SLM as an Adaptive Cache Layer for LLMs

Anonymous ACL submission

Abstract

Large Language Models (LLMs) are powerful but computationally expensive, making them impractical for latency-sensitive or resource-constrained applications. This paper presents SLMCache, an adaptive caching framework that uses Small Language Models (SLMs) as semantic caches to reduce the frequency and cost of LLM invocations. Queries are first matched against a local vector store; if a semantically similar query is found, an SLM generates the response. Otherwise, the query is forwarded to the LLM, and its output is logged for future caching. The cache uses LRU and LFU eviction policies, and the SLM is periodically retrained using logged queries to expand its response coverage. Evaluated on the Bitext customer support dataset, SLMCache achieves up to 2.8× speedup and 10× lower GPU memory usage compared to LLM-only baselines, while maintaining high semantic fidelity. The framework is practical for edge deployment and significantly reduces the operational cost of LLM-based systems.

1 Introduction

Large Language Models (LLMs) like GPT-4 can be used to create chatbot conversations, write articles, and even answer questions in a more human-like manner, enabling applications in customer support(Pandya and Holia, 2023), content generation, education(Hsain and Housni, 2024), and research (Gottweis et al., 2025). However, they come with high computational costs and latency, making them less practical for real-time applications or resource-constrained environments(Hadi et al., 2023).

To address these challenges, this paper proposes a novel framework that comprises a Small Language Model (SLM) at the edge and a cloud-hosted LLM. The SLM acts as an Adaptive Cache Layer, a lightweight solution to handle frequent or straightforward queries locally, significantly reducing response latency while maintaining high-quality interactions. The system minimizes cloud dependency by hosting a localized knowledge base alongside the SLM.

A commonly raised consideration in the design of conversational agents is whether a vector database alone can suffice for effective query retrieval. While vector databases are adept at storing dense representations (embeddings) of prior queries and retrieving semantically similar entries via similarity metrics such as cosine similarity, they fundamentally lack generative capabilities. These systems are inherently static and cannot condition responses on evolving dialogue history or fine-grained query-specific context. Consequently, their applicability is limited in dynamic, multi-turn dialogue settings where contextual coherence and adaptability are critical.

Rule-based systems, driven by handcrafted logic, guarantee determinism but lack flexibility. Their dependence on predefined rules prevents adaptation to novel inputs without manual updates, and scaling to broader conversational domains incurs substantial engineering overhead. This rigidity limits their applicability in real-world scenarios that demand conversational diversity and adaptability.

Furthermore, many real-world applications demand domain-specific knowledge integration (Yang et al., 2023). For instance, customer support systems deployed by financial institutions predominantly handle inquiries related to financial services, such as account management, transaction issues, and regulatory compliance. In such contexts, deploying a full-scale large language model (LLM) with broad but largely generic knowledge can be computationally expensive. Instead, a domain-specialized small language model (SLM), fine-tuned on financial domain data, can deliver more precise and contextually appropriate responses.

Several methods exist for creating more lightweight and efficient Large Language Models (LLMs) by reducing their size, memory consump-

tion, and computational requirements while maintaining their performance. A few of the well-known approaches include:

- **Knowledge Distillation**: This technique involves training a minor "student" model to replicate the behavior of a more prominent "teacher" model, significantly reducing computational requirements while retaining most of the original model's performance (Hinton, 2015).
- **Model Quantization**: By reducing the precision of model weights and activations (e.g., from 32-bit floating-point to 8-bit integers), quantization decreases memory usage and inference latency (Jacob et al., 2018).
- **Pruning**: This involves identifying and removing redundant weights or neurons in the model. Techniques like structured and unstructured pruning help reduce the size of the model (Ma et al., 2023).
- **Low-Rank Factorization**: Weight matrices in the model are approximated using low-rank decomposition methods like Singular Value Decomposition (SVD), significantly reducing parameters and computational cost (Saha et al., 2024b).

Beyond model compression, caching and edge computing enhance efficiency. Adaptive caching reduces redundant computations by storing frequent query-response pairs (Wang and Friderikos, 2020), while lightweight models and local databases at the edge minimize reliance on cloud infrastructure (Satyanarayanan, 2017).

While these methods address specific aspects of the challenges, they often face limitations in adaptability and efficiency. For instance, caching mechanisms may struggle with query diversity, and edge computing frameworks often trade accuracy for speed.

Early adoption across research and industry favored Key-Value (KV) cache architectures, which map queries to embedding vectors for approximate retrieval. A cached response is returned if a match exists; otherwise, the LLM generates a new response (Li et al., 2024a).

There have been efforts such as (Stogiannidis et al., 2023), (Zhu et al., 2023), (Bang, 2023) using cache systems for LLM chat services to mitigate these challenges. These cache systems store dialogues, including user queries and LLM responses. Whether operating their own LLMs or utilizing public ones, LLM chat services benefit from cache hits through reduced processing needs. A key metric for LLM chat services is the number of tokens processed. In this context, a token represents a unit of text within a query, indicating computational workload. This metric, connected to GPU usage or the expenses of forwarding queries to public LLMs, is important to the financial sustainability of LLM-based automated chat services.

2 Literature Survey

Small Language models (SLMs), typically those with up to 8 billion parameters, have received comparatively less attention in academia than large language models (LLMs). They are particularly suited for deployment on edge devices such as smartphones, laptops, and microprocessors.

Recent research explores collaborative edge computing, where LLM inference is facilitated by partitioning the model across distributed devices to optimize latency and throughput (Zhang et al., 2024).

Google's BERT (Devlin et al., 2019) has been fine-tuned and optimized for deployment on mobile devices, showcasing the feasibility of using SLMs for natural language processing tasks. These models are specifically designed to operate efficiently on resource-constrained devices that enable real-time user interaction without relying on cloud-based processing.

Several studies have proposed hybrid approaches that combine the strengths of Small Language Models (SLMs) and Large Language Models (LLMs). For instance, (Bergner et al., 2024) introduced a method where a pre-trained, frozen LLM encodes all prompt tokens in parallel, generating representations that subsequently condition and guide an SLM. Another example involves dynamically selecting between SLM and LLM based on reward token modeling, as demonstrated by (MS et al., 2024).

