


# Trident: Accelerating Speculative Decoding with Similarity-Driven Context Reuse

Zhuoer Shao
zhuoershao@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Shengan Zheng*
shengan@sjtu.edu.cn
MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
Shanghai, China

Xingda Zhou
qaqleonard@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Zhiyuan Huang
hzy364@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Shi Shu
shu-shi@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Cong Zhou
cong258258@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Guifeng Wang
wangguifeng@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Linpeng Huang*
lphuang@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

## Abstract

Speculative decoding (SD) reduces the autoregressive latency of large language model (LLM) inference, but tree-based verification remains costly in this paradigm. We present Trident, a similarity-driven context-reuse framework that accelerates SD by exploiting semantic locality across independent requests. Trident retrieves the closest historical prefix for each incoming prefix and dynamically selects among Sequence Reuse, Probability Reuse, and SD-based Recomputation. The reuse paths leverage historical sequences, target probabilities, and KV cache through fuzzy matching, guided by an approximation that constrains reuse to cases with bounded generation loss. To scale historical-context storage, Trident stores contexts in an RDMA-enabled remote layer indexed with OMH-verified LSH and hides access latency through asynchronous retrieval and layer-wise context restoration. Evaluations demonstrate that Trident reduces TBT by 69.2% on average over autoregressive engines across models and datasets. It also outperforms existing training-free SD frameworks by an average of 30.3% across the same scenarios.

***CCS Concepts:*** • **Computing methodologies** → **Machine learning approaches**; *Distributed computing methodologies*; **Natural language generation**; • **Computer systems organization** → Architectures; • **Mathematics of computing** → Probability and statistics.

***Keywords:*** LLM, Speculative Decoding, Fuzzy Matching, LSH, OMH, Distributed System, RDMA, KV cache Reuse

**ACM Reference Format:**
Zhuoer Shao, Shengan Zheng, Xingda Zhou, Zhiyuan Huang, Shi Shu, Cong Zhou, Guifeng Wang, and Linpeng Huang. 2027. Trident: Accelerating Speculative Decoding with Similarity-Driven Context Reuse. In *22nd European Conference on Computer Systems (EuroSys '27), April 19–23, 2027, Rabat, Morocco.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3842654.3848549

*Corresponding authors.



This work is licensed under a Creative Commons Attribution 4.0 International License.

*EuroSys '27, Rabat, Morocco*

© 2027 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2971-3/2027/04
https://doi.org/10.1145/3842654.3848549

## 1 Introduction

Large Language Models (LLMs) have demonstrated unprecedented capabilities across a wide range of natural language tasks [7, 14, 50, 58, 61], yet their inference efficiency remains constrained by autoregressive decoding [34, 46]. Since each output token depends on the previously generated prefix, conventional decoding invokes the target model sequentially, limiting throughput and increasing time-between-tokens (TBT) as models scale. Speculative Decoding (SD) [34, 38, 39, 56, 57, 64] mitigates this bottleneck by decoupling generation into a two-stage execution flow. A lightweight draft model first proposes multiple candidate tokens, and the target model then verifies those candidates in parallel so that the final output distribution remains identical to standard autoregressive decoding. Tree-based SD further improves the probability of accepting draft tokens by organizing candidates into a hierarchical draft tree [46]. However, tree expansion also increases the number of draft tokens that must be produced and verified. Since self-attention computation grows with

token volume, tree-based SD can shift the bottleneck to draft generation [23] and target-model verification, particularly in resource-constrained inference environments [72].

KV cache [31, 53, 60] reduces redundant self-attention computation in LLM inference by storing the key and value tensors of previous tokens. Once cached, these tensors can be reused during later decoding steps, avoiding repeated self-attention computation over the same prefix. However, the reuse of KV tensors inherently relies on exact prefix matching [2, 47, 49], confining it to intra-request inference where cached states correspond to the same immutable prefix. Modern workloads, including autonomous-agent traces [51, 68] and multi-tenant conversations [69, 75], often contain independent requests with nearly identical semantic prefixes that differ only in small surface details. These small variations invalidate exact prefix matching, forcing full attention recomputation for prefixes whose model states and likely continuations are still close enough to be reused.

Fuzzy matching opens a broader reuse opportunity by allowing historical intermediate states from semantically similar requests to be reused across requests. By retrieving historical prefixes that are close to an incoming prefix, Locality-Sensitive Hashing (LSH) relaxes the exact-match constraint of KV cache reuse and avoids redundant attention-state recomputation [73]. When integrated with SD, this reuse opportunity extends beyond KV cache to a broader historical context[1] including generated token sequences and target probabilities, which accelerate complementary stages of decoding with modest storage overhead. Turning this broader reuse opportunity into a practical design, however, requires overcoming three key challenges:

**Challenge 1: Inflexibility of Fuzzy-Matching Reuse Policies.** Semantic similarity varies continuously, but existing context-reuse policies are mostly binary. Current fuzzy-matching approaches [65, 73] follow a binary policy that either fully reuses historical contexts or fully recomputes them. Such a rigid policy is poorly matched to SD since it cannot handle the broad spectrum of semantic similarity across prefixes. Aggressive reuse [3, 67] may introduce semantic deviations from a mismatched prefix, leading to significant generation quality degradation. Conservative recomputation, on the other hand, discards useful historical context and falls back to full tree-based SD [12, 38, 46]. As a result, a binary reuse policy collapses a spectrum of partially reusable cases into a coarse tradeoff between unsafe reuse and unnecessary recomputation, missing opportunities to reuse valid historical context without compromising generation quality.

**Challenge 2: Absence of Quantified Similarity-Quality Correlation.** Reliable context reuse requires a quantitative criterion that relates prefix similarity to the generation loss induced by reusing historical states. Existing fuzzy-matching approaches estimate similarity using signals such as attention deviation, token overlap, or embedding distance [65, 73, 77]. These signals are useful for retrieving candidate prefixes, but they do not characterize how a similarity mismatch between the current and historical prefix perturbs the target model's output distribution. This leaves no principled basis for deciding whether historical contexts remain accurate enough to reuse. Without a quality-aware criterion, reuse decisions remain heuristic, making it difficult to protect generation fidelity while consistently exploiting reusable historical context.

**Challenge 3: Insufficiency of Local Storage for Massive Historical Contexts.** Effective cross-request fuzzy matching depends on retaining a large and diverse history of reusable inference states. In modern LLM serving deployments, independent requests continuously produce new historical contexts, causing the reuse database to grow rapidly with workload volume. Keeping this history on compute nodes is impractical because local memory is already shared by model execution, active requests, and runtime buffers. Under high load, limited capacity forces aggressive eviction, reduces the chance of finding high-similarity prefixes, and increases index misses. Retaining a large volume of historical contexts improves matching coverage, yet local storage pressure prevents it from scaling and drives the inference back to costly recomputation.

We propose **Trident**, a similarity-driven context reuse framework that accelerates tree-based speculative decoding by reusing historical contexts across semantically similar requests. Trident decouples fast-path decoding from historical-context storage and retrieval through a disaggregated compute-storage architecture. Compute nodes evaluate prefix similarity and execute decoding, while remote storage nodes maintain a large LSH-indexed repository of historical contexts and return the closest reusable prefix through RDMA. The disaggregated design allows Trident to exploit cross-request semantic locality without forcing each compute node to retain a large local reuse database.

For each incoming prefix, Trident retrieves the most similar historical prefix using OMH-verified LSH, computes a lightweight OMH-verified Jaccard reuse similarity score, and routes the request to one of three execution branches. Sequence Reuse is used for high-similarity prefixes and reuses historically generated tokens as draft candidates. Probability Reuse is used for medium-similarity prefixes and reuses historical target probabilities while generating fresh draft tokens for the current prefix. SD-based Recomputation is used for low-similarity prefixes and executes standard tree-based SD. Together, Trident's three execution branches form a similarity-aware context-reuse strategy that exploits strong

[1] In this paper, the historical context comprises KV cache, target probabilities, and generated tokens of the historical prefixes.

matches aggressively, weaker matches conservatively, and novel prefixes through recomputation.

Trident grounds reuse decisions in an approximate correlation between prefix similarity and generation quality. The approximation maps Jaccard similarity to hidden-state cosine similarity, decomposes hidden-state deviation into aligned and orthogonal components, and derives a safe-reuse bound from the stability of high-probability next-token candidates. Trident enforces this criterion throughout decoding rather than only at the initial match. After each decoding step, the compute node appends the accepted tokens, recomputes similarity to the retrieved historical prefix, and redirects execution when the continuation begins to diverge. Such recursive checking prevents a transient fuzzy match from committing the request to overly aggressive reuse.

To make historical context storage scalable, Trident separates context management from compute-node execution. Remote storage nodes index historical prefixes with Ordered MinHash, which preserves token order as well as token overlap during LSH lookup. Context retrieval is performed asynchronously over RDMA and overlapped with local SD computation. When Probability Reuse later transitions to recomputation, the compute node may lack the local KV cache that would have been produced by target-model forwarding. Trident closes this gap with a dynamic KV-cache restoration pipeline that fetches the aligned historical KV segment from remote memory and restores it while local computation proceeds. As a result, Trident scales historical-context reuse beyond local memory limits while keeping remote lookup and context restoration off the latency-critical decoding path.

In summary, this paper makes the following contributions:

- We identify cross-request semantic similarity as an underexploited opportunity for accelerating speculative decoding, and present Trident, a similarity-driven context reuse framework that adaptively selects the appropriate reuse strategy based on prefix similarity.
- We make Trident practical at scale by placing historical contexts in RDMA-enabled remote storage, using OMH-verified LSH to retrieve order-preserving matches, and overlapping remote access with layer-wise context restoration.
- We augment Trident with a quality-aware reuse criterion that approximates the correlation between prefix similarity and generation quality, allowing it to selectively reuse historical contexts and dynamically detect semantic drift during decoding.
- We implement an asynchronous retrieval and restoration pipeline that overlaps remote context access with local speculative computation and restores missing KV cache segments layer by layer when execution switches back to recomputation.
- We evaluate Trident on LLaMA2-7B, Vicuna-13B, and LLaMA2-70B, showing that it reduces TBT by 69.2% on average over autoregressive decoding and outperforms existing training-free SD frameworks by 30.3% on average under different workloads.

