Enhancing Resemblance Matching using Structural Awareness for Hierarchical LLM Caching

***Dr. Chaitanya Udatha[1], Krithi Chippada[2], Satvik Dabbara[3]**

[1,2,3] Information Technology, Mahatma Gandhi Institute of Technology (MGIT), Hyderabad, India.

DOI: https://doi.org/10.51584/IJRIAS.2026.11050049

Received: 03 May 2026; Accepted: 08 May 2026; Published: 27 May 2026

ABSTRACT

Large Language Models (LLMs) have become an integral part of our daily lives; they are used for tasks such as chatbots in customer services and require a lot of computing power. If the user base is large, generating different responses to similar queries results in slower performance and increased computational latency. Hence hierarchical caching systems like GPTCache and MinCache were introduced to reduce redundant inference using exact matching, resemblance matching and semantic matching of the prompts with stored queries to reuse LLM responses for similar queries. However, Unigram-based resemblance caching mechanisms are susceptible to adversarial lexical reordering leading to excessive false positive cache hits. The proposed research introduced structural-aware resemblance matching to improve the robustness of the system without violating the ideology of MinCache by using lightweight and fast similarity caching mechanisms. It has achieved 7.39x safer cache reuse compared to the standard 1-g Minhash while preserving 79.5% of resemblance layer throughput and maintained overall accuracy.

Keywords: Structural Shingling, Skip-Gram Similarity, Cache Eviction Policies, Large Language Models, Low-Latency Inference, Paraphrase Identification, Adversarial Text Reordering**.**

INTRODUCTION

LLMs have demonstrated their applications in wide range of Natural Language Processing tasks such as question-answering systems, text generation, summarization, reasoning, translation tasks, conversational agents, sentiment analysis, decision support and information retrieval systems, They are associated with risks such as high deployment cost and high energy consumption problem [1,2].

LLM users often input long and complex prompts which requires larger models and longer outputs this results in increased latency, additionally transformer attention also causes quadratic memory growth. Multiple optimization techniques have been introduced that reduce per token cost. These methods do not eliminate regeneration for redundant queries [3] and increasing hardware or model level solutions doesn't always improve performance as LLM inference in memory-bandwidth bound [4].

System level solutions such as caching are introduced into LLM systems that help solve redundant query problems. User queries are repetitive in nature, a small fraction of those queries are actually unique queries in such cases caching query and it's corresponding response increases response retrieval time.

Most often the recently asked queries have higher chances of getting asked again this demonstrates temporal locality in user prompts [5]. To make cache memory work extremely fast, only a limited number of most important query pair must be stored in the cache for quick retrieval. To avoid overloading cache memory Least Recently Used (LRU) eviction policy can be used to clean the cache by keeping only the most recently used queries and responses [6].

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

RELATED WORKS

Caching mechanisms and eviction policies have been used for diverse applications such as search engines and cache memory systems. Before caching, search engines were slow, overloaded and computed answers each time for redundant queries [7]. LLM inference is slow even for single query, as LLM's tokens are generated sequentially and each LLM's token is dependent on the previous token this makes the process slow and increases latency. To overcome this limitation small, fast model are used to predict the future tokens and larger accurate models are used to verifies these tokens in parallel. It is one of a system level optimizations to reduce latency in LLMs but it still triggers full generation for each query [8].

LLM (like GPT-3 and GPT-4) call API for redundant or slightly modified prompts and each API call triggers full LLM inference which makes it slow and expensive. Cache and Distil introduced the first system level caching optimization technique to reduce latency in LLMs. The research cached repeated prompts and responses for quicker access (Cache) and trained a small local model to imitate LLM behavior for future similar prompts (Distil) [9]. It retrieves the response from cached prompts by comparing its semantic similarity and significantly lowering the cost but false positive reuse is not analyzed deeply. Heavy – Hitter Oracle (H2O) paper observed a very important detail about LLM "A small number of prompts account for a large fraction of total LLM inference cost." To prevent wastage of computation for repeated prompts, it introduced H2O. Heavy Hitter is a frequently accessed item in a stream; in this scenario prompts are the stream and heavy hitter are frequently occurring prompts. It doesn't account for semantic similarity or correctness [10]. SCLAM proposed a caching framework for chat services. It stores previous user queries and its corresponding response using semantic embedding and performed cosine similarity for semantic matching using a fixed threshold. [11].

