Make your GenAI Application Smarter, Faster and Cheaper with LLM Caching

Kunal Banerjee

Walmart Global Tech
Bengaluru, India

22-Aug-2025

Contents

Brief Biography

- Professional Experience
  - **Walmart Global Tech:** Principal Data Scientist (Sep 2020 – Present)
  - **Intel Labs:** Research Scientist (Aug 2015 – Aug 2020)
- Educational Background
  - **PhD:** Computer Science & Engineering, IIT Kharagpur (2010 – 2015)
  - **BTech:** Computer Science & Engineering, Heritage Inst of Tech (2004 – 2008)
- Research Highlights
  - 9 peer-reviewed journal publications
  - More than 50 peer-reviewed conference/workshop publications
  - More than 1300 citations
  - Best PhD Thesis Awards, Best Paper Awards, Special Mentions
  - IEEE Senior Member (since 2020), ACM Senior Member (since 2021)
  - AV Luminary Award (Top 10 Data Scientists) at DataHack Summit 2023

Primary Reference

- **Title:** waLLMartCache: A Distributed, Multi-Tenant and Enhanced Semantic Caching System for LLMs [link]
- **Corresponding author:** Kunal Banerjee
- **Other contributors:** Soumik Dasgupta, Anurag Wagh, Lalitdutt Parsai, Binay Gupta, Geet Vudata, Shally Sangal, Sohom Majumdar, Hema Rajesh, Anirban Chatterjee
- **Venue:** *International Conference on Pattern Recognition (ICPR)*, December 2024, pages: 232-248

Generative AI and LLMs

Generative AI

Generative AI is a type of artificial intelligence that creates new content like text, images, music, audio, or videos, based on patterns learned from existing data. It uses machine learning models, often large models, that are pre-trained on vast amounts of data. Instead of just identifying patterns or making predictions, generative AI actively generates new instances or variations of the data it has been trained on.

Large Language Models (LLMs)

LLMs are advanced artificial intelligence systems designed to understand and generate human language. They leverage deep learning techniques, particularly the transformer architecture, to analyze vast amounts of text data and learn complex patterns in language. This allows them to perform various natural language processing (NLP) tasks like generating text, translating languages, and understanding the meaning of text.

Generative AI is in Focus

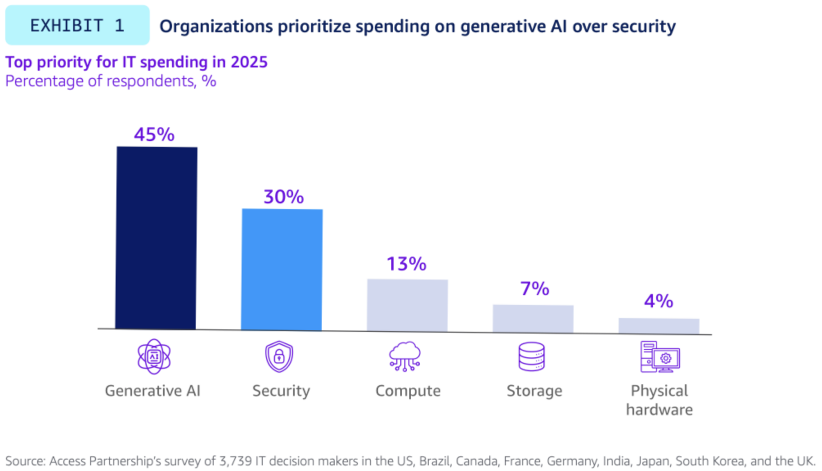


Figure 1: IT leaders rank generative AI as their top budget priority for 2025

Source: VentureBeat article

Generative AI Spending on the Rise

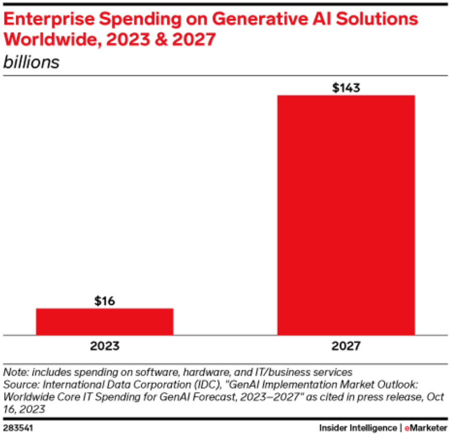
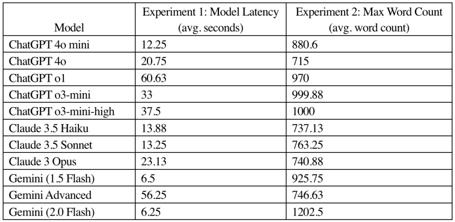


