BEYOND QUERY SIMILARITY: SAFETY-CONDITIONED CACHE ROUTING FOR LLM RESPONSE REUSE

Anonymous authors
Paper under double-blind review

ABSTRACT

Semantic response caching reuses a cached response for a semantically similar query to improve LLM inference efficiency. However, query similarity alone can lead to unsafe reuse when safety conditions change. Existing approaches improve cache safety through stricter similarity checks, context-aware validation, or response verification, yet they may still miss safety conflicts between the cached response and the new request or introduce unnecessary verification overhead. We argue that safe and efficient semantic caching requires a reuse policy that explicitly represents the conditions required by a cached response and checks whether they remain satisfied by the new request. In this paper, we propose SafeCache, a safety-aware framework that separates candidate retrieval from safety approval and formulates semantic response reuse as a safety-conditioned routing problem. SafeCache represents cached-response reuse requirements and new-request safety facts as two safety condition sets and aggregates condition-level matches into an overall compatibility score. To prevent decisive conflicts from being diluted by aggregate matching, SafeCache checks explicit contradictions separately. SafeCache further estimates matching uncertainty and combines these safety signals with query similarity to make the final reuse decision under safety and verifier-budget constraints. Experiments show that SafeCache achieves a better safety-utility trade-off among the evaluated baselines, enabling safer response reuse at comparable cache utility.

1 INTRODUCTION

LLM services often receive repeated or semantically similar requests. Semantic caching reduces the inference time on repeated or similar requests by storing query-response pairs and reusing a cached response for a semantically similar new request (Zilliz, 2023; Regmi & Pun, 2024; Gill et al., 2025; Yan et al., 2025), which can reduce serving latency and inference cost (Li et al., 2024; Liu et al., 2026). However, similarity between queries does not guarantee that the cached response remains safe for the new request. Changes in safety-relevant conditions, such as the protected subject, authorization state, supporting evidence, or tool context, may invalidate the cached response. Recent studies further show that semantic caches can be exploited to return unsafe cached responses (Zhang et al., 2026; Wu et al., 2026b;a). Therefore, semantic cache safety requires determining whether a cached response can be safely reused for each new request.

Existing work addresses semantic cache safety through several approaches. The first group tries to make reuse safer through stricter similarity matching (Zilliz, 2023; Regmi & Pun, 2024; Gill et al., 2025; Schroeder et al., 2025). These methods narrow the cache hit set by requiring the new query to be very close to a cached query. This can filter out obvious mismatches, but it still only compares the two queries and does not verify whether the cached response is actually safe to reuse, even when the queries appear semantically similar (Zhang et al., 2026; Wu et al., 2026b). The second group uses context-aware validation, adding external context, metadata, or retrieved support evidence to the reuse decision (Yan et al., 2025; Shah, 2026). These checks make the cache safer when a decisive change appears in the external context. However, the external context may remain consistent even when the cached response is internally unsafe for the new request (Yan et al., 2025; Shah, 2026). As a result, unsafe cached responses may still be reused.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To examine whether a cached response remains safe for a semantically similar new request, verified semantic caches add a verifier before returning a candidate response (Schroeder et al., 2025; Singh et al., 2026; Shah, 2026). A verifier can check response safety conditions, including the protected subject, authorization state, evidence support, and tool state, rather than relying only on query similarity or limited external context. However, existing verified caches often treat verification as a generic approval step after retrieval. They do not clearly specify what safety conditions in the cached response should be checked, what types of safety conflicts should require response regeneration, or which uncertain candidates are worth the extra cost of a verifier. Consequently, they may reuse unsafe responses, undermining safety. They may also waste verifier calls on clear cases or discard responses that could be safely reused, reducing utility. These limitations motivate a safety-conditioned reuse policy that explicitly represents response safety conditions and adapts the reuse decision to better balance safety and utility in semantic caching.

To better balance safety and utility, we argue that a safety-conditioned semantic cache needs a routing policy that makes the cached response's safety conditions explicit. Similarity should only retrieve a candidate, while safety approval should depend on whether these conditions remain satisfied for the new request, allowing efficient reuse without undermining safety.

In this paper, we propose SafeCache, a safety-aware framework for semantic response caching that makes reuse decisions based on safety conditions, aiming to improve safe reuse while preserving caching utility. Specifically, SafeCache first uses embedding similarity only to retrieve a plausible candidate, separating candidate retrieval from the final reuse decision. A safety-conditioned router then decides whether the cached response should be reused, verified, or regenerated. To capture safety changes that ordinary query similarity cannot detect, SafeCache extracts the safety conditions required by the cached response and matches them against those in the new request. This condition-level matching yields an aggregated risk score that measures the overall mismatch between the new request and the safety conditions required for reuse. To ensure that explicit safety conflicts are not overlooked, SafeCache checks them separately from the aggregated risk score and prevents reuse when such a conflict is detected. Because verification adds extra cost, SafeCache further uses an uncertainty-aware safety matching strategy to identify borderline candidates and sends them to a verifier only when they appear reusable but are not reliable enough for direct reuse. Extensive experiments show that SafeCache achieves a better safety-utility trade-off among the semantic cache baselines, with stronger control of unsafe response reuse under comparable cache utility.

We make three contributions: (1) We present SafeCache, a safety-aware framework for semantic response caching that separates candidate retrieval from the final reuse decision and explicitly targets the safety-utility trade-off in cache reuse. (2) We develop a guarded and uncertainty-aware safety routing algorithm that checks whether the cached response's safety conditions still hold, blocks direct reuse under explicit contradictions, and uses calibrated uncertainty to decide when to reuse, verify, or regenerate. (3) We introduce a semantic cache safety evaluation and show that SafeCache achieves the strongest safety-utility trade-off among the evaluated semantic cache baselines.

2 Related Work

LLM caching broadly includes prefix/KV caching and semantic response caching. Prefix/KV methods reuse attention states for repeated token sequences or shared prompt prefixes (Gim et al., 2024; Zheng et al., 2024; Liu et al., 2025; Kwon et al., 2023), whereas semantic caches reuse complete responses for embedding-similar requests (Zilliz, 2023; Regmi & Pun, 2024; Yan et al., 2025). However, LLM caching also introduces (1) reuse-validity: semantically similar requests may differ in dialogue context, retrieved evidence, or other execution states (Yan et al., 2025; Shah, 2026; Schroeder et al., 2025), and (2) adversarial-integrity risks: semantic cache keys can be exploited through collisions or poisoning (Zhang et al., 2026; Wu et al., 2026a).

Existing safeguards mainly rely on similarity-threshold calibration (Schroeder et al., 2025; Baral et al., 2026), context/evidence validation (Yan et al., 2025; Shah, 2026), or verifier-based filtering (Wu et al., 2026b;a). Adversarial studies further expose collision and poisoning risks in semantic caches (Zhang et al., 2026; Wu et al., 2026a). In contrast, SafeCache explicitly models response safety conditions and routes candidates with a calibrated policy to balance safety and efficiency. A broader review is provided in App. A.

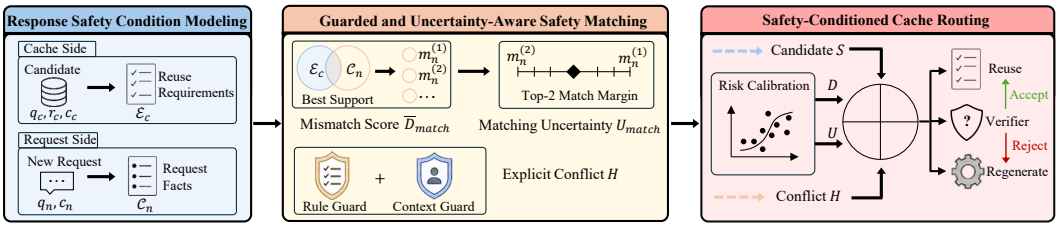

Figure 1: Overview of SafeCache. From left to right, SafeCache first extracts the reuse requirements of the cached response and the safety facts of the new request. It then matches the two condition sets to compute the condition mismatch and matching uncertainty, while rule- and context-level guards preserve explicit conflicts. Finally, risk calibration produces the routing risk and uncertainty, which are combined with query similarity and the conflict indicator to choose direct reuse, verification, or regeneration. A verified response is reused if accepted and regenerated otherwise.

