WhiFlash: Accelerating Speculative Decoding with Token-Level Cross-Paradigm Routing

Young D. Kwon **Miles Williams** **Rui Li**
Alexandros Kouris **Stylianos I. Venieris**

Samsung AI Center, Cambridge, UK
Correspondence: yd.kwon@samsung.com

Abstract

The autoregressive nature of large language models (LLMs) remains a significant bottleneck for inference, particularly in complex agentic workloads. While speculative decoding (SD) accelerates inference, current approaches rely on static drafting paradigms, utilising either autoregressive drafting models for reasoning or diffusion-based parallel drafting models for structured outputs. We empirically find that drafting accuracy fluctuates dramatically within a single sequence, leaving significant performance unrealised by static paradigms and coarse-grained routing. To address this volatility, we introduce *WhiFlash*, the first cross-paradigm SD method that unifies autoregressive and diffusion-based parallel drafting under a single token-level controller. WhiFlash adopts a fine-grained routing mechanism that employs either a lightweight entropy-based or a learned neural policy, both parametrised to provide a tunable balance between expected token gain and latency. To make high-frequency switching computationally viable, we introduce novel cache-management optimisations, *Lazy Catch-up* and *KV-only Prefill*, reducing switching overhead to below 7% of per-round latency. By capitalising on the complementary strengths of fundamentally distinct drafting architectures, WhiFlash achieves significantly higher acceptance lengths, yielding category-specific throughput gains of up to 69.6% over the state-of-the-art autoregressive EAGLE-3 and 37.3% over the diffusion-based DFlash.

1 Introduction

Recent applications of large language models (LLMs) have evolved beyond conversational interfaces that primarily rely on parametric knowledge, towards agentic systems that interact with external environments through planning, reasoning, coding, and tool calling (Agarwal et al., 2025; Liu et al., 2025; Cao et al., 2026). However, their efficient deployment is severely hindered by the inherent latency of autoregressive decoding. To mitigate this, speculative decoding (SD) (Leviathan et al., 2023; Chen et al., 2023) has emerged as a primary solution. SD utilises a smaller draft model to propose tokens, which are subsequently verified in parallel by the target LLM. Nonetheless, as agentic outputs become more sophisticated, the demands placed on these draft models have diversified. Agentic workloads typically fluctuate between open-ended reasoning and highly structured phases (Kim et al., 2026a). Consequently, achieving peak decoding speed requires drafting mechanisms capable of dynamically handling these rapidly shifting contexts.

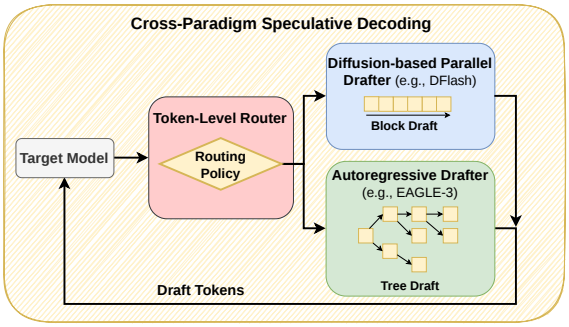

Figure 1: WhiFlash dynamically routes between both autoregressive and diffusion-based parallel drafters.

While a substantial body of work has advanced SD through various architectural approaches, state-of-the-art methods largely rely on static drafting paradigms. In the current landscape, standalone drafters typically excel in specific task categories while underperforming in others; autoregressive drafters (Li et al., 2025; Wang et al., 2025; Liu et al., 2026b) are highly effective in open-ended chat and novel reasoning, caching approaches (Ma et al., 2025; Dumitru et al., 2025; Oliaro et al., 2025) excel at repetitive tasks, whereas diffusion methods (Chen et al., 2026; Ringel and Romano, 2026) dominate highly structured outputs.

Crucially, beyond the cross-task performance variability, we make the key observation that drafting accuracy fluctuates aggressively even *within* a single generation sequence (Figure 2, Section 3.1). Existing multi-drafter orchestration methods do not adequately address this token-level volatility. For example, routing mechanisms, such as HedgeSpec (Liu et al., 2026a), formulate selection as an online learning problem designed to rapidly converge on a single, fixed expert for the remainder of the sequence. Furthermore, these frameworks operate under an assumption of architectural homogeneity, *i.e.* routing between identical types of models. Ultimately, by relying on a static drafter or coarse-grained task-level routing, current systems fail to capture token-level generation dynamics and leave significant performance gains unrealised.

In this work, we introduce WhiFlash (Figure 1), the first cross-paradigm SD method that unifies autoregressive and diffusion-based parallel drafting under a single dynamic controller. Rather than aiming to converge on a single global expert, WhiFlash capitalises on the complementary strengths of fundamentally different drafting paradigms. To perform this with minimal latency overhead, we design a token-level router that dynamically routes each decoding step to the highest-performing draft model at run time, using either an entropy-based or neural policy. We make the following core contributions:

- We present an analysis of intra-sequence generation dynamics, demonstrating that draft model accuracy fluctuates aggressively over token sequences and that coarse-grained, task-level routing is suboptimal.
- We propose WhiFlash, a unified speculative decoding method that dynamically orchestrates autoregressive and diffusion-based parallel drafting paradigms at maximum token-level granularity.
- Across a variety of tasks, WhiFlash achieves higher acceptance lengths than static SD methods, with throughput gains of up to 69.6% and 37.3% over EAGLE-3 and DFlash, respectively.

2 Related Work

Standalone Draft Models. Recent advances in SD have primarily focused on maximising the efficacy of standalone draft models through two distinct architectural paradigms. First, *autoregressive* (AR) methods capture sequential correlations between successive draft tokens and introduce dynamic, context-aware draft trees. For instance, EAGLE-3 (Li et al., 2025) abandons top-layer feature prediction for direct token prediction via multi-layer fusion. To optimise draft token exploration, methods such as TALON (Liu et al., 2026b) employ confidence-aware token trees, while OPT-Tree (Wang et al., 2025) adaptively searches for a tree structure that maximises acceptance length. These AR models excel in open-ended chat and complex reasoning tasks due to their sequential-dependence modelling. Second, *diffusion*-based parallel methods such as DFlash (Chen et al., 2026) bypass the sequential bottleneck of AR drafting by predicting tokens in blocks with a single forward pass, dominating in highly structured tasks such as coding. Despite these advances, our empirical analysis reveals that the efficacy of these individual drafters fluctuates aggressively within a single sequence (Section 3.1). Unlike these approaches that remain bound to a single paradigm, WhiFlash is a unified method that dynamically combines parallel and AR drafting at the decoding step level.

Hybrid Speculative Decoding. To overcome the limitations of purely generative draft models, a few hybrid approaches have emerged that rely heavily on retrieval mechanisms. RASD (Quan et al., 2025) fuses generative draft trees with suffix-matched retrieval trees sourced from existing datastores. SuffixDecoding (Oliaro et al., 2025) utilises suffix trees to cache long token sequences from prompts and previous outputs, while adaptively scaling its speculation length. The paper also studies a basic hybrid variant that falls back to an autoregressive drafter when historical pattern matching yields low confidence. Furthermore, ReSpec (Fang et al., 2025) transitions from heuristic retrieval to an adaptive framework via an entropy-guided trigger to decide whether to retrieve from historical context or fall back to an autoregressive drafter. While showing promise for repetitive agentic loops, retrieval inherently falters when generating novel reasoning sequences. In contrast to these retrieval-dependent systems, WhiFlash relies either on a learned neural router or on readily available runtime metrics, rather than historical text caches, to make fine-grained paradigm-switching decisions.

