International Journal of Social and Education (INJOSEDU) https://languar.net/index.php/INJOSEDU
Vol. 3 No. 4 July 2026, hal., 483-488 e-ISSN: 3047-6151

LLM LATENCY OPTIMIZATION ON NON-STANDARD QUERIES USING HYBRID SEMANTIC CACHE MIDDLEWARE

Agry Alfiah
Universitas Gunadarma, Indonesia
agry_alfiah@staff.gunadarma.ac.id

Abstract

This research aims to optimize Large Language Model (LLM) latency in handling non-standard language queries through a Hybrid Semantic Cache Middleware architecture. The primary challenges in cloud LLM integration are high network latency and semantic matching failures on noisy text. The proposed method integrates the all-MiniLM-L6-v2 model with a static dictionary-based Text Normalization module and threshold optimization in FAISS. Experimental results show that the system successfully increased the Hit Rate from 11.00% to 58.00% under optimal conditions (Threshold 0.65). Furthermore, the average response latency was reduced from 4,494.00 ms (Cloud Baseline) to 17.98 ms, achieving a 99.6% speedup. This implementation proves that local caching with normalization effectively minimizes operational costs and network bottlenecks for Generative AI services in Indonesia.

Keywords: Semantic Cache, Latency, LLM, Non-Standard Language, FAISS.

INTRODUCTION

Several previous studies have attempted to address the efficiency of LLM API calls through semantic memory storage methods, such as those implemented in the development of the GPTCache framework. These studies demonstrated that using embedding models for text feature extraction combined with high-performance vector search (such as FAISS) can drastically reduce asynchronous computation time (network latency). However, the main problem with implementing this standard is that the embedding models used are generally trained on formal language corpora, making the system highly susceptible to cache misses when receiving queries containing non-standard grammar. The critical weaknesses of global Natural Language Processing (NLP) models in addressing the characteristics of internet users in Indonesia have also been highlighted in the IndoNLU publication by Wilie et al. (2020) and the IndoCollex study by Wibowo et al. (2021). Users frequently insert slang, abbreviations, and emphasizing particles that are not present in the base model's training dictionary, thus classifying them as Out-of-Vocabulary (OOV). The presence of OOV words corrupts the numeric vector representation and significantly reduces the semantic closeness (cosine similarity) value. Although morphological transformations have been proposed to handle colloquial words, the focus of such research is still limited to offline translation or classification tasks, not real-time middleware interaction filtering architectures for LLM.

Based on the review, a research gap was found in the form of the absence of an adaptive text preprocessing layer in the standard Semantic Cache architecture to handle non-standard Indonesian language queries. As a solution, this study proposes the development of a Hybrid

Semantic Cache Middleware that innovatively combines the reliability of local memory lookups (FAISS and all-MiniLM-L6-v2) with the addition of a static mapping-based Text Normalization module. In addition, a threshold tuning method is applied to provide a relaxation of tolerance for free word particles, so that queries containing slang can still be recognized by the local cache without having to burden the Cloud API service.

METHOD

This research was conducted through a series of systematic stages, including literature review, data collection, architecture design, program code implementation, and performance evaluation. The primary focus of this methodology is to build an intelligent intermediary layer capable of mitigating latency constraints in LLM integration through semantic caching techniques optimized for local language characteristics.

A. Research Stages

The research phase began with system requirements analysis and relevant dataset collection. The process continued with the design of a middleware architecture that integrates text normalization and vector search functionality. The implementation was carried out using the Python programming language, utilizing the FastAPI library for endpoint management, Sentence-Transformers for model embedding, and FAISS for managing vector indexes in RAM. Comparative testing was conducted between a pure cloud service and the proposed middleware system to obtain empirical data on response time efficiency and matching success rates.

B. Proposed System Architecture

The proposed architecture is a Hybrid Semantic Cache Middleware that acts as a router between the user and the Cloud LLM API. The system workflow begins with the Normalizer module intercepting user queries to transform non-standard text into a standard format through static dictionary mapping. The normalized text is then converted into a 384-dimensional vector using the all-MiniLM-L6-v2 model. The query vector is evaluated by the FAISS engine using Cosine Similarity calculations against a set of in-memory reference vectors. A decision gate determines the processing route based on a similarity threshold of 0.65. If a Cache Hit condition is met, the system immediately returns the local answer; however, if a Cache Miss occurs, the query is forwarded to the Gemini 3.1 Flash Lite API service in the Cloud.

C. Testing Datasets and Scenarios

The datasets in this study were divided into two main categories. The first was a knowledge base containing 95 standard question-answer pairs as a cache reference. The second was a test query dataset consisting of 100 unique queries with a variety of slang, abbreviations, and typos to test the system's robustness. Testing was conducted through four main scenarios: Baseline Cloud (without middleware), Scenario A (without normalization, threshold 0.75), Scenario B (with normalization, threshold 0.75), and Scenario C (with normalization and optimization threshold 0.65). Success was measured using the Hit Rate metric to evaluate semantic matching accuracy and Latency to measure response time acceleration in milliseconds.

