SpecCache: Speculative KV Cache Reuse for Efficient RAG Serving

Zijian Wen[1]*, **Tao Zhang**[1]*, **Shuangwu Chen**[1]†, **Shenghao Ye**[1], **Yu Guo**[1], **Qirui Chen**[1], **Jingxian Shuai**[1], **Yunpeng Hou**[2], **Huasen He**[1], **Jian Yang**[1]
[1]University of Science and Technology of China
[2]Institute of Artificial Intelligence, Hefei Comprehensive National Science Center
{wzj20020304,zhangtaolqy,ssh0321y,yukariguo,chenqirui,vexertron_shuai,hyp314}@mail.ustc.edu.cn
{chensw,hehuasen,jianyang}@ustc.edu.cn

Abstract

Retrieval-Augmented Generation (RAG) significantly enhances LLMs but faces high prefill latency during long-context processing. While KV cache reuse can mitigate this, current methods relying on shallow features or static heuristics often fail to identify critical tokens for recomputation, resulting in generation quality degradation. We have an insight that KV deviations are more pronounced in deep layers. However, directly extracting deep-layer features from the target model is computationally prohibitive. Crucially, we find that the deep-layer features of a lightweight speculative model exhibit strong consistency with the target model in the selection of critical tokens for recomputation. In light of these insights, we propose SpecCache, which employs deep-layer hidden-state norms from a speculative model as a proxy to guide the critical token selection for target large model. Experiments demonstrate that SpecCache outperforms state-of-the-art (SOTA) baselines. Compared to full KV recomputation, it reduces time-to-first-token (TTFT) by $2.17-3.95\times$ and increases inference throughput by $2.7-5.2\times$, with negligible degradation in generation quality relative to full recomputation.

1 Introduction

Retrieval-Augmented Generation (RAG) has emerged as a critical paradigm to enhance Large Language Models (LLMs) (Achiam et al., 2023) by supplementing user queries with extensive text chunks retrieved from a knowledge base. However, processing such long contexts significantly drives up inference latency, as the computational complexity of the attention mechanism scales quadratically with input length, leading to substantially higher time to first token (TTFT). To accelerate RAG serving, reusing the KV caches of the same retrieved texts for different LLM inputs rather than recomputing them becomes a promising solution.

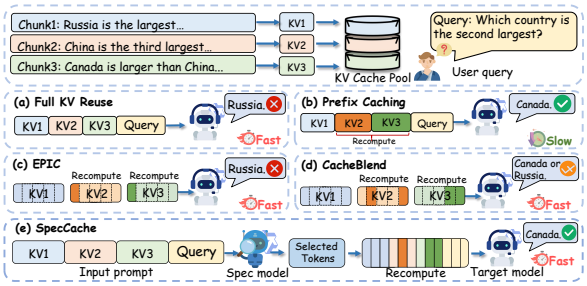

Figure 1: **Comparison of KV cache reuse strategies.** (a) **Full KV Reuse**: reuses all retrieved KV caches directly. (b) **Prefix Caching**: reuses matched prefixes and recomputes suffixes. (c) **EPIC**: recomputes fixed-position tokens of every chunk and reuses the rest. (d) **CacheBlend**: recomputes tokens selected from the first layer and reuses others. (e) **SpecCache (Ours)**: leverages a speculative model to precisely select critical tokens for recomputation.

Despite their promising performance, current KV cache reuse methods present distinct limitations. By directly concatenating all precomputed KV caches of retrieved chunks, Full KV Cache Reuse (Gim et al., 2024) (Figure 1(a)) avoids redundant computation but fails to capture cross-chunk attention (Yao et al., 2025; Hu et al., 2025). Consequently, the precomputed KV cache exhibits substantial divergence from the actual values obtained via full recomputation. Such *KV deviation* leads to severe degradation in generation quality. To make a step forward, Prefix Caching (Zheng et al., 2024; Kwon et al., 2023), which is widely adopted by current systems, reuses the LLM input's prefix (Figure 1(b)) by matching the longest common prefix. However, in RAG scenarios involving multiple text chunks, only the first chunk can serve as the prefix. By contrast, the KV caches for all other chunks remain unusable, resulting in low reuse rates and limited inference acceleration.

To overcome these limitations, recent re-

*Equal contribution.
†Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 18861–18871

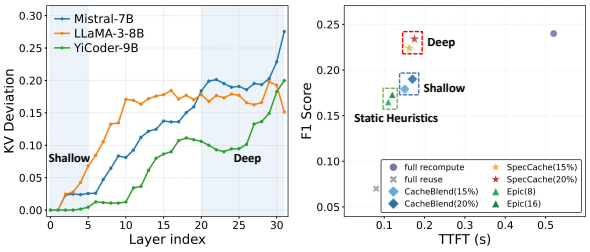

Figure 2: **Layer-wise KV Deviations.** Left: Shallow layers fail to capture the drift of KV deviations, while deeper layers show high sensitivity. Right: SpecCache (deep guidance) significantly outperforms methods (CacheBlend) relying on shallow-layer features or static heuristics (EPIC) on Llama-3-8B evaluated on the MuSiQue dataset.

