DCHORD: SCHEMA-AWARE SPECULATIVE DECODING FOR STRUCTURED GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Agents and embodied systems increasingly rely on large language models to produce structured generation for information extraction, tool invocation, and action commands. Recurring workloads are widely applied in practical scenarios, where requests from the same task share predefined field names, field order, and serialization, while the model predicts short, input-dependent field values. Recent work has explored speculative decoding to accelerate these structured generation workloads. However, these methods typically draft the complete serialized token sequence, allocating prediction positions to schema-determined structure and risking avoidable rejections on tokens that need not be predicted. We present DChord, a schema-aware speculative decoding framework with two components: PairDraft and PairCut. PairDraft concentrates prediction exclusively on field values and end markers, while a pair state machine supplies schema-determined structure to expand a compact value-only prediction into a complete draft. For concurrent serving, PairCut treats schema-determined structure as certain in its verification decisions, increasing acceptance length with negligible additional target latency. We evaluate DChord on Qwen3.5-4B, Qwen3.6-35B-A3B, and Gemma4-12B across four schema-determined tasks spanning multimodal and text-only inputs, comparing against autoregressive decoding (AR) and four recent speculative decoding methods. At concurrency 16, DChord achieves up to $3.63\times$ throughput speedup over AR and up to $1.63\times$ the throughput of the strongest competing speculative method.

1 INTRODUCTION

LLMs increasingly produce structured generation for agents and embodied systems, including schema-guided information extraction, tool calls, and action commands (Rombach & Fettke, 2025; Ding et al., 2026). Among these applications, schema-determined structured generation tasks are widely used to provide predictable, machine-readable interfaces (Wang et al., 2026; Liu et al., 2024a). Structured outputs in these workloads consist of several components: **field names, field values, delimiters, and other serialization symbols, arranged in a prescribed field order** (Dunn et al., 2022). As these applications become more latency-sensitive, improving the efficiency of structured generation becomes increasingly important (Dong et al., 2025; Geng et al., 2025).

Compared with autoregressive decoding (AR), speculative decoding provides a more efficient way: a lightweight drafter proposes multiple future tokens, and the target model verifies together in a single forward pass (Chen et al., 2023; Li et al., 2025). Methods such as DFlash further use diffusion drafting to predict multiple tokens in parallel within a single forward pass (Chen et al., 2026). However, general-purpose speculative decoding methods do not fully exploit the deterministic and reusable structure of these workloads: **requests within the same task share a predefined schema, while only short field values vary with the input**. Specifically, deterministic schema tokens consume draft positions that could otherwise be used to predict input-dependent field values. Moreover, prediction errors on structure that is already determined by the schema can trigger target rejection and terminate the accepted prefix. Thus, speculative decoding wastes draft capacity on known structure and incurs avoidable rejections. This mismatch raises a natural question:

If the schema already determines the stable output structure, can the drafter focus on field values while the schema supplies the deterministic structure for batched target verification?

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

To answer this question, we introduce DChord, a schema-aware speculative decoding framework for recurring structured generation workloads with stable schemas and short field values. It has two components, PairDraft and PairCut. PairDraft removes schema-determined structure from learned drafting and predicts only field values. By concentrating drafter prediction positions on input-dependent content, PairDraft allows the same prediction capacity to cover more field values.

Predicting values alone, however, introduces a new challenge: *the drafter must know which field is currently being generated, and where each field value ends before the corresponding schema structure can be inserted.* To address this, PairDraft jointly predicts field values and their end markers, conditioned on schema information and the current decoding state. A pair state machine and a decoding state encoder jointly track the decoding state. The pair state machine further inserts the corresponding schema structure, expanding the compact value-only prediction into a complete draft for batched target verification.

However, PairDraft introduces a second challenge under concurrent serving: *the expanded schema increases target verification queries and computational cost.* Existing hardware-aware pruning controls this cost at token granularity (Hu et al., 2026; Liu et al., 2026), but does not account for where a cut falls relative to field value boundaries. DChord therefore introduces PairCut, which uses pair structure together with hardware execution characteristics to determine how much of the expanded draft should be verified. When additional verification queries incur negligible incremental latency, PairCut uses them to extend across schema-determined structure toward the next field value.

Our main contributions are:

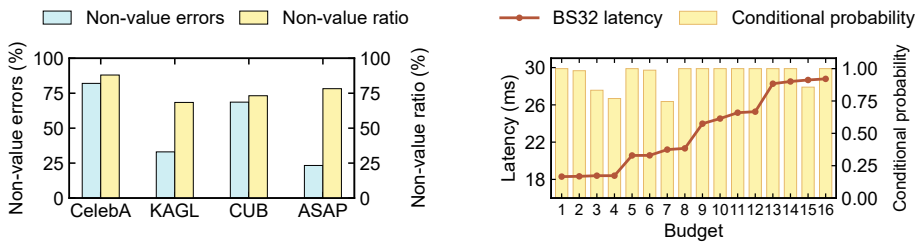
- We identify that, in recurring structured generation workloads, schema-determined structure is reusable across inputs, while only short field values depend on the input. Therefore, we apply this regularity to speculative decoding. Specifically, deterministic structure is separated from learned value prediction, thus drafter capacity is concentrated on content that must be inferred.
- We introduce DChord, a schema-aware speculative decoding framework with two components. PairDraft addresses the challenge of drafting values without predicting schema-determined structure. It jointly predicts field values and their end markers, while a decoding state encoder and pair state machine track the current state and reconstruct a complete draft. PairCut further controls the increasing verification cost from expanded drafts by exploiting the difference between schema-determined structure and uncertain values. This yields longer acceptance length with negligible additional target latency.
- We evaluate DChord on Qwen3.5-4B, Qwen3.6-35B-A3B, and Gemma4-12B across four diverse multimodal and text-only schema-determined workloads. At concurrency 16, DChord delivers up to $3.63\times$ the throughput of AR and $1.63\times$ that of the strongest speculative method.

2 MOTIVATION

To characterize the mismatch between conventional speculative decoding and schema-determined structured generation, we examine how token-level drafting and verification allocate computation between schema-determined structure and input-dependent field values, using the Qwen3.5-4B model.

Token-level drafting wastes capacity on determined structure. We analyze where the first rejection occurs in DFlash drafts under schema constraints (Chen et al., 2026). As shown in Figure 1(a), schema-determined non-value tokens average 76.97% of output, yet the first rejection occurs at such positions up to 82.04% of rejection-containing rounds. Because speculative decoding commits only a contiguous accepted prefix (Leviathan et al., 2023), these rejections can block subsequent field values. Thus, token-level drafting both spends prediction capacity on determined structure and introduces avoidable rejection, motivating direct schema injection and value-focused drafting.

Token-level pruning overlooks structural progress. Figure 1(b) measures DFlash's position-wise conditional acceptance probability and target verification latency on the CUB dataset at batch size 32. Conditional acceptance is lower at field value positions and higher at schema-determined key positions. Yet increasing the verification budget from 1 to 4 or from 9 to 12 leaves latency



(a) Non-value output and rejection rate. **(b) DFlash budget profile at BS32.**

Figure 1: Motivation for DChord. (a) DFlash's first rejections occur during drafting across four datasets (b) DFlash's position-wise conditional acceptance probability and target verification latency at batch size 32.

nearly unchanged, at approximately $18.1ms$ and $25.0ms$, respectively. This motivates using a small increase in verification budget to retain more high-confidence key positions, and extend the accepted prefix toward subsequent values without appreciably increasing target latency.

3 PRELIMINARIES

Schema-determined structured generation. We consider a collection of recurring tasks, each associated with a predefined schema that is shared across its training and evaluation inputs. Within each task, the schema determines field names, field order, and serialization, while field values depend on the input. Field values are short in the workloads considered here, but their token lengths can vary across inputs. PairDraft predicts their boundaries jointly with their content.