Figure 2: In 2027, enterprise spending on generative AI solutions worldwide will grow nearly ninefold from its total in 2023

Source: EMarketer article

Latencies for Different LLMs



Model	Experiment 1: Model Latency (avg. seconds)	Experiment 2: Max Word Count (avg. word count)
ChatGPT 4o mini	12.25	880.6
ChatGPT 4o	20.75	715
ChatGPT o1	60.63	970
ChatGPT o3-mini	33	999.88
ChatGPT o3-mini-high	37.5	1000
Claude 3.5 Haiku	13.88	737.13
Claude 3.5 Sonnet	13.25	763.25
Claude 3 Opus	23.13	740.88
Gemini (1.5 Flash)	6.5	925.75
Gemini Advanced	56.25	746.63
Gemini (2.0 Flash)	6.25	1202.5

Figure 3: A summary of both average latency and word count metrics

Source: Aidocmaker article

- Latency and verbosity are *ideally* inversely related
- Latency is more for LLMs that are meant for reasoning (e.g., ChatGPT o1)
- Latency should be greater if images/audios/videos are generated
- **All these latencies may be unacceptable for solutions catering to large number of requests**

Motivation

- Large Language Models (LLMs) are everywhere
- Its applications include question-answering, translation, summarization among many others
- Unfortunately, LLMs incur high usage costs and latency
- **An LLM focused caching system can significantly reduce usage costs along with latency**
- Additionally, too many requests coming from a user may lead to (temporary) suspension which hampers developer productivity and customer experience significantly – a cache may help alleviate this problem as well

GPTCache

★ Fu Bang, *“GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings,”* ACL 2023

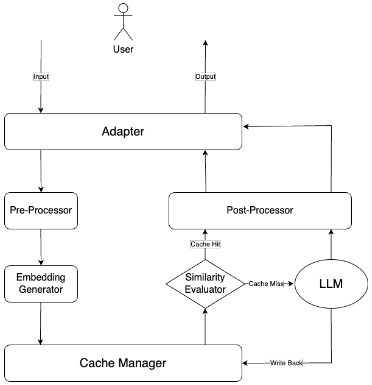


Figure 4: The overall architecture of GPTCache

Cache Manager in GPTCache

Cache manager lies at the core of GPTCache:

- **L1 Cache Storage:** When a user query arrives, it is first converted into an embedding vector and stored in a vector database; along with an embedding vector, a unique scalar id is also generated
- **L2 Cahce Storage:** Stores the unique scalar ids generated by the L1 cache along with the corresponding LLM responses
- **Eviction Management:** Clears the cache by following a pre-determined policy, e.g., LRU, FIFO, to maintain the cache capacity

N.B. The original GPTCache paper uses the terms *vector storage* and *cache storage* whereas, we use *L1 cache storage* and *L2 cache storage*, respectively. We deviate from the original terminology for two reasons: (i) we found the terms vector storage and cache storage confusing because both are part of GPTCache, and (ii) L1 and L2 storages are commonly used terms in the context of caching and also imply the order in which these storages are accessed (similar to standard caches).