## 2 Background and Motivation

### 2.1 Speculative Decoding

Speculative decoding (SD) [29, 34, 43, 56, 76] accelerates LLM inference by replacing strictly token-by-token target-model execution with a propose-and-verify workflow. Instead of invoking the target model once for every output token, SD first uses a lightweight draft model to autoregressively propose a short continuation and record the corresponding draft probabilities. The target model then evaluates these tokens in parallel and computes their exact probabilities under the target distribution. The verification procedure compares the draft and target probability distributions and accepts a valid prefix of the drafted sequence when the candidates are consistent with the target model. By amortizing one target-model invocation over multiple accepted tokens, sequence-based SD significantly reduces decoding latency without compromising generation quality.

The lossless guarantee of SD depends on the verification rule used to convert approximate draft proposals into exact target-model samples. Greedy validation, adopted in early SD works [56], accepts a draft token only when its target probability is no lower than its draft probability. This deterministic rule provides a straightforward correctness condition, but it also rejects many useful candidates whenever the draft and target distributions differ moderately.

Speculative sampling [34] replaces deterministic verification with an exact probabilistic correction. A draft token is accepted when $p_{\text{target}}/p_{\text{draft}}$ exceeds a random threshold drawn uniformly from $[0, 1]$. Thus, tokens underestimated by the draft model are accepted, whereas overconfident draft tokens are accepted only when sufficiently supported by the target model. Rejected tokens are replaced by samples from the residual positive difference between the target and draft distributions. This probabilistic verification rule substantially improves acceptance over greedy validation while preserving the lossless semantics of SD, and therefore serves as the verification primitive for later SD frameworks.

To further improve acceptance, recent SD frameworks replace the linear draft sequence with a tree of candidate tokens [12, 38, 40, 46]. During drafting, the draft model autoregressively builds a token tree in which each node represents a candidate token conditioned on the root-to-node prefix and is associated with a draft probability distribution. The target model then verifies the tree in a single parallel forward pass, producing target distributions for all candidate nodes under the tree-structured dependencies. The verifier traverses the tree in prefix order, applying speculative sampling only to nodes whose ancestors have already been accepted. Rejections prune the affected branch and invoke the residual

sampling rule, while accepted paths can continue toward deeper nodes. By exploring multiple continuations in one verification step, tree-based SD improves the probability of accepting a longer valid prefix, but it also increases the volume of draft candidates that must be generated and verified.

Recent tree-based SD frameworks mainly focus on either the producer of draft candidates or the structure in which those candidates are verified. One line of work moves draft generation into the target model to reduce the overhead of invoking a separate draft model. Medusa [8] attaches multiple decoding heads to predict future tokens in parallel, and Hydra [1] strengthens this by allowing later heads to depend on earlier speculative predictions. Kangaroo [41] instead uses an adapter and early-exit policy to avoid unnecessary shallow-layer computation. EAGLE3 [40] further replaces feature prediction with direct token prediction over multi-layer internal features. Another line of work improves the structure of the candidate set and the efficiency of verification. SpecInfer [46] verifies token trees in parallel using tree-aware attention masks. Sequoia [12] formulates draft-tree construction as a dynamic-programming problem to allocate candidate branches more effectively.

However, the key inefficiency of existing SD frameworks lies in their fundamentally isolated per-request execution model, which limits optimization to the computation performed within a single decoding instance. Even when incoming prompts are semantically close to historical contexts, these frameworks still regenerate candidate tokens, rebuild intermediate states, and repeat target-model verification from scratch. Tree-based SD amplifies this inefficiency because improving acceptance typically requires a larger candidate tree, which increases both draft-model autoregression and target-model verification work.

As inference workloads shift toward agentic execution [68], multi-turn dialogue [22], and template-heavy services [74], deployments increasingly combine larger models with wider or deeper draft trees to sustain acceptance, further raising the cost of purely intra-request optimization. Yet such workloads often issue prompts that differ in surface form while preserving similar intent, structure, and continuation behavior [9, 28, 35]. Existing SD frameworks leave this locality outside their design scope, and therefore repeatedly pay for computation that could be avoided when a current prefix is sufficiently aligned with historical contexts. This motivates similarity-driven context reuse as a complementary direction to intra-request SD optimization.

### 2.2 Cross-Request Context Reuse

Cross-request reuse [24, 65, 73] extends context reuse beyond the boundary of a single request by exploiting semantic locality among independent prefixes. Production LLM services rely on exact-match prompt caching [2, 18, 49], which avoids redundant computation when a new request shares an identical prefix. However, exact matching captures only a small subset of the locality present in real workloads. Requests in modern LLM services often express the same instruction [24, 28], follow the same dialogue template [22, 70], share semantically equivalent user intents [35], or ask structurally similar questions [9] while differing in individual tokens. Exact matching treats these prefixes as cache misses, even though their internal states and next-token distributions may remain close enough to support reuse.

Fuzzy matching broadens the reuse surface by treating semantic alignment, rather than token-level equality, as the condition for retrieving historical context. Under this paradigm, a current prefix reuses the intermediate states associated with a historical prefix when the two are close enough to induce similar model states and continuation behavior. Locality-Sensitive Hashing (LSH) [11, 17, 25, 30, 62] makes such retrieval efficient by mapping each prefix to a hash signature under which similar prefixes collide with high probability. The resulting buckets turn semantic similarity into a low-overhead indexing decision for locating reusable historical contexts at runtime.

Recent LSH-based inference optimizations differ in both the granularity at which they apply hashing and the type of state they make reusable. HashAttention [19] and HashEvict [42] use binary hash signatures inside a single request. The former casts attention sparsification as supervised Hamming-distance retrieval, whereas the latter applies SimHash before attention to filter KV entries that are unlikely to affect the current token. These designs reduce intra-request attention or cache-management overhead, but they do not expose reusable computation across independent prompts. SemShareKV [73] instead uses token-level LSH matching to enable cross-request KV cache sharing. It reuses the reference KV cache when the target-reference similarity exceeds an empirical threshold, and otherwise performs recomputation. Nevertheless, existing approaches have two key limitations that prevent them from fully exploiting cross-request semantic locality to accelerate SD.

First, existing context reuse approaches mostly follow a binary reuse policy that either fully reuses historical contexts or recomputes tokens from scratch. This rigidity stems from treating KV cache as the only reusable context [24, 73]. Once reuse is restricted to KV states, a matched prefix must either be trusted as a whole or discarded completely, since partially trusting a mismatched KV state offers no natural way to separate safe reuse from unsafe semantic drift [67].

Speculative decoding provides the opportunity to exploit this spectrum, where different artifacts tolerate different degrees of mismatch. Highly similar prefixes can support draft reuse by treating historically generated tokens as draft candidates, and their aligned KV cache segments can also be reused when recomputation resumes from a matched prefix. Moderately similar prefixes may still benefit from target-probability reuse while generating fresh draft tokens for the current request. By matching the granularity of reuse to the

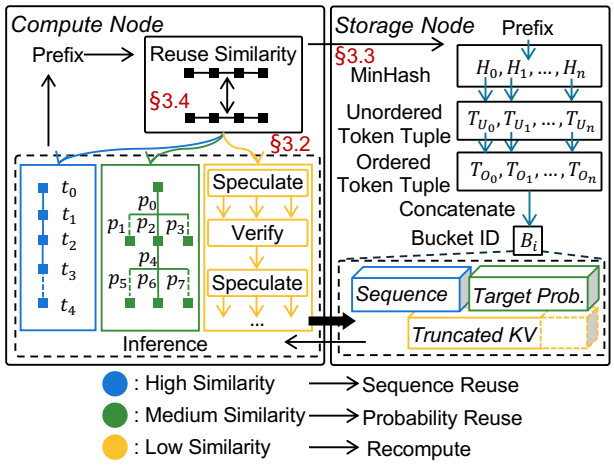



**Figure 1.** Overview of Trident's compute-storage architecture. Compute nodes evaluate prefix similarity and execute the selected reuse path, while storage nodes organize and serve reusable historical contexts.

strength of prefix similarity, SD provides the opportunity to turn fuzzy matching from an all-or-nothing cache decision into a similarity-driven execution policy.

Second, existing LSH-based reuse often uses order-agnostic hash signatures for matching, overlooking the token order that distinguishes semantically different prefixes. Prefixes with similar token sets can still lead to different model states when their tokens occur in different positions or participate in different dependency structures. MinHash [5, 6, 36, 37] is a representative LSH implementation for prefix similarity [4, 33, 52]. It summarizes a prefix by applying several independent hash functions, selecting the minimum-hash element for each function, and concatenating selected elements into a signature whose collision behavior estimates set-level resemblance. The efficiency comes from ignoring token positions, which makes it unable to capture positional information needed to distinguish semantically different sequences with similar token membership.

Ordered MinHash (OMH) [10, 44] offers a promising basis for addressing this mismatch by making the LSH signature order-aware without abandoning compact hashing. Rather than storing a single unordered minimum per hash function, OMH forms low-hash token tuples and sorts each tuple by original position ID before concatenation. The resulting order-aware hash value captures both token membership and relative ordering, allowing the LSH index to retrieve historical prefixes with semantic and state-level alignment.

## 3 Design

### 3.1 Overview

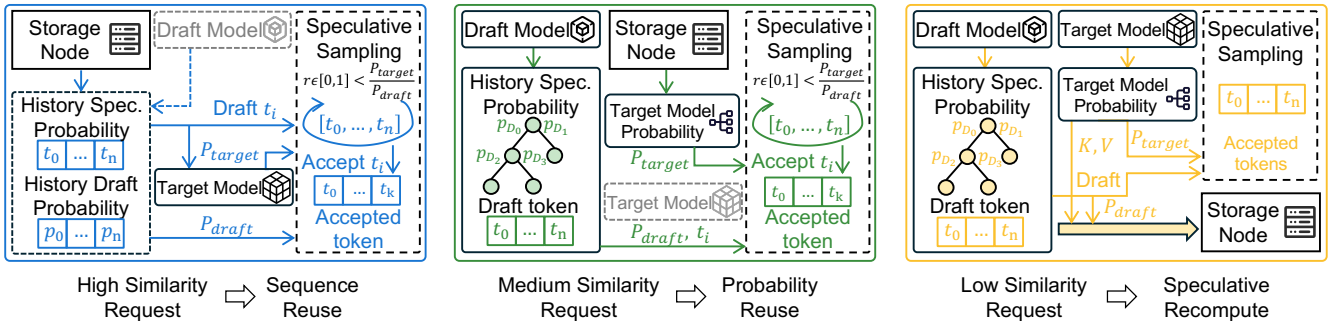
Trident addresses speculative decoding inefficiency by turning cross-request similarity into a routing decision that selects the most effective reuse strategy at runtime. It couples local compute nodes with remote storage nodes in a disaggregated context-reuse framework, so that each incoming prefix can reuse the historical context best aligned with its semantics rather than following a fixed reuse-or-recompute policy. The disaggregated design separates latency-sensitive speculation-verification execution from historical-context management and enables Trident to leverage semantically relevant reuse opportunities across a broad range of prefixes.