Given an input x and a deterministic schema S, the schema determines the serialization structure, while field values are inferred from x. We organize the serialized output field by field into *pairs*, each consisting of an input-dependent *schema-value* and its associated *schema-key*. Schema-keys include field names and schema-determined serialization tokens such as braces, quotation marks, colons, and commas. The final pair includes remaining structure.

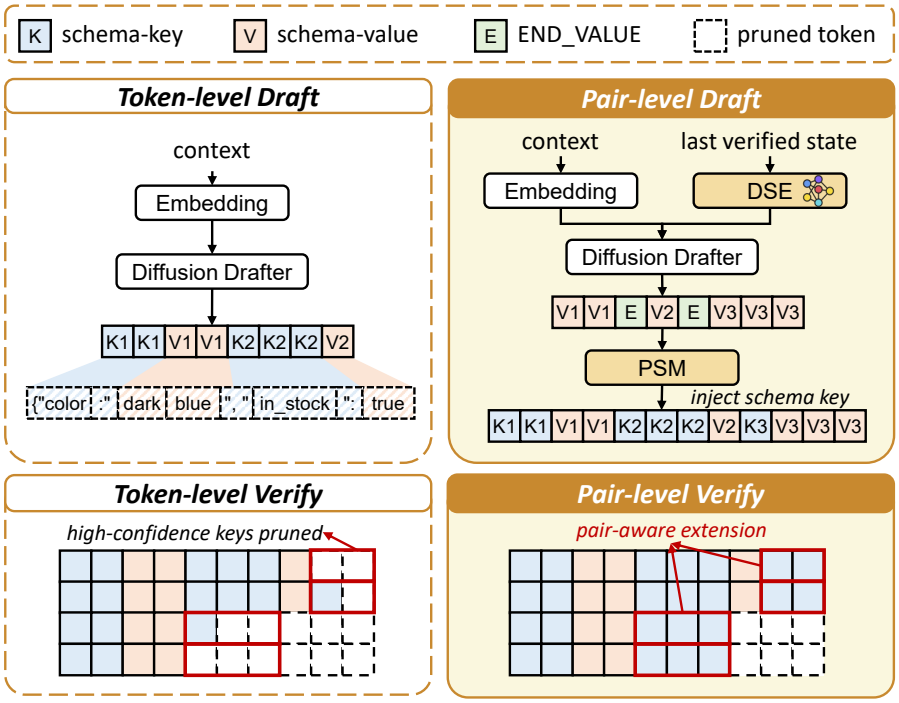
For example, in {`"color":"dark blue","in_stock":true`}, `dark blue` and `true` are schema-values, while all remaining content is schema-key. Token boundaries follow the target model's tokenizer: if a token merges schema-key content with value-dependent content, we classify the entire token as a schema-value because its realization depends on the field value.

Speculative decoding and block-diffusion drafting. Speculative decoding uses a lightweight drafter to propose multiple candidate tokens for batched verification by the target model (Chen et al., 2023). Let h denote the committed output prefix and its final token the current *anchor*. In each speculative round, the target verifies the draft in one forward pass and commits the longest contiguous accepted prefix. A rejection produces a target-generated correction token, while full acceptance permits an additional bonus token.

We denote by τ the *acceptance length*, defined as the average number of newly committed final-output tokens per speculative round, including the accepted prefix and the target-generated correction or bonus token. Under the same definition, DChord's accepted prefix may contain both predicted schema-values and injected schema-keys.

We use *token-level drafting* to refer to drafting in which prediction positions correspond to positions in the full serialized output, covering both schema-key and schema-value tokens ($e.g.$, DFlash (Chen et al., 2026)). DFlash predicts such positions with a block-diffusion drafter. DChord adopts the same drafting backbone but replaces the token-level representation with the schema-aware representation introduced in Section 4.

Draft length. For request i, we use d_i to denote the number of target-model tokens in the complete draft available for speculative verification. In conventional token-level speculative decoding, this corresponds directly to the drafter's token-level proposal. In DChord, the draft is constructed after



(a) Token-level Decode (b) Pair-level Decode

Figure 2: Overview of DChord for schema-determined structured generation. (a) Conventional token-level decoding drafts schema-keys and schema-values uniformly and prunes verification at token granularity. (b) PairDraft predicts only schema-values and their end markers. The Pair State Machine (PSM) provides the last verified state to the Decoding State Encoder (DSE), which encodes it to condition the drafter. The PSM then tracks the latest state and inserts schema-keys to complete drafts. PairCut (pair-level verification) uses pair structure to extend verification across injected schema-keys toward the next field value, increasing acceptance length with negligible additional target latency.

schema expansion and therefore contains both predicted schema-value tokens and injected schema-key tokens, while end markers are excluded, as detailed in Section 4.1.

Verification width. We use W to denote the maximum number of target verification queries per request in one round, including one position for a target-generated correction or bonus token. For a complete draft of length d_i, the selected draft width w_i satisfies $0 \leq w_i \leq \min\{d_i, W-1\}$. The target verifies the first w_i draft tokens and uses the remaining query position for correction or bonus generation.

4 METHOD

Figure 2 provides an overview of DChord. We first describe PairDraft, including the pair state machine, decoding state encoder, drafting, and training, then introduce PairCut.

4.1 PAIRDRAFT

Pair State Machine (PSM). Since PairDraft no longer predicts schema-keys, PSM explicitly manages schema structure and per-request generation state. It has two main responsibilities: storing the schema information of the deterministic target workload set, and tracking and updating each request's decoding state to support schema-key injection and draft expansion.

For each workload, PSM parses its schema and stores the structural information according to the canonical field order, including the ordered pairs, tokenizer-resolved schema-key content, and schema-value types. This schema information is reused during both training and inference. Af-

ter PairDraft predicts schema-values and their ending boundaries, PSM injects the corresponding schema-key content according to the pair structure, constructing the complete draft.

For each request, PSM maintains an independent cursor that records the current pair, the anchor's position type (schema-key or schema-value), and the number of committed tokens in the current schema-value. For example, suppose the committed output ends after the schema-value token `dark` in the `color` field. If `dark` is one token, the cursor is inside the schema-value of pair 0 at offset 1. The pair index, token offset, and anchor type jointly form the decoding state supplied to DSE. After target verification, PSM updates the cursor according to the committed accepted prefix and the target-generated correction or bonus token. Rejected draft suffixes do not advance the request state.

Decoding State Encoder (DSE). DSE converts the state information maintained by PSM into a state vector for the drafter. At decoding round t, DSE receives the current pair index j_t for the current request, the number of committed tokens in the current schema-value o_t, and the anchor's position type a_t, and constructs state features:

$$r_t = \mathrm{StateFeatures}(j_t, o_t, a_t), \qquad z_t = W_{\mathrm{state}} r_t. \tag{1}$$

Here, $\mathrm{StateFeatures}$ denotes deterministic state-feature construction, W_{state} is a trainable linear projection, and z_t has the same dimensionality as the drafter's hidden representations.

Pair-level drafting. PairDraft adopts a diffusion-based drafter and predicts a compressed sequence of schema-values and end markers:

$$u = s_0 \oplus e \oplus s_1 \oplus e \oplus \cdots, \tag{2}$$

where s_j denotes the schema-value token sequence of pair j, and $e = \langle \mathrm{END_VALUE} \rangle$ is an additional class in the drafter's output space.

The drafter conditions on target-model context features and the DSE state. Target context features derived from the prompt and committed prefix provide the input and schema semantics, while DSE encodes decoding states. At each decoding round, the DSE state vector is added to the mask embeddings. Conditioned on these representations, the anchor, and target context features, the drafter predicts schema-values and end markers in parallel in a single forward pass.