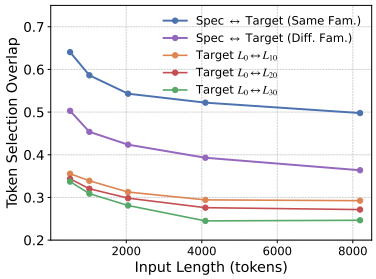

Figure 3: **Token Selection Overlap.** Jaccard similarity of the top-20% important tokens on 2WikiMQA (Spec: Llama3-1B, Qwen2.5-1.5B; Target: Llama3-8B). Spec ↔ Target measures cross-model overlap (same/different family), and Target $L_0 \leftrightarrow L_{10/20/30}$ reports the Shallow–Deep intra-model baseline, measuring token-selection overlap between layer 0 and deeper layers of target model (L_k denotes layer k).

searches (Yao et al., 2025; Hu et al., 2025; Yang et al., 2025a,b; Agarwal et al., 2025) have pivoted towards selective KV cache reuse. They selectively recompute the KV values for critical tokens while reusing the precomputed KV cache for remaining tokens, achieving a favorable trade-off between generation quality and inference latency. EPIC (Hu et al., 2025) adopts a static selection policy, that designates the first few tokens of each chunk as critical (Figure 1(c)). Although EPIC is computationally efficient, its generation quality during inference is limited because critical tokens are not always confined to the beginning of each chunk. In contrast, CacheBlend (Yao et al., 2025) (Figure 1(d)) employs a dynamic selection policy that recomputes only those tokens exhibiting large KV deviations in the first layer. However, shallow layers, which mainly capture syntactic or surface features, convey less semantic content than deeper layers (Chuang et al., 2023). This raises a fundamental concern: ***can shallow-layer accurately identify truly critical tokens?***

To investigate this concern, we analyze the layer-wise KV deviation by measuring the cosine distance between the precomputed KV cache and the full-recomputed KV cache at each layer. As revealed in Figure 2 (left), the KV deviations across diverse models are often subtle at shallow layers but become salient at depth. In light of this, we use the deep-layer features for reliable KV deviation detection. As shown in Figure 2 (Right), our deep-guided approach significantly outperforms shallow-guided methods such as CacheBlend.

Directly extracting the deep-layer features from the large target model for token selection is computationally prohibitive, as it incurs substantial latency that negates the benefits of KV cache reuse. Drawing inspiration from speculative decoding (Xia et al., 2023; Chen et al., 2023), we observe a strong alignment between the deep semantic features of the lightweight speculative model and the large target model. To quantify this, Figure 3 presents the token selection overlap of the top-20% tokens selected by importance scores estimation (detailed in Sec 2.2). Since token importance is derived from these features, this overlap serves as a proxy for semantic alignment. The low Shallow–Deep baseline indicates that shallow-layer token selection fails to represent deep-layer preferences. In contrast, the speculative model achieves higher overlap. Notably, a cross-family speculative model still outperforms the Shallow–Deep baseline, confirming that the speculative model serves as a proxy for the target model to identify important tokens.

Motivated by these insights, we propose SpecCache, a novel selective KV cache reuse framework for efficient RAG serving. SpecCache leverages the aggregated hidden-state norms from deep layers of a speculative model to guide the target model's token selection for recomputation. SpecCache further employs smoothness-based span-level token selection to mitigate the fragmentation of coherent multi-tokens caused by independent token selection. Finally, SpecCache incorporates a cross-tokenizer alignment mechanism to map the selected tokens from the speculative model to the target model, ensuring correct KV cache correspondence. We implement SpecCache on vLLM (Kwon et al., 2023) and evaluate it against state-of-the-art (SOTA) reuse schemes such as CacheBlend and

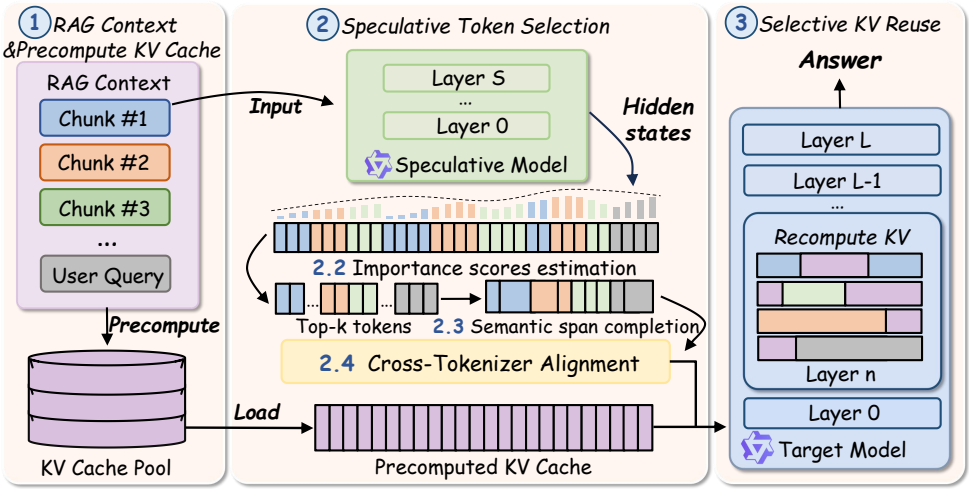

Figure 4: **Overview of the SpecCache Framework.** The pipeline of speculative token selection consists of: (1) Importance Score Estimation to identify critical tokens; (2) Semantic Span Completion to preserve coherent multi-token spans; and (3) Cross-Tokenizer Alignment to accurately map selections to the target model.

EPIC across six tasks and three model architectures. SpecCache improves generation quality, surpassing CacheBlend by 2.1–23.2% and EPIC by 3.4–30.08% in task scores. Meanwhile, compared to full recompute, SpecCache improves TTFT by 2.17–3.95× and boosts throughput by 2.7–5.2×.