Limitations of GPTCache / Contributions of waLLMartCache

1. We introduce the support for a **new database Redis** in GPTCache – this is used as L2 cache in our designed system (our PR is already merged with GPTCache)
2. Currently, GPTCache is implemented to be run on a single node which we enhance to **span across multiple nodes** to handle industry-scale requests and consequently, we also designed a **distributed eviction manager**
3. We further create partitions for individual tenants (clients) so that these can be hosted together while **maintaining semantic separations**
4. We present a **decision engine** (i.e., an enhanced semantic similarity checker) that decides whether to cache an LLM response based on retail business use-cases
5. We showcase that **preloading FAQs** (which can be set to be stored persistently in the memory) while booting the LLM cache can be a simple yet effective strategy to boost cache hits significantly

Dataset Description

- Collected from in-house *Generative AI Playground*
- Any associate can pose a query and get its response from LLM
- Covers a large spectrum of queries related to retail industry
- The chosen queries are in the range from 500 to 1000 tokens
- The response lengths vary and can be very large

Incorporating Redis as a Database

Table 1: Experimental results with Redis as an L2 cache storage

Content Size (#tokens)	**#Concurrent Users**	**#Requests**	**#RPS**	**Average Response Time (ms)**	**Median Response Time (ms)**
<2K	500	135758	156	119	45
	1000	170530	258	614	190
2K to 5K	500	83035	155	141	55
	1000	180694	239	1051	180
5K to 10K	500	81536	158	152	91
	1000	122708	217	1523	240
>10K	500	91331	154	227	140
	1000	103840	180	2463	380
All buckets	500	135365	150	173	48
	1000	190668	230	952	200

☞ Due to our internal non-competition policy, we refrain from mentioning the other databases that we explored; nevertheless, it may be noted that *the closest competition scaled to only 30% of the Requests Per Second (RPS) that was registered for Redis for the same configuration*

To Cache or Not to Cache (Example 1)

User1

Query: How much income tax did I pay last year?
Response: You paid Rs. X as income tax for FY 2024-25.

To Cache or Not to Cache (Example 1)

User1

Query: How much income tax did I pay last year?
Response: You paid Rs. X as income tax for FY 2024-25.

User1

Query: How much income tax did I pay last year?
Response: You paid Rs. X as income tax for FY 2024-25.

To Cache or Not to Cache (Example 1)

User1

Query: How much income tax did I pay last year?
Response: You paid Rs. X as income tax for FY 2024-25.

User1

Query: How much income tax did I pay last year?
Response: You paid Rs. X as income tax for FY 2024-25.

☞ In this case, caching should be helpful!

To Cache or Not to Cache (Example 2)

User1

Query: How much income tax did I pay last year?
Response: You paid Rs. X as income tax for FY 2024-25.

To Cache or Not to Cache (Example 2)

User1

Query: How much income tax did I pay last year?
Response: You paid Rs. X as income tax for FY 2024-25.

User2

Query: How much income tax did I pay last year?
Response: You paid Rs. X as income tax for FY 2024-25.

To Cache or Not to Cache (Example 2)

User1

Query: How much income tax did I pay last year?
Response: You paid Rs. X as income tax for FY 2024-25.

User2

Query: How much income tax did I pay last year?
Response: ~~You paid Rs. X as income tax for FY 2024-25.~~

☞ In this case, caching can be erroneous!

To Cache or Not to Cache (Example 2)

User1

Query: How much income tax did I pay last year?
Response: You paid Rs. X as income tax for FY 2024-25.

User2

Query: How much income tax did I pay last year?
Response: ~~You paid Rs. X as income tax for FY 2024-25.~~

- ☞ In this case, caching can be erroneous!
- ☞ **Semantic separation** needs to be maintained across tenants

Distributed LLM Cache with Multi-Tenancy

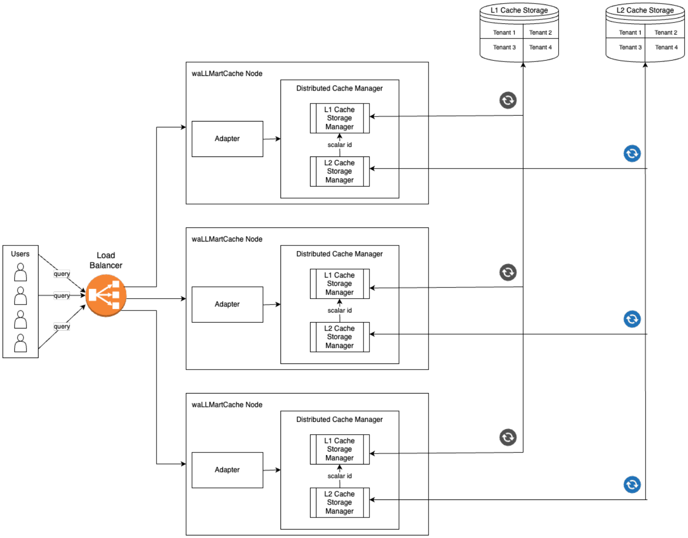


Figure 5: Design of our distributed cache for LLMs

To Cache or Not to Cache (Example 3)

User1

Query: What was yesterday's average temperature?
Response: It was 30°C.