RESULTS AND DISCUSSION

This section presents the overall findings from testing the Hybrid Semantic Cache Middleware system, as well as an in-depth analysis of the impact of the normalization and threshold tuning components on system performance.

A. Implementation Environment and Baseline

The system was implemented using Python 3.11 with the FastAPI framework. The embedding model used was all-MiniLM-L6-v2, which maps text into 384-dimensional vectors, while FAISS served as a similarity search engine in RAM. As a baseline, testing was conducted directly against the Gemini 3.1 Flash Lite API endpoint.

Table 1. Baseline Cloud Test Results

Test Parameters	Description / Results
Target API Model	Gemini 3.1 Flash Lite Preview
Execution Location	Cloud (Google API Server)
Total Sample	Queries: 15 Queries
Response Status	(HTTP) 100% Successful (HTTP 200 OK)
Average Response Time (Latency)	4,494.00 ms (~4.5 seconds)

Baseline testing results showed an average network latency of 4,494.00 ms for 15 sample queries. This figure confirms the presence of a significant network bottleneck if the system relies entirely on cloud services.

B. Local Scenario Testing Results

Testing was conducted in three main scenarios (A, B, and C) to determine the effectiveness of the local middleware module in handling non-standard queries.

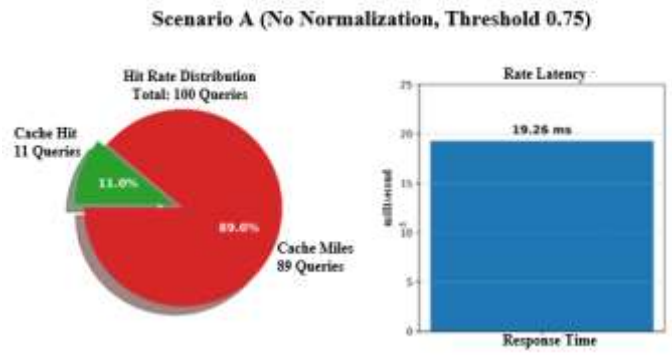

Figure 1. Graph of Scenario A Results

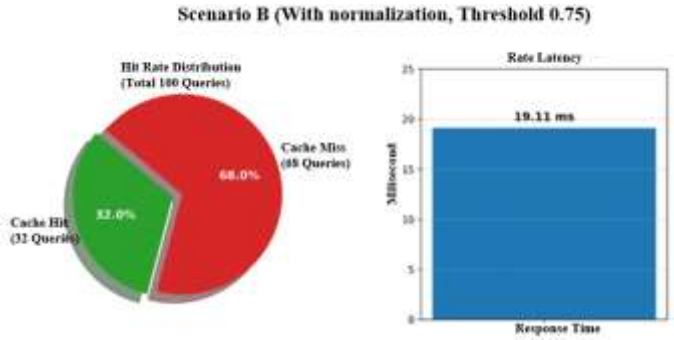

Figure 2. Graph of Scenario B Results

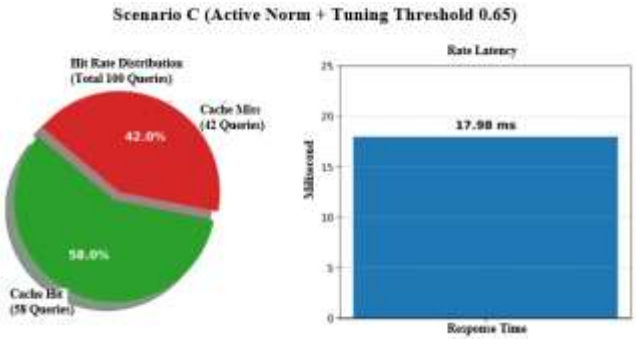

Figure 3. Graph of Scenario C Results

In Scenario A, the system used pure vector search without text normalization, resulting in a low Hit Rate of 11.00% with a latency of 19.26 ms. After activating the Normalizer module in Scenario B, the Hit Rate increased to 32.00% with a stable latency of 19.11 ms. Optimal conditions were achieved in Scenario C by setting the threshold to 0.65, which successfully boosted the Hit Rate to 58.00% and reduced latency to 17.98 ms.

C. Analysis and Discussion

Based on empirical data, the on-premises middleware implementation provided a markedly faster response time compared to the pure cloud service.

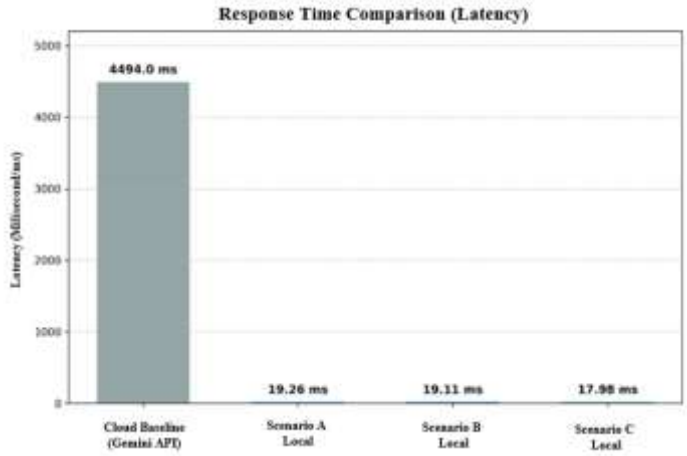

Figure 4. Latency Comparison Between Scenarios

As seen in Figure 4, shifting the load to local memory reduced the response time from 4,494.00 ms to 17.98 ms, equivalent to a time efficiency of 99.6%. This demonstrates that computational costs and internet latency can be drastically reduced through vector processing in local RAM.