Draft expansion and verification. PSM expands the compressed prediction into complete drafts by inserting schema-keys at the boundaries indicated by the predicted end markers. The end marker themselves are not passed to the target.

The expanded draft is submitted to the standard speculative verification procedure, subject to the verification-width selection in Section 4.2. Schema expansion determines the complete draft tokens, while target verification determines which tokens are committed. In particular, a predicted end marker does not by itself commit the end of a schema-value.

At the first rejection, verification commits the accepted prefix and a target-generated correction token, and discards the remaining draft candidate suffix. If the submitted candidate is fully accepted, the target generates an additional bonus token. PSM updates its persistent cursor only from the tokens actually committed, so the next round resumes from the target-verified and corrected prefix. Since DChord retains standard target-model verification, it preserves the target sampling distribution under standard speculative decoding assumptions.

Pair-level training. DChord constructs its training supervision through self-distillation from a frozen target model. For each training sample, PSM injects the schema-key content according to the corresponding schema information, while the frozen target generates schema-values in the complete serialized context, producing a full teacher output. We then add an $\langle \mathrm{END_VALUE} \rangle$ boundary target to each schema-value, yielding the schema-value and end marker prediction targets represented by Eq. 2. The complete teacher output, including both schema-key and schema-value content, is retained throughout training to provide the original serialized context. Schema-key positions are not used as drafter prediction targets and are excluded from the loss through a supervision mask.

We sample anchor positions from the complete teacher outputs during training, an anchor may lie in either a schema-key or a schema-value. For each anchor, PSM recovers the current pair, the decoding progress within the current schema-value, and the anchor position type, from which DSE

constructs the same decoding state representation used at inference time. The drafter accesses only the target context features available at that anchor and learns to predict the subsequent schema-value tokens and end markers in parallel.

We optimize the drafter with cross-entropy loss over valid value-and-boundary prediction positions, assigning equal weight to schema-value tokens and end markers. The drafter, DSE, and end marker output parameters are jointly optimized, while the target model remains frozen.

4.2 PAIRCUT

PairDraft's expanded proposals increase target verification work, so PairCut begins with the hardware-aware baseline widths $\mathbf{w}^0$ defined above. Including one correction or bonus query per request gives the logical query count $Q_0 = \sum_{i=1}^{N}(w_i^0 + 1)$. Measured target latency grows in steps with the query count, so a small increase beyond Q_0 can incur little additional time. Let Q_G denote the upper query count of the cost step selected at Q_0, and let $B_{extra} = Q_G - Q_0$ be the additional budget PairCut may consider. PairCut selectively spends this budget to extend request prefixes while preserving every baseline width.

PairCut scores each newly represented learned-value prediction with its drafter confidence $p_{i,j}$ and each injected position with $p_{i,j} = 1$. These scores give the prefix score $s_i(w) = \prod_{j=1}^{w} p_{i,j}$. From the current width w_i, PairCut can extend to the *value tail* at the next value's last learned token. It can also take a shorter *fixed bridge* through the longest contiguous injected key or syntax prefix that fits. The value tail retains a new learned prediction, whereas the bridge advances through known structure. The value-tail endpoint excludes any following boundary-testing token, which may occupy the target bonus position. An endpoint ep is feasible when $w_i < ep \le L_i = \min\{d_i, W-1\}$ and $ep - w_i \le B_{\mathrm{rem}}$, the remaining capacity.

Let $G_i(ep \mid w_i)$ sum the prefix scores at the first occurrence of each newly retained distinct learned prediction, and let $R_i(ep \mid w_i) = s_i(ep)/s_i(w_i)$ denote the conditional prefix score (zero for an invalid prefix). The two endpoint scores are

$$U_i(ep) = \begin{cases} R_i(ep \mid w_i) G_i(ep \mid w_i)/(ep - w_i), & \text{value tail,} \\ R_i(ep \mid w_i), & \text{fixed bridge.} \end{cases} \tag{3}$$

A value-tail candidate requires positive learned gain and conditional prefix score. A bridge contains no learned prediction and is scored by its conditional prefix score alone. Both types compete in the same ranking, even when the value tail is feasible. PairCut selects the highest score, breaks ties by lower query cost and then request order, and repeats until no candidate remains. The final widths satisfy $w_i^0 \le w_i \le L_i$ and $\sum_i (w_i - w_i^0) \le B_{extra}$. Unused capacity may remain. We set W using representative requests from the target workload set and keep it fixed during serving.

5 EXPERIMENTS

Models and evaluation. We evaluate Qwen3.5-4B (Team, 2026b), Qwen3.6-35B-A3B (Qwen Team, 2026), and Gemma4-12B (Team, 2026a) under greedy decoding ($T = 0$) and stochastic sampling ($T = 1$). Single-request experiments report mean acceptance length τ and decoding speedup over AR based on mean time per output token (TPOT), while concurrent serving experiments report aggregate output throughput.

Datasets. We use four schema-determined tasks: 40 Boolean attributes on CelebA (Liu et al., 2015), 7 product attributes on KAGL (Jung & Clark, 2024), 28 bird attributes on CUB (Wah et al., 2011), and 18 aspect-level sentiment labels on ASAP (Bu et al., 2021). Each dataset uses its own prompt comprising task instructions and a fixed one-shot example. More details in Appendix A.

Implementation. Experiments run on two NVIDIA A800-SXM4-80GB GPUs with vLLM 0.27.0. For each target model, a single DChord drafter is initialized from the corresponding open-source DFlash checkpoint and jointly trained on the training splits of all datasets, with the target model frozen. Evaluation uses held-out inputs disjoint from the training and validation splits, with each task's schema unchanged across splits.

Table 1: Aggregate throughput speedup over autoregressive decoding and average acceptance length (τ) under continuous batching at concurrency $C = 16$, using 512 requests per dataset. Throughput speedup is normalized to AR under the same model, temperature, and dataset. Parenthesized values denote the speculative block size. AVG is the unweighted arithmetic mean across the four datasets.

	Model	Method	CelebA		KAGL		CUB		ASAP		AVG	
			Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
$T = 0$	Qwen3.5-4B	MTP (8)	2.85×	6.348	1.44×	5.011	2.14×	5.359	2.06×	6.366	2.12×	5.771
		DFlash (16)	2.39×	8.439	1.27×	6.403	2.19×	8.168	2.31×	11.579	2.04×	8.647
		DChord	**3.63×**	**15.289**	**1.53×**	**11.078**	**2.98×**	**14.958**	**2.60×**	**14.304**	**2.69×**	**13.907**
	Qwen3.6-35B-A3B	MTP (8)	1.83×	7.118	0.92×	4.813	1.33×	5.711	2.04×	7.006	1.53×	6.162
		DFlash (16)	1.57×	10.377	0.85×	5.099	1.47×	8.699	2.05×	12.433	1.48×	9.152
		DChord	**2.38×**	**15.359**	**1.50×**	**11.105**	**1.99×**	**14.107**	**2.40×**	**14.448**	**2.07×**	**13.755**
	Gemma4-12B	EAGLE3 (8)	2.44×	6.461	1.33×	4.224	1.96×	4.853	1.99×	6.630	1.93×	5.542
		DFlash (8)	2.53×	6.355	1.39×	3.980	1.88×	4.258	2.10×	6.665	1.98×	5.314
		DSpark (8)	2.78×	6.953	1.53×	5.386	2.21×	5.161	2.20×	6.960	2.18×	6.115
		DFlash (16)	2.51×	7.233	1.35×	4.005	1.80×	4.348	2.06×	8.102	1.93×	5.922
		DChord	**3.27×**	**15.506**	**1.57×**	**12.555**	**2.94×**	**13.604**	**2.39×**	**14.640**	**2.54×**	**14.076**
$T = 1$	Qwen3.5-4B	MTP (8)	2.82×	6.281	1.41×	4.829	1.83×	4.573	1.77×	6.079	1.96×	5.441
		DFlash (16)	2.32×	8.401	1.31×	6.277	1.70×	5.982	2.25×	10.972	1.89×	7.908
		DChord	**3.51×**	**14.870**	**1.51×**	**8.569**	**2.31×**	**10.920**	**2.43×**	**12.882**	**2.44×**	**11.810**
	Qwen3.6-35B-A3B	MTP (8)	1.97×	6.961	1.17×	5.383	1.37×	5.464	1.95×	6.836	1.62×	6.161
		DFlash (16)	1.71×	10.253	1.00×	6.022	1.44×	7.557	1.98×	12.112	1.53×	8.986
		DChord	**2.29×**	**14.846**	**1.33×**	**8.515**	**1.85×**	**12.000**	**2.11×**	**13.520**	**1.89×**	**12.220**
	Gemma4-12B	EAGLE3 (8)	2.44×	6.467	1.29×	4.320	1.95×	4.711	1.97×	6.607	1.91×	5.526
		DFlash (8)	2.51×	6.372	1.34×	4.026	1.86×	4.097	2.05×	6.649	1.94×	5.286
		DSpark (8)	2.75×	6.952	1.48×	5.505	2.23×	5.022	2.15×	6.937	2.15×	6.104
		DFlash (16)	2.51×	7.258	1.31×	4.039	1.76×	4.178	2.01×	8.070	1.90×	5.886
		DChord	**3.25×**	**15.424**	**1.52×**	**12.775**	**2.81×**	**12.293**	**2.32×**	**14.497**	**2.47×**	**13.747**