3 METHODOLOGY

3.1 OVERVIEW

Our proposed SafeCache (Fig. 1) consists of three parts. The first is Response Safety Modeling, which extracts separate safety condition sets for the cached response and the new request. The second is Guarded and Uncertainty-Aware Safety Matching, which uses condition-level semantic matching to measure the degree to which the cached response conditions remain supported by the new request, while separately checking for explicit contradictions and estimating the reliability of the matching result. The third is Safety-Conditioned Cache Routing, which combines retrieval similarity with calibrated safety evidence to make routing decisions under safety and verifier-call constraints.

Overall, SafeCache adopts a three-stage cache-routing framework. It first constructs separate safety condition sets for the cached response and the new request, and then evaluates their safety compatibility to derive mismatch risk and matching uncertainty. Finally, a calibrated router combines safety evaluation with query similarity to make cache reuse decisions under safety and utility constraints.

3.2 RESPONSE SAFETY CONDITION MODELING

To make response safety explicit before cache reuse, SafeCache constructs separate safety condition sets for the cached response and the new request. The motivation is that query similarity only indicates that two requests have similar context, whereas safe reuse also depends on the protected subject, authorization state, supporting evidence, tool state, and so on. Because these conditions may change even when the requests remain semantically similar, SafeCache models them explicitly before making the reuse decision.

SafeCache takes a cached entry (q_c, r_c, c_c), where q_c is the cached request, r_c is the cached response, and c_c is the associated context, together with a new request-context pair (q_n, c_n), where q_n is the new request and c_n is its context. The cache-side extractor produces the cached-response condition set $\mathcal{E}_c = \mathrm{Extract}_{\mathrm{cache}}(q_c, r_c, c_c)$, which records the conditions required to reuse r_c safely. The request-side extractor produces the new-request condition set $\mathcal{C}_n = \mathrm{Extract}_{\mathrm{request}}(q_n, c_n)$, which records the safety-relevant facts and states in q_n and c_n. The condition representation, extraction rules, and edge-case handling are detailed in App. C. Moreover, we also evaluate robustness to condition-extraction errors in App. D. By separating cached response requirements and new-request facts, this modeling enables detection of unsupported conditions and safety conflicts. The next module compares $\mathcal{E}_c$ and $\mathcal{C}_n$ to determine whether the cached response remains safe to reuse.

3.3 GUARDED AND UNCERTAINTY-AWARE SAFETY MATCHING

To assess whether the safety conditions for the cached response remain supported by the new request, SafeCache combines aggregate safety condition matching with explicit contradiction checks and matching uncertainty estimation. The motivation is that safe reuse depends on the joint support

of multiple cached-response conditions, which SafeCache summarizes with an aggregate mismatch score. Because the aggregate score may hide explicit conflicts or be unreliable under ambiguous matches, SafeCache adds explicit contradiction checks and a separate uncertainty estimate.

3.3.1 AGGREGATE CONDITION SUPPORT MATCHING

To obtain an overall measure of safety compatibility, SafeCache matches each cached-response condition against the new-request condition set and aggregates the resulting per-condition mismatches. SafeCache examines each condition in $\mathcal{E}_c$ one at a time, treating it as a safety requirement, and searches $\mathcal{C}_n$ for the condition that provides the strongest support. The resulting per-condition mismatches are then averaged into a continuous overall mismatch score.

Let M and N denote the numbers of conditions in the cached-response set $\mathcal{E}_c$ and the new-request set $\mathcal{C}_n$, respectively. For each cached-response condition e_i, SafeCache compares it with every new-request condition c_j using the similarity calculation sim_ϕ, where ϕ is the frozen condition representation model. SafeCache retains the highest pairwise score $m_i^{(1)}$ as the strongest support in $\mathcal{C}_n$ for e_i. The per-condition mismatch is then defined as $1 - m_i^{(1)}$, so weaker support yields a larger mismatch. A cached-response condition with no support in $\mathcal{C}_n$ is treated as unmet and blocks direct reuse. SafeCache then averages the per-condition mismatches to keep the aggregate on the same scale across different numbers of cached-response conditions. This gives the raw condition mismatch $\overline{D}_{\mathrm{match}}$ as

$$m_i^{(1)} = \max_{1 \le j \le N} \mathrm{sim}_\phi(e_i, c_j), \quad \overline{D}_{\mathrm{match}} = \frac{1}{M} \sum_{i=1}^{M} \left(1 - m_i^{(1)}\right). \tag{1}$$

Because each pairwise similarity lies in $[0, 1]$, the raw condition mismatch also lies in $[0, 1]$ and increases as more cached-response conditions receive weak or no support. By aggregating condition-level mismatch scores, this score provides a normalized and continuous measure of overall safety compatibility, capturing different degrees of support.

3.3.2 EXPLICIT CONTRADICTION CHECKS

To prevent decisive safety conflicts from being diluted by aggregate matching, SafeCache detects explicit contradictions as an independent signal against direct reuse. It checks both extracted condition records and original inputs, in which the former captures structured conflicts, while the latter covers conflicts missed during extraction. The two checks are combined into an independent conflict indicator that blocks direct reuse whenever an explicit contradiction is detected.

At the condition level, SafeCache applies conflict rules $\mathcal{R}_{\mathrm{rule}}$ (App. C) to the condition sets of the cached response and the new request (i.e., $\mathcal{E}_c$ and $\mathcal{C}_n$). Each rule $r \in \mathcal{R}_{\mathrm{rule}}$ first determines whether two conditions concern the same safety decision and then checks whether their values, states, or required response actions conflict. The resulting indicator $H_{\mathrm{rule}}(x) = 1$ whenever at least one condition-level rule detects an explicit contradiction: $H_{\mathrm{rule}}(x) = \bigvee_{r \in \mathcal{R}_{\mathrm{rule}}} r(\mathcal{E}_c, \mathcal{C}_n)$, where $x = (q_c, r_c, c_c, q_n, c_n)$ denotes the cache candidate and each rule $r(\cdot)$ returns a binary value.

At the input level, SafeCache applies conflict rules $\mathcal{R}_{\mathrm{ctx}}$ directly to the cached and new request-context pairs (q_c, c_c) and (q_n, c_n). These rules compare corresponding safety-relevant attributes, such as authorization state, tool outcome, and requested time, to detect explicit conflicts that may not have been preserved during condition extraction. The resulting indicator $H_{\mathrm{ctx}}(x) = 1$ whenever any input-level rule detects such a conflict: $H_{\mathrm{ctx}}(x) = \bigvee_{r \in \mathcal{R}_{\mathrm{ctx}}} r(q_c, c_c, q_n, c_n)$.

As a result, SafeCache retains H_{rule} and H_{ctx} separately to record the level at which a conflict is detected, and combines them into the final conflict indicator,

$$H(x) = H_{\mathrm{rule}}(x) \vee H_{\mathrm{ctx}}(x). \tag{2}$$

Thus, $H(x)$=1 whenever either level detects an explicit contradiction. Unlike the aggregate condition mismatch, H serves as an independent guard that cannot be diluted by averaging in Eq. 1 and directly blocks reuse.

3.3.3 UNCERTAINTY-AWARE RISK ESTIMATION

To assess the reliability of condition matching, SafeCache estimates matching uncertainty U_{match} by measuring how clearly each cached-response condition is distinguished from alternative matches. The raw mismatch in Eq. 1 uses the strongest support score $m_i^{(1)}$ but does not reflect whether other candidates receive similar scores. When several candidates have similar scores, the selected match is less stable.

For each cached-response condition e_i, SafeCache retains the strongest score $m_i^{(1)}$ and the second-highest score $m_i^{(2)}$. Their difference $m_i^{(1)} - m_i^{(2)}$ measures the confidence of the best correspondence (Joshi et al., 2009). A large margin therefore indicates both strong support and a clearly distinguished match, while the margin is small when the best score is low or when the two highest scores are close. SafeCache averages these margins across the cached-response conditions and converts the result into the matching uncertainty:

$$U_{\text{match}}(x) = 1 - \frac{1}{M}\sum_{i=1}^{M}(m_i^{(1)} - m_i^{(2)}). \tag{3}$$

Because $0 \le m_i^{(2)} \le m_i^{(1)} \le 1$, the matching uncertainty lies in $[0, 1]$ and increases as the best matches $m_i^{(1)}$ become weaker or less distinct.

This matching module summarizes each candidate using three complementary signals: $\overline{D}_{\text{match}}(x)$ measures overall condition mismatch, $U_{\text{match}}(x)$ reflects the reliability of the matching result, and $H(x)$ identifies explicit safety conflicts. The following routing module then calibrates and combines these signals for the final cache decision.

