ACCELERATING DECODING VIA AUTOREGRESSIVE-TO-PARALLEL DISTILLATION FOR ARCHITECTURE-AGNOSTIC GENERATIVE RECOMMENDATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative Recommendation (GR) retrieves items by autoregressively generating semantic identifiers, making beam-search inference costly. Existing speculative-decoding methods for GR either retrain the target with a draft head or train an external draft model while keeping the target frozen. The former incur substantial target retraining costs, whereas the latter still suffer from costly autoregressive draft inference and are limited by architecture-specific verification pipelines. To address these limitations, we propose the **A**utoregressive-to-**P**arallel **D**istillation framework for **G**enerative **R**ecommendation (**APD-GR**). This framework distills the frozen target's autoregressive prediction knowledge into a lightweight draft that computes SID position representations in parallel. Specifically, we design a two-stage distillation strategy that first establishes parallel prediction through SID supervision and then aligns the draft's prefix-conditioned distributions with those of the frozen target. To further support heterogeneous GR backbones, we propose a unified interface across decoder-only and encoder–decoder targets. In the experiments, to verify effectiveness and versatility of our proposed accelerating framework, we conduct extensive experiments across three datasets and five GR backbones. The results show that our method achieves an average end-to-end inference speedup of $9.22\times$ while largely preserving recommendation performance. To ease reproducibility, we release the code online[1].

1 INTRODUCTION

Recently, generative recommendation (GR) has emerged as a promising recommendation paradigm. It represents items as discrete semantic IDs (SIDs) and formulates recommendations as sequence generation conditioned on user histories (Rajput et al., 2023; Tay et al., 2022; Wang et al., 2022; Singh et al., 2024; Wang et al., 2024; Liu et al., 2025; Kong et al., 2025; Wei et al., 2025; Fu et al., 2026). Then, to produce top-K recommendations for each user, GR models typically use beam search to decode SIDs autoregressively, expanding and pruning candidate prefixes at each position. However, this process requires repeated forward passes through deep generative models across decoding positions and candidate prefixes. The resulting inference cost grows with model size, SID length, and beam width (Stern et al., 2018; Leviathan et al., 2023; Xia et al., 2023; Miao et al., 2023). To alleviate this bottleneck, recent studies have explored speculative decoding for GR (Lin et al., 2025b). They adopt a lightweight draft model or head (*Draft*) to predict SID tokens, reducing repeated computation of the GR backbone (*Target*).

Building on this idea, existing approaches to efficient drafting in GR can be broadly grouped into two categories, depending on whether the Target is retrained or kept frozen. **i) Retraining-based Methods**. These methods modify and retrain the Target to support multi-token prediction within a single backbone forward pass. For example, NEZHA (Wang et al., 2025) integrates a self-drafting head into the Target and jointly trains both components. Such designs improve decoding efficiency but require changes to the original model and its training procedure. *Consequently, they cannot be directly applied to already-trained GR models*. **ii) Training-free Methods**. To avoid retraining the Target, these methods (Lin et al., 2025b; Chen et al., 2026) freeze the pretrained Target and

[1] https://anonymous.4open.science/r/anonymous-code-release-DC63

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.

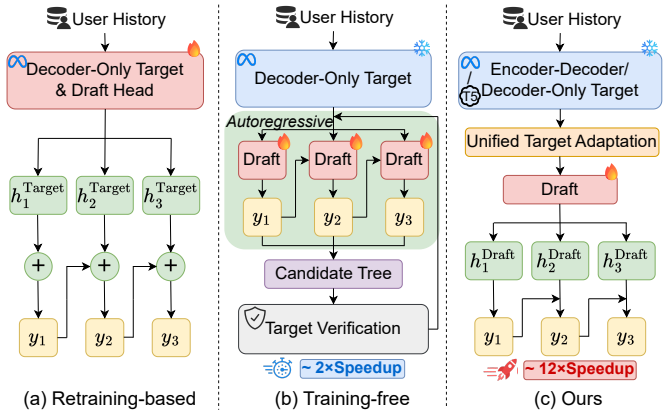

Figure 1: Comparison of existing speculative decoding approaches and our framework for GR.

train a lightweight Draft to approximate its predictions. The Draft proposes candidates, which the Target then verifies (illustrated by Figure 1(b)). However, this approach has two limitations. First, *the Draft still generates SID tokens autoregressively, thereby requiring repeated computation of the Draft backbone across decoding positions*. Second, existing tree-based verification implementations rely on model-specific attention masks and KV-cache management. Consequently, *the drafting framework developed for one GR backbone cannot be directly applied to another with a different architecture*. For example, AtSpeed (Lin et al., 2025b) uses a Llama-based verification pipeline, while its applicability to encoder–decoder GR backbones remains unexplored.

To achieve a more flexible and pluggable accelerating framework, we aim to construct SID position representations in parallel while keeping the Target frozen and supporting heterogeneous GR backbones. However, there exist two challenges: **i) Autoregressive-to-Parallel Distillation Challenge**. The frozen Target predicts SID tokens sequentially, conditioning each prediction on the preceding tokens. The Parallel Position Draft, however, must construct representations for all SID positions in a single forward pass, without access to those tokens. The challenge is to distill the Target's prefix-dependent prediction knowledge into these parallel position representations. **ii) Architecture-Agnostic Conditioning Challenge**. Since the Draft requires user-history information to construct parallel position representations and an initial generation state to support recurrent prefix modeling, features from heterogeneous frozen Targets must be adapted to meet these requirements. The challenge is to expose these signals through a common representational interface across Targets without modifying or retraining them.

To address the above challenges, we propose **APD-GR**, a lightweight decoding framework for heterogeneous frozen Targets. To address the autoregressive-to-parallel distillation challenge, we develop a two-stage knowledge transfer strategy for the Parallel Position Draft. The first stage establishes parallel position representations through multi-position SID supervision, and the second refines them through prefix-conditioned distribution distillation. The parallel draft computes these representations in a single forward pass, while the Prefix-Conditioned Head incorporates SID prefix information through lightweight recurrent prefix states. Furthermore, to support heterogeneous GR backbones, a unified cross-architecture conditioning interface provides adapted context features for the parallel draft and a root representation to initialize the recurrent prefix state. During inference, Target features are extracted once per input, and the parallel draft runs once; subsequent constrained beam search requires only lightweight computation in the Prefix-Conditioned Head.

In summary, our main contributions are as follows:

- We propose a two-stage knowledge transfer strategy from a frozen autoregressive Target to a parallel Draft, enabling faster recommendations while preserving recommendation quality.
- We introduce a unified, architecture-agnostic conditioning interface that enables the Draft to work with heterogeneous GR backbones without modifying or retraining them.
- We conduct extensive experiments across three datasets and five heterogeneous GR backbones. The results demonstrate that APD-GR achieves an average end-to-end inference speedup of $9.22\times$ while largely preserving recommendation performance.

2 PRELIMINARY

2.1 GENERATIVE RECOMMENDATION WITH SEMANTIC IDS

Generative recommendation (GR) formulates next-item recommendation as Semantic ID (SID) generation conditioned on a user's interaction history (Rajput et al., 2023; Singh et al., 2024; Wang et al., 2024). Let $\mathcal{I}$ denote the item set. Each item $i \in \mathcal{I}$ is represented by a fixed-length SID $\mathbf{c}_i = (c_{i,1}, \ldots, c_{i,M})$, where $c_{i,m} \in \mathcal{V}_m$ and $\mathcal{V}_m$ is the token space at position m. The valid SID set is $\mathcal{C} = \{\mathbf{c}_i \mid i \in \mathcal{I}\}$. Given the history input x, an autoregressive Target with frozen parameters assigns a probability to a candidate SID $\mathbf{y} = (y_1, \ldots, y_M)$ as follows:

$$p_{\mathrm{T}}(\mathbf{y} \mid x) = \prod_{m=1}^{M} p_{\mathrm{T}}(y_m \mid x, \mathbf{y}_{<m}), \tag{1}$$

where $\mathbf{y}_{<m} = (y_1, \ldots, y_{m-1})$. Top-$K$ recommendation seeks the K valid SIDs with the highest Target log-probabilities. Beam search approximates this objective by iteratively extending a bounded set of candidate prefixes (Freitag and Al-Onaizan, 2017), while an item-code trie restricts extensions to those that can form valid SIDs (Rajput et al., 2023). However, this process requires repeated Target backbone computation, resulting in decoding overhead.

2.2 SPECULATIVE DECODING

Speculative decoding reduces sequential Target calls through a draft-then-verify procedure (Leviathan et al., 2023; Xia et al., 2023; Miao et al., 2023). A lightweight Draft proposes a continuation from the accepted prefix, and the Target evaluates the proposed positions together. Generation advances through accepted tokens and resumes at the first rejection. With an exact acceptance rule, this procedure preserves the Target distribution.

For Top-K GR, proposals can be organized into a candidate tree so that the Target scores multiple continuations together before updating the beam (Lin et al., 2025b; Chen et al., 2026). These approaches keep the Target in an online verification loop, while autoregressive drafting still requires successive Draft computations. Furthermore, tree verification relies on architecture-specific attention masks and key–value cache operations. These dependencies motivate learning parallel Draft representations from frozen Targets through a common representation interface.