Baselines. We compare DChord with AR, MTP (Team, 2026b; Qwen Team, 2026), EAGLE3 (Li et al., 2025), DFlash (Chen et al., 2026), and DSpark (Cheng et al., 2026), using publicly available checkpoints compatible with each target model. In Table 1, parenthesized values indicate the total per-request verification width, including one position for a target-generated correction or bonus token. For MTP, we use (8), which achieves the best end-to-end decoding speed among the settings we evaluated. For the other methods, we use the default configurations recommended by their released checkpoints. All methods are evaluated under identical schema constraints to ensure comparable, format-valid outputs. Further experimental details are provided in Appendix A.

5.1 Main Results

Concurrent serving at concurrency 16. Table 1 reports aggregate throughput speedup over AR and average acceptance length τ at concurrency $C = 16$ across three target models, four datasets, and both sampling temperatures. DChord achieves both the highest throughput and the longest average acceptance length in all 24 model–dataset–temperature combinations. At $T = 0$, its average throughput speedups over AR are **2.69**×, **2.07**×, and **2.54**× on Qwen3.5-4B, Qwen3.6-35B-A3B, and Gemma4-12B, respectively. Compared with the strongest speculative baseline for each model, DChord further improves average throughput by **1.17–1.35**× and average acceptance length by **1.50–2.30**×. The same advantage persists at $T = 1$, where the corresponding improvements are **1.15–1.24**× in throughput and **1.36–2.25**× in acceptance length.

These gains show that DChord's longer acceptance length translates into higher serving throughput at $C = 16$. By concentrating drafter positions on schema-values and injecting schema-keys during proposal assembly, DChord advances more output per target verification step while eliminating prediction errors at schema-key positions.

Single-request performance. Figure 3 shows that DChord achieves the longest acceptance length in all configurations. Compared with the strongest speculative acceptance-length baseline on each dataset, its mean acceptance length is $2.54\times$, $2.10\times$, and $2.49\times$ that of the baseline on Qwen3.5-4B, Qwen3.6-35B-A3B, and Gemma4-12B, respectively. The same runs yield mean-TPOT speedups of $1.98\times$, $2.13\times$, and $1.45\times$ over the fastest competing speculative method on each dataset.

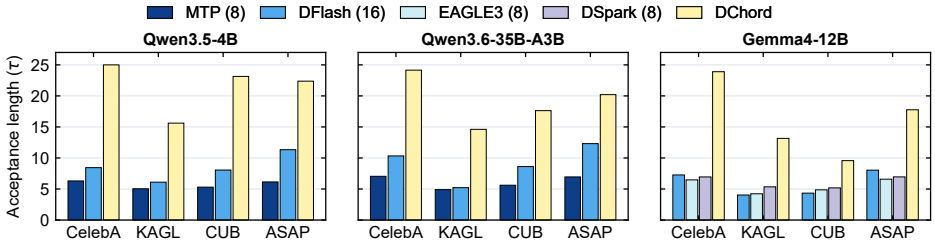


Figure 3: Average acceptance length τ under single-request greedy decoding across three target models and four datasets. AR is omitted since it does not use speculative drafts. Higher is better.

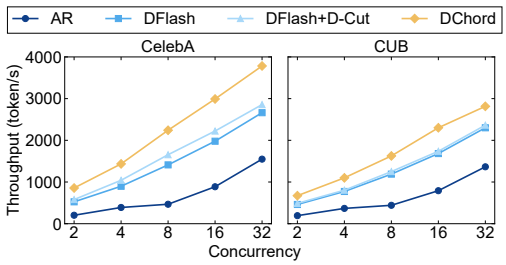


Figure 4: Scaling with concurrency.

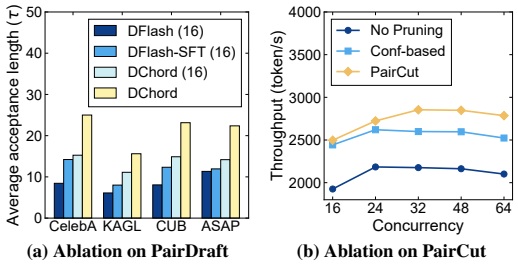


Figure 5: Ablations on SFT and verify pruning.

Scaling with concurrency. We evaluate aggregate output throughput on CelebA and CUB at request concurrency levels $C \in 2, 4, 8, 16, 32$, using the same setup described above. Figure 4 compares DChord with AR, DFlash, and DFlash+D-Cut, which applies adaptive verification pruning to DFlash.

Throughput increases with concurrency for all methods. DFlash+D-Cut generally outperforms DFlash, with larger gains on CelebA than on CUB. DChord achieves the highest throughput at every evaluated concurrency level on both datasets. Relative to AR, its throughput is **2.44–4.80**$\times$ higher on CelebA and **2.06–3.68**$\times$ higher on CUB. PairDraft concentrates prediction on schema-values, while PairCut retains more of the expanded proposal for verification. We isolate PairCut's contribution in Section 5.2.

5.2 Ablations and Analysis

Effect of verification width. A key advantage of DChord is that it drafts only schema-values and expands them with schema-keys, allowing a compact prediction sequence to form a longer target-side draft length. Therefore, we compare DChord(16) with DFlash(16) under the same physical verification-width limit. Both variants submit at most 15 draft tokens for target verification, together with one target-generated correction or bonus token. As shown in Figure 5(a), DChord(16) consistently outperforms DFlash(16) across all four datasets. This shows that DChord remains effective under the same verification-width constraint as DFlash(16).

Effect of fine-tuning. Because DChord is trained on the target datasets, part of its gain may come from improved agreement with the target model through fine-tuning. To isolate this effect, we fine-tune DFlash jointly on the same training data with comparable training settings, yielding DFlash-SFT(16). As shown in Figure 5(a), DFlash-SFT(16) improves over DFlash(16), confirming that fine-tuning is beneficial. DChord(16) reduces mean TPOT by 20.95% relative to DFlash-SFT(16). Both DChord(16) and DChord consistently outperform DFlash-SFT(16). Appendix Table 4 further compares schema-key rejection rates before and after fine-tuning.