As illustrated in Figure 1, Trident's compute node is responsible for similarity evaluation and speculation-verification execution, while the storage node indexes reusable history and returns the contexts associated with the closest prior prefix. At its core, Trident employs a similarity-driven context-reuse engine that evaluates each incoming prefix against the retrieved historical prefix and routes it to the execution path best matched to its reuse opportunity (§3.2). When a new query arrives, the compute node forwards its prefix to the corresponding storage node, which applies OMH-verified LSH to locate the bucket, identify the highest-similarity historical prefix, and return the associated reusable contexts (§3.3). The compute node then computes a lightweight OMH-verified Jaccard reuse similarity score (§3.4) and dispatches the prefix to the corresponding execution path.

The selected execution path is determined by how much of the historical context can be reused safely. For high-similarity prefixes, *Sequence Reuse* adopts the historically-generated sequence as draft tokens and verifies them in linear SD, removing draft-model autoregression from the critical path and improving acceptance. For medium-similarity prefixes, *Probability Reuse* leverages stored historical target probabilities during tree-based SD to bypass expensive target-model forward passes. For low-similarity prefixes, Trident falls back to standard *SD-based Recomputation* and persists the newly generated reusable contexts to storage nodes for future access. After each SD step, Trident appends the newly accepted tokens to the prefix, re-evaluates the similarity, and can switch to a different path if the reuse opportunity changes.

Trident's storage layer makes similarity-driven routing practical at scale by organizing historical contexts across distributed storage nodes and indexing them with OMH-verified LSH. The key advantage of OMH over order-agnostic LSH is that it retains both token overlap and token order, rather than approximating similarity from token membership alone. Concretely, OMH uses MinHash to select token tuples, sorts each tuple by the tokens' original positions, and concatenates the ordered tuples into an order-aware hash value. The resulting order-aware hash value drives bucket placement and lookup, enabling accurate retrieval of reusable historical prefixes with low lookup overhead.

Trident further hides remote-access overhead with asynchronous RDMA-based context retrieval. Historical contexts are transmitted in parallel with local speculative decoding, so remote access does not stall the critical path. In addition,




**(a)** Workflow of Sequence Reuse. **(b)** Workflow of Probability Reuse. **(c)** Workflow of SD-based Recomputation.

**Figure 2.** Trident's similarity-driven reuse workflow under different prefix-similarity levels.

Trident addresses the KV-cache gap introduced by Probability Reuse. Since Probability Reuse leverages historical target probabilities to bypass target-model forward passes, switching back to SD-based Recomputation may leave the compute node without the local KV cache required to continue standard SD. Trident resolves this with a dynamic KV-cache restoration pipeline that fetches the highest-similarity historical KV tensors and truncates them according to the current generated length, reconstructing a semantically aligned KV state without regenerating it from scratch.

Finally, Trident grounds its routing policy in a mathematical approximation of the correlation between input-prefix similarity and output-generation quality to avoid redundant recomputation while preserving generation accuracy. Trident first maps token-level Jaccard similarity to hidden-state cosine similarity, then decomposes hidden-state deviation into a semantically aligned component and an orthogonal perturbation. Based on this decomposition, it derives a semantic-loss inequality that constrains the perturbation so that the top-$m$ next-token probabilities remain stable. The resulting cosine-similarity bound is translated into a Jaccard threshold, yielding a lightweight but principled routing evaluation metric that prevents redundant recomputation of reusable prefixes without compromising generation quality.

### 3.2 Similarity-Driven Context Reuse

Trident performs similarity-driven context reuse by routing each incoming prefix according to the similarity of the best-matched historical prefix. For each incoming prefix, the compute node first queries the OMH-indexed remote storage nodes to retrieve the most semantically similar historical prefix. It then computes the OMH-verified Jaccard similarity between the incoming and retrieved prefixes and uses this evaluation score to determine the reuse strategy for the current decoding step. Specifically, Trident selects Sequence Reuse for high-similarity prefixes, Probability Reuse for medium-similarity prefixes, and SD-based Recomputation for low-similarity prefixes. Rather than applying a fixed reuse-or-recompute policy, Trident matches each incoming request to the execution path that best balances acceleration with approximate correctness bounded by semantic-loss constraints.

**Sequence Reuse.** When the incoming prefix is highly similar to a stored historical prefix, Trident routes the request to the Sequence Reuse path. Figure 2a illustrates the workflow of Sequence Reuse. Instead of invoking the draft model to generate draft tokens and draft probabilities, the compute node retrieves the historical draft sequence and its probabilities from the remote storage nodes and directly uses them as the draft for the current request. The target model then performs a single parallel forward pass over the retrieved draft tokens to compute the corresponding target probabilities. Trident then applies speculative sampling and accepts tokens until the first rejection based on the ratio between target and draft probabilities.

Sequence Reuse removes draft-model autoregression from the critical path, because Trident no longer needs to regenerate draft tokens for highly similar prefixes. Since Sequence Reuse is invoked only when the current prefix is strongly aligned with the retrieved history, the reused sequence is likely to achieve a high acceptance rate and therefore improves throughput. At the same time, generation quality is preserved because every reused token is still verified by the target model through the speculative-sampling procedure.

**Probability Reuse.** When the incoming prefix has lower but still substantial similarity to a historical prefix, Trident routes the request to Probability Reuse. This path retains the tree-based SD structure, but replaces expensive target-model verification with reusable historical target probabilities. As illustrated in Figure 2b, the compute node generates a fresh draft tree and its draft probabilities for the current prefix, while the storage node returns the historical target probabilities associated with the closest matching prefix. Trident then traverses the draft tree from the root and applies speculative sampling at each node using the ratio between the retrieved target probability and the locally generated draft probability. Verification continues along a branch only when the parent

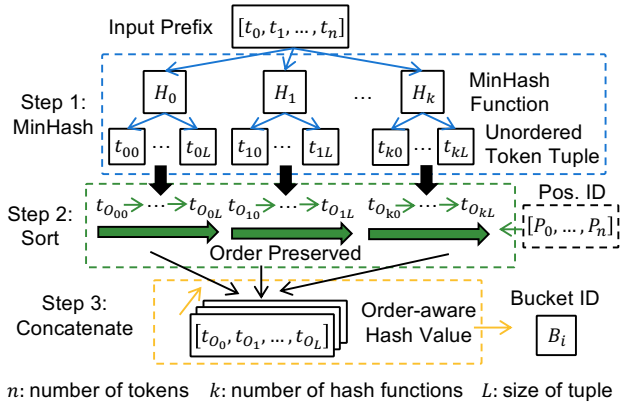



**Figure 3.** Ordered MinHash construction for prefix indexing

node is accepted, and stops when all children of a node are rejected, or when a leaf node is reached.

Probability Reuse removes the most expensive part of verification, the target-model forward pass, without discarding prefix-specific drafting. This makes it a better fit for medium-similarity cases, where reused probabilities remain useful but directly reusing the historical token sequence would be too aggressive. At the same time, because the draft tokens are produced specifically for the current prefix, Probability Reuse remains better aligned with the current request than direct sequence reuse and is therefore appropriate for medium-similarity cases. The path therefore improves efficiency while keeping generation quality aligned with the current prefix.

**SD-based Recomputation.** When the incoming prefix has low similarity to all available historical contexts, Trident falls back to standard SD-based Recomputation. As shown in Figure 2c, the draft model generates the draft tree and draft probabilities, the target model computes the exact target probabilities, and speculative sampling verifies the draft in the standard tree-based SD manner. After verification, the compute node persists the new prefix and its generated historical context to remote storage nodes for future reuse.

The SD-based fallback is essential for Trident's correctness and coverage. It guarantees that low-similarity requests that do not admit safe reuse are still processed correctly, while also expanding the historical database that supports later Sequence Reuse and Probability Reuse decisions.

**Semantic Mismatch Detection.** Under similarity-driven context reuse, an incoming prefix may occasionally be matched to a semantically misaligned historical prefix. In that case, the similarity between the reused historical sequence and the newly generated sequence decreases as decoding continues. To detect this decay and protect generation quality, the compute node recursively re-evaluates the extended prefix and dynamically selects the execution path for the next iteration.

If such a prefix mismatch occurs, reused historical target probabilities will lead to draft rejections during verification. As discussed in Section 2.1, these rejections force Trident to sample replacement tokens from the residual probability distribution. This resampling quickly reduces the similarity between the newly generated and historical token sequences and triggers a switch to SD-based Recomputation. Trident therefore preserves generation quality by reverting to standard recomputation when a semantic mismatch is detected.

However, the execution-path transition introduces a local KV cache gap on the compute node, since Probability Reuse skips the target model's forward pass and thus does not build KV cache entries for the newly generated tokens. To close this gap, Trident retrieves the historical KV cache of the highest-similarity prefix from the remote storage nodes and reuses the aligned portion instead of rebuilding the entire cache from scratch. The historical KV cache is truncated along the token dimension to match the length of newly generated tokens from Probability Reuse and transmitted to the compute node. Since Probability Reuse operates only under medium-to-high similarity, the restored KV state remains sufficiently aligned and speeds up the subsequent SD-based Recomputation.

### 3.3 Scalable Historical Context Management

Trident offloads historical-context storage and retrieval to a dedicated remote layer that keeps latency-sensitive decoding on the compute node while scaling the reusable context database across storage nodes. Each storage node maintains historical prefixes together with generated token sequences, target probabilities, and KV cache, and exposes them through OMH-verified LSH lookup. This separation allows Trident to search and retrieve reusable history without requiring each compute node to keep a large local context database. Once a storage node identifies the closest historical prefix, Trident transfers the corresponding contexts through asynchronous RDMA and overlaps the transmission with local speculative decoding, thereby reducing the impact of remote access on end-to-end inference latency.