GPTCache proposed semantic caching framework for LLM application by storing each query in its vector embedding form along with the first LLM response in the cache. It computed cosine similarity to semantically compare the new query with the cache query and only return the response if the similarity score exceeds the threshold. Although it reduced the latency for LLM it performs semantic similarity even for paraphrased queries [12]. MinCache introduced resemblance matching in LLM caching system that follows a hierarchical process. It first performs exact matching, followed by resemblance matching and only if these two layers fail it performs semantic matching. This significantly reduced the need for semantic matching and multiple lookups in cache. However, it has been observed that unigram-based resemblance matching is susceptible to adversarial lexical reordering leading to excessive false positive cache hits [13].

MinCache Architecture and Hierarchical Matching Workflow

The proposed research is built upon MinCache's three-layer, hierarchical caching mechanism with exact matching, resemblance matching and semantic matching used to reduce computational latency in LLMs as shown in Fig. 1.

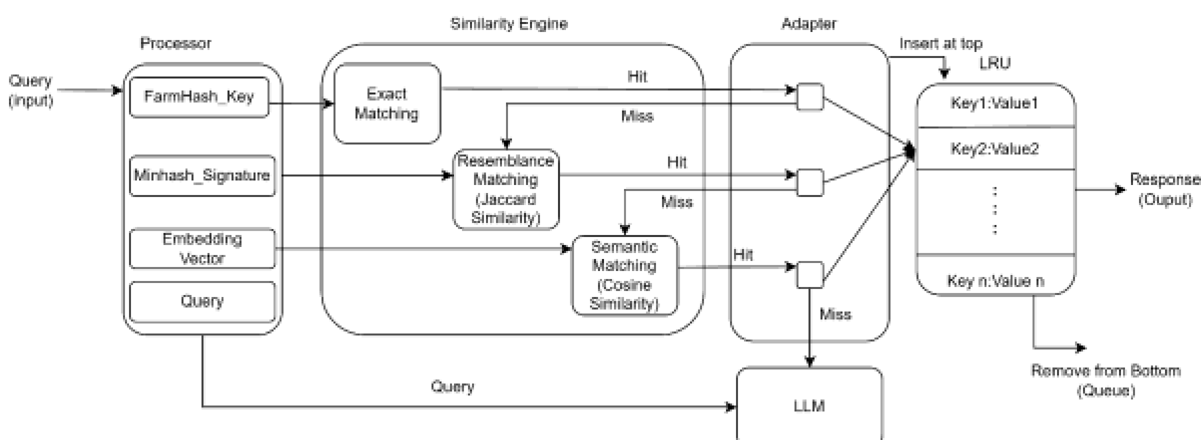

Figure 1. Hierarchical Caching Architecture in LLM Systems.

In fig. 1, the Architecture consists of five modules – Processor module, Similarity Engine module, Adapter module, LLM module and LRU Cache module.

Processor Module receives user queries. It processes the Natural Language format of the query and modifies it into machine understandable format by using Natural Language Processing techniques. It then generates the three keys, each of which will be used for different matching system.

Farmhash key is a fast hash fingerprint used for Exact Matching. Minhash Signature is fingerprint of the prompt where the text is broken into pieces called shingles. It hashes those pieces and keeps only a small fixed size vector of hash values, these hash values are compared to perform Resemblance matching. Embedding Vector is a numeric representation of the meaning of the prompt and it is used for Semantic Matching. Additionally, it also stores the original unprocessed query ready to be sent to the LLM where all the matching fails.

Similarity Engine module performs the main operation of measuring the similarity between the incoming prompt and cached entries. It performs in three hierarchical and progressive manner Exact matching, Resemblance matching and Semantic Matching. If either of the initial similarities is matched, cache is hit and none of other similarities are checked. If none of the similarities matched, cache is missed and LLM is notified to generate the response for the new user prompt.

Exact matching is computed using farmhash keys. The farmhash key generated by the processor module is compared with the farmhash keys retrieved by the adapter from the LRU cache one by one. This matching compares the keys and only the exact replica of this key retrieved by the adapter will be matched.

e.g. “What is AI ?”

Farmhash key: 0xA12F8C9E...