In the context of hardware innovations, Deeploy has demonstrated a high-efficiency, end-to-end deployment of SLMs on microcontroller-class chips without external memory access (Scherer et al., 2024). By leveraging a multicore RISC-V (RV32) MCU augmented with machine learning instruction extensions and a neural processing unit (NPU), the

Deeploy compiler generates highly optimized C code for efficient execution. This method achieves leading-edge energy and throughput efficiency of 490μJ per token, processing up to 340 tokens per second for an SLM trained on the TinyStories dataset(Eldan and Li, 2023). Deeploy highlights the feasibility of deploying SLMs in resource-constrained environments.

The integration of SLMs into edge computing environments not only enhances performance but also addresses critical concerns regarding security and privacy (Saha et al., 2024a). Operating SLMs on the edge minimizes the transfer of sensitive information to cloud servers, which is particularly beneficial in regulated industries such as healthcare and finance.

Advancements in hardware, such as developing specialized Neural Processing Units (NPUs) like Arm's Ethos-U85, have further facilitated the deployment of SLMs on edge devices. These NPUs are designed to efficiently handle AI workloads that enable real-time processing and reduce reliance on cloud-based infrastructure. (Ltd., 2025)

3 Architecture

3.1 Problem Setup and High-Level Summary

The proposed architecture employs Small Language Models (SLMs) as efficient intermediaries to cache and serve responses, which helps in reducing reliance on Large Language Models (LLMs). The workflow follows these stages:

- **Query Reception and Similarity Check:** When a user query is received, it is first embedded and compared against entries in a vector database located at the edge. This database contains embeddings of previously seen queries.
- **Threshold-Based Routing:** If a semantically similar query is found—i.e., the similarity score exceeds a predefined threshold—the system infers that the small language model (SLM) is capable of handling the query. The query is then routed to the SLM for response generation.
- **LLM Fallback and Logging:** If no sufficiently similar query is found (i.e., a cache miss), the query is forwarded to a large language model (LLM) hosted in the cloud. The LLM generates a response, which is returned to the user and concurrently stored—along with the query embedding—in the cloud-based vector database.
- **SLM Adaptation:** Periodically, the system evaluates whether the volume and diversity of LLM-handled queries warrant retraining the SLM. If so, the SLM is incrementally updated using this data to improve its performance and coverage over time.

3.2 Knowledge Base Management

Vector Database Selection. We employ ChromaDB as the underlying vector database (Balushi et al., 2025), chosen for its ease of integration, scalability, and efficient support for approximate nearest neighbor (ANN) search in high-dimensional spaces. ChromaDB provides optimized indexing and retrieval mechanisms suitable for a real-time conversational system.

Embedding Model. For embedding generation, we adopt `all-MiniLM-L6-v2` (Wang et al., 2020), a lightweight yet semantically robust model from the Sentence Transformers library. This model is capable of producing high-quality, dense vector representations while maintaining a minimal computational footprint, making it well-suited for edge deployments. The resulting embeddings are stored in the chromadb index and used to support rapid semantic similarity checks during query processing.

3.3 Cache Eviction Policies

To maintain the performance and scalability of the system, cache eviction policies are implemented within the LLM's vector database:

Least Recently Used (LRU): The system tracks the last access time of each entry. The least recently accessed entries are evicted when the cache exceeds its storage limit. This policy prioritizes retaining frequently accessed queries.

Least Frequently Used (LFU): The system tracks the frequency of access for each entry. Entries with the lowest access counts are evicted when the cache reaches capacity. This approach ensures that popular queries remain in the cache.

3.4 Size and Scaling

SLMs operate with significantly fewer parameters than LLMs. The effectiveness of language models in various tasks is often guided by scaling laws, which describe the relationship between model performance, dataset size, and computational resources.

Kaplan et al. (Kaplan et al., 2020) empirically demonstrated that the performance of autoregressive language models follows a power-law relationship with respect to three key factors: model size

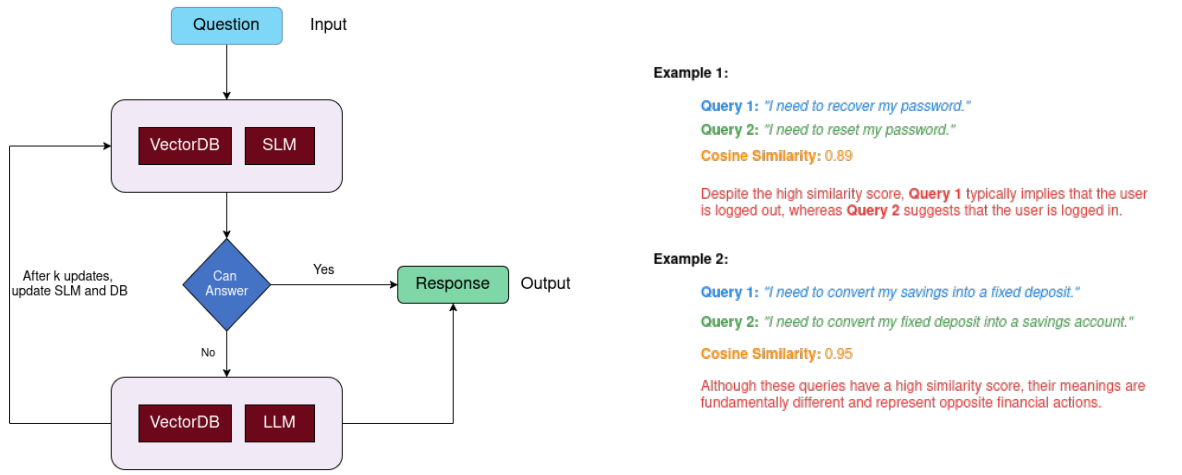

Figure 1: (a) The workflow of the proposed framework. (b) Example showing different responses for queries with high cosine similarity.