**Order-Aware LSH Indexing.** Trident uses OMH-verified LSH to locate reusable historical prefixes efficiently while preserving both token overlap and token order. Figure 3 decomposes this process into three steps. Given an input prefix, OMH first applies multiple MinHash functions to the prefix tokens and selects several minimum-hash tokens under each function, producing unordered token tuples. It then sorts each tuple by the tokens' original position IDs in the prefix, converting the tuple into an order-preserving representation. Finally, Trident concatenates the ordered tuples into an order-aware hash value and uses this value to determine the target hash bucket.

This order-aware construction is important for prefix reuse because token membership alone is insufficient to characterize semantic alignment. Two prefixes may contain many of the same tokens but arrange them in different orders, leading

to different model states and different continuation distributions. By sorting MinHash-selected tokens according to their original positions before concatenation, OMH embeds positional information into the LSH key. As a result, prefixes with similar token content and similar ordering are mapped to the same bucket with high probability, while prefixes with weaker order alignment are less likely to be treated as reusable matches.

To scale lookup across storage nodes, Trident replicates the OMH index on each storage node and partitions the historical contexts across them. When a new prefix arrives, the compute node computes its LSH hash value, derives the bucket ID, and routes the lookup directly to a single storage node. The storage node searches only the corresponding bucket in its OMH-indexed database and returns the most semantically similar historical prefix. Its reusable contexts are then fetched from the node that holds them. The compute node then computes the OMH-verified Jaccard reuse similarity between the incoming prefix and the retrieved prefix and uses this similarity score to select Sequence Reuse, Probability Reuse, or SD-based Recomputation. The OMH-verified LSH indexing approach eliminates database-wide scans and cross-node broadcasts, reducing lookup latency and communication overhead while preserving retrieval accuracy through the order-aware candidate selection.

**Historical-Context Population, Capacity, and Eviction.** Trident bounds the size of its historical-context database through on-demand population, distributed context sharding, and per-bucket LRU eviction. The historical-context database is initialized as empty. When a lookup finds no reusable context, the compute node executes SD-based Recomputation and inserts new contexts into the OMH bucket after generation. Trident partitions the historical contexts across storage nodes, allowing aggregate storage capacity to scale with the number of storage nodes. When a bucket reaches its capacity limit, the storage node responsible for that bucket evicts its least recently used context. If a subsequent lookup cannot find a reusable context because of eviction, Trident simply recomputes the request.

**Stall-Free Context Retrieval.** Remote context storage introduces a potential communication bottleneck because each reuse path depends on historical data that resides outside the compute node. Trident avoids making these remote contexts a serial dependency by retrieving them asynchronously through an RDMA transport augmented with GPUDirect RDMA (GDR). After the storage node identifies the closest historical prefix, the compute node issues non-blocking RDMA reads for the context objects required by the selected execution path. With GDR, retrieved contexts can be placed directly into GPU-accessible memory, avoiding extra host-side staging and reducing the CPU-mediated copy overhead that would otherwise appear on the inference critical path.

For compact contexts, Trident schedules retrieval according to the point at which each reuse path consumes them. In Sequence Reuse, Trident fetches the historically generated token sequence together with its draft probabilities as the draft verified by the target model. In Probability Reuse, it fetches historical target probability distributions while the local draft model constructs the current draft tree and probabilities. Given the small footprint of these compact contexts, Trident transfers them through RDMA into VRAM while local drafting proceeds, ensuring that the retrieved sequence and probability data are ready when verification begins.

KV cache requires additional handling because it is much larger and is needed mainly when execution-path transitions expose missing target-model state. In Probability Reuse, the compute node verifies draft tokens using historical target probabilities and therefore bypasses the target model's forward pass. If recursive similarity re-evaluation later routes the request to standard recomputation, the compute node may not have the local KV tensors needed to continue target-model execution from the current prefix.

To close this gap, Trident makes KV-cache restoration range-aware and asynchronous. The compute node first derives the token range needed to resume from the current generated prefix and includes this range in the KV-cache fetch request. The storage node then slices the KV cache of the highest-similarity historical prefix before transmission, retaining only the entries aligned with the current prefix and filtering out future-token states that would be stale for the current request. During a transition from Probability Reuse to SD-based Recomputation, this slicing preserves the semantically aligned KV-cache segment and truncates the mismatched segment. Removing the divergent entries prevents mismatched KV states from propagating into subsequent decoding. The sliced KV segment is transferred asynchronously over a GDR-backed RDMA path directly into VRAM. Trident restores the cache layer by layer as KV tensors arrive, appending each restored segment to the existing local KV cache and allowing remote transfer to proceed in parallel with local recomputation. This pipelined restoration enables the target model to resume from the reconstructed prefix state without waiting for the full KV cache to be fetched or rebuilding the missing cache from scratch.

The truncated KV tensors are transferred through a layer-wise restoration pipeline. Instead of waiting for the entire KV cache to arrive before resuming computation, Trident restores the cache layer by layer, allowing remote fetching and local target-model execution to proceed in a pipelined manner. Because path transitions are triggered after recursive similarity checks, the restored historical KV segment corresponds to a prefix that remains closely aligned with the current request up to the transition point. By applying asynchronous RDMA retrieval to generated token sequences, target probabilities, and KV cache, Trident keeps remote context access off the critical path whenever independent local work is available, while avoiding full KV-cache reconstruction during execution-path changes.

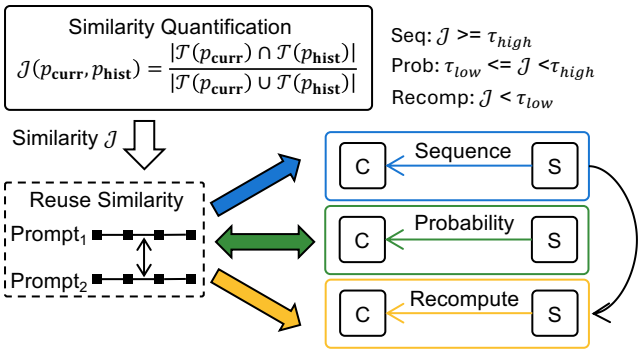



**Figure 4.** Trident's execution-path selection under different OMH-verified Jaccard reuse similarity levels.

### 3.4 Approximation of the Similarity-Quality Correlation

Trident's similarity-driven path selection relies on a lightweight similarity metric that relates input-prefix alignment to generation quality. We approximate this relationship across token and representation spaces. At runtime, Trident uses Jaccard similarity to compare an incoming prefix with the best retrieved historical prefix. To relate the token-set similarity score to generation behavior, Trident maps it to the cosine similarity between the corresponding hidden states, where semantic deviation can be bounded. The resulting bound provides the quantitative basis for distinguishing safe historical-context reuse from SD-based Recomputation and for selecting the appropriate reuse path. Figure 4 shows how this similarity drives Trident's execution paths: high-similarity prefixes enter Sequence Reuse, medium-similarity prefixes enter Probability Reuse, and low-similarity prefixes fall back to SD-based Recomputation. Trident recomputes the similarity after each decoding step and switches paths when the continuation drifts from the retrieved history.

**Similarity Metric.** Trident uses token-level Jaccard similarity as a lightweight score for identifying potential reuse candidates. We denote the set of unique tokens derived from a prefix $\mathbf{p}$ by $\mathcal{T}(\mathbf{p})$. The similarity score $\mathcal{J}$ is defined as:

$$\mathcal{J}(\mathbf{p}_{curr}, \mathbf{p}_{hist}) = \frac{|\mathcal{T}(\mathbf{p}_{curr}) \cap \mathcal{T}(\mathbf{p}_{hist})|}{|\mathcal{T}(\mathbf{p}_{curr}) \cup \mathcal{T}(\mathbf{p}_{hist})|} \tag{1}$$

Trident interprets this score as the routing signal to select distinct execution paths for the current decoding step.

- **High-similarity prefixes** ($\mathcal{J} \geq \tau_{high}$) are strongly aligned with the historical prefix and are routed to Sequence Reuse, where Trident uses the historical sequence as the draft for verification.
- **Medium-similarity prefixes** ($\tau_{low} \leq \mathcal{J} < \tau_{high}$) retain enough semantic alignment for probability-level reuse and are routed to Probability Reuse, where Trident leverages historical target probabilities while generating prefix-specific draft tokens.
- **Low-similarity prefixes** ($\mathcal{J} < \tau_{low}$) lack a safe reuse opportunity and fall back to standard SD-based Recomputation, preserving correctness and producing new historical contexts for future reuse.

**Mapping from Jaccard Similarity to Cosine Similarity.** The Jaccard score is suitable for online routing, but generation quality is affected by the model's hidden-state deviation. Trident therefore maps token-level Jaccard similarity to hidden-state cosine similarity using the empirical relationship derived from statistical analysis. Specifically, we sample 600K prompts from the real-world ShareGPT and QQP datasets. For each prefix, we compute the Jaccard similarity and the cosine similarity between the corresponding hidden states, and derive a statistical mapping between the two measurements. This empirical relationship allows Trident to translate the derived cosine-similarity bounds into Jaccard thresholds for online routing. For the retrieved hidden state $h_r$ and the new hidden state $h_n$, we denote their Cosine Similarity as $cs$.

$$\text{CosSim}(h_r, h_n) = cs \tag{2}$$

This mapping bridges the gap between an efficient token-level routing metric and a continuous measure of semantic similarity in the model representation space.

**Semantic Loss Constraint.** After obtaining the hidden-state cosine similarity, we establish a semantic-loss inequality by decomposing the hidden state and constraining the deviation of the top-probability tokens. Given cosine similarity $cs$, we decompose the current hidden state into an aligned component and an orthogonal perturbation:

$$h_n = cs \cdot h_r + \beta h_{r\perp} \tag{3}$$

Here $h_{r\perp}$ is orthogonal to $h_r$, so the inner product $h_n \cdot h_r$ isolates the projection coefficient of $h_r$, and the unit-norm constraint $|h_r| = |h_n| = 1$ identifies the coefficient as the cosine similarity. This orthogonal decomposition separates the portion of $h_n$ that preserves the retrieved prefix semantics from the portion that may perturb the next-token distribution.