3.4 SAFETY-CONDITIONED CACHE ROUTING

To convert a retrieved candidate into a cache decision, SafeCache combines query similarity with response-level safety evidence to choose direct reuse, verification, or regeneration. Query similarity indicates whether the cached request remains relevant, but does not determine whether the response is safe to reuse or whether verification is necessary. SafeCache therefore calibrates the matching risk and routing parameters on a separate calibration set and fixes them for testing. Clear candidates are reused directly, selected borderline candidates are verified, and the rest are regenerated.

For the cache candidate $x = (q_c, r_c, c_c, q_n, c_n)$, the retriever computes the query similarity $S(x) = \text{sim}_\psi(q_c, q_n)$, where ψ is the query encoder. The router uses S only to measure whether the cached request remains relevant to the new request. The preceding matching module provides the raw condition mismatch $\overline{D}_{\text{match}}$ in Eq. 1, explicit conflict indicator H in Eq. 2, and matching uncertainty U_{match} in Eq. 3. Before routing, SafeCache converts the raw matching outputs into a calibrated risk D and uncertainty U. The router then uses S, D, U, and H to make the final cache decision.

3.4.1 RISK CALIBRATION

The raw condition mismatch $\overline{D}_{\text{match}}$ in Eq. 1 measures how poorly the new request supports the cached-response conditions, but it does not directly represent unsafe-reuse risk (Guo et al., 2017). SafeCache therefore learns a monotonic calibration function $\mathcal{K}$ from a labeled calibration set to map mismatch scores to observed unsafe-reuse rates, ensuring that a larger mismatch cannot produce a lower calibrated risk.

Because each calibrated risk is estimated from a limited number of examples, SafeCache also measures its uncertainty. It uses the width of the corresponding Wilson interval (Wilson, 1927) as $U_{\text{cal}}(x) \in [0, 1]$, in which a narrower interval indicates a more reliable estimate, while a wider interval indicates greater uncertainty. The monotone risk-map construction and routing-policy calibration are detailed in App. E.

SafeCache then uses $U_{\text{cal}}(x)$ to balance the calibrated risk and the raw mismatch. When calibration uncertainty is low, the final routing risk relies more on $\mathcal{K}(\overline{D}_{\text{match}}(x))$. As uncertainty increases, it retains more of $\overline{D}_{\text{match}}(x)$. Accordingly, the final routing risk score is defined as $D(x) = \big(1 - U_{\text{cal}}(x)\big)\mathcal{K}\big(\overline{D}_{\text{match}}(x)\big) + U_{\text{cal}}(x)\overline{D}_{\text{match}}(x)$.

The final uncertainty $U(x)$ accounts for uncertainty in both condition matching and risk calibration. We define $U(x) = \max\{U_{\text{match}}(x), U_{\text{cal}}(x)\}$, to preserve high uncertainty in either step. The resulting $D(x)$ and $U(x)$ are used as the risk and uncertainty signals for routing.

3.4.2 BUDGET-AWARE CACHE DECISION

SafeCache uses a calibrated decision policy Θ to assign each candidate x to exactly one of three actions: direct reuse, selective verification, or regeneration. The decision is based on query similarity $S(x)$, reuse risk $D(x)$, uncertainty $U(x)$, and explicit safety conflicts $H(x)$. The policy first defines the direct-reuse and verification regions, while candidates outside these two regions are regenerated. Because verification incurs additional cost, SafeCache further constrains verification with a verifier-call budget, yielding the final budget-aware cache decision.

Direct reuse. Let τ_r denote the minimum query similarity for direct reuse, and let δ_r and υ_r denote the maximum allowed risk and uncertainty, respectively. A candidate is directly reused only if the cached request remains sufficiently relevant, $S(x) \geq \tau_r$; its calibrated risk and uncertainty are sufficiently low, $D(x) \leq \delta_r$ and $U(x) \leq \upsilon_r$; and no explicit contradiction is detected, $H(x) = 0$. These conditions define the direct reuse region

$$\mathcal{R}_\Theta = \{x \mid H(x) = 0, S(x) \geq \tau_r, D(x) \leq \delta_r, U(x) \leq \upsilon_r\}. \tag{4}$$

Candidates in $\mathcal{R}_\Theta$ return the cached response directly.

Selective verification. Selective verification handles candidates that are not reliable enough for direct reuse but remain sufficiently relevant to justify an additional verifier check. Let τ_v denote the minimum similarity for verification, and δ_v and υ_v the maximum allowed risk and uncertainty. Compared with direct reuse, verification uses a lower similarity threshold ($\tau_v \leq \tau_r$) and higher limits on risk ($\delta_v \geq \delta_r$) and uncertainty ($\upsilon_v \geq \upsilon_r$), admitting less reliable but still relevant candidates.

An explicit conflict $H(x) = 1$ blocks direct reuse but may still be resolved by verification when the cached and new requests are sufficiently related. App. K shows that H provides complementary safety information and that verification recovers valid conflict candidates. SafeCache therefore applies a stricter similarity threshold $\tau_v^c \geq \tau_v$ to conflict candidates. The verification threshold for each case is defined as

$$\widetilde{\tau}_v(x) = \begin{cases} \tau_v^c, & \text{if } H(x) = 1, \\ \tau_v, & \text{if } H(x) = 0. \end{cases} \tag{5}$$

The condition $S(x) \geq \widetilde{\tau}_v(x)$ provides the similarity requirement for verification.

The router then combines the similarity requirement with the verification limits of risk δ_v and uncertainty υ_v, requiring the candidate risk and uncertainty to remain within the admitted range:

$$\mathcal{V}_\Theta = \{x \mid x \notin \mathcal{R}_\Theta, S(x) \geq \widetilde{\tau}_v(x), D(x) \leq \delta_v, U(x) \leq \upsilon_v\}. \tag{6}$$

Thus, $\mathcal{V}_\Theta$ contains candidates that are not eligible for direct reuse but remain suitable for an additional verifier check.

Regeneration. Candidates outside both the direct-reuse and verification regions are regenerated.

To limit the additional cost of verification, SafeCache applies a *budget-aware final routing* strategy that constrains verifier calls with a budget B and falls back to regeneration when the budget is exhausted. In a routing window containing N_W candidates, at most $\lfloor BN_W \rfloor$ verifier calls are allowed. Let $b_W(x)$=1 indicate that verifier budget remains when candidate x is processed. App. L shows that the resulting policy is stable across different request orders. The complete routing decision is

$$a_\Theta(x) = \begin{cases} \text{reuse}, & x \in \mathcal{R}_\Theta, \\ \text{verify}, & x \notin \mathcal{R}_\Theta, x \in \mathcal{V}_\Theta, b_W(x) = 1, \\ \text{regenerate}, & \text{otherwise}. \end{cases} \tag{7}$$

When verification is selected, the verifier checks whether the cached response r_c remains safe for the new request q_n and context c_n. App. F specifies the verifier inputs, deterministic decision protocol, and conservative acceptance/failure rules. Accepted responses are reused, while rejected responses, candidates outside both routing regions, and verification candidates after budget exhaustion are regenerated. These regions define the three-way decision for a given routing parameter Θ, which is selected on the calibration set under safety and verifier-call constraints in the following.

3.4.3 RISK-CONSTRAINED ROUTING CALIBRATION

To determine the routing policy Θ, SafeCache evaluates a fixed set of candidates on a calibration set of N samples. It aims to maximize safe cache reuse while limiting unsafe reuse and verifier calls. Rather than using manually weighted objectives, SafeCache determines Θ under two explicit safety and efficiency constraints, in which the accepted-unsafe rate must not exceed α, and the verifier-call rate must not exceed B. Policies violating either constraint are discarded.

Let $N_{\text{safe}}(\Theta)$ and $N_{\text{unsafe}}(\Theta)$ denote the numbers of safe and unsafe cached responses returned either directly or after verification. Their sum $N_{\text{acc}}(\Theta) = N_{\text{safe}}(\Theta) + N_{\text{unsafe}}(\Theta)$ is the total number of cached responses returned. Let $N_{\text{verify}}(\Theta)$ denote the number of verifier calls. The accepted-unsafe rate $\widehat{\rho}$ measures the fraction of returned cached responses that are unsafe, while the verifier rate $\widehat{b}_v$ measures the fraction of calibration candidates sent to the verifier: $\widehat{\rho}(\Theta) = \frac{N_{\text{unsafe}}(\Theta)}{\max\{N_{\text{acc}}(\Theta),1\}}, \widehat{b}_v(\Theta) = \frac{N_{\text{verify}}(\Theta)}{N}$.