(N), dataset size (D), and compute budget (C). This relationship is given by:

$$L(N, D, C) = AN^{-\alpha} + BD^{-\beta} + CC^{-\gamma} \quad (1)$$

where $L(N, D, C)$ is the model loss (e.g., cross-entropy loss), N is the number of model parameters, D is the dataset size (in tokens), and C is the compute budget (in FLOPs). The coefficients A, B, C are empirically determined, and α, β, γ are the scaling exponents that quantify how loss improves with increased resources.

This equation suggests that reducing parameters, data, or compute leads to performance degradation. However, by optimizing architectures, leveraging knowledge distillation, and training on domain-specific data, SLMs can sustain strong performance at a fraction of the cost of LLMs.

Kaplan's scaling law emphasizes that model performance depends not only on parameter count but also on data efficiency and computational optimization. Instead of merely reducing model size, strategies such as fine-tuning on domain-specific datasets, distillation from larger models, and caching frequently used responses enable SLMs to achieve high efficiency while maintaining accuracy.

4 Observations and Results

4.1 Dataset

For evaluating the effectiveness of our caching strategies and LLM query routing approaches, we primarily utilized an open-source dataset made available under the Community Data License Agreement – Sharing – Version 1.0 (CDLA-Sharing-1.0) **Bitext Customer Support Dataset** (Bitext, n.d.a) [1]. Our use complies with the license terms and is strictly for academic research purposes. This dataset is tailored for customer service scenarios and comprises 26,872 query-response pairs distributed across 27 distinct intents and 10 broader categories. Each intent includes an average of 1,000 samples, annotated with 30 unique entity types and 12 language generation tags to simulate diverse linguistic structures and user behaviors. It served as the principal benchmark for assessing various caching policies, including LRU and LFU, as well as various model performances.

Additionally, we used an open-source dataset made available under the Community Data License Agreement – Sharing – Version 1.0 (CDLA-Sharing-1.0) **Bitext Insurance LLM Chatbot Training Dataset** (Bitext, n.d.b) [2]. Our use complies with the license terms and is strictly for academic research purposes. This dataset is domain-specific to the insurance sector, for further evaluation. This dataset enabled us to test the generalizability of our caching mechanisms and model selection strategies in a specialized subdomain of customer support.

Instead of solely relying on manually curated training data, we augment the existing dataset with synthetic data to ensure that the Small Language Model (SLM) provides detailed and consistent re-

[1] https://huggingface.co/datasets/bitext/Bitext-customer-support-llm-chatbot-training-dataset

[2] https://huggingface.co/datasets/bitext/Bitext-insurance-llm-chatbot-training-dataset

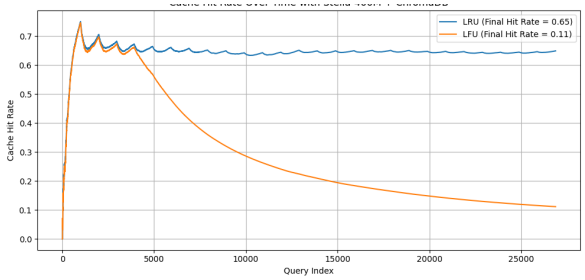

Figure 2: Cache Hit Rate over time

sponses while maintaining efficiency. Queries and responses are synthetically generated using a Large Language Model (LLM), which helps to standardize response formats and align SLM behavior with LLM outputs. We leverage two key augmentation strategies, LLM-based augmentation and Algorithmic Augmentation.

4.1.1 LLM-Based Augmentation

- **Synonym Replacement** – Replaces words with synonyms while maintaining medical context (Cegin et al., 2024).
- **Paraphrasing** – Rewrites the text (Sharma et al., 2022).
- **NER Replacement** – Substitutes named entities with similar terms.
- **Spelling & Grammar Errors** – Introduces minor typos to improve model robustness.
- **Contrastive Question Generation** – Creates challenging variations of existing questions.
- **Embedding-Based Sentence Augmentation** – Generates similar sentences while preserving meaning(Li et al., 2024b).
- **Synthetic Q&A Generation** – Produces high-quality question-answer pairs for better training(Dhruva et al., 2024).

4.1.2 Algorithmic Augmentation

- **Random Deletion** – Removes words probabilistically to create sentence variations.
- **Random Insertion** – Inserts random words from the sentence at new positions.
- **Random Swap** – Swaps positions of two words to change sentence structure.
- **Sentence Shuffling** – Randomizes the order of sentences in a paragraph.
- **Noise Injection** – Introduces character-level distortions to simulate real-world errors (Shorten et al., 2021).

4.2 Results

To evaluate the effectiveness of the Small Language Model (SLM) in generating high-quality responses, we employ three key metrics: BLEU-2, ROUGE-L, and BERTScore F1. BLEU-2 (Bilingual Evaluation Understudy with 2-grams) measures the precision of generated text by comparing it to reference outputs using n-gram overlap (Wieting et al., 2019). Specifically, BLEU-2 focuses on bigrams, ensuring the model captures local coherence and phrase-level accuracy. The BLEU score is computed using the geometric mean of n-gram precisions with a brevity penalty to penalize excessively short outputs. The formula for BLEU-N is given as:

$$BLEU_N = BP \times \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \quad (2)$$

where BP is the brevity penalty, p_n is the precision of n-grams, and w_n is the weighting factor.

ROUGE-L (Recall-Oriented Understudy for Gisting Evaluation - Longest Common Subsequence) (Rehman et al., 2025) measures the recall of generated responses by identifying the longest common subsequence (LCS) between the generated text and reference responses. This metric is particularly useful for evaluating fluency and relevance . It is computed as:

$$ROUGE\text{-}L = \frac{LCS(X, Y)}{|Y|} \quad (3)$$

where $LCS(X, Y)$ is the length of the longest common subsequence between the generated text X and the reference Y, and $|Y|$ is the length of the reference text.

BERTScore F1 (Zhang et al., 2019) leverages contextual embeddings from pre-trained BERT models to compute token-level similarity, capturing semantic accuracy beyond surface-level matching. It computes precision, recall, and F1-score based on cosine similarity between token embeddings, making it robust against paraphrasing and lexical variation.

