MemBoost: A Memory-Boosted Framework for Cost-Aware LLM Inference

Anonymous Authors[1]

Abstract

Large Language Models (LLMs) deliver strong performance but incur high inference cost in real-world services, especially under workloads with repeated or near-duplicate queries across users and sessions. In this work, we propose **MemBoost**, a memory-boosted LLM serving framework that enables a lightweight model to reuse previously generated answers and retrieve relevant supporting information for cheap inference, while selectively escalating difficult or uncertain queries to a stronger model. Unlike standard retrieval-augmented generation, which primarily grounds a single response, MemBoost is designed for interactive settings by supporting answer reuse, continual memory growth, and cost-aware routing. Experiments across multiple models under simulated workloads show that MemBoost substantially reduces expensive large-model invocations and overall inference cost, while maintaining high answer quality comparable to the strong model baseline.

1. Introduction

Large Language Models (LLMs) (Hurst et al., 2024; Team et al., 2023) have demonstrated strong capabilities in natural language understanding, instruction following (Chung et al., 2024; Ouyang et al., 2022), coding (Gao et al., 2023; Ni et al., 2024), and decision-making (Anil et al., 2023; Wei et al., 2022). However, deploying frontier-scale models in real-world services remains expensive, as inference often requires multiple high-end GPUs, especially for long-form reasoning and explanation-heavy responses (Kwon et al., 2023; Park et al., 2025). These costs are amplified in production settings where many user queries are repeated or near duplicates, causing redundant computation. A widely adopted direction to mitigate these costs is to shift part of the workload to smaller models with retrieval-augmented generation (RAG) (Lewis et al., 2020; Guu et al., 2020; Borgeaud et al., 2022). RAG allows a model to externalize knowledge into a searchable index and condition generation on retrieved evidence, reducing the need to store everything in parameters (Izacard et al., 2023; Fan et al., 2025). For example, Atlas (Izacard et al., 2023) demonstrates that the RAG model with far fewer parameters can be competitive with, and in some regimes outperform, much larger models on knowledge-intensive tasks.

However, most existing RAG work primarily targets knowledge grounding, i.e., improving the answer to a single query by retrieving external documents (Lewis et al., 2020; Guu et al., 2020). In contrast, interactive LLM services exhibit additional properties that are not addressed by standard RAG alone. First, many queries that are semantically equivalent are repeatedly requested across users and sessions. Recomputing these same answers wastes GPU time. This motivates semantic caching, which retrieves a previously generated response when a new query is semantically similar (Gill et al., 2025; Yu et al., 2025; Liu et al., 2025). However, making this reliable requires careful control. Second, deployed LLMs continuously generate new high-quality answers. When these answers are valuable, they should be written back into the retrieval memory so that a semantic cache can serve future similar queries cheaply. However, this write-back mechanism raises practical questions, such as what/when to store it. Finally, although small models augmented with retrieval can be competitive on knowledge-intensive tasks, their capabilities remain limited on more challenging requests. A practical system should therefore enable the small model to escalate to a stronger model when retrieval is insufficient, aligning with recent work on routing across multiple LLMs (Ong et al., 2024; Zhang et al., 2025; Moslem & Kelleher, 2026) to balance quality and cost.

These observations suggest a missing system-level paradigm: a unified approach that enables a small model to cheaply reuse supporting knowledge or prior answers from semantic memory, selectively defer to a stronger but costly model when retrieval is insufficient, and continuously write back useful new information into the retrieval memory, while explicitly balancing answer quality and inference cost. In this paper, we introduce **MemBoost**, a memory-boosted architecture with three components: (1)

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by *The Impact of Memorization on Trustworthy Foundation Models* Workshop @ ICML. Do not distribute.

an Associative Memory Engine (AME) that performs fast semantic retrieval and supports write-back of newly generated answers; (2) a high-capability Large-LLM Oracle that provides an accurate fallback when memory is insufficient; and (3) a lightweight Meta Controller, which is a small LLM that composes the final response by either reusing from memory or routing the query to the oracle, and support write back mechanism for future reuse, thereby balancing quality and cost. Together, MemBoost turns inference into a "retrieve-or-escalate" decision problem, substantially reducing expensive large-model calls while preserving answer quality. Experiments on the MMLU-Pro dataset (Wang et al., 2024) under a simulated workload with repeated and near-duplicate queries show that MemBoost substantially reduces calls to the costly large LLM while largely preserving the answer quality of the oracle model.