If in the LRU Cache has the exact same farmhash key “0xA12F8C9E...” cache will be hit.

This reduces the need for LLM to generate a response for redundant prompts.

Resemblance matching is computed using minhash signature. It is a unigram signature that computes resemblance based on the similarity between the words of a prompt. The minhash Signature generated by the processor module is compared with the minhash Signatures retrieved by the adapter from LRU cache one by one. This matching compute the Jaccard Similarity which compares how similarly worded, are two sentences.

Jaccard Similarity

Two sets Aand B

$$\text{Jaccard Similarity}\ (\mathrm{A}, \mathrm{B}) = \frac{|\mathrm{A} \cap \mathrm{B}|}{|\mathrm{A} \cup \mathrm{B}|}$$

Where:

$|A \cap B|$= number of elements common to both sets

$|A \cup B|$= total number of unique elements across both sets

Range: 0 (no overlap) to 1 (identical sets)

e.g. Query 1 → "what is artificial intelligence"

Query 2 → "what is ai"

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

Sig(Q1) = [a1, a2, a3, ..., a128]

Sig(Q2) = [b1, b2, b3, ..., b128]

Suppose out of 128 positions, 85 positions are equal

Jaccard ≈ 85 / 128 ≈ 0.66

If the threshold is 0.65 after comparing 0.66 ≥ 0.65

Resemblance is Hit.

This significantly improves cache hits as most of the time similar prompts are just worded differently.

Semantic Matching is computed using vector embeddings. The vector Embeddings generated by the processor module are matched with vector embeddings retrieved by the adapter from LRU cache one by one. This matching compute cosine similarities between two vectors to check its semantic similarity.

Cosine Similarity

Used for vectors (e.g., TF-IDF, embeddings).

If you have two vectors u and v

$$\text{Cosine Similarity}(u, v) = \frac{u \cdot v}{|u| \,.\, |v|}$$

$$= \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \sqrt{\sum_{i=1}^{n} v_i^2}}$$

Where:

u · v= dot product of vectors

∥ u ∥= magnitude (Euclidean norm) of vector u

n= number of dimensions

Range = -1 to 1 (for general vectors), but usually 0 to 1 for embeddings with non-negative values.

e.g. Query 1 → "what is artificial intelligence"

Query 2 → "explain the concept of ai"

Emb(Q1) = [0.12, -0.44, 0.87, ..., 0.03]

Emb(Q2) = [0.10, -0.41, 0.85, ..., 0.05]

cosine_similarity(Emb(Q1), Emb(Q2)) ≈ 0.82

If the threshold is 0.75 after comparing 0.82 ≥ 0.75

Semantic is hit.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

Semantic matching compares the meaning of the sentences. It is extremely useful when words or sentence structures are different and phrases are paraphrased.

Adapter Module acts as the intermediatory module between LRU cache which is the main storage and the rest of the system. It is responsible for all lookups into the cache and storing and retrieving data from cache. Whenever similarity engine requests data to perform similarity comparisons adapter retrieves the data from cache in the order of most recently used data to least recently used, to obtain more cache hits.

LRU cache is the main storage of the entire system. It stores the data in the form of key-value pairs such that the three keys – farmhash key, minhash signature and vector embeddings are hashed to the response which is the value. LRU eviction policy is used such that only the most frequently used data is stored in the cache in the order of their importance which makes lookups more efficient and optimizes storage space. LLM module is used handle the edge case where cache is missed which occurs if the prompt is new or its corresponding response has been evicted due to its infrequent use, in such cases when cache is missed, Adapter notifies LLM, which request the processor module for the query, it generates the response and stores it in LRU Cache.

Limitations of Existing System

MinCache's resemblance matching significantly improved the speed of caching in LLM systems. It computes unigram shingling minhash signature for each prompt that computes similarity between the user query minhash signature and the stored minhash signature by comparing the number of same words in each sentence. Real-world user prompts have high lexical reordering of words which changes the meaning of a sentence.

e.g.: A sample taken from PAWS dataset states that the following example sentence pair is dissimilar although same words have been used in both the sentences [14].

Sentence 1: "In Paris, in October 1560, he secretly met the English ambassador, Nicolas Throckmorton, asking him for a passport to return to England through Scotland."