Our Contributions. (1) *New Insight.* We identify that deep-layer features are highly sensitive to KV deviations. Furthermore, the strong deep-layer alignment between lightweight speculative and target models enables the former to effectively guide important token selection. (2) *Novel Framework.* We propose SpecCache, a novel train-free framework that leverages deep-layer features from a lightweight speculative model to guide selective KV recomputation. (3) *SOTA Performance.* We conduct extensive experiments on multiple tasks and model architectures, showing that SpecCache consistently achieves SOTA performance and accelerates RAG serving.

2 SpecCache

2.1 Overview of SpecCache

We develop SpecCache to improve selective KV cache reuse in RAG serving. Figure 4 illustrates the workflow of SpecCache, which consists of three stages: (1) retrieving relevant chunks and loading their precomputed KV caches; (2) using a lightweight speculative model to estimate token importance and select tokens for recomputation; and (3) executing selective KV reuse in the target model based on the selected tokens.

RAG Context & Precompute KV Cache. SpecCache initially retrieves chunks relevant to the user query, whose KV caches have been precomputed offline and stored in a KV Cache Pool. SpecCache then fetches these corresponding caches while feeding the full assembled input (combining retrieved chunks and the query) into the speculative model for speculative token selection.

Speculative Token Selection. Given the assembled input, the speculative model identifies the set of tokens requiring recomputation through a three-step pipeline: (1) Importance Score Estimation (detailed in Sec 2.2) based on deep-layer hidden-state norms; (2) Semantic Span Completion (detailed in Sec 2.3) to reduce fragmentation of multi-token units; and (3) Cross-Tokenizer Alignment (detailed in Sec 2.4) maps the selected tokens from the speculative model to the target model's tokenizer space, ultimately yielding the final recomputation token set.

Selective KV Reuse. Given the recomputation tokens and precomputed KV cache, the target model performs a selective KV cache reuse during the prefill phase. First, SpecCache performs positional recovery (Yao et al., 2025), ensuring that the precomputed KV caches correspond correctly to their absolute positions in the assembled input. The target model then recomputes the KV caches for the selected tokens and fuses them with the precomputed cache, thereby completing the selective KV cache reuse process.

2.2 Importance Scores Estimation

To identify the critical tokens during the selective KV cache reuse, we need to measure the importance score of each token. We derive token importance score from the hidden states of the speculative model. Let $X^s = [x_1^s, \ldots, x_N^s]$ denote the input sequence formed by concatenating the retrieved chunks with the user query, where N is the number of tokens under the speculative model tokenizer. Feeding X^s into an S-layer speculative model yields a sequence of hidden states $\{h_i^{(\ell)} \in \mathbb{R}^d\}$ across the range of deep layers $\ell \in \{D, \ldots, S\}$, where D denotes the starting index of the deep layers. In this work, we define the deep layers as the last one-third of the model. We then compute the importance score $\mathcal{I} = \{I_1, \ldots, I_N\}$, where each I_i corresponding to the token x_i^s is calculated as the averaged ℓ_2-norm of its hidden states across deep layers:

$$I_i = \frac{1}{S - D + 1} \sum_{\ell=D}^{S} \left\| h_i^{(\ell)} \right\|_2, i = 1, \ldots, N. \quad (1)$$

This formulation is grounded in a magnitude-based view of token influence. Since value vectors are linear projections of hidden states, larger hidden-state norms inherently induce larger-magnitude value vectors. Given that attention outputs are formed by a weighted mixture of value vectors, the magnitude of these vectors directly scales their contribution to the final output (Kobayashi et al., 2020). Recent evidence in KV cache reduction further suggests that value-vector norms capture critical importance signals (Guo et al., 2024). Consequently, we treat large-norm tokens as carrying higher reuse-induced perturbation risk, as KV deviation in these tokens is magnified in the final attention output. We specifically restrict this aggregation to deep layers, as our insight reveals their unique sensitivity to KV deviations. Additionally, the strong alignment observed in (Figure 3) enables the lightweight speculative model to serve as a reliable proxy for selecting important tokens. Given a global recomputation ratio $\rho \in (0, 1)$, we select the initial TopK ($K = \lceil \rho N \rceil$) tokens with the highest I_i values for recomputation, while the remaining tokens retrieve their precomputed KV caches. This selection process is formulated as follows:

$$T_{\text{init}}^s = \{x_i^s \in X^s \mid i \in \text{TopK}(\mathcal{I}, K)\}. \quad (2)$$

Subsequently, the initial token set T_{init}^s is refined via the Semantic Span Completion process.

2.3 Semantic Span Completion

The initial token set T_{init}^s is obtained by ranking tokens independently, which can break multi-token semantic units. For instance, a TopK policy may select *"New"* and *"City"* but miss *"York"*, resulting in a partially refreshed entity and degraded semantics under KV reuse. We therefore apply a lightweight post-processing step that (i) segments tokens into locally smooth spans based on the importance scores, and (ii) compensates a span by filling in its remaining unselected tokens when it is already sufficiently covered by T_{init}^s.

Span Segmentation. Token-level importance scores can be noisy, and their absolute scale may vary across prompts and datasets. To robustly identify the boundaries of multi-token semantic units, we operate on a normalized signal. Specifically, we normalize token importance scores to be comparable across inputs:

$$\tilde{I}_i = \frac{I_i - \min I_k}{\max I_k - \min I_k}. \quad (3)$$

We then identify span boundaries to partition the sequence based on the relative variation between adjacent tokens, formulated as follows:

$$\delta_i = \frac{\left| \tilde{I}_{i+1} - \tilde{I}_i \right|}{\tilde{I}_i}, \quad i = 1, \ldots, N - 1. \quad (4)$$

Based on the relative variation, we merge consecutive tokens into a span as long as $\delta_i < \tau_{\text{smooth}}$, producing spans $\mathcal{P} = \{p_1, \ldots, p_J\}$. Each span $p_j = [u_j, v_j]$ is an index interval, where u_j and v_j denote the start and end token indices. We limit the maximum span length to 16 tokens to prevent overly long merges.