To Cache or Not to Cache (Example 3)

User1

Query: What was yesterday's average temperature?
Response: It was 30°C.

User1

Query: What was yesterday's average temperature?
Response: It was 30°C.

To Cache or Not to Cache (Example 3)

User1

Query: What was yesterday's average temperature?
Response: It was 30°C.

User1

Query: What was yesterday's average temperature?
Response: It was 30°C.

☞ Even if the exact same question is asked the next day (by the same user), returning the cached answer would be wrong!

To Cache or Not to Cache (Example 3)

User1

Query: What was yesterday's average temperature?
Response: It was 30°C.

User1

Query: What was yesterday's average temperature?
Response: It was 30°C.

☞ Even if the exact same question is asked the next day (by the same user), returning the cached answer would be wrong!
☞ Therefore, we do not cache those queries whose responses may differ with time

To Cache or Not to Cache (Example 4)

User1

Query: Optimize the following code:
const = 2.0; pi = 22.0/7.0; circumference = const * pi * radius;
Response: circumference = 6.2857 * radius;

To Cache or Not to Cache (Example 4)

User1

Query: Optimize the following code:
const = 2.0; pi = 22.0/7.0; circumference = const * pi * radius;
Response: circumference = 6.2857 * radius;

User1

Query: Optimize the following code:
pi = 22.0/7.0; circumference = 2.0 * pi * radius;
Response: circumference = 6.2857 * radius;

To Cache or Not to Cache (Example 4)

User1

Query: Optimize the following code:
const = 2.0; pi = 22.0/7.0; circumference = const * pi * radius;
Response: circumference = 6.2857 * radius;

User1

Query: Optimize the following code:
pi = 22.0/7.0; circumference = 2.0 * pi * radius;
Response: circumference = 6.2857 * radius;

☞ In this case, caching should be helpful!

To Cache or Not to Cache (Example 5)

User1

Query: Optimize the following code:
const = 2.0; pi = 22.0/7.0; circumference = const * pi * radius;
Response: circumference = 6.2857 * radius;

To Cache or Not to Cache (Example 5)

User1

Query: Optimize the following code:
const = 2.0; pi = 22.0/7.0; circumference = const * pi * radius;
Response: circumference = 6.2857 * radius;

User1

Query: Optimize the following code:
pi = 22.0*7.0; circumference = 2.0 * pi * radius;
Response: ...

To Cache or Not to Cache (Example 5)

User1

Query: Optimize the following code:
const = 2.0; pi = 22.0/7.0; circumference = const * pi * radius;
Response: circumference = 6.2857 * radius;

User1

Query: Optimize the following code:
pi = 22.0*7.0; circumference = 2.0 * pi * radius;
Response: circumference = 6.2857 * radius;

☞ Vector embedding based semantic similarity between two queries having code snippets is a notoriously difficult task that often leads to false positives, i.e., **erroneous cache hits**!

To Cache or Not to Cache (Example 5)

User1

Query: Optimize the following code:
const = 2.0; pi = 22.0/7.0; circumference = const * pi * radius;
Response: circumference = 6.2857 * radius;

User1

Query: Optimize the following code:
pi = 22.0*7.0; circumference = 2.0 * pi * radius;
Response: circumference = 6.2857 * radius;

☞ Vector embedding based semantic similarity between two queries having code snippets is a notoriously difficult task that often leads to false positives, i.e., **erroneous cache hits**!
☞ The percentage of true cache hits for queries containing codes was only ∼ 10% in our experiments

Improved Decision Engine for Caching

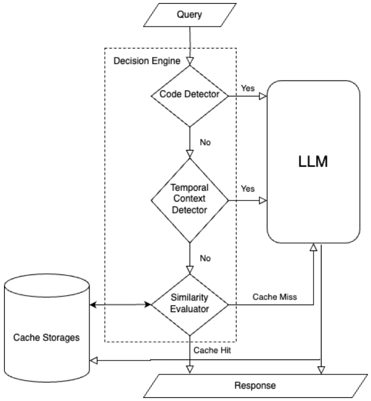


Figure 6: Design of our decision engine

- **Code Detector:** Detects code snippets
- **Temporal Context Detector:** Detects temporal dependence
- **Similarity Evaluator:** This is identical to that of GPTCache

To Cache or Not to Cache (Example 6)

User1

Query: What is the price of iPhone 16?
Response: It is Rs. X.