Metrics like BLEU and ROUGE rely on exact token matches, which may be affected if the tokenizer splits or joins tokens differently than standard tokenizers. BERTScore, being embedding-based, is generally more robust to these differences. The evaluation scores obtained for our SLM Cache model are consistent with the nature of templated question-answer generation. The BLEU-2 scores,

Policy	Final Hit Rate	Cache Size	Total Queries
LRU	0.1133	1000	26872
LFU	0.6548	1000	26872

Table 1: Cache performance comparison between LRU and LFU policies

ranging from 0.0488 to 0.1273, appear lower due to the structured nature of responses, where placeholders such as `Order Number` disrupt exact bigram matching. Similarly, ROUGE-L scores (0.2066 to 0.4024) reflect moderate recall, as variations in phrasing or tokenization can lead to slight mismatches despite conveying the intended meaning. However, the high BERTScore F1 scores (0.8493 to 0.9036) indicate strong semantic alignment, confirming that the generated responses effectively capture the meaning of the reference answers.

We trained multiple SLM models with varying vocabulary sizes, embedding dimensions, number of layers, and attention heads to systematically analyze their impact on performance metrics and identify the optimal configuration 3. Overall, these scores justify the model's ability to maintain coherence and intent despite minor syntactic variations.

Our dataset consists of approximately 100 MB of text, translating to nearly 10^8 bytes. Using standard byte-pair encoding (BPE) or WordPiece tokenization, an average of 4 bytes per token suggests a total token count of approximately 25 million. However, given the domain-specific nature of our corpus, which exhibits high redundancy, we can optimize tokenization to achieve an estimated 3 bytes per token, yielding a more efficient representation (Song et al., 2020).

For general-purpose models, vocabulary sizes typically range between 30k to 100k tokens. However, given our domain-specific focus, we optimize our tokenizer to reduce vocabulary to $V = 5,000$ to $10,000$ tokens. A smaller vocabulary directly reduces the embedding matrix size, which scales linearly with V, thus lowering memory and computational requirements (Vaswani et al., 2017).

In a decoder-only Transformer model, the embedding parameters are given by:

$$\text{Embeddings} = V \times d_{model}, \tag{4}$$

where d_{model} is the hidden size of the transformer layers. If input and output embeddings are tied, the parameter count remains $V \times d_{model}$, whereas untied embeddings would double this count. åThe core transformer consists of L layers, each containing a multi-head self-attention mechanism and a feed-forward network. Ignoring biases and normalization layers, the parameter count per layer is approximately:

$$\text{params(layer)} \approx 4 \cdot d_{model}^2 + 2 \cdot d_{model} \cdot d_{ff}, \tag{5}$$

where d_{ff} is the intermediate size of the feed-forward network, often set as $d_{ff} = 4 \times d_{model}$. Thus, for L layers, the total parameter count is:

$$P \approx V \times d_{model} + L \times \left(4d_{model}^2 + 8d_{model}^2\right). \tag{6}$$

To maintain efficiency, we prioritize the use of a domain-specific tokenizer to optimize token count, ensuring a compact and practical vocabulary. We recommend selecting a moderate number of layers ($4 \leq L \leq 8$) to strike a balance between model performance and computational efficiency. Additionally, applying 8-bit or 16-bit quantization helps reduce storage and computational requirements without significantly impacting performance. Keeping the model dimension (d_{model}) within the range of 256 to 512 allows sufficient expressiveness while controlling parameter growth. Detailed calculations supporting these recommendations are provided in Appenix. Given that our corpus exhibits redundancy and compressibility, a model within the range of 10–30 million parameters, quantized to 8-bit, should be sufficient for capturing domain-specific patterns while avoiding excessive memory overhead.

All results reported are averaged over 3 runs with different random seeds. Standard deviations were found to be negligible and are omitted for brevity.

We ran the model on an Intel i5 12th gen processor with RTX 2050 4GB GPU. We found an approximate 2.8x improvement in execution time and a 10x reduction in average GPU Memory compared to utilizing only the llama3.1 8B model (Touvron et al., 2024). We tested the model on the Bitext Customer Support Dataset, containing 26872 queries. Our architecture could answer all the queries in 261061.48 seconds or 3.02 days, whereas llama3.1 8B took 731178.4 seconds or 8.46 days.

Dataset	BLEU-2	ROUGE-L	BERTScore F1
Bitext Customer	0.1273	0.4024	0.9036
Bitext Insurance	0.3530	0.5619	0.9134

Table 2: Evaluation metrics for Bitext Customer and Insurance datasets

Model	Vocab	Embed Dim	Layers	Heads	BLEU-2	ROUGE-L	BERTScore
1	5000	384	3	3	0.1043	0.3621	0.8872
2	5000	512	6	8	0.1273	0.4024	0.9036
3	8000	256	4	4	0.1146	0.3766	0.8966
4	10000	768	8	12	0.0924	0.3491	0.8850
5	12000	1024	12	16	0.0488	0.2066	0.8493

Table 3: Performance evaluation of different SLM configurations.

4.2.1 Cache Policy Evaluation

To assess the impact of caching on model inference efficiency, we implemented and evaluated two classical cache replacement strategies: Least Recently Used (LRU) and Least Frequently Used (LFU). Using a fixed cache size of 100 entries over a sequence of 10,000 queries, we observed notable differences in final hit rates. As shown in Table 1, the LFU policy significantly outperformed LRU, achieving a final hit rate of 0.6548 compared to 0.1133 for LRU. This improvement highlights LFU's advantage in scenarios with repetitive query patterns, as it preferentially retains frequently accessed items in the cache. In contrast, LRU, which evicts items based solely on recency, struggled to capture long-term frequency trends in the query stream. These results suggest that frequency-aware caching mechanisms can play a vital role in optimizing LLM-backed customer service systems by reducing redundant model invocations and improving overall latency.

5 Applications

The proposed caching mechanism improves response times for chatbots, customer support, and virtual assistants by offloading frequent queries to small language models (SLMs). This reduces reliance on expensive LLMs, making it suitable for both resource-constrained and large-scale deployments. Its integration with a structured knowledge base also supports applications like documentation, education, and adaptive learning systems (Wang et al., 2024).