For the Probability Reuse path, reusing historical target probabilities perturbs the output distribution and degrades generation quality. To bound this deviation, we require the normalized logits of the top-$m$ probability tokens to remain close under the retrieved and current hidden states [29]. Let $L_n$ denote the normalized logit of the current prefix, $L_r$ the normalized logit of the retrieved prefix, and $L_{min}$ the normalized logit of the lowest-probability token among the top-$m$ candidates. A normalized logit is the projection of the prefix hidden state onto the corresponding token direction. Under the hidden-state decomposition, the logit deviation comprises changes along the retrieved direction and an orthogonal perturbation, whose magnitude we bound using a high-dimensional isotropic approximation. This analysis

relates hidden-state cosine similarity to the top-$m$ logit deviation, with $L_{min}$ serving as a conservative stability margin. Since the top-ranked candidates have a nontrivial margin over $L_{min}$, we impose the following sufficient constraint:

$$|L_n - L_r| \leq |L_{min}| \tag{4}$$

This constraint limits the hidden-state perturbation to a range that preserves the stability of the high-probability candidate set.

For the Sequence Reuse path, the execution strategy fundamentally shifts. Instead of generating draft tokens sequentially, Sequence Reuse adopts an entire historical token sequence. Therefore, without a draft model dynamically conditioning on the prefix at each step, any initial semantic perturbation between the current and retrieved hidden states accumulates across the sequence. These errors cascade through the sequence, causing the expected semantic deviation to scale with the square root of the accepted sequence length $\sqrt{K}$. To guarantee that the cumulative deviation stays within the margin $L_{min}$, we tighten the single-step tolerance.

$$|L_n - L_r| \leq \frac{|L_{min}|}{\sqrt{K}} \tag{5}$$

Combining Inequalities (4) and (5) with Equation (3) yields a cosine-similarity inequality.

**Cosine Similarity Lower Bound and Jaccard Selection.** Using the semantic-loss inequality, we derive lower bounds on the hidden-state cosine similarity required by the Probability Reuse and Sequence Reuse paths and then map these values back to the corresponding Jaccard similarity. We first define $L_k$ as the $k$-th largest normalized logit of the top-$m$ probability tokens from the retrieved prefix. Then, the Lambert Function $W$ [15, 32] is used to represent $L_k$ as a function of the vocabulary size $N$, the hidden dimension $D$, and the number of top candidates $m$. Finally, we use $L_k$, $D$, $m$, and $cs$ to represent $|L_n - L_r|$ and $L_{min}$, converting the semantic-loss constraint into a numerical lower bound on the cosine similarity required for reuse.

$$|L_n - L_r| \leq (1 - cs) \cdot L_k + \sqrt{1 - cs^2} \cdot 2\sqrt{\frac{1 - L_k^2}{D - 1}} \tag{6}$$

With these settings, the cosine-similarity requirement derived from Inequalities (4) and (5) stabilizes around $cs = 0.803$ for Sequence Reuse and $cs = 0.561$ for Probability Reuse. Subsequently, we evaluate prompts across the ShareGPT and QQP datasets to statistically map the cosine similarity to Jaccard similarity. The two cosine similarities above are mapped to Jaccard similarities of 0.89 and 0.51, respectively.

We therefore set $\tau_{high} = 0.89$ and $\tau_{low} = 0.51$ as correctness-preserving bounded-reuse thresholds for Sequence Reuse and Probability Reuse. During inference, prefixes below $\tau_{low}$ are recomputed, while prefixes above it can reuse historical context through Probability Reuse or, if they exceed $\tau_{high}$, Sequence Reuse. Because Trident recomputes $\mathcal{J}$ after each decoding step, the threshold also supports dynamic path correction when a reused prefix begins to diverge from the current generation.

**Model Adaptation and Applicability.** Trident's reuse thresholds transfer across the evaluated workloads without workload-specific calibration. It derives the empirical Jaccard-to-cosine mapping offline from a large, diverse prompt set and observes that the mapping remains consistent across the evaluated workloads. Trident can therefore apply the thresholds derived for a target model to different workloads without per-workload profiling or tuning.

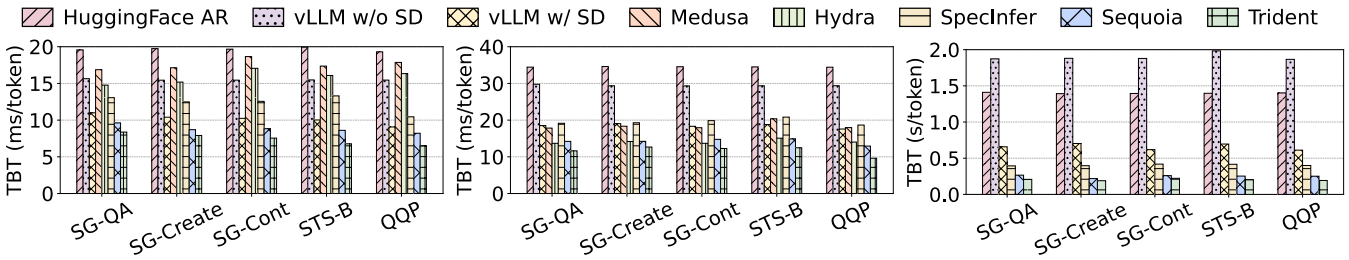
Trident adapts reuse thresholds to a new target model through a one-time offline procedure that requires no costly online recalibration. For each model, Trident combines the vocabulary size, hidden dimension, number of top-probability tokens, and Sequence Reuse length to solve the semantic-loss inequalities for both reuse paths. It then converts the resulting cosine-similarity requirements into $\tau_{low}$ and $\tau_{high}$ with the Jaccard-to-cosine relationship. Although these thresholds depend on the target model's configuration, substituting the configurations of the evaluated models into the semantic-loss inequalities changes the resulting bounds only marginally. The models therefore produce nearly identical thresholds, allowing Trident to use one shared threshold pair. A new configuration only requires rerunning this procedure before inference, without introducing any online overhead.

# 4 Evaluation

## 4.1 Experimental Setup

**Models.** We evaluate Trident on three target LLMs that cover different serving scales: LLaMA2-7B [59], Vicuna-13B [13], and LLaMA2-70B [59]. For training-free SD engines that require a draft model, we pair LLaMA2-7B with JF68M, Vicuna-13B with JF160M, and LLaMA2-70B with LLaMA2-7B.

**Testbed.** Our experiments use a disaggregated compute-storage deployment with a single compute-storage pair for end-to-end model evaluation and extend to multiple compute and storage nodes for the storage-node scalability analysis (Section 4.5). The compute-node GPU is selected according to each model's memory footprint. LLaMA2-7B and Vicuna-13B run on a 48GB NVIDIA RTX 6000 Ada GPU with a 2.6GHz Intel Xeon Gold CPU. LLaMA2-70B runs on a 96GB NVIDIA RTX PRO 6000 GPU. Since the 70B model cannot fit within GPU memory during serving, we keep 40 FP16 layers on the GPU and offload the remaining layers to CPU DRAM. The storage node uses a 2.6GHz Intel Xeon Gold CPU and 256GB DRAM to store historical contexts. The compute and storage nodes communicate through 200 Gbps InfiniBand HDR with RDMA. We initialize Trident with an empty storage node at the beginning of each experimental run. Therefore, all reported performance measurements include the storage-node warm-up time required to populate the database. The




**(a)** LLaMA2-7B **(b)** Vicuna-13B **(c)** LLaMA2-70B

**Figure 5.** End-to-end TBT comparison of Trident against existing AR and SD baselines across datasets and model scales.

measurements use compute-local index lookup with remote RDMA context storage on a single storage node.

**Datasets.** We evaluate Trident on five workloads constructed from three public datasets, chosen to stress different forms of cross-request locality in LLM serving. The first three workloads are derived from ShareGPT [55], a collection of real conversational traces that captures the heterogeneity of deployed chat services. We partition these traces into question answering (SG-QA), open-ended creation (SG-Create), and text continuation (SG-Cont), so that the evaluation covers both structured user requests and open-ended generation. To complement these workloads, we further include the Semantic Textual Similarity benchmark (STS-B) [21, 45, 48], which provides sentence pairs with controlled lexical and semantic relatedness, and the Quora Question Pairs dataset (QQP) [26], which contains user-facing questions with naturally recurring intents and paraphrases. Together, these workloads span broad prefix-similarity patterns, allowing us to evaluate whether similarity-driven routing can exploit reusable historical contexts without exact prompt matches.

**Baselines.** We organize the baselines into two groups to separate improvements from serving-system engineering and improvements from SD. The first group consists of autoregressive baselines that execute standard token-by-token decoding. We include HuggingFace Autoregression (HF-AR) [63], an LLM inference autoregressive implementation, and vLLM [31], an optimized inference and serving system that improves serving efficiency through PagedAttention and optimized KV-cache management. These baselines quantify the performance of optimized AR serving without candidate speculation or cross-request context reuse.

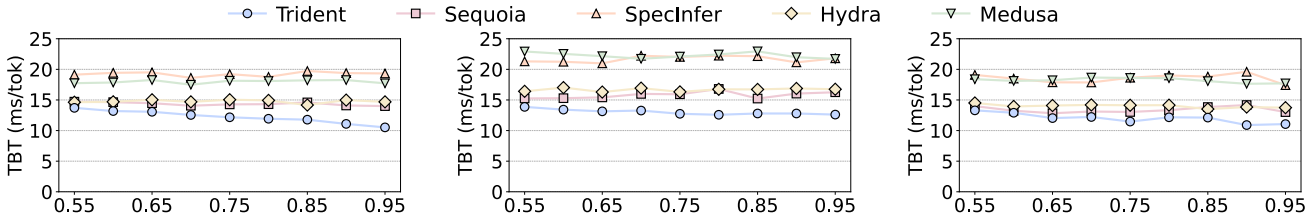
The second group consists of SD-based non-AR serving systems. We first evaluate vLLM with standard SD [31] to isolate the benefit of speculative execution atop vLLM's serving optimizations. We then compare against representative SD frameworks that improve draft generation or verification. SpecInfer [46] constructs and verifies token trees using lightweight draft models. Sequoia [12] optimizes draft-tree construction to improve acceptance. Medusa [8] attaches multiple prediction heads to the target model and generates draft tokens in parallel without a separate draft model. Hydra [1] extends this multi-head design by introducing sequential dependencies among draft heads. These frameworks represent the main design space of existing SD acceleration, while Trident adds a complementary cross-request reuse dimension. We exclude Medusa and Hydra from LLaMA2-70B experiments because training their additional prediction heads at this scale is beyond our available GPU capacity.