2. MemBoost: Memory-Boosted LLM Serving Framework

2.1. Problem Setup

Unlike standard single-turn question answering (QA) tasks studied in prior work, we consider a setting with continuous interaction between multiple users and an LLM service, where incoming queries may be exact duplicates or semantic near-duplicates over time. At each time step t, the system receives a user query x_t and produces an answer y_t. We use a ground-truth quality signal $r(x_t, y_t)$ to measure whether the response is helpful, correct, and aligned with the user's intent. Our objective is to maximize the average response quality over a horizon T, $\max \frac{1}{T}\sum_{t=1}^{T} r(x_t, y_t)$, while minimizing the overall inference cost, as formalized in Section 2.3.

2.2. Overview of MemBoost

As discussed above, in production settings, repeatedly invoking a frontier model to answer semantically redundant queries wastes substantial compute. To mitigate this inefficiency while balancing quality and cost, we propose MemBoost, a memory-boosted LLM serving system composed of three components (Figure 1): Associative Memory Engine, Large-LLM Oracle and Meta Controller.

Associative Memory Engine (AME). AME maintains an external memory that stores auxiliary knowledge as well as previously answered queries and their associated responses, and supports fast semantic retrieval. Concretely, given a new query x_t at time t, AME retrieves a small set of K candidate memory entries $\mathcal{M}_t = \{(x^{(i)}, y^{(i)}, m^{(i)})\}_{i=1}^{K}$, where $m^{(i)}$ denotes metadata (e.g., query category). The AME stores the question-answer pair for retrieval together with the qid (a unique identifier provided by the dataset), the category and the timestamp. In addition, AME supports write-back

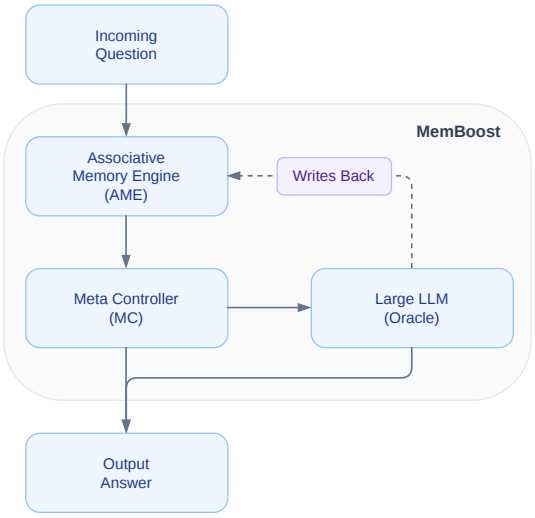

Figure 1. **Overview of MemBoost.** For each incoming query, the AME retrieves a small set of relevant memory entries. The MC then either composes an answer from the retrieved results or escalates to the Oracle. When the Oracle is used, the MC decides whether to write the new answer back into the AME for future reuse.

by storing newly generated high-quality entries (x_t, y_t, m_t) when instructed by the Meta Controller.

Large-LLM Oracle. The oracle is a high-capability model that produces high-quality answers but incurs high inference cost. We use it as a fallback: the system queries the oracle when retrieval from AME is missing, ambiguous, or deemed unreliable by the Meta Controller. We denote the oracle's response to query x_t as $y_t^\star = \mathrm{Oracle}(x_t)$.

Meta-Controller (MC). MC is a lightweight LLM that orchestrates the interaction among the user, AME, and the oracle. At each step, MC decides whether to answer using information retrieved from AME or to escalate to the oracle. When escalation occurs, MC also determines whether the newly generated query answer pair should be stored in AME for future reuse. Concretely, given a query x_t, MC first requests retrieval from AME and obtains $\mathcal{M}_t$. Conditioned on x_t and $\mathcal{M}_t$, MC either returns an answer directly, $y_t = \mathrm{MC}(x_t, \mathcal{M}_t)$, or calls the oracle to obtain $y_t^\star = \mathrm{Oracle}(x_t)$ and returns $y_t = y_t^\star$. In the latter case, MC may optionally write $(x_t, y_t^\star, m_t)$ back into AME, where m_t denotes metadata inferred by MC. This "retrieve $\rightarrow$ decide $\rightarrow$ escalate (if needed) $\rightarrow$ write-back (if needed)" loop reduces expensive oracle calls under repeated queries while preserving high answer quality when retrieval is insufficient.