Homogeneous Drafter Routing. The problem of selecting the best drafter from a pool of candidates has recently gained attention. Early ap-

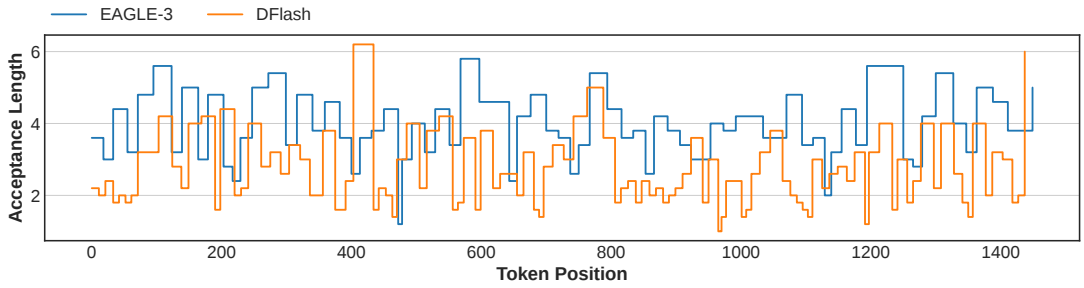

Figure 2: Token-level acceptance lengths for EAGLE-3 and DFlash. Acceptance length fluctuates substantially throughout the response, and the positions where each drafter performs well are different.

proaches, such as MetaSD-UCB (Kim et al., 2026b), framed this as a partial-information problem, utilising multi-armed bandit algorithms to route between domain-specific AR models. To optimise compute allocation, hierarchical methods like Cascade Speculative Drafting (Chen et al., 2024) introduce dynamic routing between models of varying sizes. More recently, HedgeSpec (Liu et al., 2026a) advanced this by framing drafter selection as a full-information online learning problem. It constructs an unbiased estimator of candidate acceptance lengths from the verified trajectory, which avoids extra target model calls. It then orchestrates a pool of domain experts using a regret-minimisation algorithm.

However, existing routing frameworks fundamentally operate within a single architectural paradigm and optimise for a dominant expert across a task. Because the optimal generative paradigm (autoregressive versus parallel) shifts dramatically throughout a single sequence, WhiFlash tackles a fundamentally different architectural duality. Rather than optimising to converge to a single expert, WhiFlash introduces a token-level router that continuously toggles between architectures to capitalise on entropy variability within a sequence.

3 Cross-Paradigm SD with WhiFlash

3.1 Motivation

Considering the two best-performing general-purpose drafting approaches, *i.e.* autoregressive and diffusion-based, we observe that both paradigms handle token-level dependencies differently. Autoregressive tree drafters generate tokens sequentially, with each position conditioning on earlier ones. In contrast, diffusion-based drafters generate a block in parallel, with positions conditioning on the prefix but not on each other.

To investigate further, we compare the speculation accuracy of autoregressive and diffusion approaches at each generation step. Figure 2 presents an illustrative example of the acceptance lengths across an entire generated sequence for both approaches. We observe that the acceptance length swings between one and six tokens within a single sequence, and which drafter is superior can vary substantially between positions. However, the *mean* acceptance length conceals this divergence.

The volatility in acceptance length suggests that neither the autoregressive nor the diffusion drafting paradigm is superior throughout an entire sequence. This serves as the principal motivation for WhiFlash, a new hybrid SD method that dynamically routes between heterogeneous drafters at each decoding step. By operating at this higher granularity, WhiFlash can adaptively leverage the unique strengths of each drafting paradigm, enabling even faster decoding across a variety of tasks.

3.2 Proposed Architecture

Figure 1 depicts our proposed hybrid SD method, outlining the generation process. First, the *target model* processes the current sequence (*e.g.* a user prompt) and samples the next token. The representations from the target model at the current decoding step are used to extract specific features. Next, the *step-level router* ingests these features and uses its *routing policy* to select the best-performing drafter for the upcoming speculation round. Based on this routing decision, WhiFlash activates the corresponding speculation method. In turn, drafting proceeds through either an *autoregressive drafter* which produces a tree of candidate tokens, or a *diffusion-based parallel drafter* which predicts a parallel block of tokens. This sequence of candidate tokens is then returned to the target model for parallel verification. Finally, the process repeats, continuing from the last accepted token.

3.3 Token-Level Router

To accommodate the dense intra-sequence drafting-accuracy fluctuations without inducing prohibitive computational overhead, the routing mechanism must be both highly granular and efficient. To this end, we introduce a *token-level router*, our core module that makes dynamic decisions by means of a *routing policy*. To swiftly adapt to changing contexts within a given sequence, the token-level router is designed to be invoked at every decoding step, *i.e.* once per speculation round, enabling maximum decision-making granularity.

Given the available draft models $\{m_{\text{a}}, m_{\text{d}}\}$, where m_{a} and m_{d} are the autoregressive and diffusion draft models, respectively, the routing policy aims to determine which drafter to utilise at decoding step t, such that the acceptance length is maximised. Under this setup, we propose two types of routing policies: (1) a runtime-informed policy that utilises available statistics for routing decisions, and (2) a neural policy that is learned offline prior to deployment.

3.3.1 Runtime-Informed Policy

We first introduce a runtime-informed routing policy, **WhiFlash-Entropy**, that uses the entropy of the next-token distribution from the target model as the routing signal. Our hypothesis is that entropy indicates which drafting paradigm performs better at a given step. When the next-token distribution has low entropy, the next token is more predictable, and the diffusion drafter can commit an entire block accurately whilst benefiting from its single-pass generation. When entropy is high, the next token has greater uncertainty, and the parallel drafter, which fixes a whole block at once, is more likely to mispredict. In contrast, the autoregressive drafter, which drafts sequentially and resolves one position at a time, is better suited to this regime.

Concretely, at each speculation round, the target model verifies the tokens proposed by the current drafter and produces a next-token distribution $p(x_t \mid x_{<t})$ over the vocabulary $\mathcal{V}$ for the final accepted position. The router computes the entropy, $H_t = -\sum_{x\in\mathcal{V}} p(x \mid x_{<t}) \log p(x \mid x_{<t})$, and selects the draft model for the next round as:

$$\pi(H_t, \tau_{\text{entropy}}) = \begin{cases} m_{\text{a}}, & \text{if } H_t > \tau_{\text{entropy}} \\ m_{\text{d}}, & \text{otherwise} \end{cases}$$

where τ_{entropy} is the entropy threshold. Computing the entropy requires an $O(|\mathcal{V}|)$ reduction over the target next-token probabilities at a single position, adding a softmax only when greedy decoding is used. This cost is negligible compared to a draft model forward pass, making WhiFlash-Entropy extremely lightweight for routing during inference.