Sentence 2: "In October 1560, he secretly met with the English ambassador, Nicolas Throckmorton, in Paris, and asked him for a passport to return to Scotland through England."

Hence, to prevent unsafe false positive cache hits, a structural aware resemblance matching enhancement was proposed to strengthen minhash based filtering while preserving lightweight and fast caching mechanism.

Proposed Approach

The proposed approach is used to improve the resemblance caching layer of MinCache to reduce false positive cache hits by introducing structural awareness.

Structural-Aware Resemblance Encoding

Structural awareness is incorporated into resemblance layer while maintaining its computational efficiency by using the following feature extraction strategies.

i. Unigram Shingling (baseline) (US)

In MinCache, each query is tokenized into individual words (1-gram shingles), Minhash signatures is generated from these shingles and Jaccard similarity is computed based on the collision rate between these signatures. However, it doesn't account for the positional or order information hence it produces high similarity scores and false positives.

ii. Bigram Shingling (BS)

Continuous bigrams are introduced for local ordering awareness.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

(w_i, w_{i+1})

Where, w_i is the ith word.

They are used to encode information of adjacent tokens relationships. This makes resemblance matching sensitive to adjacent word's order and placement and reduces false positives.

iii. Skip Gram Shingling (SS)

Skip gram shingling and is used to encode the information of non – adjacent tokens that are semantically linked. It is used to capture the order and placement information over a long range of words.

iv. Positional Penalty (PPS)

Even after using Bigrams and Skip grams sometimes two sentences may share many shingles or have a different global order to prevent this a similarity correction mechanism is used. It measures how many words have shifted from their original position and adds a penalty based on it.

Table 1. Structural resemblance model variants

Variants	Feature Extraction Policies
A0	US
A1	US + BS
A2	US + BS + SS
A3	US + BS + SS + PPS

In Table 1. Structural resemblance model are describes, A0 indicated the usage of unigram shingling and will be considered as baseline for further evaluations.

A1 indicates the combined usage of unigram and bigram shingling. A2 indicates the combined usage of unigram, bigram and skip gram shingling and finally A3 indicates the combined usage of unigram, bigram and skip gram shingling along with positional penalty.

Experimental Evaluation

Datasets

The experimental evaluation was tested against two open-source datasets –

PAWS - Paraphrase Adversaries from Word Scrambling

PAWS dataset contains sentence pairs that often use the same words but in different order sharing a very high lexical overlap. It is used to expose the weakness in unigram Jaccard similarity methods, Bag – of – words and surface level similarity.

QQP – Quora Question Pairs

QQP [15] is open-source dataset that has a collection of real – world question pairs collected from the Quora website. It is used in experimentation to understand how well the system generalizes to real – world scenario.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

Metrics

Metrics are selected in such a way that it is easy to decide how effectively the pipeline's decision-making system is working and simultaneously reducing latency. The results are reported using the following metrics – Recall, False Positive Rate, Precision, F1-Score, Balanced Accuracy and Throughput.

- TP = True Positives
- TN = True Negatives
- FP = False Positives
- FN = False Negatives

Recall: Measure how many actual similar pairs were correctly identifies

Recall = $\frac{TP}{TP+FN}$

False Positive Rate: Measures how many dissimilar pairs were incorrectly predicted as similar.

False Positive Rate (FPR) = $\frac{FP}{FP+TN}$

Precision: Measures how many predicted similar pairs are actually correct.

Precision = $\frac{TP}{TP+FP}$

F1-Score: Harmonic mean of Precision and Recall.

F1-Score = $\frac{2TP}{2TP + FP + FN}$

Balanced Accuracy: Average of True Positive Rate and True Negative Rate.

TNR = $\frac{TN}{TN+FP}$

Balanced Accuracy = $\frac{Recall+TNR}{2}$

Throughput: Number of samples processed per second.

Throughput $= \frac{N}{t}$

Where N is total number of samples processed and t is total execution time.

Isolated Resemblance Evaluation on PAWS Dataset

PAWS dataset is evaluated by isolating the resemblance layer from the entire pipeline to check its performance on various metrics.