Coverage-Based Completion. Following segmentation, not every span warrants completion. Completing all spans would excessively inflate the recomputation ratio, thereby undermining the efficiency gains derived from KV cache reuse. Therefore, we employ the important token coverage ratio as a simple criterion to guide selective span completion. For each span p_j, we compute the important token coverage ratio as follows:

$$r_j = \frac{\left| T_{\text{init}}^s \cap [u_j, v_j] \right|}{v_j - u_j + 1}. \quad (5)$$

If the coverage ratio exceeds the threshold (i.e., $r_j \geq \tau_{\text{cover}}$), we add the entire span to the recomputation set. The refined selection can be formulated

as follows:

$$T^s = T^s_{\text{init}} \cup \bigcup_{r_j \geq \tau_{\text{cover}}} p_j. \tag{6}$$

We set $\tau_{\text{smooth}} = 0.4$ and $\tau_{\text{cover}} = 0.7$. Hyperparameter sensitivity for $(\tau_{\text{smooth}}, \tau_{\text{cover}})$ is reported in Appendix A. To avoid over-expansion, we apply a global budget control after span completion. If the expanded set exceeds a maximum size, we keep only the highest-importance tokens under the same I_i ranking and discard the rest. This prevents excessive recomputation growth while preserving the most salient updates. Overall, this refinement ensures that important entities and phrases are recomputed rather than as fragmented token subsets, thereby reducing KV deviations under KV reuse.

2.4 Cross-Tokenizer Alignment

In practical deployments, the speculative and target models often use different tokenizers (Timor et al., 2025). As a result, a token selected by the speculative model may correspond to multiple sub-tokens under the target tokenizer. To apply selected tokens from the speculative model, we establish a positional mapping between the speculative and target models by aligning their token sequences via the shared original input. Let $X^t = [x^t_1, \ldots, x^t_M]$ denote the token sequence of the target model. We define an alignment mapping Ψ, where $\Psi(x^s_i) \subseteq \{x^t_1, \ldots, x^t_M\}$ represents the set of target token indices corresponding to the selected token x^s_i. For example, if x^s_i is split into two target sub-tokens x^t_j and x^t_{j+1}, then $\Psi(x^s_i) = \{x^t_j, x^t_{j+1}\}$. Given the selection T^s, we convert it to the target index by expanding each selected token to all of its aligned target sub-tokens. The resulting recomputation set can be formulated as:

$$T \triangleq \bigcup_{x^s_i \in T^s} \Psi(x^s_i). \tag{7}$$

This alignment prevents partial updates by ensuring that once a token is selected, all corresponding target sub-tokens representing the same text segment are recomputed, thereby maintaining strict consistency with the target tokenizer. During target model prefill, we then recompute KV caches exactly for indices in T, ensuring correct recomputation even when the two vocabularies differ.

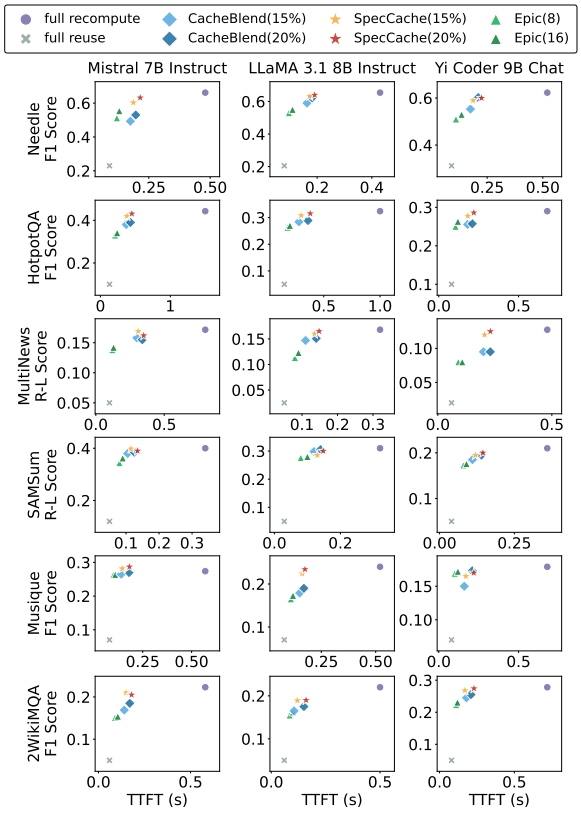

Figure 5: **Latency vs. Quality.** The suffix of EPIC denotes the number of recomputed tokens per chunk, whereas for SpecCache and CacheBlend, it indicates the recomputation ratio.

3 Experiments

3.1 Experimental Setup

Models. To evaluate SpecCache across diverse generation scenarios, we employ **LLaMA3-1B-Instruct** (Meta, 2024) as the speculative model to efficiently estimate token importance and guide recomputation. For the target models, we select a diverse set of models served via vLLM 0.4.1 (Kwon et al., 2023), including **Mistral-7B-Instruct-v0.2** (Jiang et al., 2023), **LLaMA3-8B-Instruct** (Dubey et al., 2024), and **YiCoder-9B-Chat** (Young et al., 2024).