Calibration. The threshold τ_{entropy} is calibrated on a small held-out set of prompts, typically 32 to 64 prompts per category (*e.g.* Chat, Coding, Math, and Agentic), via grid search. The procedure runs WhiFlash-Entropy on each prompt to obtain throughput and selects the τ_{entropy} that maximises mean throughput under the resulting routing decisions. Note that calibration is inexpensive, requiring no model training and completing in a few minutes on a single GPU in our experiments.

3.3.2 Neural Policy

In this subsection, we introduce a learned policy, **WhiFlash-Neural**, parametrised as a neural network f_θ, designed around three core principles. First, our policy takes the final-layer hidden states of the target model at the current decoding step t, denoted by $h_t \in \mathbb{R}^d$, where d is the hidden dimension of the target model. With this approach, we aim to leverage the contextual information already embedded within the target model. Second, the network directly predicts a continuous scalar, $\Delta\widehat{AL}_t = f_\theta(h_t)$, representing the expected difference in acceptance length between the autoregressive and the diffusion-based parallel drafter at step t. In contrast to a binary classification approach, this strategy enables us to quantify the expected token gain when selecting one of the drafters over the other. Finally, to ensure that the decoding step invocation frequency does not counteract drafting gains, we parameterise the router as a lightweight multilayer perceptron (MLP) regressor, consisting primarily of two linear layers.[1]

Training. Given a target LLM, the neural router undergoes an offline process, comprising two phases: (1) dataset construction and (2) training.

In the first phase, the training set is generated by processing a task-agnostic dataset to extract (h_t, y_t) pairs at each decoding step. For the training target y_t, we opt to use the difference in acceptance length between the two draft models, measured in tokens. That is, $\Delta AL = AL_{\text{a}} - AL_{\text{d}}$, where AL_{a} and AL_{d} are acceptance lengths for autoregressive and diffusion drafting, respectively. We collect the

[1] Full architectural details are presented in Appendix A.

acceptance lengths by running a complete drafting cycle per decoding step for each drafter separately, and calculate and store their differences as targets.

After the dataset construction, we proceed with the training phase, supervising the MLP on these (h_t, y_t) pairs. This objective directly optimises the network to predict the expected token gain when switching drafting methods. Overall, as our neural router is trained on this generic, task-independent dataset and then applied across various downstream tasks, its training cost becomes amortised.

Hardware-Aware Routing Decision. During generation, we translate the router's continuous prediction into a discrete routing decision. Let f_θ denote the trained neural regressor. At decoding step t, the router predicts the expected token gain, $\Delta\widehat{AL}_t$, given the target model's hidden states h_t, as $\Delta\widehat{AL}_t = f_\theta(h_t)$.

A conventional strategy would select drafting methods based on the prediction's sign, *i.e.* switch to the autoregressive drafter if $\Delta\widehat{AL}_t > 0$. However, this assumes both draft architectures incur identical computational costs, which is rarely true in practice. Instead, WhiFlash relies on a hardware-aware decision function in order to determine the selected draft model, controlled through a routing threshold τ_{neural}:

$$\pi(h_t, \tau_{\text{neural}}) = \begin{cases} m_{\text{a}}, & \text{if } \Delta\widehat{AL}_t > \tau_{\text{neural}} \\ m_{\text{d}}, & \text{otherwise} \end{cases}$$

The value of threshold τ_{neural} is explicitly informed by system-level metrics, such as the inference latency of each draft model and the switching latency overhead on the target hardware. As a result, the router only executes a paradigm switch when the predicted gain in accepted tokens reliably outweighs the asymmetric computational overhead of the alternative drafter.

3.4 System Optimisation

Operationalising two drafters at the token level poses two engineering questions concerning the AR drafter's KV cache: (1) when it should be updated, and (2) how the update itself should be computed. We propose lazy catch-up and KV-only prefill to address these issues, respectively.

Lazy Catch-up. The naive approach continuously keeps the KV cache for the inactive drafter up to date, performing prefill over every accepted token on every round, regardless of which drafter is chosen. Switches become instantaneous, although the inactive drafter repeatedly performs small prefill operations, losing the advantage of batched computation. Instead, we defer the updates for the inactive drafter entirely: no work is performed for the inactive drafter until a switch is committed. Upon preparing for a switch, the incoming drafter prefills its KV cache over all tokens accepted during its inactivity as a single batched operation. We refer to these tokens, accepted while a drafter was inactive and therefore missing from its KV cache, as *backlog tokens*. Their quantity defines the size of the catch-up prefill at each switch.

KV-only Prefill. The computational cost of the catch-up prefill itself can be reduced further. A full forward pass on each token computes self-attention, the MLP, and the LM head. However, for catch-up, only the KV cache is required since the drafter does not need predictions for tokens that the target has already accepted. Therefore, we compute only the attention layer on tokens accepted by another drafter, processing only the final token in full, so that its post-MLP hidden state can feed the next draft round. With this technique, WhiFlash keeps switching cost below 7% of per-round latency.

4 Experimental Setup

Baselines. We employ two state-of-the-art standalone drafters, *EAGLE-3* (Li et al., 2025) and *DFlash* (Chen et al., 2026). To ascertain the maximum achievable gain at different levels of routing granularity, we report three oracles: (1) *Oracle-Task* selects, in hindsight, the single best drafter for each task/dataset; (2) *Oracle-Prompt* selects the single best drafter for each prompt, representing the upper-bound performance of prompt-level routing approaches, such as HedgeSpec (Liu et al., 2026a); and (3) *Oracle-Token* selects the best drafter at every decoding step, representing the best achievable routing quality for the two drafters. To maintain identical sequences across drafters, Oracle-Token is computed in `float32`, while all other methods use `bfloat16`. Thus, the acceptance length is a valid token-level upper bound, but the throughput is not comparable. Accordingly, we omit the speedup.

Models. To assess the generalisation of each method, we target three open-weight LLMs from two model families. We employ *Qwen3* (Yang et al., 2025) in 4B and 8B sizes, in addition to *Llama-3.1-8B* (Grattafiori et al., 2024). We se-

lect these models because they are among the best-performing models in their size class and enable a direct comparison with prior work.

Training/Calibration. Our lightweight neural policy is trained offline and independently for each target model. We construct the training set following the process from Section 3.3.2. We adopt a task-agnostic corpus, specifically the Dolci Instruct SFT post-training dataset (Ettinger et al., 2025). Our neural router is optimised with AdamW (Kingma and Ba, 2017) using an MSE loss with a peak learning rate of 1×10^{-3}. We train for a single epoch on a 25M token subset, using a batch size of 8K tokens. Because the corpus is task-independent, this cost is incurred once per target model and amortised across all downstream tasks. Our runtime-informed policy requires no training. We calibrate its single threshold τ_{entropy} with grid search over a held-out set of 32 to 64 prompts per category and retain the value that maximises mean throughput.

Evaluation. We evaluate across four task categories: (1) *Chat*, with MT-Bench (Zheng et al., 2023), Alpaca (Taori et al., 2023), UltraChat (Ding et al., 2023), and WildChat (Zhao et al., 2024); (2) *Math*, with GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2024), AIME24, and AIME25 (AIME, 2025); (3) *Coding*, with HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and LiveCodeBench (Jain et al., 2025); and (4) *Agentic*, with SWE-Bench (Jimenez et al., 2024), Tau-Bench (Yao et al., 2025), AgentBench (Liu et al., 2024), and AgentInstruct (Mitra et al., 2024). We report full dataset statistics in Appendix A. All measurements are based on NVIDIA H100 GPUs.