Table 2. Isolated Resemblance layer performance on PAWS (2,000 samples)

Model	Recall	FPR	Precision	F1	Balanced Acc	Throughput
Baseline (1-g)	0.9875	0.9660	0.4470	0.6154	0.5108	532.65/s
Improved (A3)	0.4666	0.1415	0.7228	0.5671	0.6626	411.02/s

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

In Table 2. The results of this evaluation have shown that the baseline is extremely vulnerable to lexical reordering and hence produced a false positive rate (FPR) of 0.9660 whereas Improved method has reduced it to 0.1415 corresponding to 6.8x reduction in adversarial false positives. Simultaneously, Balanced accuracy has improved from 0.5108 to 0.6626 resulting in 29% relative gain while maintaining 77% of the baseline's throughput.

Ablation Study

An Ablation Study on 5,000 samples of PAWS dataset was conducted to understand the individual effect of the four feature extraction methods on the system performance.

Table 3. Structural ablation (PAWS 5K)

Model	**Recall**	**FPR**	**Balanced Accuracy**
(A0) (Baseline)	0.987	0.961	0.513
(A1)	0.515	0.246	0.635
(A2)	0.369	0.092	0.639
(A3)	0.123	0.018	0.552

In Table 3. Ablation study has shown that the Bigram shingling (A1) has drastically reduced false positive rates. Bigram + Skip gram shingling (A2) further improved the robustness whereas the addition of Positional Penalty (A3) along with (A2) has over penalized valid matches, reducing balanced accuracy.

Based on these findings, the final model adopts Unigram + Bigram + Skip gram (A2) feature extraction methods for structural awareness.

Isolated Resemblance Evaluation on QQP Dataset

QQP dataset is used to evaluate the resemblance layer with structural awareness's ability to generalize with real – world queries.

Table 4. Resemblance-only performance on QQP (20,000 samples)

Model	**Recall**	**FPR**	**Precision**	**F1**	**Balanced Accuracy**	**Throughput**
Baseline (1-g)	0.1284	0.0769	0.4971	0.2041	0.5258	555.59/s
Improved (A2)	0.0686	0.0587	0.4087	0.1175	0.5049	485.43/s

In Table 4. QQP contains natural paraphrasing data and the improved model is more conservative, reducing false positive rate from 0.0769 to 0.0587 but slightly reduces recall from 0.1284 to 0.0686. This proves that isolated resemblance alone is not sufficient for semantic paraphrase detection and validation.

RESULTS

Full pipeline evaluation on PAWS and QQP

To evaluate the impact of proposed structural awareness (A2) improved model against the baseline MinCache model. We have built the full hierarchical progressive matching including exact matching, resemblance matching

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

and semantic matching and evaluated it against 5.000 samples of PAWS and 5,000 samples of QQP to understand the performance of both full models and compare it in under lexical reordering conditions as well as real – world scenarios.

Table 5. Full pipeline performance (PAWS 5K)

Metric	**Baseline**	**Improved (A2)**
Recall	0.9996	0.9982
FPR	0.9957	0.9920
Balanced Acc	0.5019	0.5031

Table 6. Layer Usage (PAWS Full Pipeline)

Layer	**Baseline**	**Improved (A2)**
Resemblance	84.58%	23.9%
Semantic	15.08%	75.5%

In Table 5 and Table 6. The improved resemblance layer successfully rejects the adversarial reordering as the resemblance cache hits drop from 84.58% to 23.9%. However, the semantic model reclassifies many adversarial pairs as similar due to its high lexical overlap. These results reveal a limitation that semantic models do not strongly encode structural reordering.

Table 7. Full pipeline performance (QQP 5K)

Metric	**Baseline**	**Improved (A2)**
Recall	0.7334	0.7318
FPR	0.2087	0.2037
Precision	0.6713	0.6762
F1	0.7010	0.7029
Balanced Acc	0.7623	0.7641
Throughput	15.05/s	12.74/s

Table 8. Layer Usage (QQP Full Pipeline)

Layer	**Baseline**	**Improved (A2)**
Resemblance	7.32%	5.48%
Semantic	32.84%	34.3%
Miss	~60%	~60%

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

In Table 7. and Table 8. The models are evaluated on QQP dataset to simulate a real world scenario with natural paraphrased queries and the results reveal that the improved model (A2) preserves recall (0.7334 to 0.7318) while slightly reducing false positives from (0.2087 to 0.2037). Whereas the precision increases from 0.6713 to 0.6767 resulting in slight improvement in Balanced accuracy and throughput. These results indicate that the structural-aware resemblance layer does not degrade semantic matching performance under natural paraphrase conditions.