### 4.2 Overall Performance

We evaluate Trident's end-to-end decoding efficiency using TBT across all workloads and model scales. Figure 5 shows that Trident consistently delivers the lowest latency among all evaluated AR and SD frameworks. The consistent advantage over both AR engines and SD baselines indicates that similarity-driven context reuse complements conventional speculative decoding rather than replacing only a single optimization stage. Overall, Trident reduces TBT by 69.2% on average over AR serving and outperforms existing training-free SD frameworks by 30.3%.

Trident's relative advantage increases with model scale. Averaged over all workloads, Trident reduces TBT over AR serving by 57.2% on LLaMA2-7B and by 87.4% on LLaMA2-70B. This is because Trident accelerates the parts of speculative decoding whose cost scales with model size. SD already amortizes one target-model invocation across multiple accepted tokens, which makes it increasingly valuable as target-model forwarding becomes more expensive. Trident further reduces the remaining SD cost by reusing historical sequences or target probabilities when the retrieved prefix is sufficiently aligned with the current request. As the target model grows, forward passes become increasingly dominant in the decoding budget, and the cost of verification becomes more pronounced. Therefore, bypassing draft generation and target verification yields larger absolute savings, which translate into stronger relative speedups on larger models.

AR and SD serving engines exhibit distinct scaling behavior in Figure 5. The two AR engines, HF-AR and vLLM without SD, remain bound to one target-model forward pass per generated token and therefore exhibit higher TBT than other frameworks. vLLM benefits from PagedAttention on LLaMA2-7B and Vicuna-13B, reducing TBT by 17.9% over




**(a)** ShareGPT **(b)** STS-B **(c)** QQP

**Figure 6.** TBT comparison between Trident and existing SD frameworks under different workload similarity.

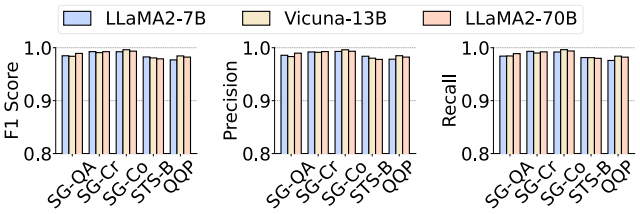



**Figure 7.** BERTScore-based semantic similarity between Trident and lossless SD outputs across workloads.

HF-AR through efficient KV-cache management. This advantage reverses on LLaMA2-70B, where vLLM becomes 35.5% slower because model serving requires offloading and turns page-level KV lookup into a visible latency bottleneck.

SD baselines mitigate the AR bottleneck by verifying multiple drafted tokens per target-model invocation. vLLM with SD therefore improves over vLLM without SD, with the largest gain on LLaMA2-70B, where it reduces TBT by 65.3%. Among trained-head methods, Hydra reduces TBT by 16.6% on average over Medusa, since each head conditions on the partial draft produced by earlier heads and thus improves acceptance. As for training-free methods, Sequoia reduces TBT by 31.4% over SpecInfer by allocating more draft-tree capacity to branches with higher acceptance potential rather than expanding candidates uniformly. These optimizations improve intra-request speculation, but they still regenerate drafts, rebuild intermediate state, and verify candidates from scratch. Trident removes this redundant work by similarity-driven context reuse, reducing TBT by 17.3% over Sequoia and achieving the lowest latency across all configurations.

### 4.3 Similarity Analysis

To isolate how cross-request locality affects Trident, we construct a controlled similarity-performance study on Vicuna-13B using ShareGPT, STS-B, and QQP. Their original average prefix Jaccard similarities are 0.33, 0.63, and 0.71, respectively, which provide limited points in the similarity space and are insufficient to cover the full similarity spectrum. We therefore synthesize workloads with target average similarities from 0.55 to 0.95 through stratified sampling over Jaccard-similarity buckets. For ShareGPT, we first compute pairwise similarities among original requests, form request pairs, and then sample from the resulting buckets. For STS-B and QQP, we use their native sentence pairs, bucket them by Jaccard similarity, and choose sampling ratios that match each target similarity. We then measure TBT and Trident's execution-path distribution on each workload, linking latency trends to shifts among SD-based Recomputation, Probability Reuse, and Sequence Reuse. Figures 6 and 8 report these results.

Existing SD frameworks show little sensitivity to workload similarity in Figure 6. Their execution remains confined within each request, so they continue to build draft candidates and invoke verification even when many incoming prefixes closely resemble prior ones. Trident converts the same cross-request locality into lower TBT. As the average prefix similarity rises from 0.55 to 0.95, more retrieved histories remain compatible with the current prefix, increasing the opportunity to reuse historical sequences or target probabilities. Consequently, Trident's average TBT reduction over Sequoia and SpecInfer increases from 19.1% to 31.1%. Figure 8 confirms the source of this gain. The share of requests handled by SD-based Recomputation drops from 83.6% to 57.3%, while Sequence Reuse and Probability Reuse expand accordingly. This shift indicates that OMH-verified LSH retrieves prefixes with sufficient alignment for reuse, and that Trident can convert those matches into lower decoding latency without placing remote retrieval on the critical path.

The effectiveness of similarity-driven reuse varies with the extent to which lexical overlap captures reusable model-state and continuation alignment. The largest TBT reduction appears on ShareGPT, where Trident reduces TBT by 23.3% as the workload similarity grows from 0.55 to 0.95, because real conversational requests often reuse templates and discourse structure. OMH-verified LSH therefore retrieves historical prefixes that remain compatible with the current request, allowing more queries to enter Sequence Reuse and Probability Reuse to sustain high draft acceptance during target-model verification. QQP and STS-B expose weaker forms of reusable locality, with TBT reductions of 16.9% and 9.0%, respectively. QQP retains moderate reuse opportunities because paraphrases often share intent, while STS-B weakens reuse by pairing lexically similar sentences with divergent continuation behavior.

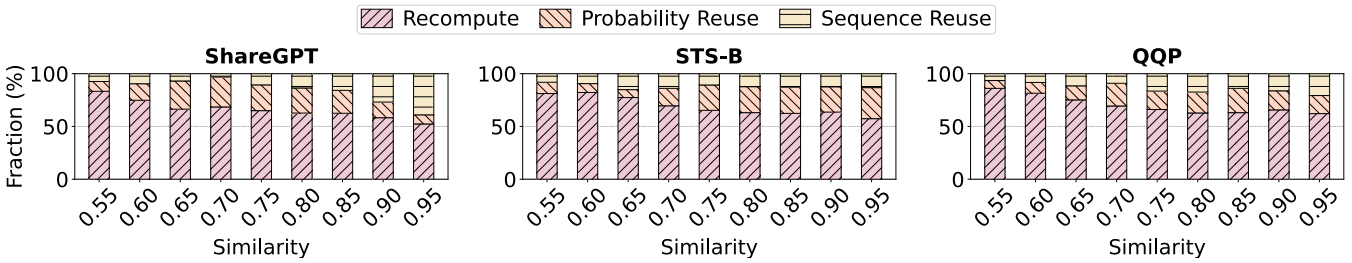
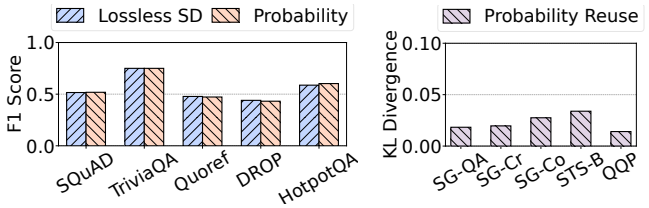



**Figure 8.** Trident's execution-path distribution across workloads and prefix-similarity levels.




**(a)** QA F1 **(b)** KL Divergence

**Figure 9.** Generation quality and distributional fidelity of Probability Reuse compared with lossless SD.

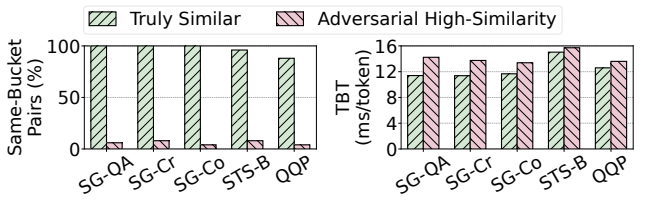



**Figure 10.** OMH-verified LSH filtering and TBT performance under adversarial-robustness evaluations.

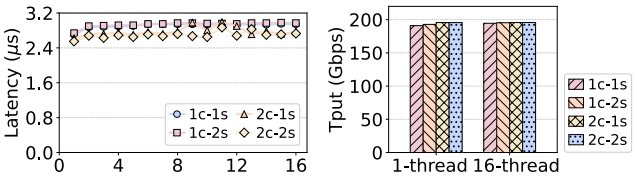
Trident preserves generation quality despite imperfect matches. We evaluate quality by comparing Trident's outputs with those of lossless standard SD using BERTScore [71]. As shown in Figure 7, precision, recall, and F1 remain above 0.95 for every workload and model size. Verification rejects mismatched drafts, and recursive similarity checks route execution back to SD-based Recomputation when reuse becomes unsafe. This allows Trident to exploit genuine matches while preventing inaccurate reuse from degrading output quality.

### 4.4 Probability Reuse Analysis

We conduct generation-quality and adversarial-robustness evaluations to assess the semantic fidelity and performance characteristics of Probability Reuse. The end-to-end quality evaluation compares Probability Reuse with lossless SD on Vicuna-13B using QA F1 across SQuAD [54], TriviaQA [27], Quoref [16], DROP [20], and HotpotQA [66]. The distribution-fidelity evaluation measures per-token KL divergence between reused and target probabilities recomputed by lossless SD, over all verified decoding positions of Probability-Reuse requests, on SG-QA, SG-Create, SG-Cont, STS-B, and QQP. For each dataset, we sample original prompts to construct adversarial high-similarity pairs. We create each adversarial variant by replacing words with antonyms or inserting negations, thereby introducing polarity reversals, semantic negation, or conflicting instructions. We submit these adversarial pairs and truly similar pairs from the same dataset to Trident. The adversarial-robustness evaluation then applies OMH-verified LSH to both groups and compares their same-bucket placement rates. Finally, the end-to-end performance evaluation compares the TBT of both groups to determine how OMH-verified LSH affects decoding latency.