To Cache or Not to Cache (Example 6)

User1

Query: What is the price of iPhone 16?
Response: It is Rs. X.

User1

Query: What is the price of iPhone 16?
Response: It is Rs. X.

To Cache or Not to Cache (Example 6)

User1

Query: What is the price of iPhone 16?
Response: It is Rs. X.

User1

Query: What is the price of iPhone 16?
Response: It is Rs. X.

☞ If the retailer has dynamic pricing, then cached response can be erroneous!

To Cache or Not to Cache (Example 6)

User1

Query: What is the price of iPhone 16?
Response: It is Rs. X.

User1

Query: What is the price of iPhone 16?
Response: It is Rs. X.

☞ If the retailer has dynamic pricing, then cached response can be erroneous!
☞ In this case, there was no *temporal context* in the query
☞ There is further scope to improve the decision engine based on the business requirements

Pre-loading FAQs into the Cache

- Chat bot applications often encounter same questions repeatedly – such common questions and their responses can be collated into Frequently Asked Questions (FAQs) and can be pre-loaded into the cache to boost up its hits
- The FAQs can be volatile or non-volatile; the following experiment is with non-volatile configuration

Table 2: Experimental results for finding the efficacy of pre-loaded FAQs

#Concurrent Users	**#Requests**	**#RPS**	**Average Response Time (ms)**	**Median Response Time (ms)**	**Cache Hit w/o FAQ**	**Cache Hit w/ FAQ**
1000	293304	329	45	35	80%	90%

Ablation Study Dataset

- We chose a set of
  1. 100 prompts containing code
  2. 100 textual prompts containing temporal context
  3. 100 (regular) textual prompts without any temporal context or code
- For each prompt, we create
  1. 5 semantically similar prompts – leading to *correct cache hits*
  2. 5 dissimilar prompts – leading to *correct cache misses* – note that these prompts need to be *pairwise* semantically different as well
- We use GPT-4 to generate the semantically similar and dissimilar prompts from a given textual prompt
- For similar prompts: active to passive voice, compound to multiple simple sentences, positive to double negative (e.g., “present” to “not absent”), replace a single or multiple words by their synonyms
- For dissimilar prompts: positive to negative, replace a single or multiple words by their antonyms, replace the entire prompt by some random unrelated Wikipedia sentence(s)

Ablation Study Dataset (contd.)

- For prompts containing code
  - to generate similar prompts, we replaced equations by their mathematical equivalent ones (e.g., $y + y$ to $2 * y$), or added some constant and later subtracted the same constant, etc.
  - to generate dissimilar prompts, we deliberately changed some of the operators, or removed equations partially or totally
- We have manually checked whether the resulting prompts were indeed similar or dissimilar
- Thus, each prompt led to 10 additional prompts, and overall we had 3300 prompts (including the original ones)
- The entire set is randomly shuffled before invoking the LLM cache

Ablation Experiment 1

Table 3: Effects of Redis, distributed cache, decision engine and pre-loading FAQs on semantic caching

Method	**Correct Hit**			**Incorrect Hit**			**Correct Miss**			**Incorrect Miss**			**Acc(%)**	
	Reg	**Cde**	**Tmp**	**Reg**	**Cde**	**Tmp**	**Reg**	**Cde**	**Tmp**	**Reg**	**Cde**	**Tmp**	Reg	All
Oracle	500	500	0	0	0	0	600	600	1100	0	0	0	100	100
GPTCache	401	422	0	59	435	77	550	222	545	90	21	478	86.4	64.8
WMC(1N)	401	422	0	59	435	77	550	222	545	90	21	478	86.4	64.8
WMC(4N)	399	418	0	61	438	77	548	222	543	92	22	480	86.1	64.5
WMC(4N)+DE	399	0	0	61	0	0	548	600	1100	92	500	0	86.1	80.2
WMC(4N)+DE+FAQ	488	0	0	65	0	0	498	600	1100	49	500	0	89.6	81.4