2.3. System Cost

To model system efficiency, let $c_O(x_t)$ denote the cost of an oracle call at time t (e.g., GPU time, energy, or monetary cost), $c_M(x_t)$ the cost of running the lightweight

MC, and $c_R(x_t)$ denote the cost of retrieval (e.g., CPU time). In typical deployments, $c_O(x_t)$ is much larger than $c_M(x_t) + c_R(x_t)$, since frontier-model inference dominates GPU compute, whereas retrieval is primarily CPU-bound and inexpensive relative to large-model generation. Let $I_t \in \{0, 1\}$ indicate whether the retrieval information from AME is used at time t ($I_t = 1$ if memory is used, and $I_t = 1$ if the system escalates to the oracle). The total cost over T steps is

$$C_T = \sum_{t=1}^{T} \big(c_M(x_t) + c_R(x_t)\big) + \sum_{t=1}^{T} (1 - I_t)\, c_O(x_t).$$

By contrast, an oracle-only baseline would cost approximately $C_T^{\text{oracle}} = \sum_{t=1}^{T} c_O(x_t)$. So our framework achieves cost savings whenever $C_T < C_T^{\text{oracle}}$. Equivalently, this condition can be written as

$$\sum_{t=1}^{T} I_t\, c_O(x_t) > \sum_{t=1}^{T} \big(c_M(x_t) + c_R(x_t)\big).$$

Putting quality and cost together, our goal is to achieve oracle-level quality with significantly fewer oracle invocations:

$$\frac{1}{T}\sum_{t=1}^{T} r(x_t, y_t) \approx \frac{1}{T}\sum_{t=1}^{T} r\big(x_t, y_t^{\star}\big), C_T < C_T^{\text{oracle}}$$

3. Experiments

In this section, we evaluate MemBoost in terms of both (i) reducing inference cost and (ii) retaining high response quality.

3.1. Experiment Setup

We evaluate the proposed MemBoost framework on the MMLU-Pro dataset (Wang et al., 2024), a widely used and challenging benchmark that covers diverse disciplines and is designed to more rigorously assess LLM capabilities. We use the Business category of the MMLU-Pro dataset and ground truth label answers as our benchmark for comparing the accuracy of the meta-controller (MC) with the Large LLM (Oracle). As the Business category contains 768 examples it is large enough to fit 5000 requests of the chosen Zipf distributions. To emulate real-world LLM-serving workloads where many requests are repeated or near-duplicated, we generate a query stream by sampling MMLU-Pro questions according to a Zipf distribution (Zipf, 1949). This produces a heavy-tailed access pattern in which a small number of questions occur frequently while most questions appear rarely, capturing the repetition behavior commonly observed in practice. We vary the Zipf parameter (α) to obtain different repetition rates and study how workload skew affects MemBoost. For the lightweight Meta Controller (MC), we evaluate several small-scale LLMs, including Qwen-3.5-2B (Qwen Team, 2026), Ministral-3-3B-Instruct-2512 (Liu et al., 2026), and Qwen3-4B-Instruct-2507-FP8 (Yang et al., 2025). For the Large-LLM Oracle, we use Qwen3-14B-FP8-dynamic (Yang et al., 2025; Micikevicius et al., 2022). To support fast retrieval in the Associative Memory Engine (AME), we embed stored query-answer pairs using all-MiniLM-L6-v2 (Wang et al., 2020; Reimers & Gurevych, 2019) and perform approximate nearest-neighbor search with a FAISS cosine-similarity index (Douze et al., 2024).

