SLMCoach: Context-Adaptive LM-Head Caching with Cross-Turn Reuse for Efficient Small Language Model Inference

Anonymous authors
Paper under double-blind review

Abstract

Small language models (SLMs) have emerged as a promising solution for privacy-preserving, low-latency artificial intelligence on edge devices. However, SLM inference remains constrained by memory footprint and data-movement overhead, with the vocabulary-dependent language modeling head (LM-head) constituting a substantial bottleneck. Existing vocabulary-reduction methods reduce these costs but typically treat queries independently, overlooking the temporal locality inherent in multi-turn interactions and missing opportunities to reuse LM-head rows across turns. To address this limitation, we formulate the *cross-turn LM-head caching* problem and introduce SLMCoach, a context-adaptive framework for efficient multi-turn SLM inference. By exploiting ***contextual continuity*** across dialogue turns, SLMCoach selectively retains LM-head rows with high reuse potential and incrementally updates a budget-constrained cache, reducing redundant parameter movement across consecutive queries. Empirical evaluations on a multi-turn dialogue benchmark demonstrate that SLMCoach reduces the LM-head footprint by 99.2% and peak GPU memory by 51.5% relative to full-head inference while preserving response quality. At matched response fidelity, SLMCoach further reduces LM-head data movement by 7.5× compared with query-isolated loading. These results highlight the benefit of exploiting cross-turn reuse for resource-constrained SLM inference.

1 Introduction

In recent years, large language models (LLMs) have become a cornerstone of modern artificial intelligence (AI), enabling advanced reasoning and language understanding across applications such as virtual assistants and conversational agents Zhao et al. (2026); Xi et al. (2025). However, their massive parameter counts typically require cloud-centric execution, requiring users to transmit query prompts and personal data to remote servers, which is undesirable in privacy-sensitive and latency-critical settings. Consequently, small language models (SLMs) have gained increasing attention for edge deployment due to their substantially lower computational and memory requirements, enabling low-latency and privacy-preserving AI applications Van Nguyen et al. (2025); Belcak et al. (2025).

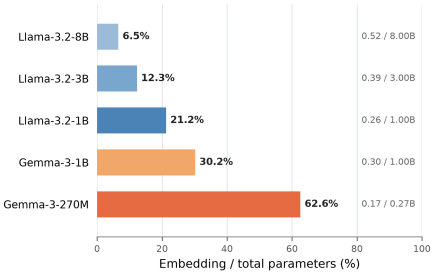
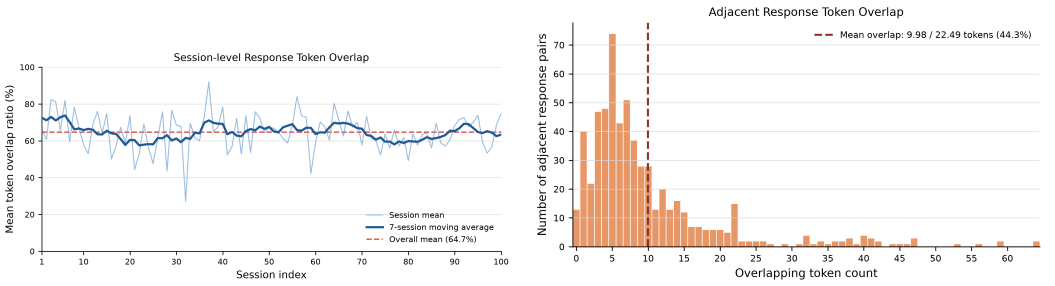


Figure 1: Proportion of vocabulary-dependent parameters (i.e., the embedding layer and LM-head) across SLMs of different sizes. Their relative footprint increases as models shrink, highlighting a growing LM-head memory bottleneck.

Despite their smaller scale, SLM inference on edge devices still faces critical memory and data-movement bottlenecks under tight resource constraints Wang et al. (2025). A key contributor is the model's *vocabulary size*, which determines the footprint of vocabulary-dependent modules, namely the *embedding table* and the *language modeling head (LM-head)* Lu et al. (2025). In weight-tied SLMs, the two share the same parameter matrix. In recent architectures such as Llama-3.2-1B Meta (2024), Gemma-3-1B Team et al. (2025), and Qwen-3-0.6B Yang et al. (2025), vocabulary-dependent param-

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



(a) Historical response-token overlap (b) Adjacent-response token-overlap distribution

Figure 2: Response-token overlap under full-head loading across 100 multi-turn dialogue sessions. (a) Mean overlap between each response and its preceding response history. (b) Distribution of token overlap between adjacent responses. The substantial cross-turn reuse motivates cross-turn LM-head caching for efficient multi-turn SLM inference.

eters account for a substantial fraction of the model, reaching 62% in Gemma-3-270M Lacombe et al. (2025), as shown in Figure 1. This proportion grows as models shrink, making vocabulary-dependent modules an increasingly important memory and data-movement bottleneck for resource-constrained edge inference.

To mitigate this issue, existing vocabulary-reduction methods retain only a compact subset of LM-head rows for inference Yang et al. (2022); Ushio et al. (2023); Zhang et al. (2026). Representative approaches include static corpus-level pruning, which applies a fixed vocabulary across inference instances Yang et al. (2022); Ushio et al. (2023), and task-adaptive methods such as VocabTailor Zhang et al. (2026), which combine a task-specific core vocabulary with rows selected from the current input. However, these methods operate at the corpus/task or individual-query level, overlooking ***contextual continuity*** across dialogue turns. In real-world user–AI interactions, topics are often revisited and incrementally refined through semantically correlated turns (e.g., from *I want to buy a cake* to *How about cheesecake?* and *Do they offer delivery?*), creating opportunities for cross-turn vocabulary reuse, as illustrated in Figure 2. In our deployment setting, the complete LM-head resides in CPU memory while only a budget-constrained subset is maintained in GPU memory, causing query-isolated methods to repeatedly transfer reusable rows across the CPU–GPU interconnect and incur avoidable data-movement overhead Song et al. (2024); Sheng et al. (2023).