- **Cde:** Prompts containing code
- **Tmp:** Textual prompts containing temporal context
- **Reg:** Regular textual prompts without any temporal context
- **Oracle:** Makes no incorrect cache hit or miss – it is used to benchmark GPTCache and our waLLMartCache
- **WMC(1N):** waLLMartCache deployed to only a single node
- **WMC(4N):** waLLMartCache distributed to four nodes
- **WMC(4N)+DE:** WMC(4N) augmented with Decision Engine
- **WMC(4N)+DE+FAQ:** WMC(4N)+DE pre-loaded with FAQs

Ablation Experiment 2

Table 4: Effects of multi-tenancy on semantic caching

Method	T	Correct Hit			Incorrect Hit			Correct Miss			Incorrect Miss			Acc(%)	
		Reg	Cde	Tmp	Reg	Cde	Tmp	Reg	Cde	Tmp	Reg	Cde	Tmp	Reg	All
Oracle	1	250	250	0	0	0	0	300	300	550	0	0	0	100	100
	2	250	250	0	0	0	0	300	250	550	0	0	0		
GPTCache	1	199	215	0	55	288	59	250	36	245	46	11	246	82	57
	2	202	207	0	52	291	61	251	36	240	45	16	249		
WMC(4N)+DE+FAQ	1	211	0	0	55	0	0	249	300	550	35	250	0	83.8	79.4
	2	212	0	0	54	0	0	250	300	550	34	250	0		
WMC(4N)+DE+FAQ+MT	1	244	0	0	33	0	0	247	300	550	26	250	0	89.5	81.3
	2	244	0	0	32	0	0	249	300	550	25	250	0		

- **Cde:** Prompts containing code
- **Tmp:** Textual prompts containing temporal context
- **Reg:** Regular textual prompts without any temporal context
- **Oracle:** Makes no incorrect cache hit or miss – it is used to benchmark GPTCache and our waLLMartCache
- **WMC(4N)+DE+FAQ:** WMC(4N)+DE pre-loaded with FAQs
- **WMC(4N)+DE+FAQ+MT:** WMC(4N)+DE+FAQ with multi-tenancy support

Conclusion

- LLMs have wide applicability but they suffer from high usage costs and latency
- An LLM cache can alleviate these hurdles
- The original work GPTCache needs lot of enhancements before it can be adopted for industry-grade applications, e.g., multi-node and multi-tenant support
- Moreover, one may need to judiciously decide which queries need to be cached
- Pre-loading FAQs can be a simple yet effective startegy to boost cache hits
- There is scope to improve semantic similarity for queries involving codes
- *Idea to explore:* Check if switching LLMs in the interim based on historical data is a good idea or not, i.e., use a more powerful LLM initially so that our cache is populated with richer responses and then transition to a less powerful LLM if we believe that most responses in the foreseeable future will be returned from the cache

References

1. Bang, F.: GPTCache: An open-source semantic cache for LLM applications enabling faster answers and cost savings. In: NLP-OSS. pp. 212–218. Association for Computational Linguistics (2023)
2. Bengio, Y., Ducharme, R., Vincent, P.: A neural probabilistic language model. In: NeurIPS (2000)
3. Dar, S., Franklin, M.J., Jonsson, B., Srivastava, D., Tan, M.: Semantic data caching and replacement. In: VLDB. pp. 330–341 (1996)
4. Handy, J.: The cache memory book. Academic Press Professional, Inc. (1993)
5. Lee, D., Chu, W.W.: Semantic caching via query matching for web sources. In: CIKM. pp. 77–85 (1999)
6. OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., et al.: GPT-4 technical report (2024).
7. Wang, J., Yi, X., Guo, R., Jin, H., et al.: Milvus: A purpose-built vector data management system. In: SIGMOD. pp. 2614–2627 (2021)

Thank You!

✉ kunal.banerjee1@walmart.com