Trident incurs negligible semantic loss from Probability Reuse across QA datasets in Figure 9a. We compare Probability Reuse with lossless SD on Vicuna-13B using end-to-end QA F1 across five datasets. Lossless SD and Probability Reuse achieve nearly identical F1 scores on the same datasets, yielding an average difference of 1.2% compared to each other. Trident limits these gaps by reusing historical target probabilities only when prefix similarity satisfies the semantic-loss bound and constructing a fresh draft tree for the current prefix. This combination bounds distributional error without copying the historical response, allowing Probability Reuse to avoid target-model recomputation while maintaining answer quality close to lossless SD.

Trident's Probability Reuse closely preserves the target-probability distributions produced by lossless SD. Figure 9b reports the median per-token KL divergence between the reused historical target probabilities and the corresponding probabilities computed by lossless SD over all verified decoding positions of Probability-Reuse requests. The per-token KL divergence remains between 0.014 and 0.034 across all five workloads, with STS-B at the upper end of this range. By contrast, historical and current distributions from prefix pairs below the reuse threshold have a median KL divergence of 1.70, up to two orders of magnitude larger. This separation shows that Trident, guided by the semantic-loss bound, rejects poorly aligned contexts and admits reuse only when historical probabilities are reasonably well aligned with the current target distribution. STS-B has higher divergence because it contains more adversarial high-similarity prompts, in which substantial lexical overlap masks semantic conflicts. These prompts can retain high token-level similarity and therefore resemble valid reuse candidates, even though their next-token distributions are less aligned. Consequently, the historical probability deviates more from the




**(a)** Token retrieval latency with increasing thread concurrency.

**(b)** Aggregated throughput for KV-cache restoration.

**Figure 11.** Performance of Trident's remote storage layer across deployment configurations.

tokens favored by the current prefix, increasing per-token KL divergence. Nevertheless, the divergence remains low among STS-B because OMH verification and the semantic-loss threshold reject stronger mismatches, while recursive checks stop reuse after divergence.

Trident distinguishes genuine reuse opportunities from adversarial lexical lookalikes across all five datasets in Figure 10. OMH-enabled LSH places 96.80% of truly similar pairs in the same bucket on average, compared with only 6.00% of adversarial high-similarity pairs. The OMH-verified LSH mechanism thus preserves high recall for reusable contexts while rejecting false matches from adversarial pairs. Figure 10 also shows that truly similar pairs achieve an average 1.15× TBT speedup over adversarial pairs. OMH-verified LSH identifies these genuine matches, enabling Probability Reuse to generate a fresh draft tree while substituting historical target probabilities for costly target-model verification. Both experiments show that Trident routes most adversarial pairs to SD-based Recomputation, which preserves quality but increases TBT by retaining target-model computation.

### 4.5 Storage Analysis

We evaluate whether Trident's disaggregated storage layer can sustain similarity-driven context reuse without turning communication into the new decoding bottleneck. Compact-context retrieval measures the latency of fetching several-KB historical token sequences, while KV-cache retrieval measures the bandwidth available for restoring several-GB cache tensors. The evaluation covers four compute-storage configurations, 1c-1s, 1c-2s, 2c-1s, and 2c-2s, where c and s denote compute and storage nodes.

For compact context retrieval, we vary the number of threads from 1 to 16 and report the average latency in Figure 11a. The average retrieval time rises from 2.65 $\mu$s to 2.85 $\mu$s across the evaluated topologies. This stability indicates that Trident's remote lookup path scales with concurrent requests and avoids per-link contention. OMH-verified LSH confines each lookup to a small candidate bucket, while the RDMA transport moves the selected context without CPU-mediated copying. The retrieval cost therefore stays low enough to be overlapped with local execution.

For KV-cache retrieval, Figure 11b shows that a single thread achieves 191 Gbps, and 16 threads sustain about 195 Gbps in aggregate, close to the 200 Gbps InfiniBand bandwidth. This behavior is consistent across all configurations, indicating that large KV-cache restoration remains efficient as compute and storage nodes scale out. GDR-based data placement avoids unnecessary host copies, read-only pipelined retrieval overlaps transfer with restoration, and asynchronous write-back keeps newly generated context insertion outside the critical path. Overall, Trident's remote storage layer delivers both compact and large historical contexts with sufficient efficiency for its decoding pipeline.

## 5 Discussion

Trident's cross-request reuse introduces potential quality degradation under imperfect semantic matches. OMH-verified LSH retrieves prefixes with similar token content, but this similarity does not guarantee semantic consistency. Probability Reuse may therefore apply mismatched historical target probabilities. Trident limits this by admitting historical probabilities only when similarity satisfies the semantic-loss bound and switching to SD-based Recomputation when similarity falls below the threshold.

Multi-tenant serving introduces a privacy risk because shared data stores may retrieve contexts derived from another tenant's request. Both Sequence Reuse and SD-based Recomputation preserve lossless generation because their drafts are verified by the target model. Therefore, they reject unsupported drafts, preventing retrieved content from altering output distributions. Probability Reuse is privacy-sensitive because it uses historical probabilities that can influence outputs. Multi-tenant deployments should therefore isolate Probability Reuse indexes by tenant.

## 6 Conclusion

We present Trident, a similarity-driven context-reuse framework for accelerating tree-based speculative decoding. Trident retrieves order-aware historical matches with OMH-verified LSH, applies quality-aware Jaccard thresholds, and routes execution through Sequence Reuse, Probability Reuse, or SD-based Recomputation. Its RDMA-backed storage layer decouples historical-context capacity from compute-node memory and hides retrieval through asynchronous restoration. Across the evaluated models and workloads, Trident reduces average TBT by 69.2% over autoregressive serving and outperforms training-free SD frameworks by 30.3%.

## 7 Acknowledgement

We thank our shepherd and the anonymous reviewers for their feedback. This work is supported by the National Key Research and Development Program of China (Grant No. 2024YFB4505203), the National Natural Science Foundation of China (Grant Nos. 92582112, 62227809, 62332012, and 62302290), the CCF-Huawei Populus Grove Fund, and the Fundamental Research Funds for the Central Universities.

## References


[1] Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. 2024. Hydra: Sequentially-Dependent Draft Heads for Medusa Decoding. In *First Conference on Language Modeling*. https://openreview.net/forum?id=FbhjirzvJG

[2] Anthropic. 2024. Prompt Caching for Claude. https://docs.anthropic.com/en/docs/build-with-claude/prompt-caching. Accessed: 2026-04-28.

[3] Fu Bang. 2023. GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, Liling Tan, Dmitrijs Milajevs, Geeticka Chauhan, Jeremy Gwinnup, and Elijah Rippeth (Eds.). Association for Computational Linguistics, Singapore, 212–218. doi:10.18653/v1/2023.nlposs-1.24

[4] Harsh Vardhan Bansal. 2025. LLMCache: Layer-Wise Caching Strategies for Accelerated Reuse in Transformer Inference. In *2025 13th International Conference on Intelligent Systems and Embedded Design (ISED)*. IEEE, 924–929. doi:10.1109/ised67359.2025.11405274

[5] A. Broder. 1997. On the Resemblance and Containment of Documents. In *Proceedings of the Compression and Complexity of Sequences 1997 (SEQUENCES '97)*. IEEE Computer Society, USA, 21.

[6] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. 2000. Min-Wise Independent Permutations. *J. Comput. System Sci.* 60, 3 (2000), 630–659. doi:10.1006/jcss.1999.1690

[7] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL] https://arxiv.org/abs/2005.14165

[8] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. MEDUSA: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 203, 27 pages.

[9] Sarthak Chakraborty, Suman Nath, Xuchao Zhang, Chetan Bansal, and Indranil Gupta. 2025. Generative Caching for Structurally Similar Prompts and Responses. arXiv:2511.17565 [cs.CL] https://arxiv.org/abs/2511.17565

[10] Moses Charikar, Ofir Geri, Michael P. Kim, and William Kuszmaul. 2018. On Estimating Edit Distance: Alignment, Dimension Reduction, and Embeddings. arXiv:1804.09907 [cs.DS] https://arxiv.org/abs/1804.09907

[11] Beidi Chen, Tharun Medini, James Farwell, Sameh Gobriel, Charlie Tai, and Anshumali Shrivastava. 2020. SLIDE : In Defense of Smart Algorithms over Hardware Acceleration for Large-Scale Deep Learning Systems. arXiv:1903.03129 [cs.DC] https://arxiv.org/abs/1903.03129

[12] Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. 2024. SEQUOIA: scalable and robust speculative decoding. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 4116, 33 pages.

[13] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. https://lmsys.org/blog/2023-03-30-vicuna/

[14] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling Language Modeling with Pathways. arXiv:2204.02311 [cs.CL] https://arxiv.org/abs/2204.02311

[15] Robert M. Corless, Gaston H. Gonnet, David E. G. Hare, David J. Jeffrey, and Donald E. Knuth. 1996. On the Lambert W Function. *Advances in Computational Mathematics* 5, 1 (1996), 329–359. doi:10.1007/BF02124750

[16] Pradeep Dasigi, Nelson F. Liu, Ana Marasović, Noah A. Smith, and Matt Gardner. 2019. Quoref: A Reading Comprehension Dataset with Questions Requiring Coreferential Reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 5925–5932.

[17] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry* (Brooklyn, New York, USA) *(SCG '04)*. Association for Computing Machinery, New York, NY, USA, 253–262. doi:10.1145/997817.997857

[18] DeepSeek. 2024. Context Caching on Disk. https://api-docs.deepseek.com/guides/kv_cache. DeepSeek API Documentation. Accessed: 2026-05-09.

[19] Aditya Desai, Shuo Yang, Alejandro Cuadron, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. HashAttention: Semantic Sparsity for Faster Inference. In *Proceedings of the 42nd International Conference on Machine Learning (ICML 2025)*.

[20] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2368–2378.