Motivated by this gap, we advocate exploiting *contextual continuity* across dialogue turns—a form of ***temporal locality***—when managing the LM-head in multi-turn SLM inference. Specifically, the system should retain rows with high reuse potential and transfer only those missing in subsequent turns. Realizing this idea is non-trivial due to three key challenges. (i) *Dialogue-specific vocabulary coverage*: identifying a compact set of rows that remains broadly useful while adapting to the ongoing dialogue; (ii) *Context- and reuse-aware row prioritization*: adapting row utility according to token prevalence, current-query relevance, recent cross-turn reuse, and residency; and (iii) *Transfer-efficient cache management*: balancing useful new rows with reusable resident rows under a fixed memory budget. Building on this intuition, we formulate the ***cross-turn LM-head caching problem***, which aims to minimize LM-head transfers across turns while maintaining high response quality under a fixed memory budget. Figure 3 provides a high-level overview of the formulated problem and contrasts it with prior query-isolated strategies.

To address these challenges, we propose SLMCoach (*Context-Adaptive Caching of LM-Head in SLMs*), a context-adaptive LM-head caching framework that exploits contextual continuity to selectively retain rows with high reuse potential and incrementally update the cache. SLMCoach incorporates three core components: (1) **To ensure dialogue-specific vocabulary coverage,** we develop *dialogue-specific base vocabulary construction*, which captures broadly useful rows while adapting to the ongoing dialogue. (2) **To adapt row priorities to the evolving dialogue,** we introduce *context- and reuse-aware adaptation*, which jointly considers token prevalence, current-query relevance, cross-turn reuse, and GPU residency. (3) **To enable transfer-efficient caching under a fixed memory budget,** we present *budgeted LM-head cache management*, which reuses resident rows, transfers only missing ones, and replaces lower-priority rows when needed. Overall, SLMCoach

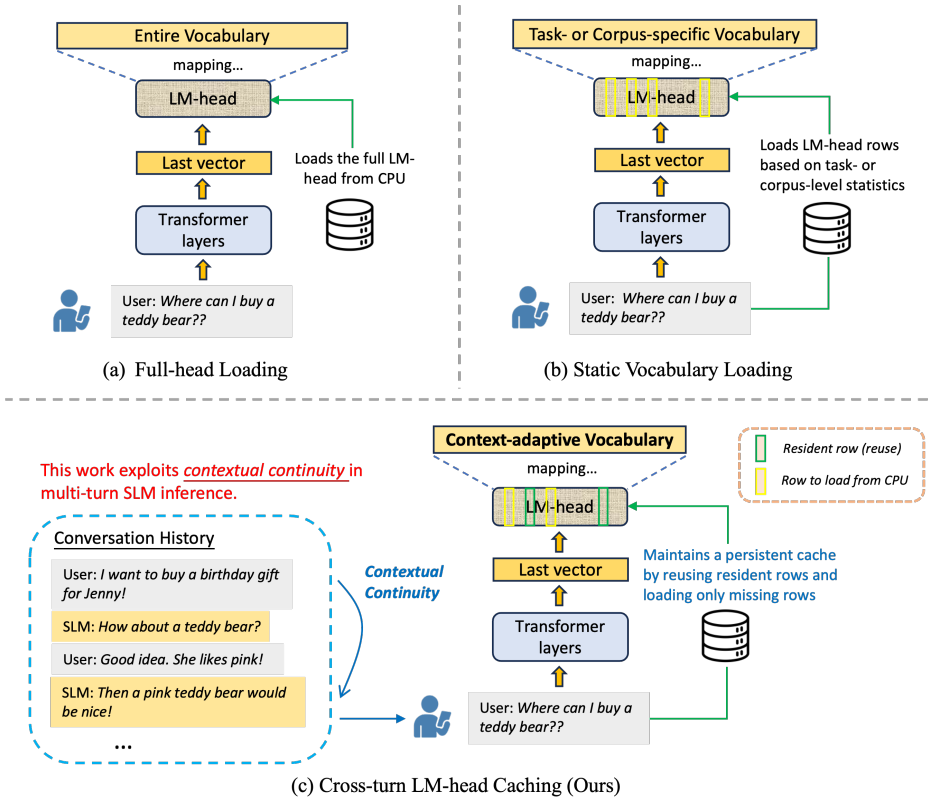


Figure 3: Comparison of LM-head loading strategies for multi-turn SLM inference. **(a)** ***Full-head Loading*** transfers the entire LM-head from CPU to GPU at every turn, incurring substantial data movement. **(b)** ***Static Vocabulary Loading*** retains a compact subset derived from task- or corpus-level statistics, reducing memory but ignoring cross-turn reuse. **(c)** ***Cross-turn LM-head Caching (Ours)*** exploits *temporal locality* (i.e., contextual continuity) by maintaining a persistent GPU-resident cache, reusing resident rows, and transferring only missing rows.

reduces redundant CPU-to-GPU LM-head transfers while preserving response quality under a fixed memory budget. We evaluate SLMCoach on MultiWOZ 2.2 Zang et al. (2020), a widely used task-oriented multi-turn dialogue benchmark, in terms of inference efficiency and response fidelity.

Our contributions are summarized as follows:

- We formulate *cross-turn LM-head caching* for efficient multi-turn SLM inference, aiming to reduce redundant LM-head transfers under a fixed memory budget while preserving response fidelity.
- We propose SLMCoach, a context-adaptive framework that exploits contextual continuity to selectively retain LM-head rows with high reuse potential and incrementally update the GPU-resident cache across turns.
- We conduct extensive experiments on a multi-turn dialogue benchmark and show that SLMCoach reduces the LM-head footprint by 99.2% and peak GPU memory by 51.5%, while reducing LM-head data movement by 7.5× at matched fidelity. Further analyses across cache budgets and serving models validate the benefit of cross-turn reuse.