Datasets. We evaluate SpecCache on five datasets from LongBench (Bai et al., 2024): MuSiQue, HotpotQA, 2WikiMQA (multi-hop QA), MultiNews, and SAMSum (summarization). We construct long-context prompts by concatenating retrieved passages with the query. Additionally, we employ Needle-in-a-Haystack (NiaH) (Briakou et al., 2023) to assess long-context retrieval capabilities across different prefill lengths. To emulate KV cache reuse in RAG, we split documents into 512-

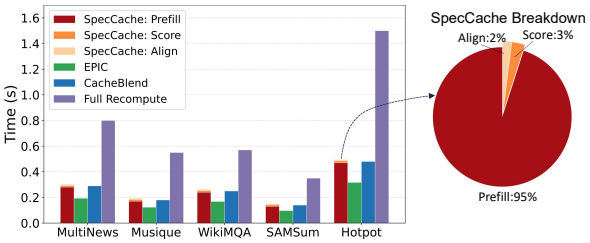

Figure 6: TTFT comparison across different methods.

token chunks using the target model tokenizer. For each dataset, we employ 60–200 test cases and precompute the KV caches for every chunk to evaluate the efficacy of cache reuse.

Metrics. We report TTFT for latency and F1 (QA) (Bai et al., 2024) / ROUGE-L (summarization) (Lin, 2004) for answer quality. We further employ a quality-aware metric, goodput (Zhong et al., 2024), adapted to our setting. While originally defined for latency adherence, we redefine goodput to consider a request successful only if it meets a dataset-specific generation quality threshold. This metric effectively captures the trade-off between latency and generation quality.

Baselines. We compare SpecCache with four baselines: full recompute as the standard prefill; full reuse (Gim et al., 2024) for direct cache reuse; CacheBlend (Yao et al., 2025) for dynamic reuse via shallow-layer features; and EPIC (Hu et al., 2025) for static selective KV reuse of a fixed-length prefix per chunk. Results represent the mean of three independent runs.

Environment. Experiments are performed on a server featuring 2× NVIDIA A100 (40GB) GPUs, an Intel Xeon Gold 5218 CPU, and 1 TB DRAM. The software environment includes Ubuntu 22.04, NVIDIA driver 535.274.02, and CUDA 12.2.

3.2 Result Analysis

We evaluate SpecCache against full recompute and representative KV cache reuse baselines. Figure 5 plots TTFT against task quality (F1 or ROUGE-L), where each marker corresponds to a baseline. Full recompute attains the highest quality but also the largest TTFT, whereas full reuse minimizes TTFT at the expense of substantial quality degradation. In contrast, SpecCache consistently stays in the high-accuracy region and improves the latency–quality frontier. With a 15% recomputation ratio, it recovers most of the full recompute quality while reducing TTFT by 2.17–3.95×, and at 20% it matches the full recompute scores while still retaining a substantial latency advantage.

Comparison of Selective KV Reuse Methods. SpecCache incurs a modest TTFT increase due to the overhead of importance-scoring, but it generally yields higher generation quality than EPIC and CacheBlend across evaluated tasks. Specifically, under the 15% recomputation ratio, SpecCache achieves significantly higher scores than CacheBlend and EPIC. Although this comes at the cost of slightly higher latency, the substantial gain in task performance indicates that SpecCache more effectively utilizes the recomputation ratio to select truly important tokens. With a 20% recomputation ratio, SpecCache continues to outperform CacheBlend in generation quality. In summary, SpecCache prioritizes generation accuracy: it serves as a high-fidelity alternative that approaches the quality of full recomputation much more closely than other approximation methods, while still offering significant speedups over the full recompute baseline.

Why Score Is Not Strictly Monotonic? In selective recomputation, task quality is not guaranteed to increase monotonically with the recomputation ratio. As the ratio grows, the recomputed set gradually expands from the most confidently identified important tokens to lower-ranked ones, whose importance estimates are noisier and whose marginal contribution is often limited. Consequently, additional recomputation may not correct the most critical KV deviations. It may also introduce more boundaries between reused and recomputed tokens, which leads to more fragmented recomputation spans and amplifies small alignment errors. This effect is especially noticeable on MuSiQue, where multi-hop reasoning is highly sensitive to small perturbations and F1 can change discretely under minor output differences.

Why SpecCache Achieves High Quality? Unlike baselines that rely on shallow-layer based selecting (CacheBlend) or static heuristics (EPIC), SpecCache leverages deep-layer features from the speculative model to detect KV deviations. This allows it to selectively recompute the influential tokens crucial for KV cache recomputation, effectively breaking the trade-off between the high latency of full KV cache recomputation and the quality degradation of full KV cache reuse.

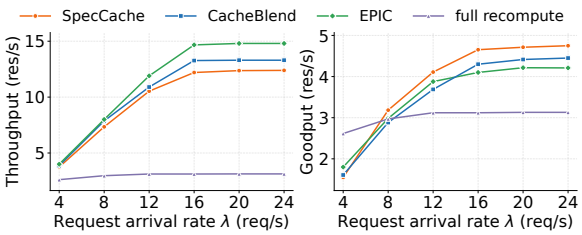

Figure 7: Throughput and goodput across methods.

Methods	Ratio	SAMSum	MuSiQue	HotpotQA
Full recompute	100%	0.327	0.243	0.334
w/o. Span	15%	0.291	0.212	0.294
w/. Span	15%	**0.295** ↑1.3%	**0.224** ↑5.66%	**0.308** ↑4.76%
w/o. Span	20%	0.292	0.208	0.306
w/. Span	20%	**0.306** ↑4.79%	**0.234** ↑12.50%	0.301 ↓1.63%
w/o. Span	25%	0.304	0.214	0.302
w/. Span	25%	**0.313** ↑2.96%	**0.236** ↑10.28%	**0.319** ↑5.63%

Table 1: Ablation of Semantic Span Completion in SpecCache on LLaMA3-8B.