3 METHOD

3.1 OVERVIEW

Figure 2 illustrates the overall framework. Our framework approximates a frozen Target's recommendation behavior by separating prefix-independent position representations from recurrent prefix states. First, a **Unified Architecture-Agnostic Conditioning** interface provides adapted context features and a root representation, which initializes the recurrent prefix state before any SID token is generated (Section 3.2). Then, the **Parallel Position Draft** uses the adapted context features to compute all SID position representations in a single forward pass (Section 3.3). Next, the **Prefix-Conditioned Head** combines these representations with recurrent prefix states for prefix-conditioned prediction and constrained top-K decoding (Section 3.4). Finally, to transfer autoregressive prediction knowledge to the Parallel Position Draft, we develop a **Two-stage Knowledge Transfer** strategy that first establishes parallel position representations through multi-position SID supervision and then refines them through prefix-conditioned distribution distillation (Section 3.5). During inference, Target features are extracted once per input, with no online Target verification. Following Section 2, the ground-truth next-item SID is $\mathbf{y}^* \in \mathcal{C}$.

3.2 UNIFIED ARCHITECTURE-AGNOSTIC CONDITIONING

To provide consistent conditioning across heterogeneous frozen Targets, we organize Target features by their functional roles: context features $\mathbf{C}$ provide user-history information for parallel position representation learning, while a root representation $\mathbf{r}$ initializes the recurrent prefix state.

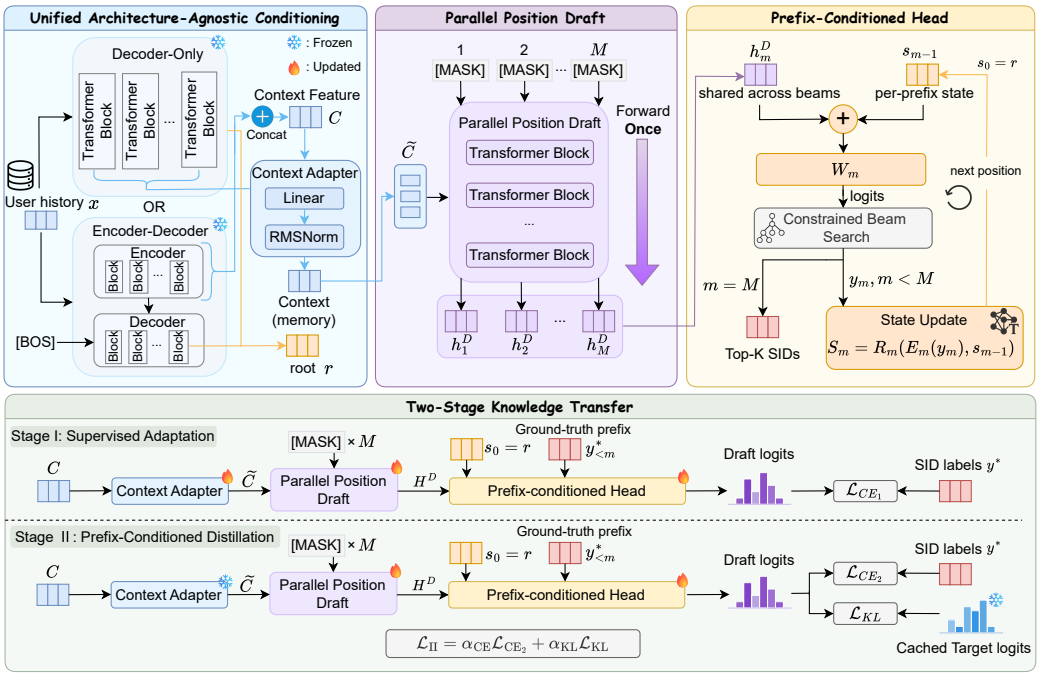

Figure 2: Overview of the proposed framework.

Architecture-Specific Extraction identifies features serving these roles, and then a trainable **Context Adapter** maps the context features into the representation space of the Parallel Position Draft. All Target parameters remain frozen during the training process.

Architecture-Specific Extraction. For decoder-only Targets, a forward pass over the user-history prompt provides the Target context states, and the final-layer hidden state at the last valid prompt position serves as $\mathbf{r}$. For encoder–decoder Targets, Target context states come from the encoder, and $\mathbf{r}$ is the decoder's final-layer hidden state at the start-token position (Vaswani et al., 2017; Raffel et al., 2020; Touvron et al., 2023). In both cases, $\mathbf{r}$ is the Target state used to predict the first SID token given the user history, unifying various backbones. We therefore use this root representation to initialize the recurrent prefix state in the Prefix-Conditioned Head.

Context Adapter. Let $\mathbf{H}^{(\ell_s)} \in \mathbb{R}^{L\times H}$ be the Target context states from layer ℓ_s, with padded context length L and hidden size H. We concatenate S selected layers and project features:

$$\begin{aligned}\mathbf{C} &= \mathrm{Concat}\big(\mathbf{H}^{(\ell_1)},\ldots,\mathbf{H}^{(\ell_S)}\big),\\ \widetilde{\mathbf{C}} &= \mathcal{A}_\phi(\mathbf{C}) = \mathrm{RMSNorm}\big(\mathbf{C}\mathbf{W}_C^\top\big),\end{aligned} \tag{2}$$

where $\mathbf{W}_C \in \mathbb{R}^{H\times(S\times H)}$ is a learnable projection matrix, and ϕ denotes the Context Adapter parameters. The projection aims to map the concatenated context features into the representation space of the Parallel Position Draft, and RMSNorm (Zhang and Sennrich, 2019) normalizes the projected features. We use equal hidden sizes for the Target and Parallel Position Draft, allowing $\mathbf{r} \in \mathbb{R}^H$ to initialize the recurrent prefix state directly. Thus, the adapted context features $\widetilde{\mathbf{C}}$ condition the Parallel Position Draft, while the root representation $\mathbf{r}$ initializes the recurrent prefix state in the Prefix-Conditioned Head. The Context Adapter, Parallel Position Draft, and Prefix-Conditioned Head are trained separately for each Target.

3.3 Parallel Position Draft

The Parallel Position Draft (Gu et al., 2018; Ghazvininejad et al., 2019; Stern et al., 2018) computes representations for all SID positions in a single forward pass, allowing them to be reused across candidate prefixes during decoding. Specifically, it constructs all position representations jointly from the adapted context features and M copies of a shared mask embedding $\mathbf{e}_{\mathrm{mask}}$:

$$\mathbf{H}^{\mathrm{D}} = \mathcal{D}_\theta\big(\widetilde{\mathbf{C}}, \mathbf{e}_{\mathrm{mask}}^{\times M}\big) = \big(\mathbf{h}_1^{\mathrm{D}},\ldots,\mathbf{h}_M^{\mathrm{D}}\big). \tag{3}$$

Each Transformer layer in the GR backbone updates only the mask-position states: queries come from these states, while keys and values include both context and mask-position states. The mask-position state at position m attends to all valid context positions and mask-position states $1, \ldots, m$. Positional encodings place the mask positions after the last valid context position. Since preceding mask-position states are derived from mask embeddings rather than generated-token embeddings, $\mathbf{H}^{\mathrm{D}}$ is independent of candidate prefixes and shared across beams.

3.4 Prefix-Conditioned Head

The Parallel Position Draft computes one representation for each SID position, shared across all candidate prefixes. To predict the next token for a specific prefix, the Prefix-Conditioned Head combines the position representation $\mathbf{h}_m^{\mathrm{D}}$ with a recurrent prefix state $\mathbf{s}_{m-1}$ (Cho et al., 2014) that summarizes the preceding SID tokens. During inference, constrained beam search uses the resulting logits to select valid prefix extensions.

Recurrent Prefix State. The recurrent prefix state is initialized as $\mathbf{s}_0 = \mathbf{r}$, and token prediction and state updates are computed as follows:

$$\begin{aligned} \mathbf{o}_m^{\mathrm{D}} &= \mathbf{W}_m\big(\mathbf{h}_m^{\mathrm{D}} + \mathbf{s}_{m-1}\big), \\ \mathbf{s}_m &= \mathcal{R}_m\big(\mathbf{E}_m(y_m), \mathbf{s}_{m-1}\big), \quad m < M, \end{aligned} \tag{4}$$

where $\mathbf{E}_m$, $\mathcal{R}_m$, and $\mathbf{W}_m$ are position-specific token embeddings, RNN cells, and bias-free output projections. The logits $\mathbf{o}_m^{\mathrm{D}}$ produced by the Prefix-Conditioned Head define the framework's prediction distribution: $p_{\mathrm{D}}(y_m \mid x, \mathbf{y}_{<m}) = \mathrm{Softmax}_{\mathcal{V}_m}(\mathbf{o}_m^{\mathrm{D}})_{y_m}$. Here, p_{D} denotes the prediction distribution produced jointly by the Parallel Position Draft and Prefix-Conditioned Head. Training updates recurrent prefix states with y_m^*; inference maintains a separate prefix state for each generated prefix. Thus, $\mathbf{h}_m^{\mathrm{D}}$ is shared across all beams at position m, while $\mathbf{s}_{m-1}$ carries each beam's prefix information. Implementation details and configurations of the Parallel Position Draft and Prefix-Conditioned Head are provided in **Appendix A.1**.