2 PRELIMINARIES

In this section, we first describe the standard SLM inference pipeline and then formulate the *cross-turn LM-head caching problem* for multi-turn SLM inference. Section 3 presents SLMCoach, our solution to this problem.

2.1 SMALL LANGUAGE MODEL INFERENCE PIPELINE

A modern small language model (SLM) typically comprises a tokenizer, an embedding table, a stack of Transformer layers, and a language modeling head (LM-head). Given a raw-text input x, the tokenizer maps it to a token sequence $Q = (q_1, \ldots, q_n)$, where $q_i \in \mathcal{V}$ and $\mathcal{V}$ denotes the vocabulary. The embedding table $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times d}$ maps these tokens to d-dimensional representations, which are subsequently contextualized by the Transformer layers. During autoregressive decoding, the resulting hidden state at each decoding step is projected by the LM-head for next-token prediction. We next focus on this vocabulary-dependent LM-head, which is the target of our cross-turn caching formulation.

LM-head and autoregressive decoding. Let $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}| \times d}$ denote the LM-head weight matrix. In weight-tied models, $\mathbf{W}$ shares its parameters with the embedding table $\mathbf{E}$. At decoding step j of turn k, the LM-head projects the final hidden state $\mathbf{h}_{k,j} \in \mathbb{R}^d$ into vocabulary logits:

$$\mathbf{z}_{k,j} = \mathbf{W}\mathbf{h}_{k,j} \in \mathbb{R}^{|\mathcal{V}|}. \tag{1}$$

For multi-turn inference, let $H_k = \{(Q_i, R_i)\}_{i<k}$ denote the dialogue history before turn k, where Q_i and R_i are the query and response token sequences of turn i, respectively. Given the current query Q_k and the previously generated response prefix $R_{k,<j} = (r_{k,1}, \ldots, r_{k,j-1})$, the next-token distribution is

$$p(r_{k,j} = v \mid H_k, Q_k, R_{k,<j}) = \frac{\exp(z_{k,j,v})}{\sum_{u \in \mathcal{V}} \exp(z_{k,j,u})}, \quad v \in \mathcal{V}, \tag{2}$$

where $z_{k,j,v}$ is the logit associated with token v. Under greedy decoding, the next token is selected as

$$r_{k,j} = \arg\max_{v \in \mathcal{V}} z_{k,j,v}. \tag{3}$$

Since the size of $\mathbf{W}$ grows linearly with $|\mathcal{V}|$ and the LM-head is invoked at every decoding step, it constitutes the vocabulary-dependent component targeted by our cross-turn caching formulation in Section 2.2.

2.2 THE PROBLEM OF CROSS-TURN LM-HEAD CACHING FOR MULTI-TURN SLM INFERENCE

In this work, we consider multi-turn SLM inference in an edge setting where the complete LM-head resides in CPU memory and only a budget-constrained subset is maintained in GPU memory. Since consecutive dialogue turns exhibit contextual continuity, useful LM-head rows can be reused across turns to avoid redundant CPU-to-GPU transfers. Specifically, we formulate the *cross-turn LM-head caching problem*, which aims to minimize cumulative LM-head row transfers under a fixed GPU memory budget while preserving response quality.

2.2.1 PROBLEM FORMULATION

Consider a multi-turn session consisting of K turns, indexed by $k = 1, \ldots, K$. At turn k, the SLM conditions on the dialogue history $H_k = \{(Q_i, R_i)\}_{i<k}$ and the current query Q_k to autoregressively generate the response R_k, following the inference pipeline in Section 2.1.

Causal row selection and cross-turn reuse. Let M be the maximum number of LM-head rows that can reside in GPU memory. At turn k, let $C_k \subseteq \mathcal{V}$ denote the rows selected for the current inference and $A_{k-1} \subseteq \mathcal{V}$ the rows already resident in GPU memory. Because C_k must be chosen before R_k is generated, it is determined causally from (H_k, Q_k) and the current cache state A_{k-1}:

$$C_k = \pi(H_k, Q_k, A_{k-1}), \qquad |C_k| \le M. \tag{4}$$

Rows already resident in the cache can be reused directly, whereas only the missing rows must be transferred from CPU memory:

$$\textsc{Reuse}_k = A_{k-1} \cap C_k, \qquad \textsc{Load}_k = C_k \setminus A_{k-1}. \tag{5}$$

After serving turn k, the cache is updated to a resident set A_k satisfying

$$A_k \subseteq A_{k-1} \cup C_k, \qquad |A_k| \le M, \quad \forall k, \tag{6}$$

where A_0 denotes the initial cache state. The first constraint allows the system to retain previously resident or newly loaded rows without introducing additional transfers solely for future reuse.

Response fidelity. Let R_k denote the full-head response and $\hat{R}_k$ the response generated using only the selected rows C_k. Along the full-head trajectory, restricting the LM-head to C_k preserves the logits of retained tokens, so the full-head next token is reproduced whenever it belongs to C_k.

We therefore define the top-1 coverage

$$\mathcal{Q}(C_k, R_k) = \frac{1}{|R_k|} \sum_{j=1}^{|R_k|} \mathbb{I}\big[r^*_{k,j} \in C_k\big], \tag{7}$$

where $r^*_{k,j} = \arg\max_{v \in \mathcal{V}} z_{k,j,v}$ is the full-head token at step j conditioned on $R_{k,<j}$. The full-head trajectory provides a fixed reference even if the budgeted response diverges. Under greedy decoding, $\mathcal{Q}(C_k, R_k) = 1$ guarantees $\hat{R}_k = R_k$.