A candidate policy is feasible only when its accepted-unsafe rate satisfies the empirical calibration target α and its verifier rate remains within the budget B. Therefore, the feasible policy set is

$$\mathcal{F} = \left\{ \Theta \;\middle|\; \widehat{\rho}(\Theta) \le \alpha, \widehat{b}_v(\Theta) \le B \right\}. \tag{8}$$

Among the feasible policies in $\mathcal{F}$, SafeCache first selects those that return the largest number of safe cached responses. If multiple policies achieve the same result, it next compares the number of unsafe cached responses and then the number of verifier calls. This ordered selection rule is represented by

$$\mathbf{o}(\Theta) = \Big(N_{\text{safe}}(\Theta), -N_{\text{unsafe}}(\Theta), -N_{\text{verify}}(\Theta) \Big), \Theta^\star = \arg\operatorname*{lex\,max}_{\Theta \in \mathcal{F}} \mathbf{o}(\Theta). \tag{9}$$

The candidate policy set and all selection rules are fixed before evaluation and reported in Sec. 4.1.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Datasets. We evaluate test pairs covering dialogue/tool-state, temporal-freshness, refusal-validity, authorization, and semantic-collision changes, constructed from Schema-Guided Dialogue (Rastogi et al., 2020), τ-bench (Yao et al., 2024), SituatedQA (Zhang & Choi, 2021), and XSTest (Röttger et al., 2024). Construction, labeling, and splits are detailed in App. B. We additionally use CacheAttack (Zhang et al., 2026) to stress-test adversarial cache reuse.

Baselines. We compare SafeCache with `GPTCache` (Zilliz, 2023), `vCache` (Schroeder et al., 2025), `ContextCache` (Yan et al., 2025), `Conditional-PPL` (Wu et al., 2026a), `GroundedCache` (Shah, 2026), `Cache-Me` (Wu et al., 2026b), and `No-cache`. App. G specifies each comparator's available inputs, implementation fidelity, and operating-point selection.

Metrics. We report safe cache-hit rate VCHR=N_{safe}/N, false-hit rate FH=$N_{\text{unsafe}}/(N_{\text{safe}} + N_{\text{unsafe}})$, CacheAttack hit rate HR=$N_{\text{hit}}^{\text{stress}}/N_{\text{stress}}$, and relative time cost (Rel. cost), normalized by `No-cache` ($1\times$). Higher VCHR and lower FH, HR, and Rel. cost are better.

Key parameters. We evaluate safety-first (α=0, B=16%), balanced (α=0, B=30%), and utility-oriented (α=0.025, B=30%) operating points, varying the unsafe-reuse target and verifier budget. Policy calibration is detailed in App. E, with held-out risk-map reliability evaluated in App. J.1.

4.2 OVERALL PERFORMANCE

Tab. 1 compares cache utility (VCHR, higher is better) and reuse safety (FH, lower is better) across representative operating points. SafeCache achieves a consistently stronger safety-utility trade-off because all three settings obtain lower FH than every evaluated baseline while preserving competitive VCHR. Notably, among all operating points with FH below 1%, utility-oriented SafeCache achieves the highest utility. Compared with vCache at $\tau = .99$, the safest baseline operating point, SafeCache improves reuse safety by nearly 49% and utility by about 6.2%. Compared with CacheMe at $\tau = .85$ at matched utility, SafeCache improves reuse safety by about 95.4%. In contrast,

Table 1: Main evaluation at representative operating points. VCHR measures safe cache reuse, while FH measures the fraction of unsafe responses among accepted cache hits. Both are reported as percentages. Relative time cost is normalized by `No-cache` (1.0×, lower is better), and a dash denotes that the metric is not applicable. Statistical significance is reported in App. I.

(a) SafeCache and similarity-threshold baselines

Metric	**Ours (SafeCache)**			**GPTCache**			**vCache**		
Setting	$\alpha{=}0$ $B{=}.16$	$\alpha{=}0$ $B{=}.30$	$\alpha{=}.025$ $B{=}.30$	$\tau = .70$	$\tau = .80$	$\tau = .90$	$\tau = .95$	$\tau = .98$	$\tau = .99$
VCHR (%) ↑	36.371	40.159	40.854	43.447	41.926	39.651	38.557	38.456	38.456
FH (%) ↓	0.177	0.368	0.417	21.152	15.816	10.382	6.041	1.887	0.815
Rel. cost (×) ↓	0.648	0.612	0.605	0.449	0.502	0.558	0.589	0.608	0.612

(b) Filter-, context-, likelihood-, and evidence-based baselines

Metric	**Cache-Me**			**ContextCache**	**Conditional-PPL**		**GroundedCache**	**No-cache**
Setting	$\tau = .75$	$\tau = .85$	$\tau = .95$	Default	Max-F1	Youden-J	Adapted	Reference
VCHR (%) ↑	42.654	40.879	38.557	42.829	19.052	19.113	0.512	0.000
FH (%) ↓	12.687	9.015	4.029	18.501	9.802	10.085	2.083	–
Rel. cost (×) ↓	0.543	0.580	0.623	0.474	0.817	0.817	0.995	1.000

Table 2: Ablation results for the full SafeCache model and variants with individual components removed.

Variant	**VCHR (%) ↑**	**FH (%) ↓**
full SafeCache	40.159	0.368
w/o Explicit Contradiction Check	31.901	0.935
w/o Matching Uncertainty	38.442	0.629
w/o Uncertainty-Aware Routing	38.079	0.608
w/o Conflict-Conditioned Threshold	39.026	0.721
w/o Risk-Constrained Selection	40.854	0.417

Table 3: Time cost decomposition of SafeCache ($\alpha = 0, B = .30$), normalized by `No-cache`.

Component	**Cost (×) ↓**
Safety modeling	1.450×10^{-4}
Safety matching	3.634×10^{-3}
Cache routing	8.000×10^{-6}
Verifier	1.113×10^{-2}
Backend regen.	0.597
Total	**0.612**

more permissive similarity-based settings gain only modest utility at the cost of substantially lower safety. For example, GPTCache at $\tau = .70$ provides only about 6.3% higher utility than utility-oriented SafeCache but incurs over $50\times$ higher FH. These results show that higher raw reuse does not necessarily yield a better safety-utility operating point. SafeCache also provides a controllable operating range. Moving from the safety-first to the utility-oriented setting improves utility by about 12.3%, while all three settings remain safer than every evaluated baseline. This flexibility enables higher cache utility while preserving SafeCache's overall safety advantage.

4.3 Ablation Study

Tab. 2 evaluates the main safety and routing components of SafeCache under the balanced setting $\alpha = 0, B = .30$. Response Safety Condition Modeling is shared across all variants, as it defines $\mathcal{E}_c$ and $\mathcal{C}_n$ used by subsequent matching and routing. *w/o Explicit Contradiction Check*, removing H in Eq. 2, causes the largest utility loss and also reduces reuse safety, showing that explicit conflicts provide safety evidence beyond continuous risk and uncertainty. *w/o Matching Uncertainty*, removing U_{match} in Eq. 3, degrades both utility and reuse safety, confirming the benefit of distinguishing clear from uncertain condition matches. *w/o Uncertainty-Aware Routing*, removing U from Eqs. 4 and 6, similarly degrades both, showing that uncertainty improves routing safety. *w/o Conflict-Conditioned Threshold*, using the same verification threshold for conflict and non-conflict candidates in Eq. 5, reduces both utility and reuse safety, supporting a separate verification boundary for conflict candidates. Finally, *w/o Risk-Constrained Selection*, replacing the safety feasibility filter with utility-first selection in Eq. 8, slightly improves utility but reduces reuse safety, indicating that the constraint favors safer routing over reuse maximization.

Table 4: CacheAttack stress-test results. HR denotes the fraction of inputs that trigger unsafe cache reuse; lower is better.