Component	Hyperparameter	Value
Traffic simulation	Workload distribution	Zipfian
	Zipf exponent (α)	$\{0.8, 1.1, 1.4\}$
	Number of requests (N)	5,000
	Random seed	1
Associative Memory Engine	Embedding model	`all-MiniLM-L6-v2`
	Similarity metric	Cosine (FAISS inner product)
	Similarity threshold (τ)	0.95
	Retrieval top-k	3
LLM generation (MC)	Temperature	0.0
	Max generation tokens	4096
	Frequency / presence penalty	0.0
	Chat template kwargs	`"enable_thinking"=false`
LLM generation (Oracle)	Temperature	0.0
	Max generation tokens	4096
	Frequency / presence penalty	0.0
Environment	Python version	3.10
	NVIDIA driver	573.57
	CUDA version	12.8

Table 1. System and hyperparameter configuration for the continuous serving simulation.

To ensure reproducibility of our experiment runs, we attach the exact hyperparameters as found in Table 1 which we used across all experiments. All models were served using the vLLM library to optimize for throughput and latency (Kwon et al., 2023). To ensure fair inference time comparisons, the Meta-Controller including the small LLM and the Associative Memory Engine and the Large LLM Oracle (solver) were each deployed on a dedicated NVIDIA A100 80GB GPU. Both the Meta-Controller and the Large LLM Oracle are configured with Temperature = 0.0 to maintain deterministic responses. In addition the Oracle and the Router model had a 4,096 max tokens setting to allow for 5-shot Chain-of-Thought reasoning. We use the Business category of the MMLU-Pro dataset and ground truth label answers as our benchmark for comparing the accuracy of the meta-controller (MC) with the Large LLM (Oracle). As the Business category contains 768 examples it is large enough to fit 5000 requests of the chosen Zipf distributions.

3.2. Preliminary Results

Table 2 reports accuracy (%) against the MMLU-Pro ground-truth labels for different methods under Zipf-sampled query streams with varying repetition rates (Zipf exponent α). Across all settings, MemBoost consistently improves over the corresponding small-model baselines and achieves performance comparable to the oracle. Notably, MemBoost with Qwen3.5-2B even outperforms the oracle in all three

Model	Accuracy (%)		
	Zipf 0.8	Zipf 1.1	Zipf 1.4
Baselines			
Qwen3.5-2B	50.0	43.5	37.1
Ministral-3-3B-Instruct-2512	53.8	46.4	38.2
Qwen3-4B-Instruct-2507-FP8	74.5	75.6	80.5
Qwen3-14B-FP8-dynamic (Oracle)	76.4	79.9	85.0
MemBoost (ours)			
MemBoost (Qwen3.5-2B)	**76.7**	**81.8**	**87.4**
MemBoost (Ministral-3-3B-Instruct-2512)	76.2	79.7	85.0
MemBoost (Qwen3-4B-Instruct-2507-FP8)	76.1	79.8	85.0

Table 2. Accuracy (%) of different methods on MMLU-Pro under Zipf-sampled query streams with varying repetition rates (Zipf α).

workloads. We attribute this to MemBoost's ability to reuse previously generated high-quality answers through semantic memory: once a question (or a near-duplicate) has been answered correctly and written back, subsequent occurrences can be served directly from memory, avoiding additional generation errors. Finally, MemBoost exhibits a clear improvement as the workload becomes more skewed (larger α), since higher repetition rates lead to more memory hits, thereby increasing the fraction of queries answered using stored correct responses.

Figure 2. Average memory-use rate $\overline{I}_t$ (200-step window) over a 5,000-step Zipf-sampled query stream. Higher $\overline{I}_t$ indicates more queries served from AME and fewer oracle calls, implying lower total inference cost.

To quantify the serving cost of MemBoost, we report the average memory-use rate over a 200-step window, i.e., $\overline{I}_t = \frac{1}{200}\sum_{s=t-199}^{t} I_s$, where $I_s = 1$ indicates that the system answers using information retrieved from AME (and $I_s = 0$ indicates escalation to the oracle). Under our cost model, where the oracle cost c_O is assumed to dominate the Meta Controller and retrieval overhead $c_M + c_R$, a larger $\overline{I}_t$ corresponds to fewer oracle calls and thus lower total cost. Figure 2 plots $\overline{I}_t$ over the 5,000-step workload for different MC choices and Zipf distributions. In all settings, $\overline{I}_t$ increases over time as AME is progressively populated via the write-back mechanism, indicating that an increasing fraction of queries are served from memory rather than escalated to the oracle. Under our cost model, this implies that MemBoost achieves lower total cost than an oracle-only baseline. Moreover, memory usage is consistently higher under more skewed workloads (larger Zipf α), where repeated queries occur more frequently, further reducing the number of expensive oracle calls.

