LLM-dCache: Improving Tool-Augmented LLMs with GPT-Driven Localized Data Caching

Simranjit Singh*[1], Michael Fore*[1], Andreas Karatzas*[2], Chaehong Lee[1], Yanan Jian[1], Longfei Shangguan[3], Fuxun Yu[1], Iraklis Anagnostopoulos[2], Dimitrios Stamoulis[1]
Email:{simsingh, mifore, chaelee, yananjian, fuxunyu, distamo}@microsoft.com
longfei@pitt.edu, {andreas.karatzas, iraklis.anagno}@siu.edu – * Equal contribution
[1]Microsoft Corporation, USA [2]Southern Illinois University, USA [3]University of Pittsburgh, USA

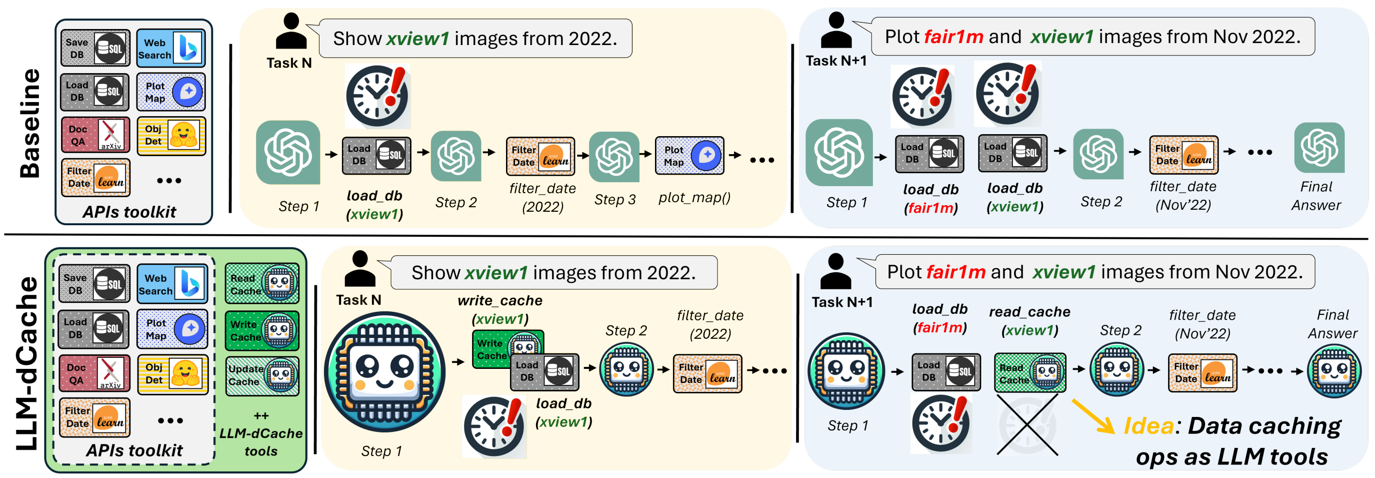

Fig. 1: GPT-driven data caching with LLM-dCache: we expose cache operations as callable API tools to GPT, enabling it to read and update cache data dynamically to respond to user queries. On a large-scale Copilot platform with hundreds of GPTs and terabytes of imagery, LLM-dCache speeds up task-completion by 1.24× on average across various prompting techniques.

***Abstract*—As Large Language Models (LLMs) broaden their capabilities to manage thousands of API calls, they are confronted with complex data operations across vast datasets with significant overhead to the underlying system. In this work, we introduce LLM-dCache to optimize data accesses by treating cache operations as callable API functions exposed to the tool-augmented agent. We grant LLMs the autonomy to manage cache decisions via prompting, seamlessly integrating with existing function-calling mechanisms. Tested on an industry-scale massively parallel platform that spans hundreds of GPT endpoints and terabytes of imagery, our method improves Copilot times by an average of 1.24× across various LLMs and prompting techniques.**

***Index Terms*—Tool-augmented agents, Large Language Models**

I. Introduction

Recent advances in Large Language Models (LLMs) have enhanced their reasoning capabilities towards solving complex problems, allowing them to manage thousands of tools and API calls efficiently [1], [2]. These improvements have unlocked their potential across large-scale systems, including UI/Web interfaces [3], [4], mobile apps [5], SQL backends [6], and remote sensing platforms [7]. These uses exemplify system-level complexity by requiring integration of various APIs for loading, filtering, processing, and visualizing data across multiple temporal and spatial dimensions [8].