Metrics. We focus our evaluation on three key metrics: (1) *mean acceptance length* (AL), the average number of tokens accepted per target verification round; (2) *throughput* (TPS), the number of generated tokens per second in wall-clock time; and (3) *speedup*, the throughput ratio versus plain autoregressive decoding for the same target model. All measurements are based on the decoding stage.

5 Results

WhiFlash consistently outperforms both DFlash and EAGLE-3 standalone drafters. Table 1 reports mean AL per category for the standalone drafters, the three oracles, and both WhiFlash routers. Neither standalone drafter is uniformly best. By mean AL, EAGLE-3 leads on Chat and Agentic across all models and DFlash leads on Math and Coding for Qwen models, whereas by speedup, DFlash leads Math, Coding, and Agentic on all models, since its lower per-round latency turns even a lower AL into a higher TPS. This highlights that a router must intelligently select the appropriate drafter; otherwise, it risks harming decoding performance. Oracle-Token, the ideal upper bound of token-level routing, sits above both standalone drafters and the coarser oracles. This quantifies the AL that only token-level routing reaches and that coarser routing leaves unrealised.

WhiFlash narrows this gap substantially: averaged across categories, WhiFlash-Neural attains a mean AL of 6.26 tokens on Qwen3-8B, a 22.3% gain over EAGLE-3 and an 18.5% gain over DFlash. It also surpasses Oracle-Task by 6.8% and Oracle-Prompt by 5.7% and closes much of the gap (short of merely 8.0%) to the performance ceiling represented by Oracle-Token.

WhiFlash throughput surpasses optimal task- and prompt-level routing strategies. Table 1 reports decoding throughput per category for each target model, with the AL advantage carrying through to wall-clock gains. We report gains relative to DFlash as it is the stronger standalone baseline at this scale. For Qwen3-4B, WhiFlash-Entropy improves average throughput over DFlash by 7.8% and WhiFlash-Neural by 10.0%, while the corresponding gains for Qwen3-8B are 7.7% and 9.1%. Both routers are calibrated per model and per category. The learned policy proves to be the stronger of the two, as it conditions on hidden states from the target model, capturing routing signals that a calibrated entropy threshold cannot. The gains are largest where the standalone drafters are weakest: relative to DFlash, WhiFlash improves throughput on Chat by up to 37.3% (Qwen3-8B), the category where diffusion drafting is least effective. Compared to EAGLE-3, it reaches up to 69.6% on Math, 63.4% on Coding, and 19.9% on Agentic tasks (Qwen3-4B).

Crucially, WhiFlash outperforms Oracle-Prompt, which represents the hindsight upper bound of any prompt-level router, including HedgeSpec (Liu et al., 2026a). These performance gains reveal that the token-level ceiling lies above what is achievable by purely prompt-level routing strategies. This finding supports our hypothesis that the optimal routing paradigm operates on a token-level basis,

Model	Method	Chat		Math		Coding		Agentic		Total Avg.	
		AL	Speedup (×)	AL	Speedup (×)	AL	Speedup (×)	AL	Speedup (×)	AL	Speedup (×)
Qwen3-4B	EAGLE-3	5.21	2.58	5.44	2.94	5.25	2.79	3.99	2.02	4.95	2.57
	DFlash	3.43	2.08	7.22	4.75	6.45	4.29	3.56	2.34	5.08	3.30
	Oracle-Task	5.21	2.58	7.22	4.75	6.45	4.29	3.99	2.34	5.67	3.44
	Oracle-Prompt	5.29	2.76	7.22	5.00	6.49	4.45	4.01	2.41	5.71	3.60
	Oracle-Token	5.68	-	8.36	-	7.79	-	4.81	-	6.59	-
	WhiFlash-Entropy	5.19	2.69	7.36	4.90	6.73	4.46	4.22	2.41	5.82	3.56
	WhiFlash-Neural	**5.41**	**2.78**	**7.66**	**4.98**	**6.97**	**4.57**	**4.40**	**2.42**	**6.05**	**3.63**
Qwen3-8B	EAGLE-3	5.24	2.73	5.58	3.13	5.29	2.90	4.42	2.38	5.12	2.78
	DFlash	3.40	2.12	7.38	5.23	6.56	4.44	4.12	2.62	5.29	3.55
	Oracle-Task	5.24	2.73	7.38	5.23	6.56	4.44	4.45	2.62	5.86	3.71
	Oracle-Prompt	5.34	2.95	7.39	5.20	6.63	4.63	4.53	2.72	5.93	3.82
	Oracle-Token	5.72	-	8.56	-	7.98	-	5.26	-	6.81	-
	WhiFlash-Entropy	5.24	2.84	7.54	5.28	6.97	4.64	4.78	2.73	6.08	3.82
	WhiFlash-Neural	**5.47**	**2.92**	**7.72**	**5.30**	**7.20**	**4.67**	**4.90**	**2.78**	**6.26**	**3.87**
Llama3.1-8B	EAGLE-3	4.99	2.22	5.25	2.42	5.62	2.71	4.28	1.95	4.99	2.30
	DFlash	3.62	2.30	4.49	2.88	4.65	3.04	3.28	2.10	3.97	2.55
	Oracle-Task	4.99	2.30	5.25	2.88	5.62	3.04	4.28	2.16	4.99	2.57
	Oracle-Prompt	4.44	2.42	4.85	2.95	4.92	3.08	3.93	2.23	4.51	2.64
	Oracle-Token	5.30	-	5.95	-	6.76	-	4.58	-	5.36	-
	WhiFlash-Entropy	4.97	2.34	5.34	2.89	5.58	**3.09**	4.30	**2.17**	5.01	**2.59**
	WhiFlash-Neural	**5.02**	**2.37**	**5.48**	**2.88**	**5.79**	3.07	**4.45**	2.16	**5.14**	**2.59**

Table 1: Per-category acceptance length (AL) and speedup over vanilla AR decoding, for three target models across four task categories. Oracle rows are hindsight upper bounds. **Bold** denotes the best deployable method. Notably, the Oracle-Token speedup is omitted as it requires `float32` inference, preventing comparison with `bfloat16` inference.

τ_{neural}	Chat	Math	Coding	Agentic
-1.0	149.5	238.4	223.1	136.7
-0.5	147.2	247.7	226.7	137.4
0.0	146.1	249.1	223.7	140.0
0.5	148.1	250.6	228.5	**142.7**
1.0	**150.0**	**263.4**	**237.5**	140.6
1.5	148.4	262.4	235.8	139.4
2.0	143.4	256.2	221.9	134.2

Table 2: Threshold sensitivity analysis with respect to the throughput of Qwen3-8B with WhiFlash-Neural.

Model	Lazy catch-up	TPS	% Increase
Qwen3-4B	No	168.3	–
	Yes	190.9	13.4%
Qwen3-8B	No	172.7	–
	Yes	193.7	12.1%
Llama3.1-8B	No	161.5	–
	Yes	183.3	13.5%

Table 3: Lazy catch-up improves throughput without altering token acceptance through deferring KV updates.

rather than at the prompt-level (Section 3.1).