Inference Process. During inference, we use constrained beam search to generate valid SIDs from the prefix-conditioned logits. At each position, only tokens that extend the current prefix toward a valid SID are allowed. The admissible next-token set is:

$$\mathcal{A}_m(\mathbf{y}_{<m}) = \big\{v \in \mathcal{V}_m \mid \exists \mathbf{c} \in \mathcal{C},\ (\mathbf{y}_{<m}, v) \preceq \mathbf{c}\big\}, \tag{5}$$

where $\preceq$ denotes the prefix relation. Constraints are stored as GPU-resident prefix transitions. Training normalizes over $\mathcal{V}_m$, whereas inference masks inadmissible logits and renormalizes over $\mathcal{A}_m$:

$$\bar{p}_{\mathrm{D}}(v \mid x, \mathbf{y}_{<m}) = \mathrm{Softmax}_{\mathcal{A}_m(\mathbf{y}_{<m})}\big(\mathbf{o}_m^{\mathrm{D}}\big)_v. \tag{6}$$

The full inference procedure is described in detail in **Appendix A.3**, and its computational complexity is analyzed in detail in **Appendix A.4**.

3.5 Two-Stage Knowledge Transfer

To address the autoregressive-to-parallel distillation challenge, we develop a two-stage knowledge transfer strategy (Hinton et al., 2015; Ba and Caruana, 2014; Kim and Rush, 2016). Stage I establishes the Parallel Position Draft's ability to jointly construct all SID position representations from the adapted context features through multi-position SID supervision. Building on this foundation, Stage II refines these representations through prefix-conditioned distribution distillation, transferring the Target's autoregressive prediction knowledge to the Parallel Position Draft. Together, the two stages transfer prefix-dependent prediction knowledge while preserving single-pass computation in the Parallel Position Draft during subsequent inference.

Stage I: Multi-Position Supervised Adaptation. This stage trains the Parallel Position Draft to construct all SID position representations jointly from the adapted context features through supervision at all SID positions. To this end, the Prefix-Conditioned Head combines each representation with the recurrent prefix state computed from the ground-truth prefix for token prediction. For a batch of B examples, the multi-position cross-entropy objective jointly optimizes the Context

Adapter, Parallel Position Draft, and Prefix-Conditioned Head:

$$\mathcal{L}_{\mathrm{I}} = \mathcal{L}_{\mathrm{CE}_2} = -\frac{1}{BM}\sum_{b=1}^{B}\sum_{m=1}^{M}\log p_{\mathrm{D}}\big(y^*_{b,m} \mid x_b, \mathbf{y}^*_{b,<m}\big). \tag{7}$$

Stage II: Prefix-Conditioned Distribution Distillation. Starting from the Stage I checkpoint, we further train the Parallel Position Draft and Prefix-Conditioned Head to match the frozen Target's token distributions under the same ground-truth prefixes. We freeze the Context Adapter to simplify the training pipeline: we precompute and cache the adapted context features, avoiding repeated projection and transfer of concatenated multi-layer features in each epoch. The Context Adapter remains in the final model to process online Target features.

For each ground-truth prefix, we also cache the frozen Target's logits $\mathbf{o}^{\mathrm{T}}_{b,m}$ restricted to $\mathcal{V}_m$. The Prefix-Conditioned Head produces predictions under the same prefix and over the same token space as the Target. Distillation matches these predictions to the Target's corresponding conditional token distributions. Its gradients update both the Prefix-Conditioned Head and the Parallel Position Draft. At temperature $\tau > 0$, define:

$$p^{(\tau)}_{a,b,m} = \mathrm{Softmax}_{\mathcal{V}_m}\big(\mathbf{o}^a_{b,m}/\tau\big), \quad a \in \{\mathrm{T}, \mathrm{D}\}. \tag{8}$$

We minimize the KL divergence between the Target's distribution and the distribution produced by the Prefix-Conditioned Head:

$$\mathcal{L}_{\mathrm{KL}} = \frac{\tau^2}{B\sum_{m=1}^{M} w_m}\sum_{b=1}^{B}\sum_{m=1}^{M} w_m D_{\mathrm{KL}}\big(p^{(\tau)}_{\mathrm{T},b,m} \| p^{(\tau)}_{\mathrm{D},b,m}\big), \tag{9}$$

where $w_m = \exp(-(m-1)/\gamma_{\mathrm{KL}})$, with $\gamma_{\mathrm{KL}} > 0$, emphasizes earlier decisions that determine subsequent search branches. Combining the supervision and distillation objectives, we optimize:

$$\mathcal{L}_{\mathrm{II}} = \alpha_{\mathrm{CE}}\mathcal{L}_{\mathrm{CE}_2} + \alpha_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}}, \tag{10}$$

with detached Target logits and nonnegative loss weights. The complete training algorithm and caching procedure are provided in **Appendix A.2**.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate our method on three real-world datasets, i.e., Amazon Instruments, Amazon Beauty and Amazon Toys (Ni et al., 2019). We follow the preprocessing and data-splitting procedures used in LETTER. More details about the datasets can be seen in **Appendix B.1**.

Baselines. To validate the flexibility of APD-GR, we evaluate it across five heterogeneous GR backbones, including two decoder-only models, LETTER (Wang et al., 2024) and MiniOneRec (Kong et al., 2025), and three encoder–decoder models, TIGER (Rajput et al., 2023), CoFiRec (Wei et al., 2025), and DIGER (Fu et al., 2026). For acceleration baselines, we include the retraining-based NEZHA and the frozen-Target-based AtSpeed. Beam Search and NEZHA are evaluated separately as reference baselines. Further details about the baselines are provided in **Appendix B.2**.

Implementation Details. All experiments are conducted on a platform equipped with Intel Xeon Gold 6426Y CPUs and NVIDIA RTX 5880 Ada Generation 48 GB GPUs. Our implementation uses Python 3.10.20 and PyTorch 2.4.1 with CUDA 12.4. More details are in **Appendix B.3**.

Evaluation Metrics. In the experiments, we adopt the metrics of Top-50 list for evaluation. Specifically, the Hit Rate (H@10) and Normalized Discounted Cumulative Gain (N@10) (Cremonesi et al., 2010; Järvelin and Kekäläinen, 2002) are mainly used.

Efficiency Evaluation. We measure mean per-example latency on an NVIDIA RTX 5880 Ada GPU in BF16, using batch size 1 and beam width 50. Timing covers Target feature extraction and constrained generation over 1,000 test examples after five warm-up runs.

Table 1: Overall performance comparison on Instruments, Beauty, and Toys across five backbones. The Ratio row reports APD-GR's recommendation performance retained relative to the backbone and the corresponding speedup.

Method	Instruments					Beauty					Toys				
	H@10	**H@20**	**N@10**	**N@20**	**Time**	**H@10**	**H@20**	**N@10**	**N@20**	**Time**	**H@10**	**H@20**	**N@10**	**N@20**	**Time**
Beam Search	0.0772	0.0915	0.0633	0.0669	192.30	0.0671	0.0996	0.0380	0.0462	209.44	0.0554	0.0714	0.0340	0.0381	204.74
NEZHA	0.0921	0.1157	0.0704	0.0763	10.67	0.0561	0.0860	0.0320	0.0393	10.78	0.0529	0.0765	0.0299	0.0356	10.82
LETTER	0.1130	0.1409	0.0868	0.0938	224.31	0.0725	0.1027	0.0419	0.0495	229.90	0.0474	0.0662	0.0288	0.0335	225.20
+*AtSpeed*	0.1134	0.1410	0.0870	0.0939	100.07	0.0727	0.1026	0.0418	0.0494	107.75	0.0479	0.0678	0.0289	0.0339	104.78
+*APD-GR*	0.1087	0.1373	0.0821	0.0893	19.37	0.0719	0.1016	0.0412	0.0486	20.26	0.0451	0.0622	0.0273	0.0316	19.13
Ratio	96%	97%	95%	95%	11.58×	99%	99%	98%	98%	11.35×	95%	94%	95%	94%	11.77×
MiniOneRec	0.1016	0.1218	0.0804	0.0855	233.09	0.0512	0.0749	0.0282	0.0341	237.56	0.0555	0.0793	0.0303	0.0363	238.78
+*APD-GR*	0.0927	0.1144	0.0733	0.0788	19.03	0.0434	0.0660	0.0230	0.0287	18.99	0.0439	0.0647	0.0228	0.0281	18.97
Ratio	91%	94%	91%	92%	12.25×	85%	88%	82%	84%	12.51×	79%	82%	75%	77%	12.59×
TIGER	0.1090	0.1378	0.0816	0.0888	63.31	0.0677	0.0993	0.0365	0.0444	63.73	0.0662	0.0964	0.0370	0.0446	62.86
+*APD-GR*	0.1037	0.1322	0.0777	0.0849	7.91	0.0613	0.0919	0.0329	0.0406	7.90	0.0625	0.0895	0.0346	0.0414	7.85
Ratio	95%	96%	95%	96%	8.00×	91%	93%	90%	91%	8.06×	94%	93%	93%	93%	8.01×
CoFiRec	0.1115	0.1417	0.0820	0.0895	65.55	0.0647	0.0948	0.0345	0.0421	63.19	0.0492	0.0752	0.0254	0.0319	60.89
+*APD-GR*	0.1092	0.1385	0.0800	0.0874	8.18	0.0588	0.0887	0.0315	0.0390	8.10	0.0407	0.0597	0.0209	0.0257	8.10
Ratio	98%	98%	98%	98%	8.01×	91%	94%	91%	93%	7.80×	83%	79%	82%	80%	7.52×
DIGER	0.1119	0.1442	0.0818	0.0900	66.52	0.0672	0.1011	0.0363	0.0449	68.00	0.0649	0.0957	0.0340	0.0418	65.31
+*APD-GR*	0.1090	0.1398	0.0801	0.0878	10.52	0.0619	0.0927	0.0335	0.0412	10.97	0.0552	0.0837	0.0280	0.0352	10.31
Ratio	97%	97%	98%	98%	6.32×	92%	92%	92%	92%	6.20×	85%	87%	82%	84%	6.34×