Cross-turn caching objective. The cross-turn LM-head caching problem seeks to minimize cumulative CPU-to-GPU row transfers while satisfying the memory budget and a target response-fidelity level $\epsilon \in [0, 1]$:

$$\begin{aligned} \min_{\pi, \{A_k\}_{k=1}^K} \quad & \sum_{k=1}^{K} |C_k \setminus A_{k-1}| \\ \text{s.t.} \quad & A_k \subseteq A_{k-1} \cup C_k, \qquad |A_k| \le M, \\ & \mathcal{Q}(C_k, R_k) \ge \epsilon, \qquad \forall k. \end{aligned} \tag{8}$$

Since every LM-head row contains d parameters, the objective is directly proportional to the amount of LM-head data transferred across turns. Section 3 develops SLMCoach to instantiate the causal policy π.

3 SLMCoach: A Context-Adaptive Caching Framework for Multi-Turn SLM Inference

To address the challenges of cross-turn LM-head caching, we develop SLMCoach (*Context-Adaptive Caching of LM-Head in SLMs*), a context-adaptive framework that retains reusable LM-head rows across dialogue turns by exploiting contextual continuity. SLMCoach incrementally maintains the GPU-resident cache by reusing resident rows and loading only missing ones. It comprises three components: (1) dialogue-specific base vocabulary construction, (2) context- and reuse-aware adaptation, and (3) budgeted LM-head cache management.

3.1 Dialogue-Specific Base Vocabulary Construction

SLMCoach first constructs a compact vocabulary base from offline profiling and conditions it on the ongoing dialogue, providing a stable prior for subsequent online adaptation.

Residual Output Token Extraction. We consider two serving models for multi-turn inference: *prompt re-rendering*, which reprocesses the dialogue history at each turn, and *KV-cached serving*, which retains previously processed context. We therefore exclude tokens already supplied by the serving context, since their rows require no additional LM-head allocation. Let $P_{s,k} = \mathrm{ChatTemplate}(H_{s,k}, Q_{s,k})$ denote the rendered prompt of turn k in profiling session s, and define

$$F_{s,k} = \begin{cases} \mathrm{SetTok}(P_{s,k}), & \text{prompt re-rendering,} \\ \mathrm{SetTok}(Q_{s,k}), & \text{KV-cached serving,} \end{cases} \tag{9}$$

where $\mathrm{SetTok}(\cdot)$ returns the unique token IDs. The residual output tokens are

$$U_{s,k} = \mathrm{SetTok}(R_{s,k}) \setminus F_{s,k}. \tag{10}$$

Thus, $U_{s,k}$ contains only output rows not already available from the serving context and can be computed entirely from completed profiling turns.

Profiling-Derived Base Vocabulary. We estimate corpus-level token prevalence by session-level occurrence. Let $U_s = \bigcup_k U_{s,k}$ and define

$$g(v) = \frac{1}{N} \sum_{s=1}^{N} \mathbb{I}[v \in U_s], \tag{11}$$

where N is the number of profiling sessions. Counting each token once per session prevents repetitive dialogues from dominating the estimate. We rank tokens by $g(v)$ and form T_{global} from the shortest prefix covering a $(1 - \tau_{\text{cov}})$ fraction of the aggregate prevalence. Accordingly, T_{global} provides a compact vocabulary prior of rows broadly useful across dialogues.

Dialogue-Specific Conditioning. To adapt the global prior to the ongoing dialogue, we estimate service-specific statistics. Let n_c denote the number of profiling turns associated with service c, and $n_c(v)$ the number containing residual token v. We define

$$g^{(c)}(v) = \frac{n_c(v) + g(v)}{n_c + 1}, \qquad \tilde{g}_{s,k}(v) = (1 - \rho) g(v) + \rho \max_{c \in \mathcal{C}_{s,k}} g^{(c)}(v), \tag{12}$$

where $\mathcal{C}_{s,k}$ contains the services observed up to turn k and $\rho \in [0, 1]$ controls service conditioning. Smoothing toward $g(v)$ stabilizes sparse service estimates, while $\tilde{g}_{s,k}(v)$ specializes the prior to the current dialogue.

3.2 Context- and Reuse-Aware Adaptation

Because token demand evolves throughout a dialogue, SLMCoach adapts the profiling-derived prior using recent prevalence, current-query relevance, and cross-turn reuse.

Online Dialogue Prevalence. Let $\mathcal{W}_{s,k} = \{j : \max(1, k - W) \le j \le k - 1\}$ denote the W most recent completed turns, and let $d_{s,j} = \text{SetTok}(Q_{s,j}) \cup \text{SetTok}(R_{s,j})$. We define the recency-weighted prevalence

$$\hat{g}_{s,k}(v) = \frac{\sum_{j \in \mathcal{W}_{s,k}} e^{-(k-1-j)/\tau} \mathbb{I}[v \in d_{s,j}]}{\sum_{j \in \mathcal{W}_{s,k}} e^{-(k-1-j)/\tau}}, \tag{13}$$

where $\tau > 0$ controls recency decay. Hence, tokens occurring frequently and recently in the current session receive higher prevalence.

Contextual Relevance and Cross-Turn Reuse. Prevalence alone does not capture relevance to the current query. We therefore measure the association between a query token u and residual output token v using positive pointwise mutual information:

$$a(u, v) = \max\left(\log \frac{\Pr(u, v)}{\Pr(u) \Pr(v)}, 0\right), \tag{14}$$

where the probabilities are empirical occurrence and co-occurrence rates over profiling turns. The context score of candidate v is

$$h_{s,k}(v) = \sum_{u \in \text{SetTok}(Q_{s,k})} \text{idf}(u)\, a(u, v), \tag{15}$$

where IDF downweights common query tokens with limited predictive value. For efficiency, we retain only informative query tokens and their strongest output associations in a sparse trigger table.