Effect of PairCut. Figure 5(b) compares three verification-width strategies within DChord on CUB at $C \in \{16, 24, 32, 48, 64\}$. No Pruning verifies the complete expanded draft. Conf-based follows D-Cut's confidence-based budget selection (Liu et al., 2026), assigning injected fixed positions a shared conditional confidence estimated from a held-out DFlash calibration set, while using drafter confidences for learned values. PairCut assigns injected fixed positions unit selector scores

and extends high-confidence draft prefixes toward value-completion boundaries. This spends additional verification time to retain more of the expanded proposal and lengthen the accepted prefix. At $C = 16$, PairCut improves throughput over Conf-based by only 2.24%. The selectors retain nearly the same proposal positions at low concurrency, and early draft rejections limit the value of verifying later positions. As concurrency rises, PairCut gains more from carrying high-confidence prefixes across fixed keys toward subsequent values. Its advantage over Conf-based remains about 10% at $C = 48$ and 64. Throughput levels off beyond $C = 32$, suggesting that higher configured concurrency provides limited additional effective parallelism.

Comparison with schema-aware AR. We compare DChord with schema-aware AR, which skips deterministic schema structure while generating field values autoregressively. As shown in Appendix Table 6, DChord achieves lower TPOT than schema-aware AR, with speedups up to $4.20\times$. These results show that DChord provides additional acceleration by combining schema injection with speculative generation of field values.

6 Related Work

Structured and constrained decoding. Grammar-constrained decoding restricts valid continuations to enforce structured outputs (Geng et al., 2023). SGLang (Zheng et al., 2024) accelerates deterministic spans through finite-state-machine-based decoding and XGrammar reduces grammar-processing overhead (Dong et al., 2025). These methods accelerate constraint enforcement but do not themselves parallelize the generation of input-dependent field values. DChord uses speculative decoding to predict multiple schema-values in parallel, accelerating their generation.

Speculative decoding. Speculative decoding accelerates autoregressive generation through draft proposals and batched target verification (Leviathan et al., 2023; Chen et al., 2023; Xia et al., 2024). Recent work improves its general efficiency through draft–target separation (Liu et al., 2025; Kumar et al., 2026b), loose verification (Narasimhan et al., 2025), partial verification (Kumar et al., 2026a), and stronger parallel drafting (Li et al., 2025; Chen et al., 2026; Cheng et al., 2026). However, these methods may underexploit stable regularities in practical domains. DistillSpec (Zhou et al., 2024), Online Speculative Decoding (Liu et al., 2024b), and Specialized Drafters (Yi et al., 2024) improve speculation by exploiting task-, query-, or language-specific distributions. These methods primarily exploit distributional regularities while retaining token-level drafting over the serialized sequence. DChord instead exploits the structural regularity of schema-determined workloads: it predicts only schema-values and their boundaries, while schema-determined structure is supplied explicitly.

Adaptive drafting and verification. Adaptive methods trade candidate coverage against drafting and verification cost (Zhen et al., 2025). AdaEAGLE adapts draft length at each decoding step (Zhang et al., 2024). D-Cut (Liu et al., 2026) and ECHO (Hu et al., 2026) use token confidence to control target verification, while DSpark (Cheng et al., 2026) uses a pretrained confidence head for its verification decisions. These methods make verification decisions at token granularity without distinguishing schema-determined keys from predicted values in their confidence estimates. PairCut assigns unit confidence to injected keys and uses the resulting prefix scores to prioritize verification toward subsequent values.

7 Conclusion

We introduce DChord, a schema-aware speculative decoding framework for structured generation. It consists of two components: PairDraft and PairCut. PairDraft builds on a diffusion-based speculative drafter to predict schema-values and their end markers. It then injects schema-keys and expands complete drafts using a decoding state encoder and a pair state machine. PairCut further addresses the increased target verification cost caused by expanding drafts under high concurrency. It exploits the difference between deterministic schema-keys and uncertain schema-values to increase acceptance length with negligible additional target latency. We evaluate DChord on three target models and four schema-determined tasks under two sampling regimes, comparing it against AR and four recent speculative decoding methods. At concurrency 16, DChord achieves up to $3.63\times$ throughput speedup over AR and up to $1.63\times$ the throughput of the strongest competing speculative method.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript polish, and feedback on experimental design and presentation. All technical decisions, experiments, results, and claims were reviewed and verified by the authors, who take full responsibility for the content of this work.

REPRODUCIBILITY STATEMENT

We have taken several steps to ensure reproducibility. More details about training data and configurations are also introduced in Appendix A.1. All models used in this work (Qwen3.5-4B, Qwen3.6-35B-A3B, and Gemma4-12B) and their related speculative drafters (MTP, DFlash, Eagle3, and DSpark) are publicly available through HuggingFace, as detailed in Appendix A.2. The detailed experimental results are also provided in Appendix A. We will release DChord modules, modified model checkpoints, and experiment scripts upon acceptance to enable exact reproduction of all tables and figures.

REFERENCES

Oscar Brown, Zhengjie Wang, Andrea Do, Nikhil Mathew, and Cheng Yu. Dynamic depth decoding: Faster speculative decoding for llms, 2024. URL https://arxiv.org/abs/2409.00142.

Jiahao Bu, Lei Ren, Shuang Zheng, Yang Yang, Jingang Wang, Fuzheng Zhang, and Wei Wu. ASAP: A Chinese review dataset towards aspect category sentiment analysis and rating prediction. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2069–2079, Online, June 2021. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2021.naacl-main.167.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, Qinyu Chen, Chengqi Deng, Jiping Yu, Damai Dai, Zhengyan Zhang, Yixuan Wei, Yixuan Tan, Wenkai Yang, Runxin Xu, Yu Wu, Zhean Xu, Xuanyu Wang, Muyang Chen, Rui Tian, Xiao Bi, Zhewen Hao, Shaoyuan Chen, Huanqi Cao, Wentao Zhang, Anyi Xu, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation, 2026. URL https://arxiv.org/abs/2607.05147.

Yihao Ding, Siwen Luo, Yue Dai, Yanbei Jiang, Zechuan Li, Qiang Sun, Geoffrey Martin, Wei Liu, and Yifan Peng. A survey on MLLM-based visually rich document understanding: Methods, challenges, and emerging trends. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 13319–13340, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.652. URL https://aclanthology.org/2026.findings-acl.652/.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ziyi Xu, Yilong Zhao, Ruihang Lai, and Tianqi Chen. Xgrammar: Flexible and efficient structured generation engine for large language models. In M. Zaharia, G. Joshi, and Y. Lin (eds.), *Proceedings of Machine Learning and Systems*, volume 7. MLSys, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/file/5c20ca4b0b20b0bd2f1d839dc605e70f-Paper-Conference.pdf.

Alexander Dunn, John Dagdelen, Nicholas Walker, Sanghoon Lee, Andrew S. Rosen, Gerbrand Ceder, Kristin Persson, and Anubhav Jain. Structured information extraction from complex scientific text with fine-tuned large language models, 2022. URL https://arxiv.org/abs/2212.05238.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured NLP tasks without finetuning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10932–10952, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.674. URL https://aclanthology.org/2023.emnlp-main.674/.

Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. Jsonschemabench: A rigorous benchmark of structured outputs for language models, 2025. URL https://arxiv.org/abs/2501.10868.

Xinyi Hu, Yuhao Shen, Baolin Zhang, Hengxin Zhang, Jun Dai, Shuang Ge, Lei Chen, Yue Li, and Mingcheng Wan. Echo: Elastic speculative decoding with sparse gating for high-concurrency scenarios, 2026. URL https://arxiv.org/abs/2604.09603.