4.2 Overall Performance

To validate the effectiveness and flexibility of APD-GR, we compare it with backbone GR models on three datasets in Table 1. Overall, APD-GR reduces inference cost while maintaining competitive performance across recommendation backbones and datasets, demonstrating strong generality and flexibility. These results show that APD-GR achieves a favorable trade-off between recommendation quality and efficiency, making it practical for accelerating sequential recommendation. We further analyze the results below in detail.

Overall Comparison. APD-GR generally preserves the performance of LETTER, TIGER, and DIGER, whereas its performance drop is more evident on MiniOneRec, especially on Toys. This may be because MiniOneRec relies on the original autoregressive decoding path, while Toys involves diverse user interests and item semantics. Such complexity may reduce the consistency between draft and Target predictions, leading to larger ranking errors. Nevertheless, APD-GR substantially reduces inference cost with limited performance loss in this setting. The speedup is greater on decoder-only backbones. This may be attributed to their larger model sizes.

Baseline Performance. AtSpeed achieves performance comparable to, and sometimes slightly better than, LETTER across the three datasets, suggesting that its draft-model alignment and relaxed verification can effectively preserve the target model outputs. In contrast, NEZHA exhibits clear dataset-dependent performance: it outperforms Beam Search on Instruments but remains weaker than LETTER, performs worse on Beauty, and becomes more competitive on Toys. A possible reason is that NEZHA relies only on a lightweight draft head for self-drafting. Its ability to approximate the target model distribution is therefore more sensitive to the data distribution, especially when user–item interactions are complex. Thus, NEZHA provides high efficiency, but its recommendation quality is less stable than methods with an explicitly aligned draft model.

Inference efficiency. NEZHA is consistently faster than APD-GR, which is expected from its architecture. It integrates a lightweight draft head into the target model and uses model-free verification, avoiding the execution of an external draft model and the associated verification overhead. By contrast, AtSpeed is slower than APD-GR because it uses an independent draft model and performs Top-K candidate expansion, alignment, and verification. These operations partly offset the benefit of speculative decoding. Therefore, AtSpeed favors near-lossless recommendation performance,

Table 2: Ablation study on the Beauty dataset with the LETTER and TIGER backbones.

Method	Decoder-only				Encoder-decoder			
	H@5	H@10	N@5	N@10	H@5	H@10	N@5	N@10
APD-GR	0.0492	0.0719	0.0338	0.0412	0.0384	0.0613	0.0256	0.0329
w/ $\mathcal{L}_{\mathrm{KL}}$ in $\mathcal{L}_{\mathrm{I}}$	0.0472	0.0709	0.0329	0.0406	0.0390	0.0607	0.0256	0.0326
w/o Stage 2	0.0394	0.0612	0.0268	0.0338	0.0318	0.0514	0.0203	0.0266
w/o $\mathcal{L}_{\mathrm{KL}}$ in $\mathcal{L}_{\mathrm{II}}$	0.0449	0.0641	0.0307	0.0368	0.0350	0.0570	0.0234	0.0305
w/o $\mathcal{L}_{\mathrm{CE}_2}$ in $\mathcal{L}_{\mathrm{II}}$	0.0134	0.0242	0.0084	0.0119	0.0363	0.0576	0.0234	0.0303
w/ random s_0 initialization	0.0419	0.0637	0.0287	0.0357	0.0359	0.0564	0.0238	0.0304
w/o state updates	0.0377	0.0568	0.0265	0.0326	0.0295	0.0472	0.0196	0.0253

whereas APD-GR offers a more practical balance between accuracy preservation and inference efficiency. More details about the training and inference latency are provided in **Appendix D**.

4.3 Ablation Study

Table 2 reports the ablation results on the Beauty dataset. First, *w/o Stage 2* performs worse than APD-GR for both decoder-only and encoder–decoder models. This shows that Stage 2 is essential for refining draft predictions after initial optimization. Second, *w/* $\mathcal{L}_{\mathrm{KL}}$ *in* $\mathcal{L}_{\mathrm{I}}$ leads to a slight but consistent degradation, suggesting that introducing prefix-conditioned distillation before the Draft learns effective parallel representations may hinder its parallel prediction capability. In contrast, removing either loss term in $\mathcal{L}_{\mathrm{II}}$ causes a larger performance drop. In particular, *w/o* $\mathcal{L}_{\mathrm{CE}_2}$ *in* $\mathcal{L}_{\mathrm{II}}$ yields the worst results among all loss-related variants, which demonstrates that the CE objective is crucial for learning accurate draft tokens in the second stage. Meanwhile, the inferior performance of *w/o* $\mathcal{L}_{\mathrm{KL}}$ *in* $\mathcal{L}_{\mathrm{II}}$ verifies that distribution alignment is also necessary during this refinement stage.

We further study the state-related designs. Compared with APD-GR, *w/ random* s_0 *initialization* consistently reduces performance across different model architectures, showing that the proposed initialization offers informative states for subsequent draft generation. More importantly, *w/o state updates* causes a substantial performance drop under both GR architectures. This indicates that updating states during generation is critical for preserving the evolving sequential context throughout the decoding process. Overall, the performance decline of each *w/o* variant confirms the necessity of the corresponding component, while their combination enables APD-GR to achieve the best results.

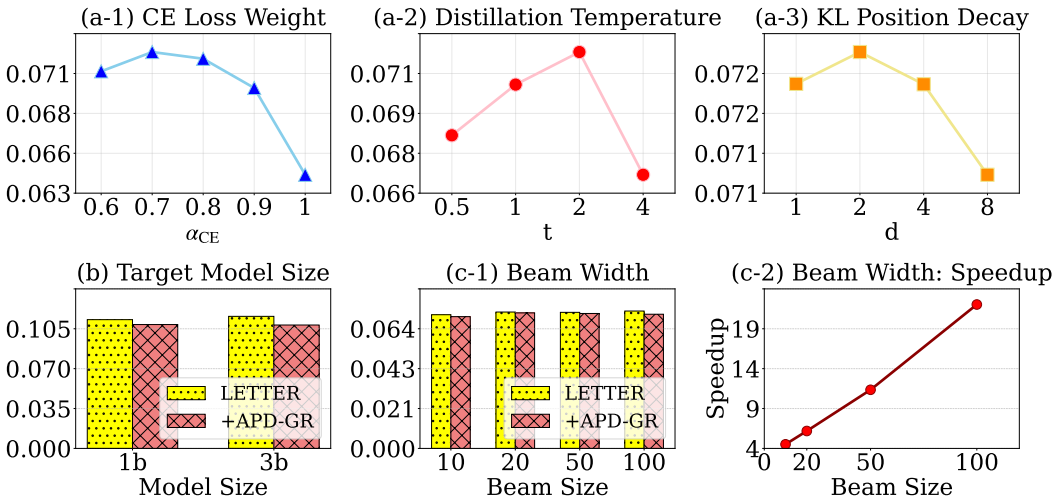

Figure 3: Effects of hyperparameters, Target model size, and beam width on APD-GR.

4.4 Hyper-parameter Analysis

We investigate three hyper-parameters in Figure 3. As shown in (a-1), performance first improves as the CE loss weight α_{CE} increases, but declines when it becomes too large. A moderate CE weight improves draft learning, whereas an excessive weight dominates the objective and weakens other

supervision signals. Figure 3(a-2) shows that a moderate distillation temperature t achieves the best performance. A small temperature produces overly sharp targets and provides limited distillation knowledge, while a large temperature over-smooths the target distribution and weakens ranking signals. Finally, Figure 3(a-3) shows that a small KL position decay d is preferred. A proper decay balances KL supervision across positions, whereas an overly large decay excessively weakens supervision on later positions and harms final recommendation quality.

4.5 Scalability Analysis

We further investigate the scalability of APD-GR with respect to target model size and beam width. Figure 3(b) shows that APD-GR consistently maintains performance close to LETTER for both target model sizes. As the target model becomes larger, both methods achieve better recommendation performance, while the performance gap between APD-GR and LETTER remains small. This indicates that APD-GR can effectively preserve the capability of larger GR models.