SLMCoach additionally captures locally reused tokens that may not appear in T_{global}. Their recent reuse multiplicity is

$$r_{s,k}(v) = \left|\{j \in \mathcal{W}_{s,k} : v \in \text{SetTok}(R_{s,j})\}\right|. \tag{16}$$

Tokens outside T_{global} with $r_{s,k}(v) \ge m_0$ form $T_{\text{temp},k}$, where m_0 filters incidental repetition. The resulting dialogue-specific token set is

$$T_{s,k} = T_{\text{special}} \cup T_{\text{global}} \cup T_{\text{temp},k}, \tag{17}$$

where T_{special} contains mandatory special tokens. $T_{\text{temp},k}$ is empty at the first turn and subsequently augments the offline base with locally reusable rows.

Unified Residency-Aware Ranking. Since the offline prior is more reliable early in a dialogue, whereas online evidence accumulates over time, we combine them as

$$p_{s,k}(v) = (1 - \eta_k)\tilde{g}_{s,k}(v) + \eta_k \hat{g}_{s,k}(v), \qquad \eta_k = \frac{n_{s,k}}{n_{s,k} + n_0}, \tag{18}$$

where $n_{s,k} = |\mathcal{W}_{s,k}|$ and $n_0 > 0$ controls the adaptation rate.

Each candidate row is then assigned the utility

$$u_k(v) = \log\big(p_{s,k}(v) + \epsilon_0\big) + \mu\, \bar{h}_{s,k}(v) + \lambda\, \mathbb{I}[v \in A_{k-1}], \tag{19}$$

where $\bar{h}_{s,k}(v)$ is the max-normalized context score, $\epsilon_0 > 0$ ensures numerical stability, and μ and λ weight context and residency, respectively. The first two terms capture dialogue demand and query relevance, while the residency bonus promotes reuse and reduces transfers.

3.3 Budgeted LM-Head Cache Management

Mandatory Rows and Contested Capacity. At turn k, SLMCoach instantiates the causal policy π by dividing the available capacity between mandatory and budgeted rows. The mandatory set is

$$C_k^{\mathrm{req}} = T_{\mathrm{special}} \cup F_{s,k}, \tag{20}$$

where $F_{s,k}$ follows Eq. (9). The remaining capacity is $b_k = M - |C_k^{\mathrm{req}}|$, with candidate pool

$$\mathcal{S}_k = \left(T_{s,k} \cup \bigcup_{c \in \mathcal{C}_{s,k}} T_c \cup T_{\mathrm{tail}} \right) \setminus C_k^{\mathrm{req}}. \tag{21}$$

where T_c denotes the service-specific vocabulary ranked by $g^{(c)}(v)$ for service c, and T_{tail} contains the remaining rows observed during profiling. When $|\mathcal{S}_k| > b_k$, SLMCoach selects the b_k highest-utility rows:

$$S_k^\star = \arg\max_{S \subseteq \mathcal{S}_k,\, |S| = b_k} \sum_{v \in S} u_k(v), \qquad C_k = C_k^{\mathrm{req}} \cup S_k^\star. \tag{22}$$

Since the objective is modular, Eq. (22) is solved by ranking candidates according to $u_k(v)$ and selecting the top b_k rows.

To avoid re-ranking the full candidate pool at every turn, SLMCoach rescores only rows affected by recent dialogue signals and those near the admission boundary whose selection may change due to residency.

Incremental LM-Head Cache Management. Given C_k, SLMCoach reuses resident rows and transfers only missing ones:

$$\mathrm{Reuse}_k = A_{k-1} \cap C_k, \qquad \mathrm{Load}_k = C_k \setminus A_{k-1}, \qquad \mathrm{Evict}_k = A_{k-1} \setminus C_k. \tag{23}$$

After serving turn k, the cache becomes $A_k = C_k$, with $|A_k| \le M$. Rows in Evict_k are released before Load_k is transferred so that GPU occupancy remains within the memory budget. Retention is therefore realized directly through the selection policy, enabling SLMCoach to reuse resident rows while adapting the cache across dialogue turns.

4 Experiments

4.1 Experimental Setup

Benchmark Dataset. We evaluate SLMCoach on MultiWOZ 2.2 Zang et al. (2020), a widely used task-oriented multi-turn dialogue benchmark that provides sequential, context-dependent interactions suitable for studying cross-turn LM-head reuse. Our evaluation uses 512 sessions containing 3,842 user queries (7.5 turns per session on average), with a disjoint 512-session split reserved for offline profiling. Each session is replayed sequentially using the original user utterances as queries, while responses generated by Full-head Resident serve as oracle references for evaluating response fidelity.

Table 1: Inference efficiency and LM-head data movement on MultiWOZ 2.2. VocabTailor and SLMCoach use $M = 2{,}048$ resident rows, whereas Static VP retains its tokenizer-compatible merge closure. Full-head variants are included for reference.

Method	Resident Rows	LM-Head (MiB)	Mean TPOT (ms)	Mean E2E (ms)	Mean H2D/query (MiB)	Peak VRAM (GiB)
Full-head Loading	262,144	320.00	7.10	260.39	320.000	0.5750
Full-head Resident (Oracle)	262,144	320.00	6.96	207.87	0.000	0.5747
Static VP	7,430	9.07	6.86	**198.96**	1.2066	0.2858
VocabTailor	2,048	2.50	**6.82**	202.95	0.3421	0.2798
SLMCoach (ours)	2,048	2.50	**6.82**	202.82	**0.3343**	**0.2785**

Table 2: Response fidelity of SLMCoach and vocabulary-reduction baselines on MultiWOZ 2.2, measured against *Full-head Resident* responses.