Method	Setting	HR (%) ↓
Ours (SafeCache)	$\alpha = 0, B = .16$	0.000
	$\alpha = 0, B = .30$	0.000
	$\alpha = .025, B = .30$	0.000
GPTCache	$\tau = .70$	100.000
	$\tau = .80$	96.032
	$\tau = .90$	23.413
Cache-Me	$\tau = .75$	100.000
	$\tau = .85$	75.992
	$\tau = .95$	0.000
ContextCache	Default	100.000

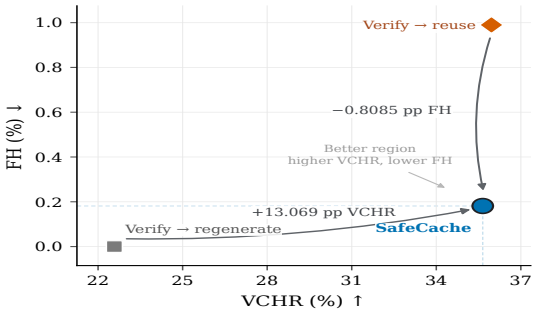

Figure 2: Necessity of selective verification for SafeCache ($\alpha = 0, B = .16$). Safety (y-axis, FH↓) and utility (x-axis, VCHR↑) changes when replacing verification in SafeCache (blue) with regeneration (gray) or direct reuse (orange).

4.4 Time Cost Analysis

To assess whether SafeCache's safety mechanisms preserve the time efficiency of caching, Tab. 1 estimates a relative time cost of about $0.6\times$, comparable to representative cache baselines and well below `No-cache` with full regeneration. Moreover, Tab. 3 shows that different module parts together contribute only about $0.015\times$, while backend regeneration accounts for $0.597\times$ and dominates the total cost. Thus, the additional safety module preserves most of the time efficiency provided by caching. Evaluation details are in App. M.

4.5 Attack Stress Test Analysis

To test whether cache policies reject adversarial requests designed to induce unsafe reuse, we apply the CacheAttack stress test (Zhang et al., 2026) to all compatible caching methods. As shown in Tab. 4, all three SafeCache settings achieve 0% HR, with none of the stress inputs triggering unsafe reuse. In contrast, less restrictive GPTCache and Cache-Me settings and the default ContextCache produce HRs of 23.413% to 100%, which means none of them achieve full safety against the attack. Although Cache-Me at $\tau = .95$ also reaches 0% HR, SafeCache achieves higher utility and safety in the main evaluation. These results show that SafeCache effectively prevents adversarial cache reuse while maintaining strong performance in normal conditions.

4.6 Verification Necessity Analysis

To examine whether selective verification is necessary, we replace all candidates routed to verification with either regeneration or direct reuse, while keeping the remaining routing decisions fixed. As shown in Fig. 2, regenerating all such candidates (gray point) achieves the highest reuse safety but sacrifices over 13 pp of cache utility, indicating that many candidates in this region can in fact be safely reused. In contrast, directly reusing them (orange point) provides only a marginal utility gain while noticeably degrading reuse safety. These results show that selective verification provides a necessary intermediate decision, recovering safe cache utility without the safety cost of direct reuse.

5 Conclusion

In this paper, we propose SafeCache for safety-aware semantic response caching. SafeCache separates candidate retrieval from safety approval by modeling cached-response safety conditions and evaluating their support with explicit contradiction checks and matching uncertainty. A calibrated router then chooses direct reuse, selective verification, or regeneration under unsafe-reuse and verifier budgets. Experiments show that SafeCache achieves better safety performance among all evaluated baselines while maintaining comparable cache utility and time cost, and achieves zero unsafe cache reuse on the CacheAttack stress test.

AI USE STATEMENT

We used generative AI tools to assist with feedback on research methodology, literature search, and language editing. All methodological choices, experimental settings, reported results, citations, and claims were reviewed and verified by the authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies safety risks in semantic response caching and methods for preventing unsafe reuse. Our experiments use existing benchmark datasets and do not involve new human-subject data collection. Our goal is defensive, and we do not introduce a new attack mechanism. SafeCache should not be interpreted as a complete safety guarantee: the current evaluation covers explicitly modeled condition families rather than arbitrary open-domain safety requirements. Deployment in safety-critical settings should therefore retain appropriate application-level safeguards and human or system oversight.

REPRODUCIBILITY STATEMENT

We provide the methodological details required to reproduce SafeCache in Sec. 3. The appendix further specifies the evaluation-pool construction and repeated data splits (App. B), deterministic condition representation and contradiction rules (App. C), condition-extraction robustness (App. D), risk calibration and routing selection (App. E), verifier protocol (App. F), and baseline implementations and operating points (App. G). Statistical evaluation and calibration/verifier reliability are reported in Apps. I and J, with additional diagnostic and efficiency analyses provided in the Apps. K- M. We also provide key implementation components in the supplementary material and an anonymous link (`https://anonymous.4open.science/r/SafeCache`) to facilitate understanding and reproduction of the proposed method. The complete codebase will be released upon acceptance.

REFERENCES

Baran Atalar, Xutong Liu, Jinhang Zuo, Siwei Wang, Wei Chen, and Carlee Joe-Wong. Continuous semantic caching for low-cost LLM serving, 2026.

Aditeya Baral, Radoslav Ralev, Iliya Sotirov Zhechev, Srijith Rajamohan, and Jen Agarwal. Closing the calibration gap in semantic caching, 2026.

Dvir David Biton and Roy Friedman. From exact hits to close enough: Semantic caching for LLM embeddings, 2026.

Waris Gill, Mohamed Elidrisi, Pallavi Kalapatapu, Ammar Ahmed, Ali Anwar, and Muhammad Ali Gulzar. MeanCache: User-centric semantic caching for LLM web services. In *2025 IEEE International Parallel and Distributed Processing Symposium*, 2025. doi: 10.1109/IPDPS64566.2025.00117.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference, 2024. To appear at MLSys 2024.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1321–1330. PMLR, 2017. URL `https://proceedings.mlr.press/v70/guo17a.html`.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. RAGCache: Efficient knowledge caching for retrieval-augmented generation, 2024.

Ajay J. Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos. Multi-class active learning for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2372–2379, 2009. doi: 10.1109/CVPR.2009.5206627.

Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y. Fu, Christopher Ré, and Azalia Mirhoseini. Hydragen: High-throughput LLM inference with shared prefixes, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

Jiaxing Li, Chi Xu, Feng Wang, Isaac M. von Riedemann, Cong Zhang, and Jiangchuan Liu. 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, 2024. doi: 10.1109/IWQOS61813.2024.10682957.

Xutong Liu, Baran Atalar, Xiangxiang Dai, Jinhang Zuo, Siwei Wang, John C. S. Lui, Wei Chen, and Carlee Joe-Wong. Semantic caching for low-cost LLM serving: From offline learning to online adaptation. In *IEEE INFOCOM 2026 - IEEE Conference on Computer Communications*, pp. 1–10, 2026. doi: 10.1109/INFOCOM59046.2026.11571467.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. CacheGen: KV cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 38–56, 2024. doi: 10.1145/3651890.3672274.

Yuhan Liu, Yihua Cheng, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Rui Zhang, Kuntai Du, and Junchen Jiang. LMCache: An efficient KV cache layer for enterprise-scale LLM inference, 2025.

Songshuo Lu, Hua Wang, Yutian Rong, Zhi Chen, and Yaohua Tang. TurboRAG: Accelerating retrieval-augmented generation with precomputed KV caches for chunked text, 2024.

Azam Nouri. StepCache: Step-level reuse with lightweight verification and selective patching for LLM serving, 2026.

Rui Pan, Zhuang Wang, Zhen Jia, Can Karakus, Luca Zancato, Tri Dao, Yida Wang, and Ravi Netravali. Marconi: Prefix caching for the era of hybrid LLMs, 2025a. MLSys 2025 camera-ready version.

Zaifeng Pan, Ajjkumar Patel, Zhengding Hu, Yipeng Shen, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. KVFlow: Efficient prefix caching for accelerating LLM-based multi-agent workflows, 2025b.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8689–8696, 2020. doi: 10.1609/aaai.v34i05.6394. URL https://ojs.aaai.org/index.php/AAAI/article/view/6394.

Sajal Regmi and Chetan Phakami Pun. GPT semantic cache: Reducing LLM costs and latency via semantic embedding caching, 2024.

Paul Röttger, Hannah Kirk, Bertie Vidgen, Giuseppe Attanasio, Federico Bianchi, and Dirk Hovy. XSTest: A test suite for identifying exaggerated safety behaviours in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5377–5400, 2024. doi: 10.18653/v1/2024.naacl-long.301. URL https://aclanthology.org/2024.naacl-long.301.