3.3 Overheads and Goodput

Overheads Evaluation. We evaluate the runtime overhead of SpecCache and compare it with representative baselines with Mistral-7B. Figure 6 decomposes the TTFT of SpecCache (15%) into three parts: importance-scoring time (*Score*), cross-tokenizer alignment time (*Align*), and target model prefill time (*Prefill*), and compares its overall latency with EPIC (16), CacheBlend (15%) and full recompute. Across all datasets, the dominant cost of SpecCache is still *Prefill*, whereas *Score* and *Align* contribute only a small overhead (about 5%) relative to *Prefill*. Moreover, in our implementation, both *Score* and *Align* are executed while the framework streams precomputed KV caches from host memory into GPU memory; since these operations are independent, their costs can be hidden behind KV loading and thus have little visible impact on TTFT. EPIC attains lowest TTFT because it avoids token selection and adopts a static policy that recomputes only a fixed subset of tokens. This design reduces latency, but it also weakens token selection fidelity and tends to incur larger quality degradation compared to approaches that adaptively identify tokens. As a result, SpecCache achieves TTFT that is close to CacheBlend yet remains 2.3–3.7× faster than full recomputation. This confirms that using a lightweight speculative model to obtain importance scores does not introduce a prohibitive overhead; instead, it adds only a modest cost, allowing SpecCache to retain most of the latency advantages of KV reuse while benefiting from more informed token selection.

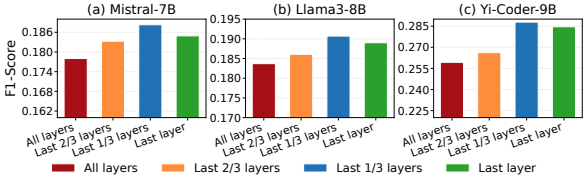

Figure 8: Impact of selected layers across models.

Goodput Evaluation. To assess performance under varying loads, we evaluate SpecCache using a quality-aware goodput metric. We simulate an asynchronous serving system where requests arrive via a *Poisson process* with arrival rate λ. We measure response quality using task score and evaluate it relative to the full recompute baseline. A response is deemed valid if its score exceeds a threshold γ. We set γ to 80% of the mean score achieved by the full recompute baseline. This threshold serves as a quality filter, allowing for minor variations while ensuring that the retrieved context is effectively utilized. Adapted from DistServe (Zhong et al., 2024), which defines goodput based on latency SLO compliance, we propose goodput to quantify the throughput of valid responses:

$$\text{Goodput}(\gamma, Q) = \frac{1}{H} \sum_{q=1}^{Q} \mathbb{I}(\text{F1}_q \geq \gamma)\,. \quad (8)$$

where H is the total simulated duration to process Q requests, and $\mathbb{I}(\cdot)$ is the indicator function. This metric distinguishes meaningful acceleration (usable outputs) from raw processing speed. We compare SpecCache (15%) with CacheBlend (15%) and Epic (32) on the MuSiQue dataset with Llama3-8B. As illustrated in Figure 7, SpecCache achieves superior goodput despite exhibiting lower throughput than CacheBlend and EPIC. While these baselines attain higher throughput, their gains are effectively offset by significant quality degradation. Consequently, SpecCache strikes a superior balance, maintaining high generation quality while sustaining high throughput at scale.

3.4 Ablation and Sensitivity Analysis.

We examine the impact of configuration variations on overall performance of SpecCache.

Ablation on Semantic Span Completion. Table 1 compares SpecCache with and without the proposed Semantic Span Completion module under different recomputation ratios. In nearly all settings across the three datasets, *w/. Span* outperforms the

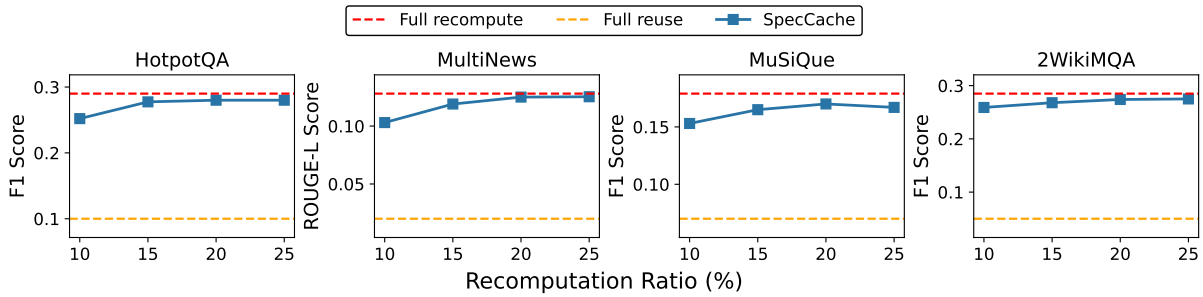

Figure 9: Impact of different recompute ratios with Yi-coder-9B.

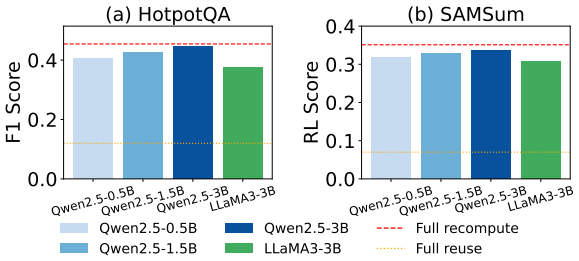

Figure 10: Impact of speculative model's family and scale.

w/o. Span variant. These results indicate that Semantic Span Completion effectively improves task performance by better preserving semantic coherence under a fixed recomputation budget.