Method	BERTScore F1	ROUGE-L	BLEU-2	BLEU-4	METEOR	Prefix Agreement (%)	Length Ratio
Static VP	50.88	53.60	43.77	35.62	53.27	22.42	0.943
VocabTailor	84.77	86.28	82.84	78.20	86.12	73.40	0.993
SLMCoach (ours)	**85.23**	**86.73**	**83.28**	**78.64**	**86.52**	**74.56**	**0.994**

Comparative Methods. We compare SLMCoach with two full-head references and two representative vocabulary-reduction baselines. *Full-head Loading* transfers the complete LM-head from CPU to GPU for every query and serves as a zero-reuse reference, while *Full-head Resident* keeps the entire LM-head in GPU memory and serves as the full-head oracle. *Static Vocabulary Pruning (VP)* Yang et al. (2022); Ushio et al. (2023) follows conventional corpus-based filtering to construct a fixed vocabulary shared across inference instances, whereas *VocabTailor* Zhang et al. (2026) is a task-adaptive approach that augments a task-specific core vocabulary with rows selected from the current input. Unlike these prior methods, which operate at the corpus/task or individual-query level, SLMCoach exploits dialogue history and cross-turn residency to adapt LM-head selection and reuse cached rows across turns. For the serving-model analysis in Table 3, we additionally include LRU Mattson et al. (1970), a standard recency-based caching baseline that retains recently used rows without modeling dialogue context or token relevance. We further include a session oracle that uses full-session output-token information as an upper bound.

Evaluation Metrics. We evaluate SLMCoach in terms of inference efficiency, LM-head data movement, and response fidelity. For efficiency, we report LM-head memory footprint, Time Per Output Token (TPOT), end-to-end latency, and peak GPU memory Zhong et al. (2024). Cross-turn caching efficiency is measured by host-to-device (H2D) traffic per query and transferred rows per turn Song et al. (2024). For response fidelity, we report top-1 coverage $\mathcal{Q}$ in Eq. (7) and exact recovery ($\epsilon = 1$), together with BERTScore F1 Zhang et al. (2019), ROUGE-L Lin (2004), BLEU-2/4 Papineni et al. (2002), and METEOR Banerjee & Lavie (2005). We additionally use prefix agreement and length ratio to measure generation-trajectory consistency and response-length preservation against the Full-head Resident oracle, respectively.

4.2 Experimental Results

Overall Efficiency Comparison. In the first set of experiments, we compare SLMCoach with two full-head references (*Full-head Loading* and *Full-head Resident*) and two representative vocabulary-reduction baselines (*Static VP* and *VocabTailor*) over the same 512 evaluation sessions. The corresponding results are presented in Table 1. Inspection of Table 1 reveals three noteworthy points. First, compared with *Full-head Resident*, repeatedly loading the full LM-head increases end-to-end latency by 25.3% while leaving TPOT nearly unchanged, and accumulates approximately 1.17 TiB of H2D traffic. This suggests that repeated parameter movement accounts for most of the additional latency. Second, with only 2,048 resident rows, SLMCoach reduces the LM-head footprint by 99.2%, peak GPU memory by 51.5%, and H2D traffic by 99.9% relative to the corresponding full-head references, while maintaining comparable end-to-end efficiency. Third, SLMCoach and VocabTailor exhibit similar runtime under prompt re-rendering, where tokens from prior turns are repeatedly

Table 3: Comparison of SLMCoach with vocabulary-reduction and recency baselines on MultiWOZ 2.2 under KV-cached and prompt re-rendering serving. Each entry reports top-1 coverage $\mathcal{Q}$ / exact recovery (ϵ=1) in %. Static VP uses its native 7,430-row footprint.

	KV-cached Serving				Prompt Re-rendering	
Method	M=256	M=512	M=1,024	M=2,048	M=512	M=2,048
Session oracle (UB)	100.00/100.0	100.00/100.0	100.00/100.0	100.00/100.0	100.00/100.0	100.00/100.0
Static VP	87.63/30.5	93.62/52.2	96.96/70.6	98.45/81.0	93.62/52.2	98.45/81.0
VocabTailor	91.03/39.5	95.54/60.0	97.69/74.7	98.73/83.5	96.55/69.9	99.04/87.6
LRU	92.02/43.0	95.30/55.5	96.84/63.0	97.61/66.7	96.69/69.5	99.07/87.5
SLMCoach (ours)	**94.47/59.0**	**96.94/71.6**	**98.39/81.8**	**99.09/87.8**	**97.00/71.3**	**99.08/87.7**

reintroduced through the dialogue context, reducing the opportunity for explicit cross-turn reuse. We further investigate this effect under KV-cached serving in Table 3.

Response Fidelity Comparison. In the second set of experiments, we evaluate the response fidelity of SLMCoach against the two vocabulary-reduction baselines, using *Full-head Resident* responses as the oracle reference. Three important observations can be drawn from Table 2, which presents five standard text-generation metrics (i.e., BERTScore F1, ROUGE-L, BLEU-2, BLEU-4, and METEOR), together with prefix agreement and length ratio. First, despite retaining only 2,048 LM-head rows, SLMCoach closely preserves full-head responses, with a length ratio of 0.994 indicating minimal change in response length. Second, SLMCoach consistently outperforms VocabTailor across all metrics, achieving an average relative gain of 0.52% across the five text-generation metrics and improving prefix agreement by 1.16 percentage points (1.58% relative). These results suggest that context-adaptive LM-head selection better preserves both response content and generation trajectory. Third, Static VP exhibits substantially lower fidelity, highlighting the limitation of a fixed vocabulary in adapting to multi-turn generation. Overall, the efficiency gains of SLMCoach are achieved without sacrificing response fidelity.