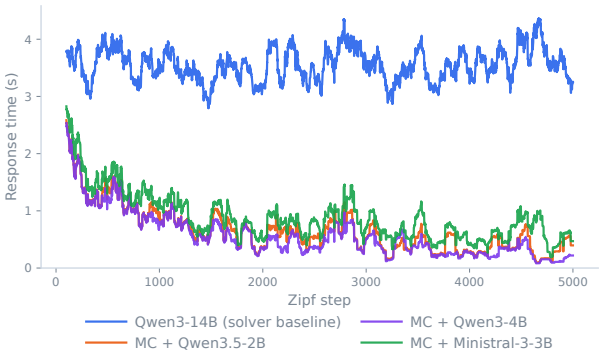

Figure 3. Response latency over time under Zipf-sampled workloads (average over the previous 100 steps). MemBoost reduces latency relative to the oracle-only baseline as an increasing fraction of queries are served from AME.

In addition to accuracy and system cost, we also report the latency of MemBoost. Figure 3 shows the average response time over the previous 100 steps during the 5,000-step Zipf-sampled workload. As MemBoost increasingly answers queries using AME over time, the average latency steadily decreases and remains well below the oracle-only baseline.

Overall, these results indicate that MemBoost is particularly effective for repeat-heavy workloads, preserving the strong answer quality of the larger oracle model while significantly reducing system cost and response latency.

4. Conclusion

In this paper, we introduced **MemBoost**, a memory-boosted architecture for efficient LLM serving under interactive, repeat-heavy workloads. By combining an Associative Memory Engine, a high-capability Large-LLM Oracle, and a lightweight Meta-Controller for routing and response composition, MemBoost reduces redundant large-model inference while preserving answer quality. We hope MemBoost provides a practical foundation for building next-generation LLM services that jointly optimize quality, cost, and responsiveness.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Asai, A., Wu, Z., Wang, Y., Sil, A., and Hajishirzi, H. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*, 2023.

Bang, F. GPTCache: An open-source semantic cache for LLM applications enabling faster answers and cost savings. In Tan, L., Milajevs, D., Chauhan, G., Gwinnup, J., and Rippeth, E. (eds.), *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pp. 212–218, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.nlposs-1.24. URL https://aclanthology.org/2023.nlposs-1.24/.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., Van Den Driessche, G. B., Lespiau, J.-B., Damoc, B., Clark, A., et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pp. 2206–2240. PMLR, 2022.

Chen, L., Zaharia, M., and Zou, J. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L., and Jégou, H. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

Du, X., Zheng, G., Wang, K., Zou, Y., Wang, Y., Deng, W., Feng, J., Liu, M., Chen, B., Peng, X., et al. Vul-rag: Enhancing llm-based vulnerability detection via knowledge-level rag. *ACM Transactions on Software Engineering and Methodology*, 2024.

Fan, T., Wang, J., Ren, X., and Huang, C. Minirag: Towards extremely simple retrieval-augmented generation. *arXiv preprint arXiv:2501.06713*, 2025.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.

Gill, W., Elidrisi, M., Kalapatapu, P., Ahmed, A., Anwar, A., and Gulzar, M. A. Meancache: User-centric semantic caching for llm web services. In *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1298–1310. IEEE, 2025.

Guu, K., Lee, K., Tung, Z., Pasupat, P., and Chang, M. Retrieval augmented language model pre-training. In *International conference on machine learning*, pp. 3929–3938. PMLR, 2020.

Hurst, A., Lerer, A., Goucher, A. P., Perelman, A., Ramesh, A., Clark, A., Ostrow, A., Welihinda, A., Hayes, A., Radford, A., et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., Riedel, S., and Grave, E. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43, 2023.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

Li, J., Xu, C., Wang, F., von Riedemann, I. M., Zhang, C., and Liu, J. Scalm: Towards semantic caching for automated chat services with large language models. In *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*, pp. 1–10. IEEE, 2024.