6 Analysis

WhiFlash is robust across a wide threshold range. Table 2 shows per-category throughput of WhiFlash-Neural on Qwen3-8B as the routing threshold τ_{neural} varies. For WhiFlash-Neural, the best throughput is observed within a tight band over $\tau_{\text{neural}} \in [0.5, 1.5]$. With Qwen3-8B, mean throughput stays above 184.5 tokens/s, within 7% of the best setting while outperforming DFlash throughout. It degrades only as the threshold approaches the extremes that recover the standalone behaviour. Per-category calibration, which selects τ_{neural} separately per-category, improves over the best single global threshold. For Qwen3-4B, it reaches 193.9 tokens/s, yielding a 10.0% gain over DFlash and 41.1% over EAGLE-3.

Lazy catch-up improves throughput at no accuracy cost. Table 3 isolates the effect of the lazy catch-up scheme (Section 3.4). The naive alternative continuously updates the KV cache of the inactive drafter. This incurs a per-token prefill cost on every round, regardless of the routing decision, ultimately losing the benefit of batched computation. In contrast, lazy catch-up, which defers these updates until the router commits to a switch, performs only a single batched catch-up prefill. This avoids the redundant per-token cost and yields a throughput improvement of 12.1–13.5% on average. The mean AL and downstream task accuracy remain unaffected, since the tokens accepted by the target are identical under both schemes.

KV-only prefill keeps switching cost below 7% of per-round latency. We profile the cost of a

	Backlog Tokens				
Prefill Tokens	1	64	128	512	1024
512	3.3%	3.8%	3.8%	3.8%	4.9%
1024	3.3%	3.8%	3.8%	3.9%	5.2%
2048	3.2%	3.8%	3.8%	4.1%	5.6%
4096	2.8%	3.7%	3.7%	4.2%	**5.8%**
8192	1.9%	3.4%	3.4%	3.8%	5.7%

Table 4: Profiling results for switching cost (% w.r.t. per-round latency) with different base KV lengths and backlog sequence lengths for Qwen3-8B.

cross-paradigm switch, which comprises the prefill of the backlog tokens for the EAGLE-3 drafter. The KV-only prefill of Section 3.4 computes attention over the backlog tokens and continues processing only the final token in full. This avoids invoking the transformer MLP layer and LM head for tokens that have already been accepted by the target model, reducing the switching cost by 15%.

We then benchmark the switching cost by running all prompts across all tasks. We observe that the average number of backlog tokens is around 80–90 across different tasks, and that switching overhead grows linearly with the number of backlog tokens. However, per-round latency also increases, keeping the relative cost bounded. Table 4 presents the switching cost considering different sequence lengths for both the base KV cache and backlog. Notably, this remains below 7% of the per-round latency across all instances, and around 3–5% at the representative 80–90 token backlog. This demonstrates that operating two drafters does not erode the throughput gains yielded by the higher AL.

The neural router adds negligible decoding step overhead. We now investigate the overhead of operationalising our lightweight neural router. Note that WhiFlash-Entropy relies on the runtime-informed policy, whose routing cost is the single-position softmax followed by entropy computation. Therefore, the overhead is negligible, taking 0.10 ms (0.24% of the per-round latency). WhiFlash-Neural adds a single MLP execution per decoding step. Table 6 shows that this execution consistently accounts for less than 1% of the per-round latency, below the switching cost characterised above and comfortably within the throughput gains of Table 1. The computational cost of the two linear layers in the neural router is negligible relative to a draft forward pass. Consequently, the learned policy improves routing quality over WhiFlash-Entropy at no meaningful runtime penalty.

Method	**Memory (GB)**	**% Increase**
Qwen3-8B (No SD)	15.26	–
EAGLE-3	16.39	7.4%
DFlash	17.62	15.4%
WhiFlash-Entropy/Neural	18.41	20.6%

Table 5: Peak memory footprint of EAGLE-3, DFlash, and WhiFlash. The memory overhead from WhiFlash over a diffusion-only system (DFlash) is 0.79 GB, merely 5.2% of the target model memory footprint.

Model	**Parameters**	**Latency**	**% Drafting**
Qwen3-4B	0.82M	0.29 ms	0.73%
Qwen3-8B	2.10M	0.30 ms	0.78%
Llama3.1-8B	2.10M	0.30 ms	0.89%

Table 6: Neural router characteristics, including the average latency relative to the overall drafting round.

WhiFlash adds single-digit memory overhead over a diffusion drafter. WhiFlash holds two drafters and the neural router in memory, incurring a greater memory footprint than a single-drafter regime. To understand the extent of this, we examine the peak memory footprint of EAGLE-3, DFlash, and WhiFlash. Table 5 reports the peak memory usage for each approach with Qwen3-8B and 2048 maximum new tokens. Overall, WhiFlash incurs merely 5.2% memory overhead compared to DFlash. The overhead is minimal since the router is a two-layer MLP and the autoregressive drafter is a single transformer layer. In comparison to the target model and diffusion drafter, both are relatively lightweight. Therefore, WhiFlash offers both drafting paradigms at a memory cost close to that of a single diffusion drafter.

7 Conclusion

We proposed WhiFlash, a speculative decoding method that routes between autoregressive and diffusion-based parallel drafters at each decoding step, driven by the observation that no single paradigm dominates an entire sequence. WhiFlash leverages either a virtually cost-free runtime-informed policy (WhiFlash-Entropy) or a lightweight neural policy (WhiFlash-Neural), and further system optimisations (lazy catch-up with KV-only prefill) to keep switching cost below 7% of per-round latency. WhiFlash improves throughput by up to 37.3% over DFlash on Chat and by up to 69.6% over EAGLE-3 on Math, achieving a high throughput that task-level and prompt-level routing cannot reach.

Limitations

WhiFlash maintains two drafters concurrently with a lightweight router, increasing memory overhead relative to a single-drafter system. However, we highlight that the autoregressive drafter is a single layer, so the added memory over a diffusion-only baseline is marginal. This is also mitigated by our lazy catch-up and KV-only prefill, which keep switching cost below 7% of per-round latency. Finally, we note our evaluation focuses on the 4B to 8B scale, leaving behaviour at larger scales yet to be examined.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, Kai Chen, and 106 others. 2025. gpt-oss-120b & gpt-oss-20b Model Card. *Preprint*, arXiv:2508.10925.

AIME. 2025. AIME problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIMEProblemsandSolutions.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. *Preprint*, arXiv:2108.07732.

Ruisheng Cao, Mouxiang Chen, Jiawei Chen, Zeyu Cui, Yunlong Feng, Binyuan Hui, Yuheng Jing, Kaixin Li, Mingze Li, Junyang Lin, Zeyao Ma, Kashun Shum, Xuwu Wang, Jinxi Wei, Jiaxi Yang, Jiajun Zhang, Lei Zhang, Zongmeng Zhang, Wenting Zhao, and Fan Zhou. 2026. Qwen3-Coder-Next Technical Report. *Preprint*, arXiv:2603.00729.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating Large Language Model Decoding with Speculative Sampling. *Preprint*, arXiv:2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. 2026. DFlash: Block Diffusion for Flash Speculative Decoding. *Preprint*, arXiv:2602.06036.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating Large Language Models Trained on Code. *Preprint*, arXiv:2107.03374.