As Copilots scale, the overhead on the underlying stack increases, from cloud endpoints to local execution devices [9], [10], catalyzing a fundamental shift in how we design large-scale LLM-based systems and software [4], [11]. However, early system optimizations primarily target simplified queries or well-defined benchmarks [12] that might not capture nuanced task patterns and data dependencies at the system level [13]. In realistic LLM workloads, data exhibits significant reusability. Consider an analyst who asks "*show me satellite images around Newport Beach, CA.*" with a subsequent prompt "*Now, detect airplanes in **this** area,*" demonstrating a scenario where data elements are repeatedly accessed.

In this work, we draw inspiration from spatiotemporal reusability patterns akin to those observed in CPU cache systems and we introduce LLM-dCache, a GPT-driven caching strategy to optimize LLM data access patterns. Our **key intuition** lies in a novel design choice to seamlessly integrate cache management as one of the LLM tools, facilitating a fully GPT-driven *plug-and-play* approach compatible with existing function-calling mechanisms with minimal overhead. Evaluated on a large-scale geospatial platform [13], LLM-dCache achieves latency reductions across various agents. We hope these findings motivate further exploration into empowering LLMs with other system level optimizations.

II. Related work

Model-level LLM optimization: several works aim to enhance LLM efficiency via model design improvements, such as quantization [14], pruning [15], KV token caching [16], or token compression [17]. Despite these advances, as motivated in [11], these techniques might have limitation in scenarios involving immutable black-box LLM models within cloud-based systems, where direct modifications to models and their inference mechanisms are limited [18]. We therefore focus on design optimizations at the system level [19], [20], which are especially important in large-scale Copilot platforms.

Application-level LLM optimization: methodologies such as MemGPT [19] and "model-as-a-resource" caching [21] align with our motivation. We also note advancements from the open-source community, with LangChain now supporting prompt-caching [22]. Similarly, drawing from hardware design and parallel computing, recent methods [11], [12], [20] explore parallel execution strategies. While these methods offer benefits for parallel or repeating tasks, they overlook the critical aspect of data locality, as they assume task chains with short-horizons [11] or template-based question-answer pairs with simplistic task interdependencies [12].

III. Methodology

Our goal is to design and assess LLM-dCache on realistic data patterns in large-scale cloud-first Copilot systems.

Cache operations: We aim to explore GPT's ability to understand *when* to read and use caching to execute a given task, as well as whether GPT is able to effectively implement a cache update policy autonomously. To allow GPT to handle system-level decisions via in-context prompting, we therefore define the operation of loading cache data as a tool in *GPT function calling*, *i.e.*, exposing its function definition in the GPT API call alongside other tool descriptions. Upon receiving a user query, GPT is informed of the current cache contents and decides whether to execute the cache loading tool.

Similarly, we experiment with an entirely prompt-based implementation of cache updating. We succinctly describe the update policy to GPT and furnish it with this round's load operations and cache contents in JSON format, then query GPT to return the updated cache state. We opt for the Least Recently Used (LRU) scheme as our primary cache update strategy, while we ablate other schemes.

Framing caching functions as GPT tools streamlines our implementation and makes it platform-agnostic. The cache read and update operations become part of GPT's decision-making process, thus requiring minimal changes. Additionally, granting the LLM autonomy over cache decisions allows our method to handle cache misses: upon a failed function call, the LLM is prompted to reassess its tool sequence, just as it would any other tool-selection missteps where the API return-message indicates a failure. This abstraction, simulating a main memory read after a cache-hit scenario and managed entirely at the LLM level, effectively positions the LLM as a memory controller. Such dynamic adaptability is key to rectifying inaccuracies in tool selection in real-time.

LLM-dCache prompting

```
As a Compiler handling geospatial data, you
have access to the following tools [..]

Tools:
- load_db(..images from database..)
- read_cache(..images from local cache..)
- ...

Given the user query, the cache content, and
the examples below, complete the task [..]

User Query: {question}
Cache: {cache content}

------
Example 1:
Query: Plot the xview1 images from 2022
Cache: {,}
Thought: The user asks for the xview1-2022
imagery. The cache is empty [..]
Action: To complete the task I will call
load_db(xview1-2022), then [..]
Answer: ..

------

Example 2:
Query: Show fair1m and xview1 imgs from 2022
Cache: {xview1-2022,}
Thought: The user wants both the fair1m-2022
and xview1-2022 images. The cache is already
contains the latter, so [..]
Action: To complete the task I will
first call load_db(fair1m-2022), then
read_cache(xview1-2022), and [..]
Answer: ..
```

Cache specifications: We represent and retrieve data as key-value pairs. As we operate on top of geospatial data, we opt to use the string template *dataset-year* as cache keys. We find this temporal granularity to be the most sensible (as opposed to longitude-latitude coordinates due to the spatial skewness of data around regions of interest like major cities). We then store as values the GeoPandas DataFrames containing the respective yearly imagery metadata – filenames, coordinates, detections, timestamps, *etc*. As is common in many geospatial platforms, the actual image files are not loaded into memory until needed for specific subsequent operations. As the yearly GeoPandas DataFrames typically occupy 50-100 MB, so we find it reasonable to set a cache size limit of 5 entries at a time. We note that such design choices are likely to be application specific, and we leave further ablations for future work.