Impact of selected layers. We evaluate the sensitivity of importance scoring to layer selection by aggregating signals over different layer subsets (all layers; the last 2/3; the last 1/3; or only the final layer). As illustrated in Figure 8, prioritizing deeper layers consistently enhances F1 scores across models, with the last 1/3 layers achieving the best overall performance. This corroborates our insight that deep-layer features capture KV deviations more effectively, thereby providing a more reliable proxy for selective KV reuse.

Impact of Recompute Ratios. As shown in Figure 9, increasing the recompute ratio from 10% to 25% yields a smooth improvement in scores across all datasets. Even at low ratios, SpecCache already recovers most of the full recompute quality while substantially reducing TTFT, and around 20–25% recomputation it achieves performance parity with the full recompute baseline, yet still with 2.17× TTFT reduction compared.

Impact of Model Families and Scales. We further investigate the impact of different speculative model families and scales. Figure 10 reports HotpotQA F1 score when we fix the target model as Qwen2.5-14B (Team, 2024) and vary the speculative model. Scaling the in-family Qwen2.5 speculative model (0.5B–3B) consistently improves task score, suggesting that larger in-family speculative models yield more accurate importance estimates. In contrast, switching to a different model family with a comparable parameter scale (LLaMA3-3B (Dubey et al., 2024)) yields noticeably lower score than Qwen2.5-3B and is even comparable to the much smaller Qwen2.5-0.5B. Overall, these results indicate that family-aligned speculative models enable more reliable selection of important tokens for recomputation.

Case Study. We present a representative example from the MuSiQue dataset to illustrate the advantage of SpecCache over KV-reuse baselines. The question is: *"What record label did the person who is part of The Bruce Lee Band start?"*, and the gold answer is *"Asian Man Records"*. Correctly answering this question requires linking two retrieved pieces of evidence: (i) *"The Bruce Lee Band ... releases of Mike Park ..."* and (ii) *"Asian Man Records is a DIY record label run by Mike Park ..."*. The key challenge is to recover the bridge entity *Mike Park* across the two chunks and establish the cross-chunk evidence chain. To analyze the behavior of different methods, we inspect the tokens and spans selected for recomputation by each baseline. Under KV reuse, both full reuse and EPIC fail to recompute spans containing the bridge entity or the correct label, leading to incorrect predictions such as *"RSA Records"* and *"Drill Records"*. CacheBlend also fails because it misses the critical bridge tokens, including *"Mike Park"* and *"The Bruce Lee Band"*, preventing the model from recovering the necessary cross-chunk connection and causing it to fall back on an incorrect prior. In contrast, SpecCache successfully selects and recomputes the deviation-sensitive bridge spans, restores the missing cross-chunk linkage, and produces the correct answer.

4 Related Work

Retrieval-Augmented Generation (RAG). RAG (Mao et al., 2021; Li et al., 2022; Gao et al., 2023; Jeong et al., 2024) enhances LLM generation by incorporating relevant context retrieved from an external corpus into the input prompt. While this mechanism improves factuality by grounding answers in evidence, it introduces a significant latency bottleneck. The retrieved chunks substantially expand the input sequence length, increasing the computational cost of the prefill phase and delaying the TTFT.

KV Cache Reuse. Precomputing and reusing KV caches has been widely studied as a systems mechanism to amortize prefill cost and reduce TTFT (Gim et al., 2024; Jin et al., 2025; Liu et al., 2024). Most prior efforts focus on prefix caching, where reuse is possible only when prompts share an identical prefix, which substantially limits its utility for RAG with highly dynamic retrieved contexts. More recent selective reuse methods, such as CacheBlend (Yao et al., 2025) and EPIC (Hu et al., 2025), move toward position-independent chunk reuse; however, their token-selection strategies rely on shallow-layer signals or static, position-based heuristics, and thus often fail to identify the tokens that most need recomputation.

Speculative Inference. Speculative inference typically uses lightweight models to accelerate decoding via draft verification or to compress prompts via token pruning (Leviathan et al., 2023; Xia et al., 2023; Liu et al., 2025; Li et al., 2024). While these methods demonstrate the value of auxiliary guidance, they focus on next-token prediction or token compression. SpecCache uniquely repurposes a speculative model to guide selective KV reuse, recovering critical cross-chunk attention in RAG without retraining or tuning.

5 Conclusion

In this paper, we present SpecCache, a novel selective KV cache reuse framework designed to accelerate RAG serving without compromising generation quality. Driven by the insight that deep-layer features are uniquely sensitive to KV deviations, SpecCache leverages a lightweight speculative model to accurately identify critical tokens. Experiments demonstrate that SpecCache significantly reduces TTFT and improves throughput over full recomputation, while consistently outperforming SOTA baselines in accuracy.

Limitations

One limitation of our current work lies in the use of generic, off-the-shelf speculative models. While effective, we have not yet explored fine-tuning the speculator to explicitly mimic the target model's attention patterns, which could potentially unlock higher alignment precision. Furthermore, our evaluation is currently concentrated on models in the 7B–13B parameter range and standard academic benchmarks. Future work is needed to verify the scalability of SpecCache on significantly larger architectures (e.g., 70B parameters) and to assess its robustness across a broader spectrum of complex, domain-specific datasets.