Ziyi Chen, Xiaocong Yang, Jiacheng Lin, Chenkai Sun, Kevin Chen-Chuan Chang, and Jie Huang. 2024. Cascade speculative drafting for even faster llm inference. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NeurIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *Preprint*, arXiv:2110.14168.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023. Enhancing chat language models by scaling high-quality instructional conversations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3029–3051, Singapore. Association for Computational Linguistics.

Razvan-Gabriel Dumitru, Minglai Yang, Vikas Yadav, and Mihai Surdeanu. 2025. CopySpec: Accelerating LLMs with speculative copy-and-paste. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 26301–26332, Suzhou, China. Association for Computational Linguistics.

Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, and 48 others. 2025. Olmo 3. *Preprint*, arXiv:2512.13961.

Min Fang, Zhihui Fu, Qibin Zhao, and Jun Wang. 2025. When, What, and How: Rethinking Retrieval-Enhanced Speculative Decoding. *Preprint*, arXiv:2511.01282.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The Llama 3 Herd of Models. *Preprint*, arXiv:2407.21783.

Dan Hendrycks and Kevin Gimpel. 2023. Gaussian Error Linear Units (GELUs). *Preprint*, arXiv:1606.08415.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues? In *The Twelfth International Conference on Learning Representations*.

Jiin Kim, Byeongjun Shin, Jinha Chung, and Minsoo Rhu. 2026a. The Cost of Dynamic Reasoning: Demystifying AI Agents and Test-Time Scaling from an AI Infrastructure Perspective. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

Taehyeon Kim, Hojung Jung, and Se-Young Yun. 2026b. Multi-Drafter Speculative Decoding with Alignment Feedback. *Preprint*, arXiv:2604.05417.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. *Preprint*, arXiv:1412.6980.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast Inference from Transformers via Speculative Decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025. EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test. In *Advances in Neural Information Processing Systems*, volume 38, pages 136737–136756. Curran Associates, Inc.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's Verify Step by Step. In *The Twelfth International Conference on Learning Representations*.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, and 244 others. 2025. DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models. *Preprint*, arXiv:2512.02556.

Hongyi Liu, Jiaji Huang, Zhen Jia, Youngsuk Park, and Yu-Xiang Wang. 2026a. Not-a-Bandit: Provably No-Regret Drafter Selection in Speculative Decoding for LLMs. In *The Fourteenth International Conference on Learning Representations*.

Tianyu Liu, Qitan Lv, Yuhao Shen, Xiao Sun, and Xiaoyan Sun. 2026b. TALON: Confidence-Aware Speculative Decoding with Adaptive Token Trees. *Preprint*, arXiv:2601.07353.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2024. AgentBench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*.

Zhiyao Ma, In Gim, and Lin Zhong. 2025. Cacheback: Speculative decoding with nothing but cache. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 31079–31084, Suzhou, China. Association for Computational Linguistics.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei ge Chen, Olga Vrousgos, Corby Rosset, Fillipe Silva, Hamed Khanpour, Yash Lara, and Ahmed Awadallah. 2024. AgentInstruct: Toward generative teaching with agentic flows. *Preprint*, arXiv:2407.03502.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. 2025. SuffixDecoding: Extreme Speculative Decoding for Emerging AI Applications. In *Advances in Neural Information Processing Systems*, volume 38, pages 126326–126354. Curran Associates, Inc.

Guofeng Quan, Wenfeng Feng, Chuzhan Hao, Guochao Jiang, Yuewei Zhang, and Hao Henry Wang. 2025. RASD: Retrieval-augmented speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6167–6177, Vienna, Austria. Association for Computational Linguistics.

Liran Ringel and Yaniv Romano. 2026. Accelerating Speculative Decoding with Block Diffusion Draft Trees. *Preprint*, arXiv:2604.12989.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. `https://github.com/tatsu-lab/stanford_alpaca`.

Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. 2025. OPT-tree: Speculative decoding with adaptive draft tree structure. *Transactions of the Association for Computational Linguistics*, 13:188–199.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao,

Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 Technical Report. *Preprint*, arXiv:2505.09388.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. 2025. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *The Thirteenth International Conference on Learning Representations*.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. 2024. WildChat: 1M ChatGPT interaction logs in the wild. In *The Twelfth International Conference on Learning Representations*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and Chatbot Arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NeurIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Method	Hyperparameters
EAGLE-3	depth 8, top-k 10, draft tokens 60
DFlash	block size 16 (Qwen3), 10 (Llama-3.1-8B)
WhiFlash	Identical to the above

Table 7: Drafter hyperparameter settings for EAGLE-3, DFlash, and WhiFlash.

A Detailed Experimental Setup

Neural Router Architecture. The neural policy is a multilayer perceptron that maps the target model's final hidden state to a scalar output. Given a final hidden state $h_t \in \mathbb{R}^d$ where d is the target model dimensionality, the router first projects h_t to an intermediate dimension $m = \frac{d}{8}$. This intermediary representation is then supplied to a Gaussian Error Linear Unit (GELU) activation (Hendrycks and Gimpel, 2023), followed by dropout. Finally, the intermediary representation is projected to a scalar output $y_t \in \mathbb{R}$, used for the router policy.

Drafter Hyperparameters. For the autoregressive drafter, we follow the original EAGLE-3 paper (Li et al., 2025), using a draft-tree depth of 8, top-k of 10, and 60 total draft tokens. For the diffusion drafter, DFlash (Chen et al., 2026) uses a block size of 16 for the Qwen3 models (4B and 8B) and 10 for Llama3.1-8B, the default block size for each drafter model. WhiFlash routes between these two drafters under identical configurations, so each standalone baseline is matched to its counterpart inside WhiFlash. Table 7 summarises the hyperparameter settings.

Datasets. We report statistics for our evaluation datasets in Table 8 and group datasets based on their task category.

Software. In all experiments, we use the model and tokeniser implementations for *Qwen3* and *Llama3.1* provided by the Hugging Face Transformers framework (Wolf et al., 2020), `v4.57.1`.

B Additional Results

In this section, we present the full results for all the models employed in our study, in addition to the results presented in Section 5.

Tables 9 and 10 present per-dataset acceptance length (AL) and speedup (×) results for three target models across 15 datasets from four task categories.

Category	Dataset/Task	Examples
Chat	MT-Bench	80
	Alpaca	128
	UltraChat	80
	WildChat	80
Math	GSM8K	128
	MATH-500	128
	AIME24	30
	AIME25	30
Coding	HumanEval	164
	MBPP	128
	LiveCodeBench	128
Agentic	SWE-Bench	128
	Tau-Bench	162
	AgentBench	103
	AgentInstruct	80
Total		1577

Table 8: Number of examples in each dataset of our evaluation suite.

C Additional Analysis

In addition to the results presented in Section 6, we present the complete results for all the models employed in our study.

Impact of Lazy Catch-up. Table 11 shows the full throughput results regarding the impact of our proposed lazy catch-up spanning 15 datasets from four categories and three target models.

Switching Cost across All Models. We present profiling results of switching cost (per cent w.r.t. per-round latency) with different prefill lengths and backlog token lengths for all the target models evaluated in our study (i.e., Qwen3-4B, Qwen3-8B, and Llama3.1-8B). Table 12 demonstrates that the cost stays below 7% of per-round latency across all the models, and around 3–5% at the representative 80–90 token backlog.