IV. Experimental Setup

We use GeoLLM-Engine [13], a large-scale parameterizable LLM engine for geospatial tasks. Designed to capture agentic performance across hundreds of tools, the platform is equipped with long-horizon multi-tool LLM operations that require frequent data retrieval and filtering, a comprehensive suite of open-source APIs, interactive map UI, RAG, and data retrieval tools with over 1.1 million satellite images.

Benchmark. We expand the GeoLLM-Engine `sampler` to obtain variants of the `GeoLLM-Engine-1k` dataset. Specif-

TABLE I: LLM-dCache achieves latency reductions across models and prompting techniques with no degradation in overall agentic performance, as agent metrics are within established variance bounds [20].

Model	LLM-dCache	Success Rate (%) ↑	Correctness Rate (%) ↑	Obj. Det F1 (%) ↑	LCC R (%) ↑	VQA Rouge-L ↑	Avg Tokens / Task ↓	Avg Time / Task (s) ↓	Speedup ↑
GPT-3.5 Turbo									
CoT - Zero-Shot	✗	49.45	38.47	70.68	70.19	56.62	25.23k	6.96	–
	✓	49.40	37.96	69.71	71.23	55.57	25.55k	5.67	1.23 ×
CoT - Few-Shot	✗	54.42	70.50	89.03	82.19	62.58	30.81k	6.52	–
	✓	54.07	69.61	88.12	81.31	62.08	30.02k	5.29	1.23 ×
ReAct - Zero-Shot	✗	50.85	70.04	87.94	89.12	61.41	27.09k	7.29	
	✓	50.47	68.91	80.42	89.31	60.78	27.65k	5.47	1.33 ×
ReAct - Few-Shot	✗	63.45	71.06	82.59	92.36	69.35	34.40k	6.64	–
	✓	63.14	69.17	81.19	88.41	65.76	34.86k	5.77	1.15 ×
GPT-4 Turbo									
CoT - Zero-Shot	✗	70.48	82.04	86.34	84.91	69.78	26.81k	6.79	–
	✓	70.08	82.25	87.64	84.42	70.14	26.91k	5.16	1.32 ×
CoT - Few-Shot	✗	72.89	84.87	83.75	97.29	72.15	28.49k	6.76	–
	✓	72.16	85.48	82.95	99.72	72.72	28.92k	5.09	1.33 ×
ReAct - Zero-Shot	✗	74.30	85.80	88.49	94.52	72.18	30.51k	6.67	–
	✓	74.70	85.46	89.27	92.89	71.85	30.45k	5.70	1.17 ×
ReAct - Few-Shot	✗	76.71	85.67	64.49	98.95	74.23	36.62k	6.71	–
	✓	76.28	85.46	65.17	99.50	74.13	36.68k	5.72	1.17 ×

TABLE II: Zero-shot CoT (GPT-3.5 Turbo) runtime shows that overall latency reduction is highly dependent on data reuse rates. At high reuse, we observe only slight variability among different cache policies.

Cache Policy	No Cache	LRU					LFU	RR	FIFO
Data Reuse Rate	–	0%	20%	40%	60%	80%	80%	80%	80%
Avg Time/Task (s) ↓	5.81	5.81	5.84	5.62	5.03	4.92	5.16	5.36	5.25

ically, we extend the sampling-rate parameters and we incorporate rates that control the likelihood of data reuse. We selectively sample prompts with an 80% probability of requiring data already present in the cache, constructing a test dataset of 1,000 multi-step prompts (with an overall set of approximately 50,000 tool calls). Additionally, we prepare a mini 500 query set for ablations. Last, we use the `model-checker` module to verify the functional correctness of the generated tasks.

Metrics. For agent performance, we adhere to established evaluation practices [1], [7], measuring the *Success Rate* (proportion of tasks successfully completed), the *Correctness Ratio* (proportion of correct tool calls, since an erroneous tool might not affect successful task completion), and the *ROUGE-L* score. We also report performance on the underlying remote sensing tasks, with F1 and recall for object detection and land coverage classification (LCC), respectively, and ROUGE for visual question answering (VQA) [13].