Effect of the Serving Model. To further examine the benefit of cross-turn reuse, we evaluate SLMCoach under KV-cached serving and prompt re-rendering, comparing it with *Static VP*, *VocabTailor*, and a recency-based *LRU* baseline; the session oracle is included as an upper bound. The corresponding results are presented in Table 3, from which we can draw three notable observations. First, under KV-cached serving, SLMCoach consistently outperforms all baselines across cache budgets. At $M = 512$, it improves top-1 coverage over VocabTailor by 1.40 percentage points and exact recovery by 11.6 points, demonstrating the benefit of retaining dialogue-dependent vocabulary across turns. Second, the gains become more pronounced under tighter budgets: at $M = 256$, SLMCoach achieves a 49.4% relative improvement in exact recovery over VocabTailor, indicating that cross-turn caching is particularly valuable when cache capacity is constrained. Third, this advantage largely diminishes under prompt re-rendering, where dialogue history is repeatedly reintroduced as serving context. At $M = 2{,}048$, the exact-recovery gain over VocabTailor becomes negligible, indicating that SLMCoach is most beneficial when useful LM-head rows must be explicitly retained and reused across turns. At matched response fidelity, SLMCoach reduces LM-head data movement from 2.50 to 0.3343 MiB/query, corresponding to a 7.5× reduction relative to query-isolated loading (*cf.* Table 1).

5 Conclusion

In this paper, we introduce *cross-turn LM-head caching* as a task for efficient multi-turn SLM inference. We present SLMCoach, a context-adaptive framework that exploits ***contextual continuity*** to reuse LM-head rows across dialogue turns. Empirical evaluations on a multi-turn dialogue benchmark demonstrate that SLMCoach reduces the LM-head footprint by 99.2% and peak GPU memory by 51.5% while preserving response fidelity. At matched fidelity, it further reduces LM-head data movement by 7.5×. These results demonstrate the benefit of cross-turn reuse for resource-constrained SLM inference. Future directions include exploring more sophisticated techniques, such as efficient graph-based modeling, to capture richer linguistic and discourse structures and identify tokens with high cross-turn reuse potential, further improving the efficiency of multi-turn SLM inference.

Reproducibility Statement

We provide detailed descriptions of the SLMCoach framework, experimental setup, datasets, baselines, evaluation metrics, and implementation settings in the main paper and appendix. Additional experimental details and analyses are included in the appendix to facilitate reproduction of our results.

AI Use Statement

Generative AI tools were used to assist with language editing, improving readability, and refining the presentation of the manuscript, including feedback on the exposition of methodology and experimental results. All AI-assisted content was reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

References

Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pp. 65–72, 2005.

Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. Small language models are the future of agentic ai. *arXiv preprint arXiv:2506.02153*, 2025.

Olivier Lacombe, Kathleen Kenealy, Kat Black, Ravin Kumar, Francesco Visin, and Jiageng Zhang. Introducing gemma 3 270m: The compact model for hyper-efficient ai. Google Developers Blog, August 2025. URL `https://developers.googleblog.com/introducing-gemma-3-270m/`.

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

Zhenyan Lu, Xiang Li, Dongqi Cai, Rongjie Yi, Fangming Liu, Wei Liu, Jian Luan, Xiwen Zhang, Nicholas D Lane, and Mengwei Xu. Demystifying small language models for edge deployment. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14747–14764, 2025.

Richard L. Mattson, Jan Gecsei, Donald R. Slutz, and Irving L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems journal*, 9(2):78–117, 1970.

Meta. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models. `https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/`, September 2024.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pp. 590–606, 2024.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Asahi Ushio, Yi Zhou, and Jose Camacho-Collados. Efficient multilingual language model compression through vocabulary trimming. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 14725–14739, 2023.

Chien Van Nguyen, Xuan Shen, Ryan Aponte, Yu Xia, Samyadeep Basu, Zhengmian Hu, Jian Chen, Mihir Parmar, Sasidhar Kunapuli, Joe Barrow, et al. A survey on small language models. In *Proceedings of the 15th International Conference on Recent Advances in Natural Language Processing-Natural Language Processing in the Generative AI Era*, pp. 807–821, 2025.

Fali Wang, Zhiwei Zhang, Xianren Zhang, Zongyu Wu, Tzuhao Mo, Qiuhao Lu, Wanjing Wang, Rui Li, Junjie Xu, Xianfeng Tang, et al. A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness. *ACM Transactions on Intelligent Systems and Technology*, 16(6):1–87, 2025.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China information sciences*, 68(2):121101, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Ziqing Yang, Yiming Cui, and Zhigang Chen. Textpruner: A model pruning toolkit for pre-trained language models. In *Proceedings of the 60th annual meeting of the association for computational linguistics: System demonstrations*, pp. 35–43, 2022.

Xiaoxue Zang, Abhinav Rastogi, Srinivas Sunkara, Raghav Gupta, Jianguo Zhang, and Jindong Chen. Multiwoz 2.2: A dialogue dataset with additional annotation corrections and state tracking baselines. In *Proceedings of the 2nd workshop on natural language processing for conversational AI*, pp. 109–117, 2020.

Hanling Zhang, Yayu Zhou, Tongcheng Fang, Zhihang Yuan, Guohao Dai, Wanli Ouyang, and Yu Wang. Vocabtailor: Dynamic vocabulary selection for downstream tasks in small language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 28453–28471, 2026.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Zican Dong, Yupeng Hou, Beichen Zhang, Yingqian Min, Junjie Zhang, Peiyu Liu, et al. A survey of large language models. *Frontiers of Computer Science*, 20(12):2012627, 2026.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 193–210, 2024.

APPENDIX

Table 4: Memory footprint and runtime of SLMCoach compared with full-head inference on Gemma-3-270M-IT at $M = 2{,}048$.