6 Discussion

From an information-theoretic perspective, if a 100 MB text corpus can be heavily compressed (e.g., to 20 MB) due to redundancy, then the Kolmogorov complexity, or the genuinely irreducible information content, might be less than 20 MB. A model with approximately 10–20 MB of parameters could encapsulate this knowledge if it effectively exploits the repeated patterns within the dataset and represents them compactly. This aligns with previous estimates, particularly when considering half-precision storage or further quantization techniques. However, it is crucial to recognize that a model's parameters do not function as a one-to-one "bits = bits" memory. Unlike a lossless compression algorithm, such as a ZIP file, neural networks store and organize information in a distributed manner, introducing overhead in how weights must be structured to generate accurate next-token probabilities. Nevertheless, in practical settings, a domain-specific model containing approximately 5–50 million parameters (stored in 8-bit or 16-bit format) can achieve an overall size in the tens of megabytes or lower. This is often sufficient to encapsulate the domain-specific text, mainly if a significant portion exhibits repetition.

Reducing the model size introduces fundamental trade-offs. A smaller number of parameters reduces the model footprint but also constrains its ability to recall obscure or rare details. If a domain contains unique or one-off sentences, the model faces two options: either memorizing them explicitly, which increases parameter requirements, or discarding them, potentially losing fine-grained knowledge. By strategically reducing the vocabulary size and

selecting a relatively shallow architecture with a moderate embedding dimension, the model's footprint can be constrained to the single- to low-tens-of-megabyte range. This remains feasible even when trained on a dataset exceeding 100 MB, particularly if the dataset exhibits redundancy or high compressibility. Despite this reduction, the model can still capture a substantial fraction of the domain's linguistic structure, albeit at the potential cost of fine-grained or domain-specific nuances. If such a trade-off is deemed acceptable, this approach mitigates the risk of excessive parameter growth into the hundreds of megabytes.

LoRA (Low-Rank Adaptation) enables fine-tuning of large language models (LLMs) by adapting them for specific tasks while preserving the core structure of the original model (Chavan et al., 2023). While this technique enhances performance for particular applications, a Lora-finetuned LLM retains much of the redundancy present in the base model. As a result, it remains parameter-heavy, making real-time inference on edge devices challenging. In contrast, by leveraging model compression techniques and strategically minimizing the vocabulary size, an efficient small language model (SLM) can be designed to function within constrained computational environments while preserving core linguistic capabilities.

Recent advancements in hybrid language models, like the Uncertainty-aware Hybrid Language Model (U-HLM)(Oh et al., 2024), offer new ways to optimize SLM-based caching. U-HLM helps the SLM decide when it is confident enough to skip using the larger LLM, reducing unnecessary processing while maintaining accuracy. Integrating a similar uncertainty-based approach into the SLM Cache could further improve efficiency and token processing speed.

However, applying this method to 12MB SLMs is challenging. The original U-HLM was designed for a much larger 1.1 billion-parameter model, and shrinking it significantly leads to a loss of knowledge and generalization. Large models store more patterns and details, whereas extreme compression can result in errors and a weaker understanding of language. The key challenge is ensuring a much smaller model retains enough accuracy and context without introducing mistakes.

7 Conclusion

This paper presents SLMCache, a system that uses small language models (SLMs) as an intelligent caching layer to reduce the need for expensive large language models (LLMs). By matching incoming queries to previously seen ones and responding locally when possible, SLMCache speeds up responses and lowers memory usage without sacrificing accuracy. It achieved up to 2.8× faster processing and used 10× less GPU memory compared to using LLMs alone. The approach works especially well in real-time systems like chatbots or customer support, where many queries are repetitive. While the system still faces challenges like retraining overhead and limited ability to handle long conversations, it offers a practical, low-cost solution for many applications. Future improvements could include better decision-making about when to use the SLM or LLM, more secure on-device processing, and ways to make it work even better on edge devices.

8 Limitations

While the proposed architecture offers significant improvements in response latency, several limitations remain:

- **Low Cache Efficiency in Sparse Workloads:** For deployments with low query volume or highly diverse input distributions, the cache hit rate may remain consistently low. In such cases, the benefits of the SLM cache are diminished, and the system frequently defaults to the more expensive LLM path.
- **Retraining Overhead:** The periodic retraining of the SLM on newly logged LLM-handled queries introduces both computational and engineering overhead. Ensuring the quality, consistency, and timeliness of this retraining loop is non-trivial, particularly in large-scale or distributed deployments.
- **Limited Conversational Depth:** The cache mechanism and static nature of the SLM limit its ability to handle complex, multi-turn conversations or adapt to rapidly evolving user context. As a result, conversational coherence and personalization may degrade in long or dynamic interactions.
- **Memory Constraints at the Edge:** Although SLMs are computationally lightweight, the storage and indexing of millions of embeddings (e.g., 25M tokens $\approx$ 20–25M entries) using chro-

madb at the edge still incurs a nontrivial memory footprint. This can be prohibitive for memory-constrained devices such on embedded systems.

A significant risk in this work arises from the possibility of hallucinated responses generated by both the Small Language Model (SLM) and the fallback Large Language Model (LLM). Since the caching mechanism relies on semantic similarity to reroute queries to the SLM, there is a risk that semantically similar but contextually distinct queries may receive inaccurate or misleading answers. Furthermore, LLMs are also known to produce factually incorrect or fabricated content, especially in low-resource or domain-specific scenarios.

Such hallucinations can be particularly problematic in sensitive applications like customer support, healthcare, or finance, where erroneous outputs may mislead users or erode trust. Without rigorous verification or grounding mechanisms, these issues may propagate through the caching layer, giving an illusion of correctness due to fast response times. Future work must incorporate uncertainty estimation, output verification, or grounding in trusted knowledge bases to mitigate these risks.

In summary, while the proposed architecture effectively reduces response latency and computational load, its performance is bounded by practical constraints. Sparse workloads, retraining overhead, limited conversational depth, and memory constraints at the edge highlight trade-offs between efficiency and scalability. Addressing these limitations is crucial for broader applicability, particularly in dynamic, large-scale, or resource-constrained environments.