Liu, A. H., Khandelwal, K., Subramanian, S., Jouault, V., Rastogi, A., Sadé, A., Jeffares, A., Jiang, A., Cahill, A., Gavaudan, A., et al. Ministral 3. *arXiv preprint arXiv:2601.08584*, 2026.

Liu, J., Lin, J., and Liu, Y. How much can rag help the reasoning of llm? *arXiv preprint arXiv:2410.02338*, 2024.

Liu, X., Atalar, B., Dai, X., Zuo, J., Wang, S., Lui, J., Chen, W., and Joe-Wong, C. Semantic caching for low-cost llm serving: From offline learning to online adaptation. *arXiv preprint arXiv:2508.07675*, 2025.

Mansurova, A., Mansurova, A., and Nugumanova, A. Qarag: Exploring llm reliance on external knowledge. *Big Data and Cognitive Computing*, 8(9):115, 2024.

Micikevicius, P., Stosic, D., Burgess, N., Cornea, M., Dubey, P., Grisenthwaite, R., Ha, S., Heinecke, A., Judd, P., Kamalu, J., Mellempudi, N., Oberman, S., Shoeybi, M., Siu, M., and Wu, H. Fp8 formats for deep learning, 2022. URL https://arxiv.org/abs/2209.05433.

Moslem, Y. and Kelleher, J. D. Dynamic model routing and cascading for efficient llm inference: A survey. *arXiv preprint arXiv:2603.04445*, 2026.

Ni, A., Yin, P., Zhao, Y., Riddell, M., Feng, T., Shen, R., Yin, S., Liu, Y., Yavuz, S., Xiong, C., et al. L2ceval: Evaluating language-to-code generation capabilities of large language models. *Transactions of the Association for Computational Linguistics*, 12:1311–1329, 2024.

Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., Wu, T., Gonzalez, J. E., Kadous, M. W., and Stoica, I. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*, 2024.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Park, S., Jeon, S., Lee, C., Jeon, S., Kim, B.-S., and Lee, J. A survey on inference engines for large language models: Perspectives on optimization and efficiency. *arXiv preprint arXiv:2505.01658*, 2025.

Qwen Team. Qwen3.5-2b, 2026. URL https://huggingface.co/Qwen/Qwen3.5-2B. Hugging Face model card for the Qwen3.5-2B post-trained checkpoint.

Regmi, S. and Pun, C. P. Gpt semantic cache: Reducing llm costs and latency via semantic embedding caching. *arXiv preprint arXiv:2411.05276*, 2024.

Reimers, N. and Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Wang, C., Liu, X., Zhu, Y., Youssef, A., Nagpurkar, P., and Chen, H. Category-aware semantic caching for heterogeneous llm workloads. *arXiv preprint arXiv:2510.26835*, 2025a.

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., and Zhou, M. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, volume 33, pp. 5776–5788, 2020.

Wang, X., Liu, Y., Cheng, W., Zhao, X., Chen, Z., Yu, W., Fu, Y., and Chen, H. Mixllm: Dynamic routing in mixed large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 10912–10922, 2025b.

Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandra, A., Guo, S., Ren, W., Arulraj, A., He, X., Jiang, Z., Li, T., Ku, M., Wang, K., Zhuang, A., Fan, R., Yue, X., and Chen, W. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Yu, C., Wang, T., Shao, Z., and Jiang, S. Smartcache: Context-aware semantic cache for efficient multi-turn llm inference. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Zhang, H., Feng, T., and You, J. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Zipf, G. K. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Addison-Wesley Press, 1949.

A. Limitations

While MemBoost shows promising accuracy and reduced inference cost on MMLU-Pro, the framework has not yet been evaluated in more diverse and demanding settings, such as open-ended long-form question answering or coding tasks, where outputs are longer, and error modes may differ. In addition, although Zipf sampling is used to simulate repeat-heavy workloads, the current simulation largely reflects repeated questions drawn from a fixed benchmark. In real deployments, queries are often not exact duplicates but only semantically similar, which can increase the risk of retrieval errors. A more realistic evaluation would therefore include workloads with paraphrased or semantically overlapping queries to stress-test robustness, particularly with respect to potential false hits in AME retrieval.