Memory Analysis across All Models. In addition to Table 5, we further include peak memory measurements for Qwen3-4B and Llama3.1-8B as target models in Table 13. This confirms our observation that the memory overhead of WhiFlash is only marginally higher than that of DFlash, for different target models.

Model	Category	Dataset	Method						
			EAGLE-3	**DFlash**	**Oracle -Task**	**Oracle -Prompt**	**Oracle -Token**	**WhiFlash -Entropy**	**WhiFlash -Neural**
Qwen3-4B	Chat	MT-Bench	5.14	4.30	5.14	5.44	6.08	5.46	5.63
		Alpaca	5.06	2.90	5.06	5.01	5.31	4.90	5.12
		UltraChat	5.79	2.93	5.79	5.79	5.95	5.57	5.81
		WildChat	4.83	3.58	4.83	4.91	5.39	4.82	5.07
	Math	GSM8K	5.90	6.33	6.33	6.34	7.80	6.57	7.07
		MATH-500	5.18	7.88	7.88	7.89	8.78	7.90	8.02
		AIME24	5.44	7.58	7.58	7.58	8.80	7.77	7.91
		AIME25	5.25	7.09	7.09	7.09	8.06	7.21	7.63
	Coding	HumanEval	5.56	6.39	6.39	6.42	7.93	6.85	7.13
		MBPP	5.73	6.08	6.08	6.13	7.73	6.43	6.76
		LiveCodeBench	4.45	6.87	6.87	6.93	7.70	6.91	7.01
	Agentic	SWE-Bench	4.04	3.55	4.04	3.88	4.95	4.32	4.51
		Tau-Bench	3.13	2.79	3.13	3.04	3.68	3.23	3.29
		AgentBench	4.57	3.82	4.57	4.59	5.34	4.72	4.94
		AgentInstruct	4.21	4.07	4.21	4.54	5.28	4.59	4.87
Qwen3-8B	Chat	MT-Bench	5.17	4.26	5.17	5.46	6.13	5.41	5.68
		Alpaca	5.26	3.04	5.26	5.29	5.50	5.10	5.34
		UltraChat	5.82	2.84	5.82	5.82	5.93	5.62	5.84
		WildChat	4.70	3.47	4.70	4.79	5.30	4.83	5.02
	Math	GSM8K	6.21	6.48	6.48	6.56	8.21	6.91	7.35
		MATH-500	5.53	7.91	7.91	7.91	9.03	7.98	8.11
		AIME24	5.29	7.82	7.82	7.82	8.79	7.81	7.93
		AIME25	5.28	7.29	7.29	7.29	8.20	7.46	7.49
	Coding	HumanEval	5.65	6.52	6.52	6.52	8.22	7.08	7.42
		MBPP	5.78	5.97	5.97	6.10	7.76	6.47	6.83
		LiveCodeBench	4.44	7.20	7.20	7.27	7.97	7.35	7.34
	Agentic	SWE-Bench	4.07	3.57	4.07	3.81	5.02	4.44	4.60
		Tau-Bench	4.44	4.59	4.59	4.84	5.24	4.88	4.82
		AgentBench	4.48	3.74	4.48	4.44	5.06	4.63	4.80
		AgentInstruct	4.67	4.58	4.67	5.03	5.72	5.17	5.38
Llama3.1-8B	Chat	MT-Bench	5.19	3.97	5.19	4.54	5.59	5.20	5.26
		Alpaca	4.96	3.37	4.96	4.37	5.18	4.93	4.96
		UltraChat	5.38	3.96	5.38	4.95	5.73	5.33	5.39
		WildChat	4.42	3.16	4.42	3.91	4.71	4.41	4.47
	Math	GSM8K	5.61	4.31	5.61	4.80	6.05	5.57	5.63
		MATH-500	4.97	4.27	4.97	4.64	5.56	4.90	5.08
		AIME24	5.20	4.70	5.20	4.97	5.85	5.31	5.37
		AIME25	5.23	4.66	5.23	4.98	6.20	5.58	5.83
	Coding	HumanEval	6.05	5.01	6.05	5.19	6.76	6.01	6.28
		MBPP	6.17	5.00	6.17	5.44	6.75	6.11	6.35
		LiveCodeBench	4.63	3.95	4.63	4.14	5.18	4.61	4.73
	Agentic	SWE-Bench	4.38	3.81	4.38	3.90	5.16	4.47	4.75
		Tau-Bench	4.59	2.80	4.59	4.44	4.79	4.59	4.64
		AgentBench	4.12	3.40	4.12	3.93	4.54	4.12	4.27
		AgentInstruct	4.02	3.12	4.02	3.44	4.41	4.02	4.12

Table 9: Per-dataset acceptance length (AL) results for three target models across 15 datasets from four task categories.

Model	Category	Dataset	Method					
			EAGLE-3	**DFlash**	**Oracle -Task**	**Oracle -Prompt**	**WhiFlash -Entropy**	**WhiFlash -Neural**
Qwen3-4B	Chat	MT-Bench	2.70	2.44	2.70	2.98	2.96	2.91
		Alpaca	2.45	1.86	2.45	2.51	2.39	2.54
		UltraChat	2.95	1.89	2.95	2.93	2.96	3.11
		WildChat	2.24	2.15	2.24	2.63	2.46	2.58
	Math	GSM8K	3.25	4.30	4.30	4.62	4.38	4.29
		MATH-500	2.73	5.15	5.15	5.10	5.38	5.40
		AIME24	2.93	4.96	4.96	5.26	5.02	5.13
		AIME25	2.84	4.57	4.57	5.02	4.83	5.09
	Coding	HumanEval	3.11	4.23	4.23	4.65	4.57	4.68
		MBPP	2.96	4.01	4.01	4.19	4.14	4.27
		LiveCodeBench	2.31	4.62	4.62	4.51	4.66	4.75
	Agentic	SWE-Bench	2.25	2.60	2.60	2.61	2.78	2.73
		Tau-Bench	1.44	1.72	1.72	1.76	1.77	1.76
		AgentBench	2.27	2.62	2.62	2.66	2.57	2.60
		AgentInstruct	2.12	2.43	2.43	2.62	2.50	2.59
Qwen3-8B	Chat	MT-Bench	2.73	2.39	2.73	2.95	2.95	3.00
		Alpaca	2.65	1.96	2.65	2.98	2.68	2.84
		UltraChat	3.30	2.02	3.30	3.32	3.23	3.32
		WildChat	2.22	2.12	2.22	2.54	2.51	2.54
	Math	GSM8K	3.49	4.54	4.54	4.59	4.50	4.69
		MATH-500	3.03	5.38	5.38	5.47	5.46	5.39
		AIME24	3.07	5.99	5.99	5.45	5.75	5.74
		AIME25	2.92	5.00	5.00	5.28	5.39	5.36
	Coding	HumanEval	3.19	4.64	4.64	4.71	4.70	4.85
		MBPP	3.08	4.11	4.11	4.19	4.07	4.34
		LiveCodeBench	2.43	4.57	4.57	4.99	5.15	4.82
	Agentic	SWE-Bench	2.32	2.60	2.60	2.50	2.65	2.71
		Tau-Bench	2.35	3.03	3.03	3.02	2.96	3.09
		AgentBench	2.37	2.35	2.37	2.66	2.66	2.67
		AgentInstruct	2.47	2.50	2.50	2.68	2.64	2.64
Llama3.1-8B	Chat	MT-Bench	2.24	2.44	2.44	2.51	2.50	2.49
		Alpaca	2.20	2.21	2.21	2.34	2.31	2.25
		UltraChat	2.50	2.50	2.50	2.64	2.48	2.54
		WildChat	1.97	2.05	2.05	2.18	2.08	2.18
	Math	GSM8K	2.69	2.92	2.92	2.97	3.00	3.04
		MATH-500	2.45	2.92	2.92	2.99	2.79	2.73
		AIME24	2.25	2.82	2.82	2.89	2.84	2.70
		AIME25	2.29	2.86	2.86	2.93	2.93	3.06
	Coding	HumanEval	2.98	3.31	3.31	3.36	3.45	3.43
		MBPP	2.90	3.16	3.16	3.22	3.33	3.29
		LiveCodeBench	2.24	2.64	2.64	2.67	2.49	2.49
	Agentic	SWE-Bench	1.98	2.49	2.49	2.51	2.41	2.42
		Tau-Bench	2.07	1.82	2.07	2.15	2.05	2.11
		AgentBench	1.85	2.00	2.00	2.11	2.06	2.02
		AgentInstruct	1.90	2.10	2.10	2.14	2.16	2.10