Figures 3(c-1) and (c-2) study the effect of beam width. As the beam width increases, APD-GR consistently follows the performance trend of LETTER, indicating that it remains effective under different decoding budgets. More importantly, the speedup of APD-GR increases substantially with a larger beam width. This is because a wider beam requires the baseline to process more candidate sequences and incurs more target-model calls, while APD-GR can reduce this repeated decoding cost through draft generation. Therefore, APD-GR is particularly beneficial for high-quality recommendation settings that require a large beam width. More numerical details are in **Appendix C**.

5 Related Work

Generative Recommendation. Generative recommendation represents items as discrete token sequences and generates the next item's SID conditioned on user histories. TIGER (Rajput et al., 2023) constructs hierarchical SIDs through residual quantization, while SEATER (Si et al., 2024) learns semantic tree-structured identifiers with contrastive objectives. LC-Rec (Zheng et al., 2024) integrates language and collaborative semantics through semantic indexing and alignment tuning; LETTER (Wang et al., 2024) incorporates collaborative regularization and code assignment diversity into learnable tokenization. ETEGRec (Liu et al., 2025) further couples item tokenization and generative recommendation through alignment objectives and alternating optimization. These approaches improve identifier learning and recommendation quality, while APD-GR accelerates SID generation from frozen GR backbones.

Efficient Inference for Generative Recommendation. Existing approaches differ in whether they retrain the Target. SETRec (Lin et al., 2025a) and RPG (Hou et al., 2025) redesign identifiers and objectives for parallel prediction; EARN (Yang et al., 2025) uses register tokens to reduce later-layer computation and KV-cache usage; NEZHA (Wang et al., 2025) integrates positional placeholders and a self-drafting head. These changes require Target training and cannot directly accelerate already-trained models. AtSpeed (Lin et al., 2025b) and PAD-Rec (Chen et al., 2026) train external Drafts while freezing the Target. This avoids Target retraining but requires Draft training. AtSpeed improves top-K alignment under strict or relaxed verification; PAD-Rec uses within-item and draft-step position embeddings for list-wise recommendation. Both retain autoregressive drafting and online verification with backbone-specific attention and cache operations. APD-GR instead combines two-stage knowledge transfer with unified conditioning across heterogeneous frozen Targets, computing position representations in one Draft pass and decoding through a lightweight Prefix-Conditioned Head without online Target verification.

6 Conclusion

We presented APD-GR, which accelerates frozen heterogeneous generative recommenders by combining single-pass position representations with lightweight prefix conditioning. Our framework supports decoder-only and encoder–decoder Targets while eliminating repeated Draft computation and online Target verification. Across three datasets and five backbones, APD-GR achieves an average end-to-end speedup of $9.22\times$ while largely preserving recommendation performance, with ablations validating its components. Future work will improve quality retention across backbones.

AI USE STATEMENT

We used generative AI tools to assist with drafting, revising, and polishing the manuscript, including improving clarity, organization, linguistic consistency, and conciseness. We also used these tools to identify relevant literature, summarize related work, and assist with describing and interpreting author-provided experimental results. The authors reviewed and revised the AI-assisted text, checked the cited literature against the original sources, and verified quantitative statements against the experimental records. The authors take full responsibility for the final manuscript, including its claims, results, interpretations, and references.

REPRODUCIBILITY STATEMENT

We provide detailed experimental settings, implementation details, and additional results in the appendix to facilitate reproducibility. These include dataset preprocessing, backbone configurations, training hyperparameters, and inference settings. An anonymous repository has been created for code release:https://anonymous.4open.science/r/anonymous-code-release-DC63.

REFERENCES

Lei Jimmy Ba and Rich Caruana. 2014. Do Deep Nets Really Need to be Deep? arXiv:1312.6184 [cs.LG] https://arxiv.org/abs/1312.6184

Jiaju Chen, Chongming Gao, Chenxiao Fan, Haoyan Liu, Qingpeng Cai, Peng Jiang, and Xiangnan He. 2026. Position-Aware Drafting for Inference Acceleration in LLM-Based Generative List-Wise Recommendation. arXiv:2604.27747 [cs.IR] https://arxiv.org/abs/2604.27747

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. 1724–1734. https://aclanthology.org/D14-1179/

Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems* (Barcelona, Spain) *(RecSys '10)*. Association for Computing Machinery, New York, NY, USA, 39–46. doi:10.1145/1864708.1864721

Markus Freitag and Yaser Al-Onaizan. 2017. Beam Search Strategies for Neural Machine Translation. In *Proceedings of the First Workshop on Neural Machine Translation*. Association for Computational Linguistics, 56–60. doi:10.18653/v1/w17-3207

Junchen Fu, Xuri Ge, Alexandros Karatzoglou, Ioannis Arapakis, Suzan Verberne, Joemon M. Jose, and Zhaochun Ren. 2026. Differentiable Semantic ID for Generative Recommendation. arXiv:2601.19711 [cs.IR] https://arxiv.org/abs/2601.19711

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-Predict: Parallel Decoding of Conditional Masked Language Models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. 6112–6121. https://aclanthology.org/D19-1633/

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-Autoregressive Neural Machine Translation. In *International Conference on Learning Representations*. https://arxiv.org/abs/1711.02281

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. arXiv:1503.02531 [stat.ML] https://arxiv.org/abs/1503.02531

Yupeng Hou, Jiacheng Li, Ashley Shin, Jinsung Jeon, Abhishek Santhanam, Wei Shao, Kaveh Hassani, Ning Yao, and Julian McAuley. 2025. Generating Long Semantic IDs in Parallel for Recommendation. arXiv:2506.05781 [cs.IR] https://arxiv.org/abs/2506.05781

Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-Based Evaluation of IR Techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446. doi:10.1145/582415.582418

Yoon Kim and Alexander M. Rush. 2016. Sequence-Level Knowledge Distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Jian Su, Kevin Duh, and Xavier Carreras (Eds.). Association for Computational Linguistics, Austin, Texas, 1317–1327. doi:10.18653/v1/D16-1139

Xiaoyu Kong, Leheng Sheng, Junfei Tan, Yuxin Chen, Jiancan Wu, An Zhang, Xiang Wang, and Xiangnan He. 2025. MiniOneRec: An Open-Source Framework for Scaling Generative Recommendation. arXiv:2510.24431 [cs.IR] https://arxiv.org/abs/2510.24431

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast Inference from Transformers via Speculative Decoding. In *International Conference on Machine Learning*. https://arxiv.org/abs/2211.17192

Xinyu Lin, Haihan Shi, Wenjie Wang, Fuli Feng, Qifan Wang, See-Kiong Ng, and Tat-Seng Chua. 2025a. Order-agnostic Identifier for Large Language Model-based Generative Recommendation. arXiv:2502.10833 [cs.IR] https://arxiv.org/abs/2502.10833

Xinyu Lin, Chaoqun Yang, Wenjie Wang, Yongqi Li, Cunxiao Du, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. 2025b. Efficient Inference for Large Language Model-based Generative Recommendation. In *International Conference on Learning Representations*. https://arxiv.org/abs/2410.05165

Enze Liu, Bowen Zheng, Cheng Ling, Lantao Hu, Han Li, and Wayne Xin Zhao. 2025. Generative Recommender with End-to-End Learnable Item Tokenization. arXiv:2409.05546 [cs.IR] https://arxiv.org/abs/2409.05546

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2023. SpecInfer: Accelerating Generative Large Language Model Serving with Tree-Based Speculative Inference and Verification. arXiv:2305.09781 [cs.LG] https://arxiv.org/abs/2305.09781

Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations Using Distantly-Labeled Reviews and Fine-Grained Aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. 188–197. https://aclanthology.org/D19-1018/

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67. https://jmlr.org/papers/v21/20-074.html

Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan H. Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. 2023. Recommender Systems with Generative Retrieval. In *Advances in Neural Information Processing Systems*, Vol. 36. https://arxiv.org/abs/2305.05065

Zihua Si, Zhongxiang Sun, Jiale Chen, Guozhang Chen, Xiaoxue Zang, Kai Zheng, Yang Song, Xiao Zhang, Jun Xu, and Kun Gai. 2024. Generative Retrieval with Semantic Tree-Structured Item Identifiers via Contrastive Learning. arXiv:2309.13375 [cs.IR] https://arxiv.org/abs/2309.13375

Anima Singh, Trung Vu, Nikhil Mehta, Raghunandan Keshavan, Maheswaran Sathiamoorthy, Yilin Zheng, Lichan Hong, Lukasz Heldt, Li Wei, Devansh Tandon, Ed H. Chi, and Xinyang Yi. 2024. Better Generalization with Semantic IDs: A Case Study in Ranking for Recommendations. In *Proceedings of the 18th ACM Conference on Recommender Systems*. https://arxiv.org/abs/2306.08121

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise Parallel Decoding for Deep Autoregressive Models. In *Advances in Neural Information Processing Systems*, Vol. 31. https://arxiv.org/abs/1811.03115

Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. 2022. Transformer Memory as a Differentiable Search Index. In *Advances in Neural Information Processing Systems*, Vol. 35. https://arxiv.org/abs/2202.06991