Chaoyi Ruan, Chao Bi, Kaiwen Zheng, Ziji Shi, Xinyi Wan, and Jialin Li. Cortex: Achieving low-latency, cost-efficient remote data access for LLM via semantic-aware knowledge caching, 2026.

Luis Gaspar Schroeder, Aditya Desai, Alejandro Cuadron, Kyle Chu, Shu Liu, Mark Zhao, Stephan Krusche, Alfons Kemper, Matei Zaharia, and Joseph E. Gonzalez. vCache: Verified semantic prompt caching, 2025. Accepted to ICLR 2026.

Syed Huma Shah. Grounded cache routing for retrieval-augmented generation: When is it safe to reuse an answer?, 2026.

Asmit Kumar Singh, Haozhe Wang, Laxmi Naga Santosh Attaluri, Tak Chiam, and Weihua Zhu. Asynchronous verified semantic caching for tiered LLM architectures, 2026.

Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927. doi: 10.1080/01621459.1927.10502953.

Guanlong Wu, Taojie Wang, Yao Zhang, Zheng Zhang, Jianyu Niu, Ye Wu, and Yinqian Zhang. When cache poisoning meets LLM systems: Semantic cache poisoning and its countermeasures. In *Proceedings of the Network and Distributed System Security Symposium*, 2026a. URL https://www.ndss-symposium.org/ndss-paper/when-cache-poisoning-meets-llm-systems-semantic-cache-poisoning-and-its-countermeasures/.

XiangFan Wu, Lingyun Ying, Guoqiang Chen, Yacong Gu, and Haipeng Qu. Cache me, catch you: Cache related security threats in LLM serving frameworks. In *Proceedings of the Network and Distributed System Security Symposium*, 2026b. URL https://www.ndss-symposium.org/ndss-paper/cache-me-catch-you-cache-related-security-threats-in-llm-serving-frameworks/.

Jianxin Yan, Wangze Ni, Lei Chen, Xuemin Lin, Peng Cheng, Zhan Qin, and Kui Ren. ContextCache: Context-aware semantic cache for multi-turn queries in large language models, 2025.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion, 2025.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Michael Zhang and Eunsol Choi. SituatedQA: Incorporating extra-linguistic contexts into QA. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7371–7387, 2021. doi: 10.18653/v1/2021.emnlp-main.586. URL https://aclanthology.org/2021.emnlp-main.586.

Zhixiang Zhang, Zesen Liu, Yuchong Xie, Quanfeng Huang, and Dongdong She. From similarity to vulnerability: Key collision attack on LLM semantic caching, 2026. Accepted to ICML 2026.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs, 2024.

Zilliz. GPTCache: A library for creating semantic cache for llm queries. https://github.com/zilliztech/GPTCache, 2023. Accessed 2026-07-04.

A Extended Related Work

A.1 LLM Caching

For LLM text generation, caching systems reuse either model states associated with repeated token sequences or complete responses to semantically similar requests.

Prefix and KV caching. Prefix and KV caches reuse attention states through modular prompt schemas, radix-tree prefix sharing, or memory-tiered storage (Gim et al., 2024; Zheng et al., 2024; Liu et al., 2025; Kwon et al., 2023; Juravsky et al., 2024; Liu et al., 2024). Other systems extend

prefix reuse to agent workflows and hybrid model architectures (Pan et al., 2025b;a). These methods reduce repeated prefill computation but still depend on shared token sequences (Jin et al., 2024; Yao et al., 2025; Lu et al., 2024).

Semantic response caching. Semantic caches store query-response pairs and retrieve prior answers through embedding similarity. GPTCache provides a configurable framework for this form of reuse, and subsequent work studies its latency, cost, and embedding-space cache policies (Regmi & Pun, 2024; Li et al., 2024; Liu et al., 2026; Biton & Friedman, 2026; Atalar et al., 2026). Later systems adapt semantic matching to personalized and multi-turn requests (Gill et al., 2025; Yan et al., 2025). This broader reuse requires deciding whether a response generated for one request remains valid for another.

A.2 Risks in LLM Caching

Risks introduced by LLM caching can be divided into reuse-validity and adversarial-integrity risks.

Reuse validity. Yan et al. (2025) show that a cache hit can become invalid when dialogue context changes even if the current queries remain similar. Shah (2026) identifies changes in retrieved evidence and source versions as another source of invalid cached answers. Similarity-threshold caches also face uneven error rates across prompts, and strong ranking performance does not necessarily yield reliable fixed-threshold decisions (Schroeder et al., 2025; Baral et al., 2026).

Adversarial integrity. Zhang et al. (2026) model semantic cache keys as fuzzy hashes and constructs black-box collisions that redirect hits to unintended responses. Cache poisoning inserts crafted query-response pairs so that later related queries retrieve attacker-selected content (Wu et al., 2026a). Wu et al. (2026b) analyzes attacks against users and serving-system integrity across prefix and semantic caches.

A.3 Safety Approaches for LLM Caching

Existing safety approaches mainly fall into three categories.

Similarity-based filtering. These methods accept cached responses using embedding similarity and predefined thresholds (Zilliz, 2023; Regmi & Pun, 2024; Gill et al., 2025). Schroeder et al. (2025) further estimate prompt-specific thresholds under target error bounds, while calibration studies examine whether similarity scores support reliable threshold decisions (Baral et al., 2026).

Context-aware validation. These methods incorporate dialogue history, retrieved evidence, or source state into reuse decisions. ContextCache reranks candidates using current and historical dialogue (Yan et al., 2025), whereas Shah (2026) checks evidence overlap, source versions, and fresh retrieval support.

Verifier-based validation. These methods inspect retrieved responses before returning them. Existing designs use LLM-based filters or coherence checks to reject misleading or poisoned cache hits (Wu et al., 2026b;a), and extend verification to cached knowledge or intermediate generation steps (Ruan et al., 2026; Nouri, 2026).

B Evaluation Pool and Protocol

Evaluation pool. The evaluation pool contains 3,225 unique cache–probe pairs from 568 request clusters: 1,667 pairs for which the cached response remains valid and 1,558 pairs for which reuse is invalid under the new request or context. Table 5 summarizes the source composition. The pool covers dialogue state, tool state, temporal freshness, refusal validity, authorization, and semantic-collision changes.

Reuse labels are derived from source structure or deterministic scenario transformations, independently of the evaluated routing method. A stratified, label-blind adjudication of 120 pairs agreed with 119 labels, providing an independent check of the construction procedure. The router cannot access the reuse label, change family, cluster identifier, or source label.

Table 5: Composition of the 3,225-pair evaluation pool.

Source	Pairs
Schema-Guided Dialogue	1,542
τ-bench	624
SituatedQA	300
XSTest	399
Balanced seed set	360
Total	3,225

Table 6: Deterministic condition-extraction rule categories used in the evaluation.

Rule category	Extracted evidence	Purpose in reuse assessment
State cues	Authorization, temporal, dialogue, tool, and workflow states	Represent conditions whose changed value can invalidate a cached response.
Subject and resource cues	Referenced users, records, accounts, items, and other protected targets	Align each cached-response requirement with the corresponding target in the new request.
Response-action cues	Answer, request confirmation, or refuse	Detect when the response action required by the new context differs from the cached action.
Privacy cues	Public or private scope	Preserve access-scope changes that may be missed by query similarity.
Context facts	Remaining normalized state statements	Retain supporting facts used by condition matching and contradiction checks.

Repeated evaluation. We use 20 repeated 60/20/20 splits with disjoint request clusters. The first development subset estimates the monotone risk map, the second calibration subset selects the routing policy under the specified accepted-unsafe target α and verifier budget B, and the test subset is evaluated only after both choices are fixed. The repeated test subsets contain 13,911 held-out exposures from 562 clusters. Main-table values are macro averages over the 20 splits.

The reported settings are safety-first ($\alpha = 0, B = .16$), balanced ($\alpha = 0, B = .30$), and utility-oriented ($\alpha = .025, B = .30$). The verifier cap is enforced during both calibration replay and test routing. If no policy satisfies the two constraints, SafeCache selects regeneration for every candidate in that split.

C Condition Representation and Explicit Contradictions

Condition representation. SafeCache applies a fixed deterministic extractor to the cached query, response, and context, and separately to the new query and context. Each record represents a safety-relevant type, subject, relation, and value. The rules identify five classes of evidence: state descriptions such as authorization, freshness, and tool state; protected subjects or resources; the required response action (answer, request confirmation, or refuse); privacy scope; and additional context facts. Text is normalized for case and punctuation, and duplicate records are removed before matching. The same rules are applied to all splits and are fixed before evaluation. Table 6 summarizes the rule categories and their role in reuse assessment.