9 Ethical Considerations

This work does not involve any human subjects or private data. We used two publicly available datasets: the Bitext customer support dataset and the Bitext insurance dataset, both distributed under the Community Data License Agreement – Sharing – Version 1.0 (CDLA-Sharing-1.0). These datasets consist of synthetic, English-language dialogues related to customer service and insurance support, respectively. They do not contain any personally identifiable information or offensive content, and are intended for research purposes only. All preprocessing scripts and usage instructions are included in our code repository, and all artifacts are released under the MIT License. Our objective is to enhance the efficiency of AI systems by leveraging smaller models as intelligent caches to reduce the energy and cost associated with large language models. We acknowledge that cached responses may become outdated over time, and recommend regular updates and robust safety checks, particularly for deployments in sensitive domains like healthcare or finance.

AI tools such as ChatGPT and GitHub Copilot were used in a limited capacity during this project. Their usage was restricted to occasional code debugging, clarifying API usage, and assisting with internet-style technical searches. Additionally, minor language suggestions were taken during the editing of the paper. All core ideas, experimental design, implementation, and writing were conducted independently by the authors without reliance on generative AI for substantive contributions.

References

Ahmed-Al Balushi, AS Al-Bemani, Saleh Al Araimi, G Balaji, Uma Suresh, Asiya Najeeb, et al. 2025. Ai-driven multi-modal information synthesis: Integrating pdf querying, speech summarization, and cross-language text summarization. *Procedia Computer Science*, 258:2996–3018.

Fu Bang. 2023. GPTCache: An open-source semantic cache for LLM applications enabling faster answers and cost savings. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pages 212–218, Singapore. Association for Computational Linguistics.

Benjamin Bergner, Andrii Skliar, Amelie Royer, Tijmen Blankevoort, Yuki Asano, and Babak Ehteshami Bejnordi. 2024. Think big, generate quick: Llm-to-slm for fast autoregressive decoding. *arXiv preprint arXiv:2402.16844*.

Bitext. n.d.a. Bitext customer support llm chatbot training dataset. `https://registry.opendata.aws/bitext-customer-support-chatbot/`. Accessed: 2025-05-19.

Bitext. n.d.b. Bitext insurance llm chatbot training dataset. `https://registry.opendata.aws/bitext-insurance-chatbot/`. Accessed: 2025-05-19.

Jan Cegin, Jakub Simko, and Peter Brusilovsky. 2024. Llms vs established text augmentation techniques for classification: When do the benefits outweight the costs? *arXiv preprint arXiv:2408.16502*.

Arnav Chavan, Zhuang Liu, Deepak Gupta, Eric Xing, and Zhiqiang Shen. 2023. One-for-all: Generalized lora for parameter-efficient fine-tuning. *arXiv preprint arXiv:2306.07967*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

G Dhruva, Ishani Bhat, Sanika M Rangayyan, and P Preethi. 2024. Synthetic data augmentation using large language models (llm): A case-study of the kamyr digester. In *2024 Third International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT)*, pages 1–7. IEEE.

Ronen Eldan and Yuanzhi Li. 2023. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, et al. 2025. Towards an ai co-scientist. *arXiv preprint arXiv:2502.18864*.

Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. 2023. A survey on large language models: Applications, challenges, limitations, and practical usage. *Authorea Preprints*.

Geoffrey Hinton. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Achraf Hsain and Hamza El Housni. 2024. Large language model-powered chatbots for internationalizing student support in higher education.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Jiaxing Li, Chi Xu, Feng Wang, Isaac M von Riedemann, Cong Zhang, and Jiangchuan Liu. 2024a. Scalm: Towards semantic caching for automated chat services with large language models.

Yichuan Li, Kaize Ding, Jianling Wang, and Kyumin Lee. 2024b. Empowering large language models for textual data augmentation. *arXiv preprint arXiv:2404.17642*.

Arm Ltd. 2025. Arm ethos-u85: Machine learning processor. `https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u85`. Accessed: 2025-01-22.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.

Adarsh MS, Jithin VG, and Ditto PS. 2024. Efficient hybrid inference for llms: Reward-based token modelling with selective cloud assistance. *arXiv preprint arXiv:2409.13757*.

Seungeun Oh, Jinhyuk Kim, Jihong Park, Seung-Woo Ko, Tony QS Quek, and Seong-Lyun Kim. 2024. Uncertainty-aware hybrid inference with on-device small and remote large language models. *arXiv preprint arXiv:2412.12687*.

Keivalya Pandya and Mehfuza Holia. 2023. Automating customer service using langchain: Building custom open-source gpt chatbot for organizations.

Tohida Rehman, Soumabha Ghosh, Kuntal Das, Souvik Bhattacharjee, Debarshi Kumar Sanyal, and Samiran Chattopadhyay. 2025. Evaluating llms and pretrained models for text summarization across diverse datasets. *arXiv preprint arXiv:2502.19339*.

Dipayan Saha, Shams Tarek, Katayoon Yahyaei, Sujan Kumar Saha, Jingbo Zhou, Mark Tehranipoor, and Farimah Farahmandi. 2024a. Llm for soc security: A paradigm shift. *IEEE Access*.

Rajarshi Saha, Naomi Sagan, Varun Srivastava, Andrea Goldsmith, and Mert Pilanci. 2024b. Compressing large language models using low rank and low precision decomposition. *Advances in Neural Information Processing Systems*, 37:88981–89018.

Mahadev Satyanarayanan. 2017. The emergence of edge computing. *Computer*, 50(1):30–39.

Moritz Scherer, Luka Macan, Victor JB Jung, Philip Wiese, Luca Bompani, Alessio Burrello, Francesco Conti, and Luca Benini. 2024. Deeploy: Enabling energy-efficient deployment of small language models on heterogeneous microcontrollers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(11):4009–4020.

Saket Sharma, Aviral Joshi, Namrata Mukhija, Yiyun Zhao, Hanoz Bhathena, Prateek Singh, Sashank Santhanam, and Pritam Biswas. 2022. Systematic review of effect of data augmentation using paraphrasing on named entity recognition. In *NeurIPS 2022 Workshop on Synthetic Data for Empowering ML Research*.