Level	Metric	Full Head	**SLMCoach**	Reduction & Speedup
Memory	LM-head memory	320.00 MiB	2.50 MiB	↓99.22%
	LM-head ratio	62.57%	1.29%	↓97.94%
	Whole-model memory	511.4 MiB	193.9 MiB	↓62.08%
	Peak GPU memory	0.5747 GiB	0.2785 GiB	↓51.54%
Runtime	LM-head projection latency	0.2067 ms	0.0083 ms	24.8×
	E2E latency	207.87 ms	202.82 ms	1.02×

Table 5: Ablation study of SLMCoach on MultiWOZ 2.2 under KV-cached and prompt re-rendering serving. We evaluate the contribution of each adaptation signal in terms of response fidelity and transferred rows per turn.

	KV-cached, M=256			KV-cached, M=512			Prompt Re-rendering, M=512	
Variant	$\mathcal{Q}$ (%)	ϵ=1 (%)	Rows / Turn (↓)	$\mathcal{Q}$ (%)	ϵ=1 (%)	Rows / Turn (↓)	$\mathcal{Q}$ (%)	ϵ=1 (%)
SLMCoach (ours)	94.47	59.0	48.0	96.94	71.6	76.9	97.00	71.3
w/o service mixture	94.35	58.6	47.6	96.83	71.2	77.2	96.93	71.4
w/o context term	93.31	52.4	39.8	96.79	70.8	72.0	96.84	69.8
w/o residency term	94.54	59.1	57.1	96.97	71.9	88.8	97.02	71.6
w/o session estimate	92.57	45.9	48.9	95.96	61.6	76.5	97.00	71.3
offline prior only	91.42	40.7	38.7	95.78	60.2	71.6	96.83	69.8

A TWO-LEVEL ACCOUNTING OF THE SAVINGS

Since vocabulary reduction leaves the backbone unchanged, a single reduction ratio can obscure where the efficiency gains arise. We therefore report both LM-head-level and whole-model efficiency in Table 4. Two observations stand out. First, SLMCoach yields substantially larger gains at the LM-head level: it reduces LM-head memory by 99.22% and accelerates LM-head projection by $24.8\times$, while reducing the overall model footprint by 62.08% and improving end-to-end latency by only $1.02\times$. This gap is consistent with Amdahl's law: LM-head projection accounts for only 3.00% of each decoding step, bounding the attainable end-to-end speedup at approximately $1.03\times$. Second, the measured peak GPU memory reduction (51.54%) is smaller than the 62.08% reduction in model weights because peak memory also includes activations and the attention cache. These results clarify that SLMCoach primarily targets vocabulary-dependent memory and LM-head data movement rather than end-to-end decoding acceleration. This makes it particularly useful for memory-constrained and CPU–GPU bandwidth-sensitive SLM deployment, where repeatedly materializing the full LM-head remains costly.

B COMPONENT ANALYSIS

To assess the contribution of the adaptation signals in Eqs. (18) and (19), we conduct an ablation study by removing one component at a time while keeping all other settings fixed. Table 5 reveals three observations. First, the session estimate $\hat{g}_{s,k}$ contributes most strongly under KV-cached serving. At $M = 256$, removing it decreases top-1 coverage by 1.90 percentage points and exact recovery by 13.1 points, while its effect largely vanishes under prompt re-rendering, where prior dialogue tokens are repeatedly reintroduced. This highlights the importance of explicitly modeling cross-turn token demand when it is not preserved by the serving pipeline. Second, the context term consistently improves response fidelity. At $M = 256$, removing it reduces coverage by 1.16 points under KV-cached serving, with a smaller degradation under prompt re-rendering. By adapting LM-head selection to the current query, the context term captures turn-specific token demand beyond prevalence-based signals. The service-conditioned prior provides an additional, though smaller, gain by specializing

the vocabulary to the active dialogue services. Third, the residency term primarily reduces data movement. Removing it changes coverage by at most 0.07 points, but increases transferred rows per turn by 19.0% at $M = 256$ and 15.5% at $M = 512$. This shows that residency-aware ranking promotes cross-turn reuse without materially affecting response quality. Overall, the signals play complementary roles: prevalence and context improve dialogue-specific vocabulary selection, while residency reduces redundant LM-head transfers across turns.

C Implementation Details

To evaluate SLMCoach under highly resource-constrained conditions, we use Gemma-3-270M-IT, one of the smallest publicly available instruction-tuned language models. All experiments use bfloat16 on a single NVIDIA GeForce RTX 5090 with greedy decoding, up to 64 generated tokens per query, and one warm-up query excluded from measurement. The backbone remains GPU-resident, while the full weight-tied embedding and LM-head table is stored in CPU memory, with GPU-resident rows shared by embedding lookup and output projection. With $|\mathcal{V}| = 262{,}144$ and $d = 640$, the full table occupies 320 MiB. We set the cache budget to $M = 2{,}048$ rows (2.50 MiB), the smallest profiled budget near coverage saturation. Hyperparameters are tuned on equal fitting and development partitions of the profiling split, after which the offline component is refit on the full profiling set. We use $\tau_{\text{cov}} = 0.002$, $W = 8$, $\tau = 3$, $\rho = 0.7$, $m_0 = n_0 = 2$, $\mu = 3$, and $\lambda = 0.5$. The trigger table retains source tokens with document frequency at most $\delta = 0.5$ and their top $J = 48$ associated targets, while the residency term uses a $B = 256$-row boundary band. For fair comparison, we reimplement the vocabulary-reduction baselines within the same serving harness. VocabTailor with its default $\tau_{\text{VT}} = 0.01$ yields a 2,199-row static base, which we truncate to M for budget matching. VP cannot satisfy the same budget without altering its valid output space: its 3,260-token corpus union requires a tokenizer-compatible merge closure of 7,430 resident rows. We therefore report VP at its native footprint. This setup keeps the backbone, serving pipeline, and evaluation conditions consistent while isolating the effect of context-adaptive cross-turn LM-head caching.