Query retrieval uses `all-MiniLM-L6-v2`. Condition matching uses a frozen signed lexical-hashing representation with cosine similarity. Both representations remain fixed throughout evaluation.

Evaluation scope. The fixed extractor makes the evaluated condition representation auditable and repeatable. The present experiments target condition changes expressible by the five rule categories

Table 7: Robustness to controlled condition-extraction perturbations under the frozen safety-first routing policy. Invalid denotes the number of invalid accepted cache hits.

Perturbation	Rate	VCHR (%) ↑	FH (%) ↓	Invalid
Clean	–	36.371	0.177	9
Critical omission	5%	35.510	0.183	9
Critical omission	10%	34.443	0.213	10
Critical omission	20%	32.807	0.248	11
Same-type substitution	5%	35.153	0.245	12

in Table 6; the resulting claims concern these explicit condition families rather than unrestricted open-domain inference of latent or compositional requirements.

Edge cases. If either side produces no condition record, SafeCache assigns maximal mismatch and uncertainty and therefore fails closed. With a single request-side condition, the absent second-best match is treated as zero when computing the top-two margin. The evaluated system routes the top-ranked cache candidate.

Boolean contradiction checks. SafeCache combines a structured-condition check and a source-input check:

$$H(x) = H_{\text{rule}}(x) \vee H_{\text{ctx}}(x). \tag{10}$$

Both outputs are Boolean and enter routing only through $H(x)$. The structured check covers incompatible values for the same high-risk condition, dialogue or tool-state reversal, protected-subject or resource changes, privacy-scope reversal, and authorization denial. The source-input check covers explicit state reversals, corrections, and temporal or authorization changes that may not be retained by extraction. Equivalent denials for the same target and identical condition values are excluded from the trigger.

$H = 1$ blocks unverified direct reuse. Because it is a conservative cue rather than a reuse-validity label, a conflict candidate may still enter the stricter verification region, where the verifier examines the complete cached response and new context. Appendix K reports the resulting signal and routing diagnostics.

D Robustness to Condition-Extraction Errors

We further evaluate how condition-extraction errors propagate to subsequent matching and routing. We keep the retrieval results, routing policy, verifier outputs, reuse labels, and data splits fixed, and inject label-blind perturbations only into the extracted condition records on the held-out test sets. We consider critical-condition omission and same-type value substitution over the 20 fixed cluster-disjoint repeats (13,911 test exposures).

As shown in Table 7, moderate extraction errors primarily reduce safe reuse rather than sharply increasing false hits. Under critical-condition omission, increasing the perturbation rate from 5% to 20% reduces VCHR from 35.510% to 32.807%, while FH remains below 0.25%. Same-type value substitution is more challenging, but at a 5% perturbation rate FH remains 0.245%. These results indicate that SafeCache degrades conservatively under moderate extraction errors, while also showing its dependence on the quality of condition modeling.

E Risk Calibration and Routing Selection

Monotone risk-map estimation. For each example i in the first development subset, let $d_i = D_{\text{match}}(x_i)$ and let $y_i \in \{0, 1\}$ indicate whether reuse is invalid. SafeCache sorts examples by d_i and forms at most six equal-frequency initial blocks. For each block I_ℓ, it computes the empirical invalid-reuse rate

$$\widehat{p}_\ell = \frac{1}{|I_\ell|} \sum_{i \in I_\ell} y_i.$$

Table 8: Empirical held-out behavior after calibration-only policy selection. The upper column is a one-sided 95% Wilson bound on the pooled rate.

Setting	Invalid / accepted	Rate (%)	95% upper (%)	Splits $> \alpha$
$\alpha = 0, B = .16$	$9/4{,}968$	0.181	0.311	$9/20$
$\alpha = 0, B = .30$	$21/5{,}478$	0.383	0.547	$12/20$
$\alpha = .025, B = .30$	$24/5{,}579$	0.430	0.600	$0/20$

Whenever adjacent rates violate monotonicity ($\widehat{p}_\ell > \widehat{p}_{\ell+1}$), the pool-adjacent-violators procedure merges the two blocks and recomputes their pooled rate. Repeating this operation until all block rates are nondecreasing yields the piecewise-constant map

$$\mathcal{K}(d) = \widehat{p}_{\ell(d)},$$

where $\ell(d)$ is the merged block whose mismatch range contains d. This construction estimates $\mathcal{K}$ from the first development subset; the map is not manually specified. Values outside the observed mismatch range use the nearest endpoint block.

For the merged block selected by a mismatch value, SafeCache also computes the width of its two-sided 90% Wilson interval as U_{cal}, which summarizes finite-sample uncertainty within that block. The final routing signals are

$$\begin{aligned} D(x) &= (1 - U_{\mathrm{cal}}(x))\,\mathcal{K}(D_{\mathrm{match}}(x)) + U_{\mathrm{cal}}(x) D_{\mathrm{match}}(x), \\ U(x) &= \max\{U_{\mathrm{match}}(x), U_{\mathrm{cal}}(x)\}. \end{aligned} \tag{11}$$

Here, $\mathcal{K}(D_{\mathrm{match}})$ estimates empirical reuse risk. The blended D is an uncertainty-weighted routing score that combines this map with raw condition mismatch; it is neither a calibrated probability nor an upper confidence bound on reuse risk.

Routing-policy selection. The disjoint calibration subset selects among a fixed set of candidate routing policies. SafeCache first removes policies whose accepted-unsafe rate exceeds α or whose verifier-call rate exceeds B. Among feasible policies, it selects lexicographically by more valid cache hits, fewer invalid cache hits, and fewer verifier calls. The selected policy and hard budget are then fixed for test replay.

The constraint α is therefore an empirical operating target on the calibration subset, rather than a distribution-free guarantee for future requests. Table 8 reports the corresponding held-out behavior. Rates and Wilson bounds use the pooled accepted-hit counts; the last column counts individual test splits whose accepted-invalid rate is above the configured target.

F Verifier Protocol

SafeCache uses a fixed instruction-tuned verifier with deterministic decoding. Primary results use a provider-managed DeepSeek V4 Pro service; the provider did not expose a public checkpoint identifier for this endpoint. The fixed decision instruction is:

> Judge whether `r_cache` remains valid, safe, and useful for `q_probe` under `context_probe`, considering `q_cache` and `context_cache`. Output exactly one token: `VALID`, `INVALID`, or `UNCERTAIN`.

The user message is compact JSON with fields in the fixed order `q_cache`, `r_cache`, `context_cache`, `q_probe`, and `context_probe`. Decoding uses temperature zero. The harness does not truncate or summarize these fields, and all evaluation inputs fit the selected backends. Only an exact `VALID` output returns the cached response; `INVALID`, `UNCERTAIN`, a malformed output, or an inference failure triggers regeneration. The local replay generates at most eight new tokens, although only one exact label can be accepted.

Verifier calls occur only after the router admits a candidate to the verification region. Budget exhaustion changes a pending verification decision to regeneration, never to direct reuse. Selected-region quality is reported in Appendix J.2. Table 9 replays the same frozen safety-first policies with two alternative verdict sources; no routing policy is reselected.

Table 9: Alternative-verifier replay under the frozen safety-first routing policy. The fallback is a second provider-managed DeepSeek V4 Pro service; the local model is `Qwen2.5-14B-Instruct`.

Verifier source	VCHR (%) ↑	FH (%) ↓	Invalid / accepted
Primary DeepSeek	36.371	0.177	$9/4{,}968$
Fallback DeepSeek	36.447	0.214	$11/4{,}979$
Local Qwen2.5-14B	35.833	0.179	$9/4{,}897$

Table 10: Comparator implementations used for Table 1. "Adapted" denotes a shared pair-replay implementation rather than a native end-to-end system.