To evaluate cache effectiveness, we report GPT-hits (*i.e.*, the LLM correctly utilizes the cache over main memory). We also track the average number of tokens and time per task, with an expectation that higher cache reuse (being 5-10× faster than main memory access) will result in reduced overall API completion times. To capture latency, we follow [20] by maintaining a running average per tool operation, discarding any outliers beyond two standard deviations from the mean. To avoid congestion and ensure accurate endpoint response times, we deploy hundreds of GPT instances specifically for this evaluation, isolated from production traffic.

V. Results

LLM-dCache improves task-completion times across different configurations – GPT-4 and GPT-3.5, with Chain-of-Thought and ReAct, in both few-shot and zero-shot scenarios – by 1.24× on average (Table I). Caching does not degrade the quality of output and functionality of the agent, as agent metrics are within established variance [20]. Overall, we notice that gains primarily depend on dataset reusability patterns, not the choice of model or prompting strategy.

To corroborate this observation, we conduct an ablation with multiple `mini-val` subsets, each containing 500 queries but with varying reusability rates. Table II (top) shows higher reusability rates correlate with greater latency savings. LRU, LFU, RR, and FIFO produce no clear latency differences.

We aim to position our exploration within a broader shift towards empowering LLMs with system-level optimization decisions. To this end, we make the deliberate choice of treating cache operations as prompt-based GPT tools (*e.g.*, explaining the LRU scheme via prompts) instead of a direct programmatic implementation of the logic. In support of this, our ablation in Table III compares programmatic cache operations with those driven by GPT. We find that all GPT-driven variants closely match the fully programmatic approach, which could be considered an upper-bound in terms of effectiveness and reliability, with cache "hit" rates consistently around 97% and similar latency. This demonstrates the versatility and potential of LLM-guided cache management in lieu of traditional programmatic solutions. Our hope is that this perspective

TABLE III: GPT-driven cache operations produce performance metrics and latency very similar to programmatic implementation of caching, demonstrating GPT's ability to successfully execute system optimization tasks.

Model	Cache Read	Policy Imp.	Cache Hit Rate (%) ↑	Success Rt (%) ↑	Correctness Rt (%) ↑	Obj. Det F1 (%) ↑	LCC R (%) ↑	VQA Rouge-L ↑	Avg Tokens /Task ↓	Avg Time / Task (s) ↓
GPT-4 Turbo CoT - Few-Shot	Python	Python	-	72.49	85.40	85.11	99.46	72.64	28.76k	5.07
	GPT-4	Python	96.59	72.16	85.41	83.00	98.69	72.35	28.73k	5.11
	Python	GPT-4	97.73	72.29	84.75	82.79	99.59	72.09	28.64k	5.09
	GPT-4	GPT-4	96.16	72.16	85.48	82.95	99.72	72.72	28.92k	5.09

will motivate work for integrating LLMs into other system design optimizations [23], from execution at the edge [24] to energy/power optimizations and thermal management [25].

Limitations and future work. Our study focuses on agentic performance and average latency for cloud-first environments with extensive use of cloud endpoints. It is meaningful to include more system performance metrics, such as energy and power consumption. To this end, we will explore GPT alternatives that can be run locally, such as Llama-3 and Phi-3.5. Given that our approach implements cache operations as callable API tools, we should be able to seamlessly incorporate this with other non-GPT tool-augmented agents across different computational environments. Last, we plan to extend our evaluation beyond the geospatial domain to a wider range of orthogonal tasks also considered in recent system-level LLM optimization papers [10], [11].

VI. Conclusion

In this paper, we introduced LLM-dCache, a framework designed to optimize LLM data access patterns through a cache mechanism treated as callable API tools. By allowing LLMs to autonomously manage cache operations, we integrated caching with existing function-calling mechanisms, enabling improvements in system efficiency across various models and prompting techniques. Our work underscores the potential of leveraging LLMs for system-level optimizations in complex data-intensive environments.

Acknowledgements

This work is supported in part by grant NSF CCF 2324854.

References

[1] Y. Zhuang, Y. Yu, K. Wang, H. Sun, and C. Zhang, "Toolqa: A dataset for llm question answering with external tools," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[2] Y. Qin, S. Liang *et al.*, "Toolllm: Facilitating large language models to master 16000+ real-world apis," *International Conference on Learning Representations*, 2024.

[3] S. Singh, G. Pavlakos, and D. Stamoulis, "Evaluating zero-shot gpt-4v performance on 3d visual question answering benchmarks," *arXiv preprint arXiv:2405.18831*, 2024.