B. Related Work

B.1. Retrieval-Augmented Language Modeling

Retrieval-augmented generation (RAG) is a common way to improve language models by letting them look up relevant information from an external collection of documents, and then using that retrieved text to help produce an answer (Lewis et al., 2020; Liu et al., 2024; Du et al., 2024; Mansurova et al., 2024). Early work explored different ways to combine retrieval with generation, including training the retriever together with the language model (Guu et al., 2020), using retrieval to support open-domain question answering and other knowledge-intensive tasks (Lewis et al., 2020; Izacard et al., 2023), and scaling retrieval to very large datastores to improve language modeling and downstream performance (Borgeaud et al., 2022). A key advantage of these approaches is that knowledge can be updated by changing the document index, without having to retrain the model, and retrieval can improve factual coverage without simply making the model larger (Guu et al., 2020; Lewis et al., 2020; Izacard et al., 2023). More recent work studies how to make retrieval easier to use in practice. For example, some methods encourage the model to retrieve only when needed and to check whether the retrieved evidence actually supports the response (Asai et al., 2023). Overall, however, the main goal of RAG remains the same: improving the quality of a single response by grounding it in external evidence. As a result, standard RAG typically does not focus on reusing previously generated answers across users or sessions, nor does it explicitly support escalating to a stronger model when retrieval is insufficient.

B.2. Semantic Caching

Semantic caching extends classical caching to LLM services by reusing previous answers not only for exact duplicate queries, but also for queries that are semantically similar. In practice, a system stores past query-response pairs, represents queries with embeddings, and returns a cached response when a new query is close enough to an existing one, avoiding a full model call (Bang, 2023; Regmi & Pun, 2024; Li et al., 2024; Wang et al., 2025a). Recent work has shown that this can significantly reduce latency and cost in real deployments, since many user requests concentrate on a small set of recurring intents (Bang, 2023; Li et al., 2024). At the same time, semantic caching introduces new reliability challenges. Because a false hit can directly harm correctness (e.g., returning an answer for a different but similar-looking question), while a false miss loses potential savings (Gill et al., 2025). This has motivated work on improving similarity matching and cache policies, including user-centric designs (Gill et al., 2025), context-aware caching for multi-turn interactions (Yu et al., 2025), and more principled formulations of cache management and eviction under unknown workloads (Liu et al., 2025). Overall, prior semantic caching research typically focuses on when to reuse cached outputs and how to manage the cache, but it often treats the fallback generation model and the caching layer as loosely coupled components (Bang, 2023; Li et al., 2024).

B.3. Routing in LLMs

A complementary line of work studies routing across multiple LLMs to reduce inference cost (Chen et al., 2023; Ong et al., 2024; Zhang et al., 2025; Wang et al., 2025b; Fedus et al., 2022). The core idea is to maintain a pool of models with different cost-quality trade-offs and decide, for each query, which model to use. Many systems follow a cascade pattern: attempt a query with a cheaper model first and escalate to a stronger model only when needed, guided by a confidence or quality estimate (Chen et al., 2023). More recent approaches learn routing policies directly from preference data so that routing decisions better match human judgments of quality while reducing expensive model calls (Ong et al., 2024). Beyond single-step routing, recent work also explores more adaptive and sequential routing strategies. For example, some methods treat routing as a multi-round decision process and train routers that interleave reasoning with routing actions (Zhang et al., 2025). Other work considers routing in heterogeneous model pools where different models have complementary strengths,

requiring robust routing strategies under distribution shift (Wang et al., 2025b). While routing methods are effective at reducing calls to the strongest model, they typically assume that each query is still handled by some model generation pass and do not explicitly leverage cross-request answer reuse as a primary mechanism for efficiency (Chen et al., 2023; Ong et al., 2024). Separately, there is also extensive work on routing within a single model, such as mixture-of-experts architectures, which improves efficiency by activating sparse components but addresses a different level of the system stack (Fedus et al., 2022).