The improved model is more conservative at the resemblance stage, reducing resemblance-based cache hits from 7.32% to 5.48%. Consequently, a slightly higher proportion of queries are passed to semantic layer.

Table 9. Isolated Resemblance performance (PAWS 20,000 Samples)

Model	Recall	FPR	Precision	F1	Balanced Acc	Throughput
Baseline (1-g)	0.9857	0.9624	0.4460	0.6141	0.5116	518.20/s
Improved (A2)	0.4519	0.1302	0.7318	0.5588	0.6609	411.86/s

While the evaluation of full pipeline against PAWS and QQP only showed marginal improvements. In Table 9. Isolated Resemblance evaluated against PAWS reveals the true impact of structural awareness (A_2). The false positive rate has decreased 7.39x and improved precision 1.64x. Balanced accuracy increased 29% showing improved classification while maintain 79.5% of original throughput.

CONCLUSION AND FUTURE SCOPE

The proposed research has demonstrated that the improved model (A2) has shown 7.39x reduction in false positive rates improving adversarial robustness compared to MinCache, while retaining 79.5% of resemblance layer throughput and preserving overall pipeline performance on natural paraphrase data. While structural aware shingling has significantly reduced incorrect cache hits in resemblance layer the overall throughput is relatively unchanged on natural paraphrases as sentence-embedding- based semantic models remain highly sensitive to lexical overlap under reordered structure.

To overcome this limitation future works may explore structural aware semantic models, adaptive threshold learning and domain specific latency evaluation.

REFERENCES

1. Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, et al. (2020). Language models are few-shot learners. Adv. Neural Inf. Process. Syst. 33: 1877–1901.
2. Bommasani R, Hudson DA, Adeli E, Altman R, Arora S, von Arx S, et al. (2021). On the opportunities and risks of foundation models. arXiv preprint arXiv:2108.07258.
3. Zhou Z, Ning X, Hong K, Fu T, Xu J, Li S, et al. (2024). A survey on efficient inference for large language models. arXiv preprint arXiv:2404.14294.
4. Yuan Z, Shang Y, Zhou Y, Dong Z, Zhou Z, Xue C, et al. (2024). LLM inference unveiled: Survey and roofline model insights. arXiv preprint arXiv:2402.16363.
5. Xie Y, O'Hallaron D (2001). Locality in search engine queries and its implications for caching. CMU Technical Report CMU-CS-01-128.
6. Mookerjee VS, Tan Y (2002). Analysis of a least recently used cache management policy for web browsers. Oper. Res. 50(2): 345–357.
7. Markatos EP (2001). On caching search engine query results. Comput. Commun. 24(2): 137–143.

8. Miao X, Oliaro G, Zhang Z, Cheng X, Wang Z, Zhang Z, et al. (2024). SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. Proc. ACM ASPLOS: 932–949.
9. Ramírez G, Lindemann M, Birch A, Titov I (2024). Cache & distil: Optimising API calls to large language models. Findings ACL: 11838–11853.
10. Zhang Z, Sheng Y, Zhou T, Chen T, Zheng L, Cai R, et al. (2023). H2O: Heavy-hitter oracle for efficient generative inference of large language models. Adv. Neural Inf. Process. Syst. 36: 34661–34710.
11. Li J, Xu C, Wang F, von Riedemann IM, Zhang C, Liu J (2024). SCALM: Towards semantic caching for automated chat services with large language models. Proc. IEEE/ACM IWQoS: 1–10.
12. Bang F (2023). GPTCache: An open-source semantic cache for LLM applications enabling faster answers and cost savings. Proc. NLP-OSS Workshop: 212–218.
13. Haqiq K, Jahan MV, Farimani SA, Masoom SMF (2025). MinCache: A hybrid cache system for efficient chatbots with hierarchical embedding matching and LLM. Future Gener. Comput. Syst. 170: 107822.
14. Wu H, Pan J, Yang R, Zhang H, Shi G, Jiang Z, Liu Q (2025). PAWS: Passive concept drift adaptation based on instance weighting and subspace alignment in data stream. Proc. Int. Conf. Intelligent Computing: 236–247.
15. Quora Question Pairs Dataset (2017). Kaggle.