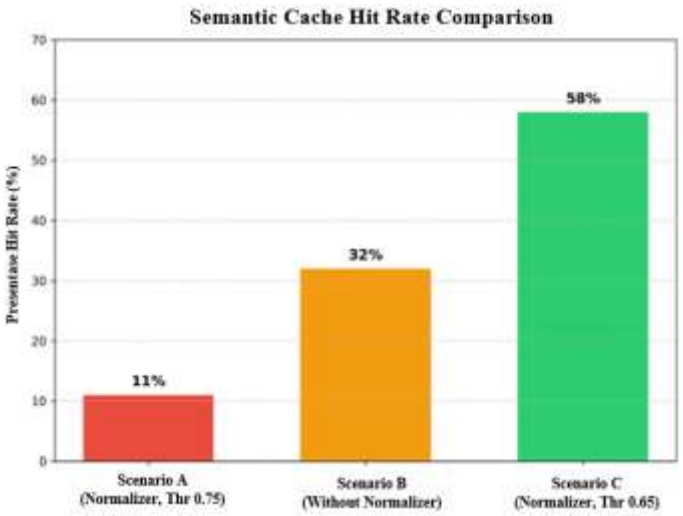

Figure 5. Comparison of Semantic Cache Hit Rates

The increase in Hit Rate from 11% to 58% (Figure 5) indicates that the Normalizer module successfully overcomes the Out-of-Vocabulary (OOV) constraint in Indonesian slang. Optimal point analysis shows that the threshold of 0.65 is the sweet spot that tolerates text noise without losing semantic accuracy. Thus, more than half of user queries can be handled locally without the burden of cloud API token costs.

CONCLUSION

Based on the results of the system design, implementation, and testing, it can be concluded that the developed Hybrid Semantic Cache Middleware architecture has successfully functioned as an effective intermediary (router) between users and LLM cloud services. This system has proven to be able to overcome network bottleneck constraints to an extreme extent by reducing the response time from an average of 4,494.00 ms to 17.98 ms, which represents a computational time efficiency of 99.6%. In addition, the implementation of a static text normalization module has proven to be crucial in handling the characteristics of Indonesian user slang and typo queries, where the system's recognition rate (Hit Rate) has increased significantly from 11.00% to 32.00%. Through optimizing the threshold tuning at 0.65 as a balance point (sweet spot), the system reached peak performance with a Hit Rate of 58.00%, which means that more than half of user queries were successfully handled locally using free resources.

RECOMEND

Although the system has yielded significant results, there is room for further research. The implementation of the normalization module can be improved from a static mapping method to a sequence-to-sequence based Machine Learning or NLP model to allow for dynamic updating of

the slang vocabulary. Furthermore, exploration of the use of embedding models specifically trained on the Indonesian language corpus, such as IndoBERT, is necessary to compare the vectorization accuracy compared to the global MiniLM model. Finally, for production-scale needs, future research is recommended to integrate cache scalability management mechanisms such as Least Recently Used (LRU) or Time-To-Live (TTL) policies to ensure RAM memory usage remains controlled as data volumes increase.

BIBLIOGRAPHY

Fu, B. dan Feng, D. 2023. *GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings*. 212–218. https://aclanthology.org/2023.nlposs-1.24/.

Google. 2025. *Generative AI Beginner's Guide*. Google Cloud Documentation. https://cloud.google.com/vertex-ai/generative-ai/docs/learn/overview.

Google Cloud. 2025. *Tiered Hybrid Pattern*. Google Cloud Documentation. https://cloud.google.com/architecture/hybrid-multicloud-patterns-and-practices/tiered-hybrid-pattern.

HuggingFace. 2023. *Sentence-Transformers: all-MiniLM-L6-v2 Model*. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.

Meta AI. 2024. *FAISS: A Library for Efficient Similarity Search and Clustering of Dense Vectors*. https://faiss.ai/.

Wibowo, H.A., dkk. 2021. *IndoCollex: A Testbed for Morphological Transformation of Indonesian Colloquial Words*. Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021. 3170–3183.

Wilie, B., dkk. 2020. *IndoNLU: Benchmark and Resources for Evaluating Indonesian Natural Language Understanding*. Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics. 843–857.

Zilliz. 2023. *GPTCache: A Library for Creating Semantic Cache for LLM Queries*. https://gptcache.readthedocs.io/en/latest/.