Myong Chol Jung and Jesse Clark. Marqo-FashionCLIP and Marqo-FashionSigLIP, August 2024. URL https://github.com/marqo-ai/marqo-FashionCLIP.

Avinash Kumar, Sujay Sanghavi, and Poulami Das. Hispec: Hierarchical speculative decoding for llms, 2026a. URL https://arxiv.org/abs/2510.01336.

Tanishq Kumar, Tri Dao, and Avner May. Speculative speculative decoding, 2026b. URL https://arxiv.org/abs/2603.03251.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 136737–136756. Curran Associates, Inc., 2025. doi: 10.52202/085713-4562. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/c7b5a35ea98b62512a869c19ea7b03cb-Paper-Conference.pdf.

Michael Xieyang Liu, Frederick Liu, Alexander J. Fiannaca, Terry Koo, Lucas Dixon, Michael Terry, and Carrie J. Cai. "we need structured output": Towards user-centered constraints on large language model output. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, CHI EA '24, New York, NY, USA, 2024a. Association for Computing Machinery. ISBN 9798400703317. doi: 10.1145/3613905.3650756. URL https://doi.org/10.1145/3613905.3650756.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. Pearl: Parallel speculative decoding with adaptive draft length. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 1085–1104, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/03b1043052700b1a471996b0baf309d4-Paper-Conference.pdf.

Tianyu Liu, Yuhao Shen, Rui Cen, Junhan Shi, Jiebin Zhang, Guangshuo Qin, Hong Liu, Song Liu, Guanghua Yu, and Jianchen Zhu. D-cut: Adaptive verification depth pruning for batched speculative decoding, 2026. URL https://arxiv.org/abs/2607.14647.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 31131–31146. PMLR, 21–27 Jul 2024b. URL https://proceedings.mlr.press/v235/liu24y.html.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Seungyeon Kim, Neha Gupta, Aditya Krishna Menon, and Sanjiv Kumar. Faster cascades via speculative decoding. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 44949–44987, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/6f43166f50f26e8d8f3edc5545b0749f-Paper-Conference.pdf.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Alexander Michael Rombach and Peter Fettke. Deep learning based key information extraction from business documents: Systematic literature review. *ACM Comput. Surv.*, 58(2), September 2025. ISSN 0360-0300. doi: 10.1145/3749369. URL https://doi.org/10.1145/3749369.

Gemma Team. Gemma 4 technical report, 2026a. URL https://arxiv.org/abs/2607.02770.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026b. URL https://qwen.ai/blog?id=qwen3.5.

C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

Maolin Wang, Yingyi Zhang, Bowen Yu, Bingguang Hao, Cunyin Peng, Yicheng Chen, Wei Zhou, Jinjie Gu, Chenyi Zhuang, Ruocheng Guo, Wanyu Wang, and Xiangyu Zhao. Function calling in large language models: Industrial practices, challenges, and future directions. *ACM Comput. Surv.*, 58(9), February 2026. ISSN 0360-0300. doi: 10.1145/3788284. URL https://doi.org/10.1145/3788284.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7655–7671, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.456. URL https://aclanthology.org/2024.findings-acl.456/.

Euiin Yi, Taehyeon Kim, Hongseok Jeung, Du-Seong Chang, and Se-Young Yun. Towards fast multilingual LLM inference: Speculative decoding and specialized drafters. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 10789–10802, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.602. URL https://aclanthology.org/2024.emnlp-main.602/.

Situo Zhang, Hankun Wang, Da Ma, Zichen Zhu, Lu Chen, Kunyao Lan, and Kai Yu. Adaeagle: Optimizing speculative decoding via explicit modeling of adaptive draft structures, 2024. URL https://arxiv.org/abs/2412.18910.

Ranran Zhen, Juntao Li, Yixin Ji, Zhenlin Yang, Tong Liu, Qingrong Xia, Xinyu Duan, Zhefeng Wang, Baoxing Huai, and Min Zhang. Taming the titans: A survey of efficient LLM inference serving. In *Proceedings of the 18th International Natural Language Generation Conference*, pp. 522–541, Hanoi, Vietnam, October 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.inlg-main.32/.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. Distillspec: Improving speculative decoding via knowledge distillation. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 32011–32050, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/8766fbc68e1ed1cdef712ce273e0a363-Paper-Conference.pdf.

A APPENDIX

A.1 TRAINING DATA AND CONFIGURATIONS

Each dataset uses a frozen prompt, field order, and serialization format. The visual tasks receive an image and a textual instruction with one format example. ASAP is a text-only task, receiving a restaurant review and a textual demonstration.

Among the four datasets, the smallest training set contains approximately 5,900 samples. To ensure a balanced training corpus, we sample 5,900 training samples from each dataset, yielding 23,600 samples in total. Training supervision is constructed by extracting schema-values and appending <END_VALUE> to each value. The end marker is used only by the drafter and is excluded in the target verification. Single-request evaluation uses 128 held-out examples per dataset, concurrent evaluation uses 512 held-out examples per dataset.

Table 2 reports the Qwen3.5-4B training configuration. Target features are extracted online on one GPU and consumed by drafter training on the other GPU. Target feature-layer indices follow the target-specific draft configuration.

Table 2: Qwen3.5-4B training configuration.

Setting	Value
Initialization	Corresponding open-source DFlash checkpoint
Training / validation examples per dataset	5,900 / 512
Epochs / random seed	3 / 20260907
Anchors per sample / windows per chunk	16 / 4
Draft prediction positions	15
Learning rate	10^{-5}
Optimizer	AdamW, $\beta = (0.9, 0.95)$
Weight decay / gradient clipping	0.01 / 1.0
Compute / trainable parameters and gradients	BF16 / FP32
Frozen parameters	Target and token embeddings
Training target backend	Hugging Face SDPA

A.2 OPEN-SOURCE CHECKPOINTS IN EXPERIMENTS

Table 3 lists the open-source checkpoints used in our experiments. For the two Qwen models, MTP uses the prediction head included in the target checkpoint and requires no separate drafter checkpoint.

A.3 END-BOUNDARY PREDICTION.

We compare DChord with two alternative strategies for determining schema-value boundaries. The *Static-length* baseline, inspired by DDD (Brown et al., 2024), assigns each pair a fixed length estimated from the training-set length distribution and does not adapt to the input at inference time. The *MLP-length* baseline, inspired by AdaEAGLE (Zhang et al., 2024), uses an additional single-layer MLP to predict the value length from the current input. We report end-position accuracy as well as Acc@2 and Acc@4, the fractions of cases in which the boundaries of two or four consecutive schema-values are all predicted correctly. Our boundary evaluation covers KAGL, CUB, and ASAP; CelebA is excluded because its schema-values are single-token Boolean values with no boundary-length variation.

As shown in Figure 6, DChord achieves average Acc, Acc@2, and Acc@4 of **98.05%**, **94.15%**, and **84.16%**, respectively, outperforming both alternatives. These results show that jointly predicting value content and end markers yields more accurate boundaries, particularly across consecutive schema-values, than either fixed lengths or a separate length-prediction head.

Table 3: Sources of the open-source speculative decoding checkpoints.

Target model	Method	Checkpoint source
Qwen3.5-4B	DFlash	`z-lab/Qwen3.5-4B-DFlash`
	MTP	Included in the target checkpoint
Qwen3.6-35B-A3B	DFlash	`z-lab/Qwen3.6-35B-A3B-DFlash`
	MTP	Included in the target checkpoint
Gemma4-12B	DFlash	`z-lab/gemma4-12B-it-DFlash`
	DSpark	`deepseek-ai/dspark_gemma4_12b_block7`
	EAGLE3	`deepseek-ai/eagle3_gemma4_12b_ttt7`

Table 4: First-rejection rate at non-value positions on Qwen3.5-4B with format constraints (%; lower is better). Among decoding rounds containing a rejection, we report the fraction whose first rejection occurs at a non-value position. AVG is the unweighted mean across the four datasets.