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, Vol. 30. https://arxiv.org/abs/1706.03762

Wenjie Wang, Honghui Bao, Xinyu Lin, Jizhi Zhang, Yongqi Li, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. 2024. Learnable Item Tokenization for Generative Recommendation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. https://arxiv.org/abs/2405.07314

Yujing Wang, Yingyan Hou, Haonan Wang, Ziming Miao, Shibin Wu, Hao Sun, Qi Chen, Yuqing Xia, Chengmin Chi, Guoshuai Zhao, Zheng Liu, Xing Xie, Hao Allen Sun, Weiwei Deng, Qi Zhang, and Mao Yang. 2022. A Neural Corpus Indexer for Document Retrieval. In *Advances in Neural Information Processing Systems*, Vol. 35. https://arxiv.org/abs/2206.02743

Yejing Wang, Shengyu Zhou, Jinyu Lu, Ziwei Liu, Langming Liu, Maolin Wang, Wenlin Zhang, Feng Li, Wenbo Su, Pengjie Wang, Jian Xu, and Xiangyu Zhao. 2025. NEZHA: A Zero-sacrifice and Hyperspeed Decoding Architecture for Generative Recommendations. arXiv:2511.18793 [cs.IR] https://arxiv.org/abs/2511.18793

Tianxin Wei, Xuying Ning, Xuxing Chen, Ruizhong Qiu, Yupeng Hou, Yan Xie, Shuang Yang, Zhigang Hua, and Jingrui He. 2025. CoFiRec: Coarse-to-Fine Tokenization for Generative Recommendation. arXiv:2511.22707 [cs.IR] https://arxiv.org/abs/2511.22707

Heming Xia, Tao Ge, Furu Wei, and Zhifang Wang. 2023. Speculative Decoding: Exploiting Speculative Execution for Accelerating Seq2seq Generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 3909–3925. https://aclanthology.org/2023.findings-emnlp.257/

Chaoqun Yang, Xinyu Lin, Wenjie Wang, Yongqi Li, Teng Sun, Xianjing Han, and Tat-Seng Chua. 2025. EARN: Efficient Inference Acceleration for LLM-based Generative Recommendation by Register Tokens. arXiv:2507.00715 [cs.IR] https://arxiv.org/abs/2507.00715

Biao Zhang and Rico Sennrich. 2019. Root Mean Square Layer Normalization. In *Advances in Neural Information Processing Systems*, Vol. 32. https://arxiv.org/abs/1910.07467

Bowen Zheng, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, Ming Chen, and Ji-Rong Wen. 2024. Adapting Large Language Models by Integrating Collaborative Semantics for Recommendation. arXiv:2311.09049 [cs.IR] https://arxiv.org/abs/2311.09049

A Additional Implementation Details

This appendix supplements Section 3 with module configurations, training and decoding pseudocode, and an inference-cost analysis. Losses, token spaces, and conditional probabilities retain their definitions in the main text.

A.1 Parallel Position Draft and Prefix-Conditioned Head

The Parallel Position Draft uses Llama-style Transformer blocks for both Target architecture families to maintain a consistent Draft architecture across backbones. Table 3 specifies the configurations used in the main Beauty experiments; these are Draft dimensions, not Target layer counts. The shared mask embedding is included among the trainable Draft parameters. Context padding masks and valid lengths accompany the extracted features so that padded positions are excluded from attention and SID position indices follow the last valid context position.

APD-GR's Prefix-Conditioned Head is a separate module from the Target and uses its own position-specific embedding tables, RNN cells, and bias-free output projections as defined in Equation 4. A state transition is required only after positions $m < M$; there is no next-token prediction after the final SID token. The output projection produces logits over the position vocabulary $\mathcal{V}_m$. At inference time, the prefix constraint is applied to these logits before normalization over $\mathcal{A}_m$. This distinguishes the projection's output space from the prefix-dependent set of admissible extensions.

Table 3: Parallel Position Draft configurations for the main Beauty experiments.

Target	Draft layers	Hidden dim.	FFN dim.	Heads
LETTER-LC-Rec	5	2,048	8,192	32
LETTER-TIGER	1	128	512	4
MiniOneRec	9	1,536	8,960	12
CoFiRec	1	128	512	4
DIGER	2	128	512	4

A.2 Two-Stage Knowledge Transfer

Algorithm 1 makes the cache dependencies and parameter updates explicit. Here, $\mathrm{Extract}_T(x)$ denotes the architecture-specific extraction in Section 3.2, including the root representation needed for the first SID prediction. All Target-derived tensors are detached. Ground-truth SIDs are read from the training data, not generated by the Target. Cached context features are trimmed to their valid lengths; the data collator reconstructs padding masks when forming minibatches. These details are suppressed in the pseudocode.

We write $\mathcal{H}_\psi(\mathbf{H}^{\mathrm{D}}, \mathbf{r}, \mathbf{y}^*)$ for teacher-forced Head evaluation: initialize $\mathbf{s}_0 = \mathbf{r}$, score position m before consuming y^*_m, and update the state with y^*_m only when $m < M$, following Equation 4. Thus, the logits at position m depend on $\mathbf{y}^*_{<m}$, not on the current label. This shorthand returns all position logits and does not denote an additional network.

The two-stage design also reduces the cost of caching features for adapter training. While the Context Adapter is trainable, the cache must retain the concatenated Target hidden states from the selected layers. We therefore train Stage I on a randomly sampled 50% subset of the training data to limit this storage overhead, while retaining full training-set coverage in the second stage. In Stage II, we freeze the learned adapter and construct a cache for the full training set, storing the adapted context features instead of the concatenated multi-layer states. For S selected layers, this reduces the cached context dimension from SH to H, lowering context-feature storage and data-transfer costs when $S > 1$ and avoiding repeated adapter computation during Stage II training. Root representations and teacher logits are cached separately; teacher and Head logits share the same position vocabulary and token ordering. The adapter remains part of online inference.

A.3 Constrained Beam-Search Decoding

Algorithm 2 presents the single-request search logic, omitting batching and output padding. It distinguishes beam width W from the requested recommendation count K, with $W \geq K$. A beam

Algorithm 1 Two-stage training of APD-GR

Require: Frozen Target T; training set $\mathcal{S} = \{(x_i, \mathbf{y}_i^*)\}_{i=1}^N$; epoch counts $E_{\mathrm{I}}, E_{\mathrm{II}}$
Require: Stage I sample indices $\mathcal{I}_{\mathrm{I}} \subseteq \{1, \dots, N\}$; settings $\tau, \gamma_{\mathrm{KL}}, \alpha_{\mathrm{CE}}, \alpha_{\mathrm{KL}}$
Ensure: Trained Context Adapter $\mathcal{A}_\phi$, Draft $\mathcal{D}_\theta$, and Head $\mathcal{H}_\psi$
Initialize ϕ, θ, ψ; keep T frozen in evaluation mode
Initialize caches $\mathcal{Q}_{\mathrm{I}} \leftarrow$, $\mathcal{Q}_{\mathrm{II}} \leftarrow$
for $i \in \mathcal{I}_{\mathrm{I}}$ **do**
    $(\mathbf{C}_i, \mathbf{r}_i) \leftarrow \mathrm{Extract}_T(x_i)$ without gradients
    Append $(\mathbf{C}_i, \mathbf{r}_i, \mathbf{y}_i^*)$ to $\mathcal{Q}_{\mathrm{I}}$
end for
for $e = 1, \dots, E_{\mathrm{I}}$ **do**
    for each minibatch from $\mathcal{Q}_{\mathrm{I}}$ **do**
        $\widetilde{\mathbf{C}} \leftarrow \mathcal{A}_\phi(\mathbf{C})$
        $\mathbf{H}^{\mathrm{D}} \leftarrow \mathcal{D}_\theta(\widetilde{\mathbf{C}}, \mathbf{e}_{\mathrm{mask}}^{\times M})$
        $\mathbf{o}_{1:M}^{\mathrm{D}} \leftarrow \mathcal{H}_\psi(\mathbf{H}^{\mathrm{D}}, \mathbf{r}, \mathbf{y}^*)$
        Update ϕ, θ, ψ using $\mathcal{L}_{\mathrm{I}}$ in Equation 7
    end for
end for
Freeze ϕ at the Stage I checkpoint
for $i = 1, \dots, N$ **do**
    Re-extract $(\mathbf{C}_i, \mathbf{r}_i) \leftarrow \mathrm{Extract}_T(x_i)$ without gradients
    $\widetilde{\mathbf{C}}_i \leftarrow \mathcal{A}_\phi(\mathbf{C}_i)$ without gradients
    Cache detached $\mathbf{o}_{i,m}^{\mathrm{T}}$ on $\mathcal{V}_m$ given $(x_i, \mathbf{y}_{i,<m}^*)$, for all m
    Append $(\widetilde{\mathbf{C}}_i, \mathbf{r}_i, \mathbf{y}_i^*, \mathbf{o}_{i,1:M}^{\mathrm{T}})$ to $\mathcal{Q}_{\mathrm{II}}$
end for
for $e = 1, \dots, E_{\mathrm{II}}$ **do**
    for each minibatch from $\mathcal{Q}_{\mathrm{II}}$ **do**
        $\mathbf{H}^{\mathrm{D}} \leftarrow \mathcal{D}_\theta(\widetilde{\mathbf{C}}, \mathbf{e}_{\mathrm{mask}}^{\times M})$
        $\mathbf{o}_{1:M}^{\mathrm{D}} \leftarrow \mathcal{H}_\psi(\mathbf{H}^{\mathrm{D}}, \mathbf{r}, \mathbf{y}^*)$
        Update only θ, ψ using $\mathcal{L}_{\mathrm{II}}$ in Equation 10
    end for
end for
return $\mathcal{A}_\phi, \mathcal{D}_\theta, \mathcal{H}_\psi$

entry stores a prefix, its cumulative constrained log-probability, and its recurrent state. At position m, each beam carries $\mathbf{s}_{m-1}$, obtained by recurrently incorporating its preceding $m-1$ tokens starting from $\mathbf{s}_0 = \mathbf{r}$. The Head combines this state with the shared Draft representation $\mathbf{h}_m^{\mathrm{D}}$ to score the next token. After pruning, each retained extension v updates its own parent state to $\mathbf{s}_m$ for the next position, provided $m < M$. The GPU-resident trie supplies admissible next tokens. Pruning operates jointly over extensions of all current beams, rather than selecting W children independently for each parent, thereby retaining the globally highest-scoring candidates at each position.