[4] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Ou, Y. Bisk, D. Fried, U. Alon, and G. Neubig, "Webarena: A realistic web environment for building autonomous agents," in *ICLR 2024 Foundation Models for Decision Making*, 2024.

[5] J. Y. Koh, R. Lo, L. Jang, V. Duvvur, M. C. Lim, P.-Y. Huang, G. Neubig, S. Zhou, R. Salakhutdinov, and D. Fried, "Visualwebarena: Evaluating multimodal agents on realistic visual web tasks," *ACL*, 2024.

[6] C. Guo, Z. Tian, J. Tang, S. Li, Z. Wen, K. Wang, and T. Wang, "Retrieval-augmented gpt-3.5-based text-to-sql framework with sample-aware prompting and dynamic revision chain," in *International Conference on Neural Information Processing*. Springer, 2023, pp. 341–356.

[7] S. Singh, M. Fore, and D. Stamoulis, "Evaluating tool-augmented agents in remote sensing platforms," in *ICLR 2024 2nd Workshop on ML for Remote Sensing*, 2024.

[8] Z. Yuan, Y. Shang, Y. Zhou, Z. Dong, C. Xue, B. Wu, Z. Li, Q. Gu, Y. J. Lee, Y. Yan *et al.*, "Llm inference unveiled: Survey and roofline model insights," *arXiv preprint arXiv:2402.16363*, 2024.

[9] Q. J. Hu, J. Bieker, X. Li, N. Jiang, B. Keigwin, G. Ranganath, K. Keutzer, and S. K. Upadhyay, "Routerbench: A benchmark for multi-llm routing system," *arXiv preprint arXiv:2403.12031*, 2024.

[10] M. Fore, S. Singh, and D. Stamoulis, "Geckopt: Llm system efficiency via intent-based tool selection," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, 2024, pp. 353–354.

[11] S. Kim, S. Moon, R. Tabrizi, N. Lee, M. W. Mahoney, K. Keutzer, and A. Gholami, "An llm compiler for parallel function calling," *arXiv preprint arXiv:2312.04511*, 2023.

[12] X. Ning, Z. Lin, Z. Zhou, H. Yang, and Y. Wang, "Skeleton-of-thought: Large language models can do parallel decoding," *arXiv preprint arXiv:2307.15337*, 2023.

[13] S. Singh, M. Fore, and D. Stamoulis, "Geollm-engine: A realistic environment for building geospatial copilots," in *CVPR Workshop EarthVision*, 2024.

[14] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[15] X. Ma, G. Fang, and X. Wang, "Llm-pruner: On the structural pruning of large language models," *Advances in neural information processing systems*, vol. 36, pp. 21 702–21 720, 2023.

[16] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.

[17] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu, "Llmlingua: Compressing prompts for accelerated inference of large language models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 13 358–13 376.

[18] M. Fore, S. Singh, C. Lee, A. Pandey, A. Anastasopoulos, and D. Stamoulis, "Unlearning climate misinformation in large language models," in *Proceedings of the 1st Workshop on Natural Language Processing Meets Climate Change (ClimateNLP 2024)*. ACL, 2024.

[19] C. Packer, V. Fang, S. G. Patil, K. Lin, S. Wooders, and J. E. Gonzalez, "Memgpt: Towards llms as operating systems," *arXiv preprint arXiv:2310.08560*, 2023.

[20] S. Singh, A. Karatzas, M. Fore, I. Anagnostopoulos, and D. Stamoulis, "An llm-tool compiler for fused parallel function calling," *arXiv preprint arXiv:2405.17438*, 2024.

[21] M. Xu, D. Niyato, H. Zhang, J. Kang, Z. Xiong, S. Mao, and Z. Han, "Cached model-as-a-resource: Provisioning large language model agents for edge intelligence in space-air-ground integrated networks," *arXiv preprint arXiv:2403.05826*, 2024.

[22] LangChain docs, "Langchain llm caching," 2024, accessed: May 2024.

[23] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-path nas: Designing hardware-efficient convnets in less than 4 hours," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2019, pp. 481–497.

[24] L. E. Erdogan, N. Lee, S. Jha, S. Kim, R. Tabrizi, S. Moon, C. Hooper, G. Anumanchipalli, K. Keutzer, and A. Gholami, "Tinyagent: Function calling at the edge," *arXiv preprint arXiv:2409.00608*, 2024.

[25] D. Marculescu, D. Stamoulis, and E. Cai, "Hardware-aware machine learning: Modeling and optimization," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.