Method	CelebA	KAGL	CUB	ASAP	AVG
DFlash	82.04	33.03	68.62	23.31	51.75
DFlash-SFT	45.32	24.41	22.73	22.90	28.84
DChord	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**

A.4 Rejections on schema-determined structure

We compare first rejections at schema-key positions for DFlash, DFlash-SFT, and DChord. Among decoding rounds containing a rejection, we measure the fraction whose first rejection occurs at a schema-key position. As shown in Table 4, this fraction decreases from **51.75%** for DFlash to **28.84%** for DFlash-SFT, averaged equally across the four datasets. Fine-tuning therefore reduces, but does not eliminate, rejections at schema-key positions. DChord records no first rejections at schema-key positions across the four evaluated datasets.

A.5 Fine-tuning and pair-level drafting.

Table 5 shows that DFlash-SFT increases average acceptance length on all four datasets, compared to DFlash. Its average TPOT decreases from $3.190ms$ to $2.462ms$. Under the same 16-token verification limit, DChord(16) achieves longer accepted sequences and lower TPOT than DFlash-SFT(16). Across datasets, mean acceptance length rises from 11.623 to 13.856 tokens, and mean TPOT falls from $2.462ms$ to $1.947ms$. With full verification limit, DChord further increases acceptance length and reduces TPOT on all four datasets.

A.6 Schema-value lengths.

We measure schema-value lengths using the Qwen3.5-4B tokenizer on 128 samples from each dataset. CelebA contains only single-token Boolean values, whereas schema-values range from 1–4 tokens in KAGL, 1–9 in CUB, and 1–3 in ASAP. These tasks cover both single-token values and variable-length, multi-token values, requiring the drafter to predict their content as well as their end boundaries.

A.7 Single-request performance.

Table 6 reports TPOT speedup over AR and average acceptance length τ across three target models, four datasets, and both greedy ($T = 0$) and stochastic ($T = 1$) decoding. Schema-aware AR reduces TPOT by using the known output structure, while DChord achieves the lowest TPOT and longest average acceptance length in all 24 model–dataset–temperature combinations. At $T = 0$, DChord's average TPOT speedups over AR are **6.31**×, **7.47**×, and **6.18**× on Qwen3.5-4B, Qwen3.6-35B-A3B, and Gemma4-12B, respectively. At $T = 1$, its average TPOT speedup over AR across the three models and four datasets is **5.86**×.

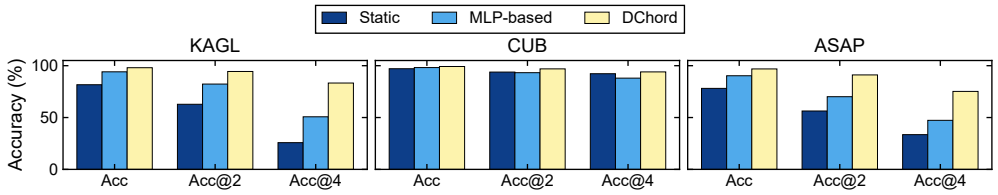


Figure 6: Dynamic value-length prediction accuracy on KAGL, CUB, and ASAP. Each sample contributes 16 randomly selected legal anchors.

Table 5: Fine-tuning ablation on Qwen3.5-4B under single-request greedy decoding. TPOT is measured in milliseconds (lower is better), and τ is average acceptance length (higher is better).

		CelebA		KAGL		CUB		ASAP		AVG	
Model	Method	TPOT	τ	TPOT	τ	TPOT	τ	TPOT	τ	TPOT	τ
Qwen3.5-4B	DFlash (16)	2.912	8.442	4.659	6.101	3.002	8.056	2.186	11.334	3.190	8.483
	DFlash-SFT (16)	1.837	14.213	3.706	8.007	2.108	12.335	2.198	11.938	2.462	11.623
	DChord (16)	1.724	15.256	2.419	11.112	1.763	14.880	1.880	14.175	1.947	13.856
	DChord	**1.320**	**24.999**	**2.206**	**15.613**	**1.464**	**23.141**	**1.467**	**22.378**	**1.614**	**21.533**

Table 6: TPOT speedup over AR and average acceptance length (τ) at temperatures $T = 0$ and $T = 1$. Speedup is the AR TPOT divided by the method's TPOT on the same model and dataset. Parenthesized values in method names indicate the draft block size. AVG is the unweighted arithmetic mean across the four datasets for each metric.

			CelebA		KAGL		CUB		ASAP		AVG	
	Model	Method	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
$T = 0$	Qwen3.5-4B	AR	1.00×	–	1.00×	–	1.00×	–	1.00×	–	1.00×	–
		Schema-aware AR	4.22×	–	1.46×	–	1.71×	–	2.30×	–	2.42×	–
		MTP (8)	2.70×	6.299	1.96×	5.039	2.29×	5.301	2.68×	6.139	2.41×	5.695
		DFlash (16)	3.32×	8.442	2.25×	6.101	3.22×	8.056	4.38×	11.334	3.29×	8.483
		DChord	**7.33×**	**24.999**	**4.76×**	**15.613**	**6.60×**	**23.141**	**6.54×**	**22.378**	**6.31×**	**21.533**
	Qwen3.6-35B-A3B	AR	1.00×	–	1.00×	–	1.00×	–	1.00×	–	1.00×	–
		Schema-aware AR	5.18×	–	1.48×	–	1.62×	–	2.32×	–	2.65×	–
		MTP (8)	2.66×	7.045	1.53×	4.905	2.09×	5.610	2.68×	6.947	2.24×	6.127
		DFlash (16)	4.36×	10.330	2.05×	5.218	3.70×	8.617	5.27×	12.312	3.84×	9.119
		DChord	**9.47×**	**24.151**	**5.76×**	**14.614**	**6.80×**	**17.628**	**7.84×**	**20.205**	**7.47×**	**19.149**
	Gemma4-12B	AR	1.00×	–	1.00×	–	1.00×	–	1.00×	–	1.00×	–
		Schema-aware AR	6.22×	–	1.74×	–	2.11×	–	2.59×	–	3.17×	–
		EAGLE3 (8)	3.38×	6.470	2.20×	4.233	2.62×	4.863	3.28×	6.582	2.87×	5.537
		DFlash (8)	3.98×	6.363	2.74×	3.975	2.76×	4.237	4.25×	6.677	3.43×	5.313
		DSpark (8)	4.19×	6.945	3.48×	5.351	3.30×	5.180	4.11×	6.953	3.77×	6.107
		DFlash (16)	4.50×	7.265	2.65×	4.032	2.79×	4.336	5.03×	8.038	3.74×	5.918
		DChord	**8.52×**	**23.900**	**5.10×**	**13.158**	**4.13×**	**9.571**	**6.97×**	**17.768**	**6.18×**	**16.099**
$T = 1$	Qwen3.5-4B	AR	1.00×	–	1.00×	–	1.00×	–	1.00×	–	1.00×	–
		Schema-aware AR	4.23×	–	1.44×	–	1.70×	–	2.33×	–	2.42×	–
		MTP (8)	2.62×	6.248	1.73×	4.648	2.00×	4.620	2.52×	6.014	2.22×	5.383
		DFlash (16)	3.30×	8.348	2.04×	5.834	2.38×	5.966	4.09×	10.864	2.95×	7.753
		DChord	**6.99×**	**22.155**	**3.85×**	**10.624**	**4.44×**	**14.588**	**5.70×**	**19.183**	**5.25×**	**16.638**
	Qwen3.6-35B-A3B	AR	1.00×	–	1.00×	–	1.00×	–	1.00×	–	1.00×	–
		Schema-aware AR	5.42×	–	1.39×	–	1.65×	–	2.29×	–	2.69×	–
		MTP (8)	2.67×	6.913	1.19×	3.769	2.05×	5.443	2.53×	6.830	2.11×	5.739
		DFlash (16)	4.37×	10.215	1.47×	3.770	3.34×	7.644	4.93×	12.023	3.53×	8.413
		DChord	**8.45×**	**21.615**	**4.09×**	**10.159**	**5.75×**	**14.461**	**7.07×**	**18.845**	**6.34×**	**16.270**
	Gemma4-12B	AR	1.00×	–	1.00×	–	1.00×	–	1.00×	–	1.00×	–
		Schema-aware AR	6.10×	–	1.74×	–	2.11×	–	2.59×	–	3.13×	–
		EAGLE3 (8)	3.30×	6.473	2.19×	4.328	2.52×	4.712	3.30×	6.577	2.83×	5.522
		DFlash (8)	3.92×	6.382	2.71×	3.974	2.63×	4.074	4.29×	6.656	3.39×	5.271
		DSpark (8)	4.15×	6.959	3.60×	5.493	3.15×	5.031	4.10×	6.936	3.75×	6.105
		DFlash (16)	4.38×	7.275	2.66×	4.060	2.64×	4.180	5.01×	7.976	3.67×	5.873
		DChord	**7.95×**	**23.488**	**5.22×**	**13.192**	**3.63×**	**8.411**	**7.12×**	**17.522**	**5.98×**	**15.653**