Here, ϵ denotes the empty prefix and $\mathbf{u} \parallel v$ denotes token concatenation. At each position, inadmissible logits are masked before log-softmax normalization, and candidate scores are obtained by adding the resulting constrained log-probabilities to their parent scores. After global pruning, each retained child computes its next prefix state from the corresponding parent state and selected token, without modifying the parent state in place. No recurrent update is needed after the final SID position. Finite-width beam search approximates the top-K ranking under the constrained distribution in Equation 6, since some candidate prefixes are discarded during pruning.

A.4 Computational Complexity

We analyze neural computation per input, focusing on the forward-pass costs of the main computational components, excluding candidate lookup, normalization, pruning, and memory movement. Let L be the padded context length, M the SID length, H the shared Target–Draft hidden size, S the number of selected Target layers, and D_D the Draft depth. APD-GR incurs Target feature-extraction cost C_T^{feat}, adapter cost C_A, and one parallel Draft pass C_D^{par}. For an encoder–decoder Target, feature extraction includes both the encoder computation and the decoder start-state computation.

Algorithm 2 Constrained top-K decoding with APD-GR

Require: Input x; frozen T; trained $\mathcal{A}_\phi, \mathcal{D}_\theta, \mathcal{H}_\psi$
Require: Trie of a nonempty valid-SID set $\mathcal{C}$; SID length M; beam width $W \geq K \geq 1$
Ensure: Up to K complete valid SIDs, ordered by cumulative score

$(\mathbf{C}, \mathbf{r}) \leftarrow \mathrm{Extract}_T(x)$
$\widetilde{\mathbf{C}} \leftarrow \mathcal{A}_\phi(\mathbf{C})$
$\mathbf{Z}_{\mathrm{mask}} \leftarrow [\mathbf{e}_{\mathrm{mask}}, \ldots, \mathbf{e}_{\mathrm{mask}}]$ ▷ M copies of the shared mask embedding
$\mathbf{H}^{\mathrm{D}} \leftarrow \mathcal{D}_\theta(\widetilde{\mathbf{C}}, \mathbf{Z}_{\mathrm{mask}})$ ▷ One parallel Draft pass
$\mathcal{B}_0 \leftarrow \{(\epsilon, 0, \mathbf{r})\}$ ▷ Empty prefix, score, state
for $m = 1, \ldots, M$ **do**
    $\mathcal{E} \leftarrow$ ▷ Candidate extensions with parent states
    for each $(\mathbf{u}, \ell, \mathbf{s}_{m-1})$ in $\mathcal{B}_{m-1}$ **do**
        $\mathbf{o}_m^{\mathrm{D}} \leftarrow \mathbf{W}_m(\mathbf{h}_m^{\mathrm{D}} + \mathbf{s}_{m-1})$
        Read $\mathcal{A}_m(\mathbf{u})$ from the trie
        $\lambda(v) \leftarrow \log \mathrm{Softmax}_{\mathcal{A}_m(\mathbf{u})}(\mathbf{o}_m^{\mathrm{D}})_v$, for $v \in \mathcal{A}_m(\mathbf{u})$
        for each $v \in \mathcal{A}_m(\mathbf{u})$ **do**
            Append $(\mathbf{u} \parallel v, \ell + \lambda(v), \mathbf{s}_{m-1}, v)$ to $\mathcal{E}$
        end for
    end for
    $\mathcal{P} \leftarrow$ the $\min(W, |\mathcal{E}|)$ highest-scoring entries of $\mathcal{E}$
    $\mathcal{B}_m \leftarrow$
    for each $(\mathbf{u}', \ell', \mathbf{s}_{m-1}^{\mathrm{parent}}, v)$ in $\mathcal{P}$ **do**
        $\mathbf{s}' \leftarrow \mathbf{s}_{m-1}^{\mathrm{parent}}$
        if $m < M$ **then**
            $\mathbf{s}_m \leftarrow \mathcal{R}_m(\mathbf{E}_m(v), \mathbf{s}_{m-1}^{\mathrm{parent}})$; $\mathbf{s}' \leftarrow \mathbf{s}_m$
        end if
        Append $(\mathbf{u}', \ell', \mathbf{s}')$ to $\mathcal{B}_m$
    end for
end for
return The $\min(K, |\mathcal{B}_M|)$ highest-scoring SIDs and scores in $\mathcal{B}_M$

Table 4: Neural computation per request. Initialization and subsequent decoding costs are counted separately.

Decoding method	Computation
Target beam search	$C_{\mathrm{init}}^{\mathrm{BS}} + O(MWc_T)$
Autoregressive draft–verify	$C_{\mathrm{init}}^{\mathrm{DV}} + O(N_D c_D) + C_T^{\mathrm{verify}}$
APD-GR	$C_T^{\mathrm{feat}} + C_A + C_D^{\mathrm{par}} + C_{\mathrm{Head}}$

In Table 4, $C_{\mathrm{init}}^{\mathrm{BS}}$ and $C_{\mathrm{init}}^{\mathrm{DV}}$ denote method-specific initialization costs, including any context encoding or prompt prefill; they are not assumed equal to APD-GR's feature-extraction cost. For fixed input and decoding lengths, c_T and c_D upper-bound the cost of one subsequent autoregressive prefix evaluation, including its output projection, under the method's caching strategy. These costs can depend on context and prefix lengths. N_D counts all subsequent Draft prefix evaluations, including repeated proposals, and C_T^{verify} aggregates all subsequent Target verification calls. Computation already included in initialization is not counted again. The draft–verify expression is a cost decomposition, not a bound determined solely by SID length or beam width.

The Context Adapter projects SH input features to H output features at each context position, giving $C_A = O(SLH^2)$. With feed-forward dimension H_{ff}, the implemented parallel Draft has cost

$$C_D^{\mathrm{par}} = O\big(D_D\left[(L+M)H^2 + M(L+M)H + MHH_{\mathrm{ff}}\right]\big). \tag{11}$$

Each layer projects context and mask states to keys and values, while queries and attention outputs are computed only for the M mask positions. This gives the projection term $O((L+M)H^2)$ and attention term $O(M(L+M)H)$. Only mask states pass through the feed-forward network, costing $O(MHH_{\mathrm{ff}})$. Context states are not recurrently updated across Draft layers, but their key/value projections are recomputed in each layer. Normalization and positional encoding contribute lower-order terms, which do not affect the dominant complexity.

Let b_m be the number of retained beams after position m, with $b_0 = 1$ and $b_m \le W$. Dense Head projections score all $|\mathcal{V}_m|$ tokens before masking, whereas recurrent updates are performed only for retained children and only when $m < M$. Thus,

$$\begin{aligned} C_{\text{Head}} &= O\left(\sum_{m=1}^{M} b_{m-1} H |\mathcal{V}_m| + \sum_{m=1}^{M-1} b_m H^2\right) \\ &= O\left(WH \sum_{m=1}^{M} |\mathcal{V}_m| + (M-1)WH^2\right). \end{aligned} \tag{12}$$

The second line is a worst-case upper bound; the initial projection is evaluated for one beam, not W beams. The vocabulary factor is $|\mathcal{V}_m|$, not the number of admissible extensions, because masking follows the dense projection over the entire position-specific token vocabulary. The implementation additionally scans $b_{m-1}|\mathcal{V}_m|$ candidate slots at each position for masking and normalization and applies global top-W selection; these search operations are outside the neural-computation bounds above. Hardware parallelism, memory traffic, and selection kernels affect wall-clock latency, so larger relative speedups at wider beams do not imply lower absolute latency.

B Additional Experimental Settings

B.1 Datasets

Table 5 distinguishes interaction counts from the next-item examples used for evaluation. The five Targets share the user sequences and item catalogs, while retaining their own SID encodings. Input construction keeps at most the 20 most recent interactions and 512 tokens; history entries use comma-space separators in the supplied input format.