Connor Shorten, Taghi M Khoshgoftaar, and Borko Furht. 2021. Text data augmentation for deep learning. *Journal of big Data*, 8(1):101.

Xinying Song, Alex Salcianu, Yang Song, Dave Dopson, and Denny Zhou. 2020. Fast wordpiece tokenization. *arXiv preprint arXiv:2012.15524*.

Ilias Stogiannidis, Stavros Vassos, Prodromos Malakasiotis, and Ion Androutsopoulos. 2023. Cache me if you can: an online cost-aware teacher-student framework to reduce the calls to large language models.

Hugo Touvron, Shruti Bhosale, Y-Lan Boureau, Tim Dettmers, Alessandro Sordoni, Armand Joulin, et al. 2024. Llama 3: Open foundation and instruction-tuned language models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Fali Wang, Zhiwei Zhang, Xianren Zhang, Zongyu Wu, Tzuhao Mo, Qiuhao Lu, Wanjing Wang, Rui Li, Junjie Xu, Xianfeng Tang, et al. 2024. A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness. *arXiv preprint arXiv:2411.03350*.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788.

Yantong Wang and Vasilis Friderikos. 2020. A survey of deep learning for data caching in edge network. In *Informatics*, volume 7, page 43. MDPI.

John Wieting, Taylor Berg-Kirkpatrick, Kevin Gimpel, and Graham Neubig. 2019. Beyond bleu: training neural machine translation with semantic similarity. *arXiv preprint arXiv:1909.06694*.

Fangkai Yang, Pu Zhao, Zezhong Wang, Lu Wang, Jue Zhang, Mohit Garg, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 2023. Empower large language model to perform better on industrial domain-specific question answering. *arXiv preprint arXiv:2305.11541*.

Mingjin Zhang, Xiaoming Shen, Jiannong Cao, Zeyang Cui, and Shan Jiang. 2024. Edgeshard: Efficient llm inference via collaborative edge computing. *IEEE Internet of Things Journal*, pages 1–1.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Banghua Zhu, Ying Sheng, Lianmin Zheng, Clark Barrett, Michael I. Jordan, and Jiantao Jiao. 2023. On optimal caching and model multiplexing for large model inference.

A Model Configurations and Storage Considerations

A.1 Token Count

Consider a dataset of 100 MB of text, corresponding to approximately 10^8 bytes. In typical English text, standard byte-pair encoding (BPE) or WordPiece tokenization results in an average of approximately 4 bytes per token. Consequently, the dataset may contain roughly 25 million tokens:

$$\frac{100 \text{ MB}}{4 \text{ bytes/token}} = 25 \times 10^6 \text{ tokens}. \quad (7)$$

However, domain-specific corpora often exhibit substantial redundancy, such as repeated disclaimers or standardized methodology sections. By training a specialized tokenizer, the bytes-per-token ratio can be reduced (e.g., 3 bytes/token or even 2 bytes/token in highly repetitive datasets). As a result, the token count remains within the range of 20–25 million tokens as a reasonable approximation.

A.2 Vocabulary Reduction

General-purpose English language models (LMs) typically employ vocabulary sizes of $V = 30$k, 50k, or even 100k subword tokens. However, in domain-specific applications—such as modeling text related to Alzheimer's disease—the diversity of subwords is substantially lower. By training a custom subword tokenizer, the vocabulary can be reduced to approximately $V = 5$k–10k tokens. This reduction significantly decreases the size of embedding matrices in transformer-based models, as their parameter count scales linearly with vocabulary size.

B Parameter Estimation for Transformer Language Models

A standard decoder-only transformer language model consists of the following primary components:

B.1 Embedding Layer

The embedding layer consists of a token embedding matrix and, optionally, a separate token output projection. The parameter count varies depending on whether embeddings are shared between input and output:

- **Untied embeddings:** Parameters = $V \times d_{\text{model}}$ (input) + $d_{\text{model}} \times V$ (output).

- **Tied embeddings:** Parameters = $V \times d_{\text{model}}$ (shared for both input and output).

B.2 Transformer Layers

Each transformer layer consists of two main components:

1. **Multi-head self-attention sublayer.**
2. **Feed-forward sublayer**, which consists of two linear transformations around a non-linearity.

Ignoring biases and layer normalization for brevity, the approximate parameter count per layer is:

$$P_{\text{layer}} \approx 4d_{\text{model}}^2 + 2(d_{\text{model}} \times d_{\text{ff}}). \quad (8)$$

The term $4d_{\text{model}}^2$ arises from the self-attention mechanism, where each layer employs three projection matrices (query, key, value) along with an output projection, summing to four times d_{model}^2. The term $2d_{\text{model}} \times d_{\text{ff}}$ results from the two linear transformations in the feed-forward sublayer.

For a model with L layers, the total parameter count for the transformer layers is approximately:

$$P_{\text{layers}} \approx L \times [4d_{\text{model}}^2 + 2(d_{\text{model}} \times d_{\text{ff}})]. \quad (9)$$

Additionally, the embedding layers contribute:

$$P_{\text{embeddings}} \approx V \times d_{\text{model}} \quad (10)$$

for the tied-embedding case, or

$$P_{\text{embeddings}} \approx 2V \times d_{\text{model}} \quad (11)$$

if the embeddings are untied.

B.3 Final Parameter Approximation

In practical configurations, the feed-forward dimension is commonly set to $d_{\text{ff}} = 4d_{\text{model}}$. Substituting this into the parameter equation:

$$\begin{aligned} P_{\text{layer}} &\approx 4d_{\text{model}}^2 + 2(d_{\text{model}} \times 4d_{\text{model}}) \\ &= 4d_{\text{model}}^2 + 8d_{\text{model}}^2 \\ &= 12d_{\text{model}}^2. \end{aligned} \quad (12)$$

Thus, the overall model parameter count can be approximated as:

$$P \approx V \times d_{\text{model}} + L \times 12d_{\text{model}}^2. \quad (13)$$

While d_{model} is typically split across multiple attention heads (e.g., if the number of heads is h, each head operates on a subspace of size d_{model}/h), this does not affect the total parameter count but merely alters the internal organization of the model.