A.8 DATASET PROMPTS

CelebA.

```
Inspect the current face image and classify all 40 CelebA attributes.
Return exactly one JSON object whose fields are the following boolean
↪ attributes in the listed order:
5_o_Clock_Shadow, Arched_Eyebrows, Attractive, Bags_Under_Eyes, Bald,
↪ Bangs, Big_Lips, Big_Nose, Black_Hair, Blond_Hair, Blurry,
↪ Brown_Hair, Bushy_Eyebrows, Chubby, Double_Chin, Eyeglasses, Goatee,
↪ Gray_Hair, Heavy_Makeup, High_Cheekbones, Male, Mouth_Slightly_Open,
↪ Mustache, Narrow_Eyes, No_Beard, Oval_Face, Pale_Skin, Pointy_Nose,
↪ Receding_Hairline, Rosy_Cheeks, Sideburns, Smiling, Straight_Hair,
↪ Wavy_Hair, Wearing_Earrings, Wearing_Hat, Wearing_Lipstick,
↪ Wearing_Necklace, Wearing_Necktie, Young.
The following synthetic object is a format-only one-shot example; its
↪ values are unrelated to the current image:
{"5_o_Clock_Shadow":true,"Arched_Eyebrows":false,"Attractive":true,"Bags
↪ _Under_Eyes":false,"Bald":true,"Bangs":false,"Big_Lips":true,"Big_No
↪ se":false,"Black_Hair":true,"Blond_Hair":false,"Blurry":true,"Brown_
↪ Hair":false,"Bushy_Eyebrows":true,"Chubby":false,"Double_Chin":true,
↪ "Eyeglasses":false,"Goatee":true,"Gray_Hair":false,"Heavy_Makeup":tr
↪ ue,"High_Cheekbones":false,"Male":true,"Mouth_Slightly_Open":false,"
↪ Mustache":true,"Narrow_Eyes":false,"No_Beard":true,"Oval_Face":false
↪ ,"Pale_Skin":true,"Pointy_Nose":false,"Receding_Hairline":true,"Rosy
↪ _Cheeks":false,"Sideburns":true,"Smiling":false,"Straight_Hair":true
↪ ,"Wavy_Hair":false,"Wearing_Earrings":true,"Wearing_Hat":false,"Wear
↪ ing_Lipstick":true,"Wearing_Necklace":false,"Wearing_Necktie":true,"
↪ Young":false}
```

KAGL.

```
Inspect the current fashion-product image and classify all seven product
↪ attributes.
Return exactly one JSON object whose string-valued fields appear in this
↪ order: gender, category1, category2, category3, baseColour, season,
↪ usage.

Use category1 for the broad product family, category2 for its
↪ intermediate family, and category3 for the most specific visible
↪ product type. Follow this hierarchy: clothing is normally Apparel ->
↪ Topwear/Bottomwear/etc. -> a specific garment; shoes are normally
↪ Footwear -> Shoes/Sandal/Flip Flops -> a specific shoe type;
↪ accessories are normally Accessories -> Bags/Watches/Eyewear/etc. ->
↪ a specific accessory. Infer baseColour as the dominant product
↪ colour. Infer season as the retail season best supported by visible
↪ style, thickness, coverage, and material. Infer usage as the most
↪ likely intended use, such as Casual, Sports, Formal, or Ethnic.

The following synthetic object is a format-only one-shot example; its
↪ values are unrelated to the current image:
{"gender":"Women","category1":"Apparel","category2":"Topwear","category3
↪ ":"Tshirts","baseColour":"Navy
↪ Blue","season":"Fall","usage":"Casual"}
```

CUB.

```
Analyze the bird in the image and describe its visible attributes. Return
↪ exactly one JSON object with all 28 keys in exactly the same order as
↪ the example. Every value must be a short lowercase English
↪ categorical label; use underscores or hyphens for multi-word values.
↪ Do not omit any key, do not use null, and do not add explanation.
Example output:
```

```
{"bill_shape":"all-purpose","wing_color":"brown","upperparts_color":"bro
↪  wn","underparts_color":"white","breast_pattern":"solid","back_color"
↪  :"brown","tail_shape":"rounded_tail","upper_tail_color":"brown","hea
↪  d_pattern":"plain","breast_color":"white","throat_color":"white","ey
↪  e_color":"black","bill_length":"shorter_than_head","forehead_color":
↪  "brown","under_tail_color":"white","nape_color":"brown","belly_color
↪  ":"white","wing_shape":"rounded-wings","size":"small_(5_-_9_in)","sh
↪  ape":"perching-like","back_pattern":"solid","tail_pattern":"solid","
↪  belly_pattern":"solid","primary_color":"brown","leg_color":"black","
↪  bill_color":"black","crown_color":"brown","wing_pattern":"solid"}
```

ASAP.

```
Read the restaurant review and classify sentiment toward each of the 18
↪  aspects listed in the example output. Use one of four labels:
↪  positive, negative, neutral, or not_mentioned. Return exactly one
↪  JSON object with all fields in the listed order. Do not include
↪  explanations or any text outside the JSON object.

Illustrative review:
The restaurant is next to the subway in a downtown shopping mall, and the
↪  entrance is clearly signposted. We waited nearly an hour, and the
↪  staff were rude. Once seated, the food arrived quickly. The prices
↪  were high, but the generous portions made the meal good value. The
↪  decoration was ordinary. The room was noisy and the seats were
↪  cramped, although the tables and floor were spotless. The food was
↪  delicious, but its presentation was unattractive. I would still
↪  recommend the dishes.

Example output:
{"Location#Transportation":"positive","Location#Downtown":"positive","Lo
↪  cation#Easy_to_find":"positive","Service#Queue":"negative","Service#
↪  Hospitality":"negative","Service#Parking":"not_mentioned","Service#T
↪  imely":"positive","Price#Level":"negative","Price#Cost_effective":"p
↪  ositive","Price#Discount":"not_mentioned","Ambience#Decoration":"neu
↪  tral","Ambience#Noise":"negative","Ambience#Space":"negative","Ambie
↪  nce#Sanitary":"positive","Food#Portion":"positive","Food#Taste":"pos
↪  itive","Food#Appearance":"negative","Food#Recommend":"positive"}
```