Ethics Statement

All datasets used in this study are publicly available and released by prior peer-reviewed work. SpecCache is a framework for accelerating long-context inference via KV cache reuse and selective recomputation; it does not introduce new safeguards against unsafe generations. We therefore encourage practitioners to apply standard safety measures and to critically verify model outputs before downstream consumption. Because cache reuse is designed for efficiency, deployments in multi-tenant settings should additionally ensure strict isolation between users and requests. In particular, implementations should prevent cross-request leakage of cached activations. Our study leverages open-source model families (e.g., Mistral, Llama, Qwen, and Yi) and associated toolchains. We adhere to their respective licenses and usage policies, and we acknowledge the contributions of these communities and prior benchmark authors that make this research possible.

Acknowledgements

This work is supported by the National Key R&D Program of China, No. 2024YDLN0004 and the National Natural Science Foundation of China (U23A20275).

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Shubham Agarwal, Sai Sundaresan, Subrata Mitra, Debabrata Mahapatra, Archit Gupta, Rounak Sharma, Nirmal Joshua Kapu, Tong Yu, and Shiv Saini. 2025. Cache-craft: Managing chunk-caches for efficient retrieval-augmented generation. *Proceedings of the ACM on Management of Data*, 3(3):1–28.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 3119–3137.

Eleftheria Briakou, Colin Cherry, and George Foster. 2023. Searching for needles in a haystack: On the role of incidental bilingualism in palm's translation capability. In *The 61st Annual Meeting Of The Association For Computational Linguistics*.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James R Glass, and Pengcheng He. 2023. Dola: Decoding by contrasting layers improves factuality in large language models. In *The Twelfth International Conference on Learning Representations*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1).

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338.

Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. 2024. Attention score is not all you need for token importance indicator in kv cache reduction: Value also matters. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21158–21166.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Hao Feng, Xusheng Chen, Yizhou Shan, Tao Xie, and 1 others. 2025. Epic: Efficient position-independent caching for serving large language models. In *Forty-second International Conference on Machine Learning*.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. 2024. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *NAACL-HLT*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Shufan Liu, Xuanzhe Liu, and Xin Jin. 2025. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *ACM Transactions on Computer Systems*, 44(1):1–27.

Goro Kobayashi, Tatsuki Kuribayashi, Sho Yokoi, and Kentaro Inui. 2020. Attention is not only a weight: Analyzing transformers with vector norms. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. 2022. A survey on retrieval-augmented text generation. *arXiv preprint arXiv:2202.01110*.

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*, pages 28935–28948.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Jingyu Liu, Beidi Chen, and Ce Zhang. 2025. Speculative prefill: Turbocharging ttft with lightweight and training-free token importance estimation. In *Forty-second International Conference on Machine Learning*.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, and 1 others. 2024. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 38–56.

Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. 2021. Generation-augmented retrieval for open-domain question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4089–4100.

Meta. 2024. meta-llama/llama-3.2-1b-instruct. Hugging Face model card. Accessed: 2025-12-20.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Nadav Timor, Jonathan Mamou, Daniel Korat, Moshe Berchansky, Gaurav Jain, Oren Pereg, Moshe Wasserblat, and David Harel. 2025. Accelerating llm inference with lossless speculative decoding algorithms for heterogeneous vocabularies. In *Forty-second International Conference on Machine Learning*.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2023. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3909–3925.

Huan Yang, Renji Zhang, Mingzhe Huang, Weijun Wang, Yin Tang, Yuanchun Li, Yunxin Liu, and Deyu Zhang. 2025a. Kvshare: An llm service system with efficient and effective multi-tenant kv cache reuse. *arXiv preprint arXiv:2503.16525*.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. 2025b. Kvlink: Accelerating large language models via efficient kv cache reuse. *arXiv preprint arXiv:2502.16002*.

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*, pages 94–109.

Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Guoyin Wang, Heng Li, Jiangcheng Zhu, Jianqun Chen, and 1 others. 2024. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, and 1 others. 2024. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583.

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. In *18th USENIX Symposium on Operating Systems Design and Implementation*, pages 193–210.

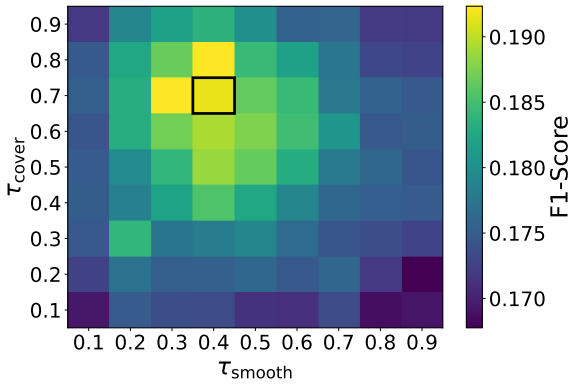

Figure 11: Heatmap of scores for the $(\tau_{\text{smooth}}, \tau_{\text{cover}})$ grid sweep on 2WikiMQA with Llama3-8B.

A Hyperparameter Sensitivity of Span Refinement.

We study the sensitivity of span refinement to two thresholds: τ_{smooth} for span segmentation (merging adjacent tokens when $\delta_i < \tau_{\text{smooth}}$) and τ_{cover} for coverage-based completion (expanding a span $p_j = [u_j, v_j]$ when its important-token coverage $r_j \geq \tau_{\text{cover}}$). We conduct a two-dimensional grid sweep over $(\tau_{\text{smooth}}, \tau_{\text{cover}})$ while holding T^s_{init} and all other components fixed, and evaluate each configuration under a fixed global recomputation budget of 20%. Figure 11 reports the scores over $(\tau_{\text{smooth}}, \tau_{\text{cover}})$, highlighting how segmentation and completion jointly influence span boundaries and semantic coherence.