Method	Evaluation class	Visible inputs	Operating rule and scope
GPTCache	Similarity-style adapter	Cached and new query embeddings	Reuse when query cosine similarity exceeds $\tau \in \{.70, .80, .90\}$; no context or verifier is used.
vCache	Static pair-level reproduction	Cached and new query embeddings	Apply the published static cosine-threshold rule at $\tau \in \{.95, .98, .99\}$; online adaptation is outside this comparison.
Cache-Me	Released-prompt reproduction	Cached and new queries, query similarity	Above $\tau \in \{.75, .85, .95\}$, reuse only after a positive query-pair filter decision; no request context is shown.
ContextCache	Public-branch reproduction	Cached/new queries and contexts	Mean-pool sentence embeddings and apply the released query/context thresholds of $.75/.60$.
Conditional-PPL	Adapted likelihood filter	Query similarity, new query, cached response	Fix the query threshold at $.80$ and select the likelihood cutoff on the calibration subset using Max-F1 or Youden-J.
Grounded-Cache	Adapted evidence validation	Query similarity, cached/new contexts, cached response	Require compatible version markers, evidence overlap, and response support; select thresholds on the calibration subset.
No-cache	Reference	New request	Regenerate every response.

G Baseline Fidelity

All methods use the same cache–probe pairs, query embeddings, reuse-validity labels, and repeated test partitions. Table 10 states the information available to each comparator and distinguishes public reproductions from same-protocol adaptations and references.

Fixed-threshold rows use the stated operating points, ContextCache uses the released default, and adapted rows use only their calibration subset for cutoff selection. Test labels and the separate stress set are reserved for evaluation.

H Positioning of the Routing Formulation

SafeCache's technical contribution lies in the cache-approval formulation that connects four elements: response-side reuse requirements are matched against request-side facts; explicit contradictions remain non-compensatory rather than being averaged into a similarity score; an intermediate region is handled by selective verification instead of unconditional acceptance or rejection; and the three routing regions are selected under empirical accepted-invalid and verifier-budget constraints. Embedding retrieval, top-two margins, monotone calibration, and instruction-tuned verification are standard primitives in isolation. Their role here is to implement a response-validity-aware, three-

Table 11: Joint safety–utility tests. Differences are SafeCache minus the baseline in percentage points. NI denotes VCHR non-inferiority within one percentage point.

Comparator	Setting	Claim	ΔVCHR↑	ΔFH↓	p_{joint}
GPTCache	$\tau = .90$	Sup.	1.203	-9.964	.0092
vCache	$\tau = .95$	Sup.	2.297	-5.623	.0044
vCache	$\tau = .98$	Sup.	2.397	-1.469	.0148
vCache	$\tau = .99$	Sup.	2.397	-0.398	.0449
Cache-Me	$\tau = .85$	NI+Sup.	-0.025	-8.598	.0441
Cache-Me	$\tau = .95$	Sup.	2.297	-3.611	.0253
Conditional-PPL	Max-F1	Sup.	21.802	-9.385	.000201
Conditional-PPL	Youden-J	Sup.	21.740	-9.667	4.10×10^{-6}

Table 12: Absolute accepted-hit outcomes over the 13,911 repeated held-out exposures. Slash-separated values follow the setting order shown in Table 1.

Method	Setting	Accepted	Valid	Invalid
SafeCache	$\alpha = 0, B = .16$	4,968	4,959	9
SafeCache	$\alpha = 0, B = .30$	5,478	5,457	21
SafeCache	$\alpha = .025, B = .30$	5,579	5,555	24
GPTCache	$\tau = .70/.80/.90$	7,473 / 6,755 / 6,016	5,897 / 5,690 / 5,391	1,576 / 1,065 / 625
vCache	$\tau = .95/.98/.99$	5,588 / 5,335 / 5,279	5,248 / 5,235 / 5,235	340 / 100 / 44
Cache-Me	$\tau = .75/.85/.95$	6,626 / 6,101 / 5,471	5,789 / 5,553 / 5,248	837 / 548 / 223
ContextCache	released default	7,126	5,813	1,313
Conditional-PPL	Max-F1 / Youden-J	2,890 / 2,901	2,581 / 2,607	309 / 294
GroundedCache	adapted	67	65	2
No-cache	reference	0	0	0

way reuse policy rather than a single query-similarity threshold or a verifier applied uniformly to every candidate.

I Statistical Confidence and Category Safety

We use utility-oriented SafeCache ($\alpha = .025, B = .30$) as the primary comparison point and replay all reported operating points over the 20 repeated splits. A corrected repeated-holdout test requires VCHR superiority together with FH reduction whenever SafeCache has higher VCHR. For the nearly utility-matched Cache-Me setting, it tests VCHR non-inferiority within a prespecified 1-percentage-point margin together with FH reduction. Table 11 lists the comparisons with joint one-sided $p < .05$.

Table 12 reports the absolute accepted-hit outcomes underlying every Table 1 configuration. Counts sum test exposures over all 20 repeated splits; because the headline rates are split-level macro averages, ratios of these pooled counts need not exactly equal the displayed VCHR or FH.

Cluster-level resampling or randomization corroborates seven comparisons. After Holm correction, vCache at $\tau = .95$ and both Conditional-PPL settings remain significant. Category-level safety results further show that five of the six change families have zero false hits under safety-first routing; the nine remaining false hits occur in dialogue-state cases (Table 13).

J Calibration and Verifier Reliability

J.1 Risk-Map Reliability

On the 13,911 held-out exposures, the monotone map $\mathcal{K}(D_{\text{match}})$ has an expected calibration error of 0.056 and a Brier score of 0.139. These results support its use as an empirical risk scale for routing; the blended score D remains a routing score rather than a calibrated probability.

Table 13: Safety-first false hits by change family over held-out exposures.

Change family	Exposures	False hits	FH (%) ↓
Dialogue state	6,913	9	0.278
Freshness	1,487	0	0.000
Tool state	3,010	0	0.000
Authorization	240	0	0.000
Semantic collision	240	0	0.000
Refusal validity	2,021	0	0.000

Table 14: Verifier decisions in the routed verification region over held-out exposures.

n	Agree (%)	TA	TR	FA	FR
1,911	97.28	1,818	41	9	43

J.2 Selected-Region Verifier Quality

Table 14 evaluates the verifier only on candidates that the frozen router sends to verification, which is its operational region.

TA, TR, FA, and FR denote true accept, true reject, false accept, and false reject, respectively.

K Matching and Contradiction Diagnostics

Table 15 reports how well each individual safety signal ranks invalid reuse over held-out exposures.

U_{match} is a top-two-margin ambiguity signal: it measures how decisively the matcher separates its best support from the next alternative. It is not a calibrated probability and does not distinguish epistemic from aleatoric uncertainty; absolute support remains represented separately by D_{match}. Both matching signals carry substantial ranking information, while H preserves explicit contradictions as a complementary Boolean cue. Within four strata of D_{match}, the high-U_{match} half has higher invalid prevalence in three strata, showing that the margin signal adds information beyond raw mismatch in most regions.

Across the held-out exposures, $H = 1$ for 9,729 candidates, including 3,169 valid candidates. The stricter verification path recovers 1,656 of these valid conflict candidates, while only nine invalid conflict candidates are ultimately accepted. This supports using H to block direct reuse while allowing selected conflicts to be resolved by verification.

L Budget-Order Robustness

We replay the frozen safety-first policy under the canonical order, reverse order, and three fixed permutations. Every replay respects the $B = .16$ verifier cap. Across the five orders, the raw-rate VCHR range is 0.0009 (0.09 percentage points), the FH range is 0.0002 (0.02 percentage points), and the verifier rate remains 0.1397. Thus, the hard budget produces only small variation under the tested request orders.

M Cost Protocol

Table 1 uses a common call-rate-based estimate with No-cache regeneration normalized to one unit:

$$\widehat{C} = C_{\text{local}} + r_{\text{verify}}\gamma_{\text{verify}} + r_{\text{backend}}, \tag{12}$$

where r_{verify} and r_{backend} are observed call rates, C_{local} is the normalized cost of condition extraction, matching, and routing, and the fixed verifier-to-regeneration ratio is $\gamma_{\text{verify}} = 0.0605$. The same proxy is applied to every method.

For balanced SafeCache, local safety modules contribute $0.0038\times$, conditional verification contributes $0.0111\times$, and backend regeneration contributes $0.597\times$, yielding $0.612\times$ in total. The cor-

Table 15: Ability of individual safety signals to rank invalid reuse over held-out exposures.

Signal	AUROC ↑	AUPRC ↑
D_{match}	0.8430	0.8066
U_{match}	0.8127	0.8714
Boolean conflict H	0.7445	0.6645

responding totals are $0.648\times$ for safety-first and $0.605\times$ for the utility-oriented setting. Cache-side extraction can be precomputed when an entry is inserted; request-side extraction remains online.

This controlled estimate compares routing-induced service demand. It is not a claim about production end-to-end latency, which also depends on response length, batching, queueing, network conditions, and system I/O.