[21] Kenneth Enevoldsen, Isaac Chung, Imene Kerboua, Márton Kardos, Ashwin Mathur, David Stap, Jay Gala, Wissam Siblini, Dominik Krzemiński, Genta Indra Winata, Saba Sturua, Saiteja Utpala, Mathieu Ciancone, Marion Schaeffer, Gabriel Sequeira, Diganta Misra, Shreeya Dhakal, Jonathan Rystrøm, Roman Solomatin, Ömer Çağatan, Akash Kundu, Martin Bernstorff, Shitao Xiao, Akshita Sukhlecha, Bhavish Pahwa, Rafał Poświata, Kranthi Kiran GV, Shawon Ashraf, Daniel Auras, Björn Plüster, Jan Philipp Harries, Loïc Magne, Isabelle Mohr, Mariya Hendriksen, Dawei Zhu, Hippolyte Gisserot-Boukhlef, Tom Aarsen, Jan Kostkan, Konrad Wojtasik, Taemin Lee, Marek Šuppa, Crystina Zhang, Roberta Rocca, Mohammed Hamdy, Andrianos Michail, John Yang, Manuel Faysse, Aleksei Vatolin, Nandan Thakur, Manan Dey, Dipam Vasani, Pranjal Chitale, Simone Tedeschi, Nguyen Tai, Artem Snegirev, Michael Günther, Mengzhou Xia, Weijia Shi, Xing Han Lù, Jordan Clive, Gayatri Krishnakumar, Anna Maksimova, Silvan Wehrli, Maria Tikhonova, Henil Panchal, Aleksandr Abramov, Malte Ostendorff, Zheng Liu, Simon Clematide, Lester James

Miranda, Alena Fenogenova, Guangyu Song, Ruqiya Bin Safi, Wen-Ding Li, Alessia Borghini, Federico Cassano, Hongjin Su, Jimmy Lin, Howard Yen, Lasse Hansen, Sara Hooker, Chenghao Xiao, Vaibhav Adlakha, Orion Weller, Siva Reddy, and Niklas Muennighoff. 2025. MMTEB: Massive Multilingual Text Embedding Benchmark. *arXiv preprint arXiv:2502.13595* (2025). doi:10.48550/arXiv.2502.13595

[22] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. 2024. Cost-efficient large language model serving for multi-turn conversations with CachedAttention. In *Proceedings of the 2024 USENIX Conference on Usenix Annual Technical Conference* (Santa Clara, CA, USA) *(USENIX ATC'24)*. USENIX Association, USA, Article 7, 16 pages.

[23] Zipeng Gao, Zhi Zheng, Qingrong Xia, Junda Lin, Ziwei Zhao, Tong Xu, Zhefeng Wang, and Enhong Chen. 2026. Unlocking Parallelism in Autoregressive Language Models via Speculative Decoding with Progressive Tree Drafting. arXiv:2607.10661 [cs.CL] https://arxiv.org/abs/2607.10661

[24] In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. Prompt Cache: Modular Attention Reuse for Low-Latency Inference. In *Proceedings of Machine Learning and Systems 6 (MLSys 2024)*. 325–338. https://proceedings.mlsys.org/paper_files/paper/2024/file/a66caa1703fe34705a4368c3014c1966-Paper-Conference.pdf

[25] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 518–529.

[26] Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. 2017. First Quora Dataset Release: Question Pairs. https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs. Accessed: 2026-09.

[27] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1601–1611.

[28] Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y. Fu, Christopher Ré, and Azalia Mirhoseini. 2024. Hydragen: High-Throughput LLM Inference with Shared Prefixes. In *ES-FoMo II: 2nd Workshop on Efficient Systems for Foundation Models at ICML 2024*. https://arxiv.org/abs/2402.05099

[29] Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. 2023. Speculative Decoding with Big Little Decoder. arXiv:2302.07863 [cs.CL] https://arxiv.org/abs/2302.07863

[30] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The Efficient Transformer. arXiv:2001.04451 [cs.LG] https://arxiv.org/abs/2001.04451

[31] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. arXiv:2309.06180 [cs.LG] https://arxiv.org/abs/2309.06180

[32] Johann Heinrich Lambert. 1758. Observationes variae in mathesin puram. *Acta Helvetica, physico-mathematico-anatomico-botanico-medica* 3 (1758), 128–168.

[33] Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. Deduplicating Training Data Makes Language Models Better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*. https://arxiv.org/abs/2107.06499

[34] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 795, 13 pages.

[35] Jiaxing Li, Chi Xu, Feng Wang, Isaac M von Riedemann, Cong Zhang, and Jiangchuan Liu. 2024. SCALM: Towards Semantic Caching for Automated Chat Services with Large Language Models. arXiv:2406.00025 [cs.CL] https://arxiv.org/abs/2406.00025

[36] Ping Li and Arnd Christian Konig. 2009. b-Bit Minwise Hashing. arXiv:0910.3349 [cs.DS] https://arxiv.org/abs/0910.3349

[37] Ping Li, Art Owen, and Cun-hui Zhang. 2012. One Permutation Hashing. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/eaa32c96f620053cf442ad32258076b9-Paper.pdf

[38] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP 2024)*.

[39] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. EAGLE: speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 1162, 14 pages.

[40] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025. EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test. In *Proceedings of the 39th Conference on Neural Information Processing Systems (NeurIPS 2025)*.

[41] Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. 2024. Kangaroo: Lossless Self-Speculative Decoding via Double Early Exiting. arXiv:2404.18911 [cs.CL] https://arxiv.org/abs/2404.18911

[42] Minghui Liu, Tahseen Rabbani, Tony O'Halloran, Ananth Sankaralingam, Mary-Anne Hartley, Furong Huang, Cornelia Fermüller, and Yiannis Aloimonos. 2024. HASHEVICT: A Pre-Attention KV Cache Eviction Strategy Using Locality-Sensitive Hashing. *CoRR* abs/2412.16187 (2024). arXiv:2412.16187 doi:10.48550/ARXIV.2412.16187

[43] Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. 2024. Online Speculative Decoding. arXiv:2310.07177 [cs.AI] https://arxiv.org/abs/2310.07177

[44] Guillaume Marçais, Dan DeBlasio, Prashant Pandey, and Carl Kingsford. 2019. Locality-sensitive hashing for the edit distance. *Bioinformatics* 35, 14 (July 2019), i127–i135. doi:10.1093/bioinformatics/btz354

[45] Philip May. 2021. Machine translated multilingual STS benchmark dataset. https://github.com/PhilipMay/stsb-multi-mt

[46] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2024. SpecInfer: Accelerating Large Language Model Serving with Tree-based Speculative Inference and Verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 932–949. doi:10.1145/3620666.3651335

[47] Microsoft Azure. 2024. How to use prompt caching with Azure OpenAI Service. https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/prompt-caching. Accessed: 2026-04-28.

[48] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. MTEB: Massive Text Embedding Benchmark. *arXiv preprint arXiv:2210.07316* (2022). doi:10.48550/ARXIV.2210.07316

[49] OpenAI. 2024. Prompt Caching in the OpenAI API. https://platform.openai.com/docs/guides/prompt-caching. Accessed: 2026-04-28.

[50] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg

Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[51] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. arXiv:2304.03442 [cs.HC] https://arxiv.org/abs/2304.03442

[52] Zhencan Peng, Zhizhi Wang, and Dong Deng. 2023. Near-Duplicate Sequence Search at Scale for Large Language Model Memorization Evaluation. *Proceedings of the ACM on Management of Data (SIGMOD)* (2023). doi:10.1145/3589324

[53] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2022. Efficiently Scaling Transformer Inference. arXiv:2211.05102 [cs.LG] https://arxiv.org/abs/2211.05102

[54] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2383–2392.

[55] ShareGPT Team. 2023. ShareGPT: Share your wildest ChatGPT conversations. https://sharegpt.com/. Accessed: 2026-04-28.

[56] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 10107–10116.

[57] Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. 2023. SpecTr: fast speculative decoding via optimal transport. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 1314, 21 pages.

[58] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL] https://arxiv.org/abs/2302.13971

[59] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023). https://arxiv.org/abs/2307.09288

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[61] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned Language Models Are Zero-Shot Learners. arXiv:2109.01652 [cs.CL] https://arxiv.org/abs/2109.01652

[62] Yair Weiss, Antonio Torralba, and Rob Fergus. 2008. Spectral hashing. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems* (Vancouver, British Columbia, Canada) *(NIPS'08)*. Curran Associates Inc., Red Hook, NY, USA, 1753–1760.

[63] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. HuggingFace's Transformers: State-of-the-art Natural Language Processing. arXiv:1910.03771 [cs.CL] https://arxiv.org/abs/1910.03771

[64] Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding. *arXiv preprint arXiv:2401.07851* (2024).

[65] Huan Yang, Renji Zhang, Mingzhe Huang, Weijun Wang, Yin Tang, Yuanchun Li, Yunxin Liu, and Deyu Zhang. 2025. KVShare: An LLM Service System with Efficient and Effective Multi-Tenant KV Cache

Reuse. *arXiv preprint arXiv:2503.16525* (2025).

[66] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2369–2380.

[67] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. 2025. CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion. In *Proceedings of the Twentieth European Conference on Computer Systems* (Rotterdam, Netherlands) *(EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 94–109. doi:10.1145/3689031.3696098

[68] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] https://arxiv.org/abs/2210.03629

[69] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 521–538. https://www.usenix.org/conference/osdi22/presentation/yu

[70] Lingfan Yu, Jinkun Lin, and Jinyang Li. 2025. Stateful Large Language Model Serving with Pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*. https://arxiv.org/abs/2312.05516

[71] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating Text Generation with BERT. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SkeHuCVFDr

[72] Yudi Zhang, Weilin Zhao, Xu Han, Tiejun Zhao, Wang Xu, Hailong Cao, and Conghui Zhu. 2025. Speculative Decoding Meets Quantization: Compatibility Evaluation and Hierarchical Framework Design. arXiv:2505.22179 [cs.CL] https://arxiv.org/abs/2505.22179

[73] Xinye Zhao and Spyridon Mastorakis. 2025. SemShareKV: Efficient KVCache Sharing for Semantically Similar Prompts via Token-Level LSH Matching. *arXiv preprint arXiv:2509.24832* (2025).

[74] 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. 2024. SGLang: Efficient Execution of Structured Language Model Programs. arXiv:2312.07104 [cs.AI] https://arxiv.org/abs/2312.07104

[75] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. arXiv:2401.09670 [cs.DC] https://arxiv.org/abs/2401.09670

[76] Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2024. DistillSpec: Improving Speculative Decoding via Knowledge Distillation. arXiv:2310.08461 [cs.CL] https://arxiv.org/abs/2310.08461

[77] Yuxuan Zhu, Ali Falahati, David H. Yang, and Mohammad Mohammadi Amiri. 2025. SentenceKV: Efficient LLM Inference via Sentence-Level Semantic KV Caching. arXiv:2504.00970 [cs.CL] https://arxiv.org/abs/2504.00970