Table 10: Per-dataset speedup ($\times$) over vanilla AR decoding for three target models across 15 datasets from four task categories. Notably, the Oracle-Token speedup is omitted as it requires `float32` inference, preventing comparison with `bfloat16` inference.

Model	Category	Dataset	Throughput		% Increase
			w/ lazy catch-up	w/o lazy catch-up	
Qwen3-4B	Chat	MT-Bench	157.02	138.98	13.0%
		Alpaca	140.55	119.44	17.7%
		UltraChat	165.62	145.04	14.2%
		WildChat	134.57	121.98	10.3%
	Math	GSM8K	240.73	204.81	17.5%
		MATH-500	279.2	249.12	12.1%
		AIME24	265.37	247.91	7.0%
		AIME25	275.33	236.6	16.4%
	Coding	HumanEval	231.09	198.06	16.7%
		MBPP	225.12	195.89	14.9%
		LiveCodeBench	238.43	208.84	14.2%
	Agentic	SWE-Bench	138.78	124.59	11.4%
		Tau-Bench	95.35	90.47	5.4%
		AgentBench	139.68	120.09	16.3%
		AgentInstruct	136.9	123.27	11.1%
Qwen3-8B	Chat	MT-Bench	158.76	142.88	11.1%
		Alpaca	144.3	133.7	7.9%
		UltraChat	154.39	142.39	8.4%
		WildChat	134.5	118.85	13.2%
	Math	GSM8K	233.56	209.39	11.5%
		MATH-500	271.82	247.17	10.0%
		AIME24	276.59	244.41	13.2%
		AIME25	264.69	231.87	14.2%
	Coding	HumanEval	237.84	211.1	12.7%
		MBPP	218.31	186.82	16.9%
		LiveCodeBench	240.49	213.34	12.7%
	Agentic	SWE-Bench	131.82	120.66	9.2%
		Tau-Bench	161	139.79	15.2%
		AgentBench	129.13	114.39	12.9%
		AgentInstruct	147.56	134.4	9.8%
Llama3.1-8B	Chat	MT-Bench	177.39	163.39	8.6%
		Alpaca	165.04	153.83	7.3%
		UltraChat	179.06	166.05	7.8%
		WildChat	156.39	130.21	20.1%
	Math	GSM8K	199.98	176.63	13.2%
		MATH-500	191.18	166.56	14.8%
		AIME24	206.95	174.37	18.7%
		AIME25	211.76	185.68	14.0%
	Coding	HumanEval	222.58	201.05	10.7%
		MBPP	234.23	200.36	16.9%
		LiveCodeBench	177.28	155.93	13.7%
	Agentic	SWE-Bench	181.14	159.69	13.4%
		Tau-Bench	152.76	135.2	13.0%
		AgentBench	145.16	129.12	12.4%
		AgentInstruct	148.93	125.02	19.1%

Table 11: Impact of lazy catch-up across all models and datasets. Deferring inactive-drafter KV updates to a single batched prefill at switch time improves throughput while leaving accepted tokens unchanged.

Model	Prefill Tokens	Backlog Token Length				
		1	64	128	512	1024
Qwen3-4B	512	3.1%	3.6%	3.5%	3.5%	4.2%
	1024	3.4%	3.5%	3.5%	3.5%	4.4%
	2048	3.0%	3.5%	3.5%	3.8%	5.1%
	4096	2.8%	3.7%	3.7%	4.1%	5.7%
	8192	2.0%	3.4%	3.4%	3.9%	5.6%
	16384	1.2%	3.0%	3.0%	3.5%	5.4%
Qwen3-8B	512	3.3%	3.8%	3.8%	3.8%	4.9%
	1024	3.3%	3.8%	3.8%	3.9%	5.2%
	2048	3.2%	3.8%	3.8%	4.1%	5.6%
	4096	2.8%	3.7%	3.7%	4.2%	5.8%
	8192	1.9%	3.4%	3.4%	3.8%	5.7%
	16384	1.2%	3.0%	3.0%	3.5%	5.4%
Llama3.1-8B	512	4.3%	5.0%	4.9%	4.9%	6.5%
	1024	4.2%	4.9%	4.8%	5.1%	6.8%
	2048	3.8%	4.6%	4.6%	5.0%	6.8%
	4096	3.2%	4.3%	4.3%	4.8%	6.7%
	8192	2.2%	3.9%	3.8%	4.4%	6.4%
	16384	1.4%	3.4%	3.4%	4.0%	6.1%

Table 12: Profiling results for switching cost (% w.r.t. per-round latency) with different base KV lengths and backlog token lengths for all three target models (*i.e.* Qwen3-4B, Qwen3-8B, and Llama3.1-8B). The cost stays below 7% of per-round latency across all the models, and around 3–5% at the representative 80–90 token backlog.

Model	Method	Memory (GB)	% Increase
Qwen3-4B	No SD	7.55	-
	EAGLE-3	8.34	10.5%
	DFlash	8.96	18.8%
	WhiFlash-Entropy/Neural	9.38	24.3%
Qwen3-8B	No SD	15.26	–
	EAGLE-3	16.39	7.4%
	DFlash	17.62	15.4%
	WhiFlash-Entropy/Neural	18.41	20.6%
Llama3.1-8B	No SD	14.96	-
	EAGLE-3	16.10	7.6%
	DFlash	17.27	15.5%
	WhiFlash-Entropy/Neural	18.06	20.7%

Table 13: Peak memory footprint of EAGLE-3, DFlash, and WhiFlash with Qwen3-4B, Qwen3-8B, and Llama3.1-8B as target models. WhiFlash's memory overheads over a diffusion-only system (DFlash) are 0.42 GB, 0.79 GB, and 0.79 GB, merely 5.5%, 5.2%, and 5.2% of the footprint of the target models, Qwen3-4B, Qwen3-8B and Llama3.1-8B, respectively.