expert medical doctor tasked with diagnosing a patient.
Analyze the patient's profile step-by-step briefly and provide the single most likely diagnosis.

Possible Diagnoses (format <number>. <diagnosis>):
{option_text}

Here are some example cases.
{fewshot_text}

Now it's your turn.

Patient Profile:
{profile}

Follow this concise process:
1. Identify only the **key symptoms**.
2. Provide a **very brief reasoning** (1-2 sentences) for your diagnosis.
3. Output the final diagnosis strictly in this format: <number>. <diagnosis>.

Example Format:
Key Symptoms: [list 2-3 main symptoms]
Reasoning: [1-2 sentences explaining the diagnosis]
Diagnosis: <number>. <diagnosis>

Now provide your diagnosis.