Table 5: Filtered dataset statistics and numbers of next-item examples.

Dataset	Users	Items	Interactions	Train	Validation	Test
Beauty	22,363	12,101	198,502	131,413	22,363	22,363
Instruments	24,772	9,922	206,153	131,837	24,772	24,772
Toys	19,412	11,924	167,597	109,361	19,412	19,412

For a user sequence of length n, the penultimate and final interactions are held out as the validation and test targets, respectively. The remaining prefix yields $n - 3$ training examples, each with a nonempty history. The first interaction serves only as history and is never a prediction target. Thus, with I interactions and U users, the full training set contains $I - 3U$ examples, and the validation and test sets each contain U examples. The split sizes in Table 5 are reported before Stage I subsampling.

B.2 Baselines and Backbones

The default context-layer selection uses $\ell_i = 3i + 1$ for $i = 0, 1, 2, \ldots$, retaining only indices available in the Target. Table 6 lists the configuration choices needed with the extraction rule in Section 3.2; the architecture-dependent root-state definition is unchanged.

Table 6: Sources of context features and SID lengths.

Target	Selected context layers	SID length
LETTER-LC-Rec	Decoder prompt states	4
LETTER-TIGER	Encoder states	4
MiniOneRec	Decoder prompt states	3
CoFiRec	Encoder states	4
DIGER	Encoder states	4

B.3 Training and Inference Hyperparameters

Table 7 lists all hyperparameter values used in the main experiments. Different Target–dataset settings use different combinations of these values. The reference configuration for the one-factor hyperparameter experiments with LETTER on Beauty is specified in Table 9.

Table 7: Hyperparameter values used in the main experiments; inference timing follows Section 4.1.

Parameter	Value
Random seed	42
Learning rate	6×10^{-4}, 2×10^{-3}, 4×10^{-3}
$\alpha_{\mathrm{CE}} : \alpha_{\mathrm{KL}}$ in Stage II	$0.7 : 0.3$, $0.8 : 0.2$
Stage I epochs	4
Stage II epochs	8 or 12
Default τ / γ_{KL}	2 / 4
Training precision / per-device batch size	BF16 / 1
Gradient accumulation steps / warm-up ratio	1 / 0.04

C Additional Quantitative Results

This section supplies derived baseline comparisons, ablation definitions, and the numerical values behind Figure 3, providing additional details for reproducibility and verification, rather than repeating the main result tables. Scores are rounded to four decimal places; percentages and speedups are rounded to two. Ratios use the underlying recorded values before display rounding.

C.1 Cross-Dataset Baseline Comparisons

Table 8 reports score ratios and speedups relative to the explicitly named reference. A score ratio above 100% indicates an improvement. We implement the Beam Search baseline using the publicly available NEZHA codebase. NEZHA changes the Target through training, so its ratios should not be interpreted as a frozen-Target acceleration result. These paired comparisons complement the absolute values in Table 1, providing additional context for interpreting relative performance.

Table 8: Score ratios and speedups for paired baseline comparisons. Each row is method / reference; APD-GR and AtSpeed use LETTER.

Dataset	Comparison	H@10	H@20	N@10	N@20	Speedup
Instruments	NEZHA / Beam	119.40%	126.54%	111.15%	114.06%	18.02×
Instruments	AtSpeed / LETTER	100.38%	100.08%	100.22%	100.12%	2.24×
Instruments	APD-GR / LETTER	96.19%	97.44%	94.59%	95.20%	11.58×
Beauty	NEZHA / Beam	83.61%	86.35%	84.21%	85.06%	19.44×
Beauty	AtSpeed / LETTER	100.28%	99.90%	99.76%	99.80%	2.13×
Beauty	APD-GR / LETTER	99.17%	98.93%	98.33%	98.18%	11.35×
Toys	NEZHA / Beam	95.56%	107.10%	87.93%	93.32%	18.93×
Toys	AtSpeed / LETTER	100.96%	102.48%	100.49%	101.30%	2.15×
Toys	APD-GR / LETTER	95.15%	93.96%	94.79%	94.33%	11.77×

C.2 Effects of Stage II Hyperparameters

Table 9 gives the complete one-factor sweeps for LETTER on Beauty. The reference is $\alpha_{\mathrm{CE}} : \alpha_{\mathrm{KL}} = 0.8 : 0.2$, $\tau = 2$, and $\gamma_{\mathrm{KL}} = 4$; each block varies one hyperparameter while keeping the others at their reference values. These are separate sweeps, not a joint search over all parameter combinations, allowing us to isolate each hyperparameter's effect.

Table 9: Recommendation quality under Stage II hyperparameter sweeps on Beauty.

$\alpha_{\mathrm{CE}} : \alpha_{\mathrm{KL}}$	0.6 : 0.4	0.7 : 0.3	0.8 : 0.2	0.9 : 0.1	1.0 : 0.0
H@5	0.0473	0.0482	0.0492	0.0478	0.0466
H@10	0.0711	0.0724	0.0719	0.0699	0.0641
N@5	0.0330	0.0333	0.0338	0.0328	0.0326
N@10	0.0406	0.0411	0.0412	0.0399	0.0395
Temperature τ	0.5	1	2	4	8
H@5	0.0443	0.0461	0.0492	0.0452	0.0474
H@10	0.0686	0.0706	0.0719	0.0670	0.0713
N@5	0.0308	0.0317	0.0338	0.0313	0.0327
N@10	0.0386	0.0396	0.0412	0.0383	0.0403
Position decay γ_{KL}	1	2	4	8	16
H@5	0.0472	0.0483	0.0492	0.0477	0.0484
H@10	0.0719	0.0722	0.0719	0.0710	0.0721
N@5	0.0331	0.0334	0.0338	0.0333	0.0333
N@10	0.0411	0.0411	0.0412	0.0408	0.0409

C.3 Effects of Beam Width

Table 10 records the values in Figure 3(c-1)–(c-2). Retention compares APD-GR with LETTER at the corresponding beam width. The table reports relative speedup rather than absolute latency as beam width changes, highlighting how acceleration scales with different decoding budgets.

Table 10: Beam-width results for LETTER and APD-GR on Beauty.

Beam width	LETTER H@10	APD-GR H@10	Retention	Speedup
10	0.0712	0.0702	98.55%	4.51×
20	0.0727	0.0722	99.36%	6.18×
50	0.0725	0.0719	99.17%	11.35×
100	0.0732	0.0715	97.68%	22.04×

C.4 Effects of Target Model Size

Table 11 supplements Figure 3(b) with H@5, NDCG, and latency on Instruments. Each APD-GR configuration uses the corresponding frozen LETTER Target, with speedup measured against that Target, thereby enabling a direct comparison across model sizes.

Table 11: Recommendation quality and latency at two LETTER Target scales on Instruments.

Scale	Method	H@5	H@10	N@5	N@10	Time (ms)	Speedup
1B	Target	0.0931	0.1130	0.0803	0.0868	224.31	1.00×
1B	APD-GR	0.0887	0.1087	0.0756	0.0821	19.37	11.58×
3B	Target	0.0947	0.1159	0.0825	0.0893	480.34	1.00×
3B	APD-GR	0.0892	0.1083	0.0767	0.0829	31.71	15.15×

D Efficiency and Training-Cost Analysis

Online inference latency. Table 12 reports the online latency breakdown on the Beauty dataset. We observe that Target computation accounts for the largest portion of the end-to-end latency for both backbones, whereas the Context Adapter incurs negligible cost and the Parallel Position Draft and Prefix-Conditioned Head introduce only moderate overhead. This is because the Target remains responsible for extracting contextual representations, while the Adapter, Draft, and Head are lightweight modules. Moreover, DIGER has lower latency than LETTER-LC-Rec in every compo-

Table 12: Online inference-latency breakdown on the Beauty dataset (ms/example).

Target backbone	Target	Adapter	Draft	Head	End-to-end
LETTER-LC-Rec	15.18	0.13	3.07	2.94	20.26
DIGER	7.45	0.12	1.57	1.45	10.97

Table 13: Offline training and caching costs on the Beauty dataset.

	Caching		APD-GR Training			Retraining Methods
Target backbone	Feature cache	Teacher-logit cache	Stage I	Stage II	Train APD-GR	Retain Target
LETTER-LC-Rec	00:10:10	00:43:02	00:19:42	00:43:27	01:26:30	10:25:58
DIGER	00:00:36	00:01:46	00:01:19	00:09:32	00:11:18	06:38:50

nent, resulting in a lower end-to-end latency. This shows that APD-GR can adapt to different Targets while preserving their relative inference efficiency.

Offline training and caching costs. Table 13 reports the one-time costs of adapting APD-GR to a frozen Target. We observe that feature caching, teacher-logit caching, and the two training stages are conducted offline and are fully separated from online serving. This design prevents these preparation costs from affecting per-request latency. In addition, training APD-GR is substantially cheaper than retraining the Target for LETTER-LC-Rec, while the cost difference is smaller for DIGER. This variation arises because the cost of adapting APD-GR depends on the backbone-specific feature and teacher-logit generation costs. Importantly, APD-GR only trains the proposed lightweight modules and does not modify or retrain the original Target.