This formulation provides a practical estimate for designing compact transformer architectures while maintaining computational efficiency.

Given our computational constraints, we explore the five possible configurations:

B.4 Configuration A

- Vocabulary size: $V = 5,000$
- Layers: $L = 3$
- Model dimension: $d_{\text{model}} = 384$
- Embedding parameters: $5{,}000 \times 384 = 1.92 \times 10^6$
- Per-layer parameters: $12 \times 384^2 = 1.77 \times 10^6$
- Total parameters: $1.92 \times 10^6 + 3 \times 1.77 \times 10^6 = 7.23 \times 10^6$
- Estimated storage:
  - float32: $\approx$ 28.9 MB
  - float16: $\approx$ 14.5 MB
  - int8: $\approx$ 7.2 MB
- Actual size after training: 30.6 MB

B.5 Configuration B

- Vocabulary size: $V = 5,000$
- Layers: $L = 6$
- Model dimension: $d_{\text{model}} = 512$
- Embedding parameters: $5{,}000 \times 512 = 2.56 \times 10^6$
- Per-layer parameters: $12 \times 512^2 = 3.15 \times 10^6$
- Total parameters: $2.56 \times 10^6 + 6 \times 3.15 \times 10^6 = 21.46 \times 10^6$
- Estimated storage:
  - float32: $\approx$ 85.8 MB
  - float16: $\approx$ 42.9 MB
  - int8: $\approx$ 21.5 MB
- Actual size after training: 88.0 MB

B.6 Configuration C

- Vocabulary size: $V = 8,000$
- Layers: $L = 4$
- Model dimension: $d_{\text{model}} = 256$
- Embedding parameters: $8{,}000 \times 256 = 2.05 \times 10^6$
- Per-layer parameters: $12 \times 256^2 = 0.79 \times 10^6$
- Total parameters: $2.05 \times 10^6 + 4 \times 0.79 \times 10^6 = 5.21 \times 10^6$
- Estimated storage:
  - float32: $\approx$ 20.8 MB
  - float16: $\approx$ 10.4 MB
  - int8: $\approx$ 5.2 MB
- Actual size after training: 21.9 MB

Figure 3: Training Loss vs step

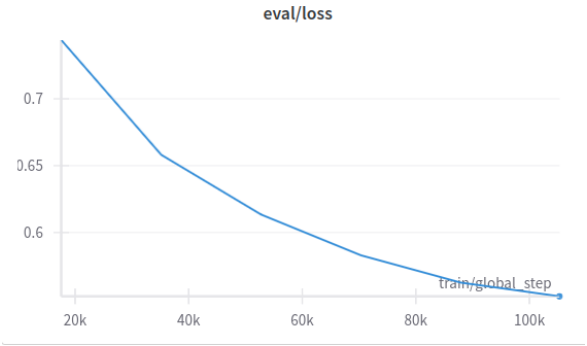

Figure 4: Eval Loss vs step

B.7 Configuration D

- Vocabulary size: $V = 10,000$
- Layers: $L = 8$
- Model dimension: $d_{\text{model}} = 768$
- Embedding parameters: $10{,}000 \times 768 = 7.68 \times 10^6$
- Per-layer parameters: $12 \times 768^2 = 7.08 \times 10^6$
- Total parameters: $7.68 \times 10^6 + 8 \times 7.08 \times 10^6 = 64.32 \times 10^6$
- Estimated storage:
  - float32: ≈ 257.3 MB
  - float16: ≈ 128.6 MB
  - int8: ≈ 64.3 MB
- Actual size after training: 260.7 MB

B.8 Configuration E

- Vocabulary size: $V = 12,000$
- Layers: $L = 12$
- Model dimension: $d_{\text{model}} = 1024$
- Embedding parameters: $12{,}000 \times 1024 = 12.29 \times 10^6$
- Per-layer parameters: $12 \times 1024^2 = 12.58 \times 10^6$
- Total parameters: $12.29 \times 10^6 + 12 \times 12.58 \times 10^6 = 163.25 \times 10^6$
- Estimated storage:
  - float32: ≈ 653.0 MB
  - float16: ≈ 326.5 MB
  - int8: ≈ 163.3 MB
- Actual size after training: 658.0 MB

From the models we trained we choose the configuration B as it gave the best results.

C Software Libraries

We used standard open-source libraries for model training, inference, and evaluation. For text generation, we employed the Hugging Face transformers library with a fine-tuned GPT-2 model and

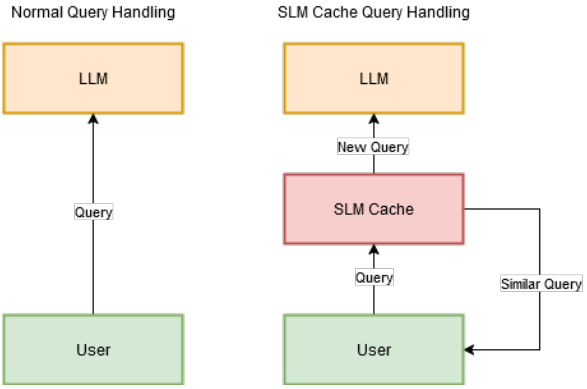

Figure 5: SLMCache workflow in comparison to LLM-Cache query handling.

GPT2TokenizerFast. Inference was done using deterministic decoding (do_sample=False) with parameters: top_k=50, top_p=0.9, temperature=1.0, and a max_length of 150 tokens.

For evaluation, we used:

- `nltk` for computing BLEU-2 scores (`sentence_bleu`) with smoothing (`SmoothingFunction().method1`)
- `rouge_score` for ROUGE-L using `RougeScorer` with stemming enabled
- `bert_score` for semantic similarity, using the `roberta-large` model with default settings

All evaluations were performed on 100 randomly sampled instances. Additional libraries used include torch, tqdm, pandas, numpy, matplotlib, cachetools, chromadb, and datasets.

Unless otherwise stated, default parameters were used throughout.