ADAPTSPEC: INPUT- AND STEP-ADAPTIVE TREE-BASED SPECULATIVE DECODING TO ACCELERATE LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates Large Language Model (LLM) inference by using a draft model to propose a *speculative length* (SL) of tokens at each decoding step, which a target model then verifies in a single forward pass. Tree-based speculative decoding further improves acceptance rates and speedup by drafting a *candidate length* (CL) of tokens in parallel at each position. However, these methods rely on fixed SL and CL. While recent work adapts SL primarily using step-level signals, prior methods do not jointly optimize SL and CL while conditioning on the request input, despite its significant impact on optimal parameter selection. Our experimental analysis identifies several influential factors, including input categories, that determine the speedup-maximizing configuration. Accordingly, we propose AdaptSpec, which initially adapts SL using machine learning and CL using binary search based on these factors, and subsequently switches to Reinforcement Learning (RL) to jointly optimize both parameters once the RL policy is trained. Experiments across four draft/target model pairs and five datasets show that AdaptSpec achieves up to 98% higher speedup than state-of-the-art methods, translating into fewer GPU-hours per request in LLM serving.

1 INTRODUCTION

Large Language Models (LLMs) have driven transformative advances in natural language processing applications such as machine translation, question answering, and text summarization (Taori et al., 2023b). Much of this deployment runs on shared GPU clusters, where per-request decoding latency directly determines GPU time, serving cost, and the request throughput a fixed cluster can sustain. The bottleneck is particularly pronounced in interactive applications where users expect fast responses, so accelerating decoding without sacrificing response quality is critical.

Speculative decoding (Leviathan et al., 2023) uses a lightweight draft model (e.g., Llama-7B) to generate a *speculative length* (SL) of tokens, which a larger target model (e.g., Llama-33B) verifies in a single forward pass (§2). *Speedup* is the ratio of target-only latency to speculative decoding latency. While a longer SL reduces target model calls, it also raises the rejection rate, causing recomputation that can negate the speedup (Kim et al., 2024). Tree-based speculative decoding (Cai et al., 2024; Spector & Ré, 2023; Hu et al., 2025; Kim et al., 2026; Miao et al., 2024; Yang et al., 2025; Lu et al., 2024; Zhang et al., 2025; Wu et al., 2025) uses top-k sampling to generate k candidate tokens—the *candidate length* (CL)—at each position; verifying all branches in one pass increases the probability that at least one token is accepted.

Previous tree-based methods (Miao et al., 2024) use static SL and CL. While recent work (Huang et al., 2025) adapts SL primarily using step-level signals, prior methods do not jointly optimize both SL and CL while conditioning on request input, despite its significant impact on optimal parameter selection. Our experimental analysis (§3) shows:

1) Speedup and acceptance ratio are influenced by input category (e.g., coding, reasoning, writing).
2) Besides the step index, acceptance ratios and the KL divergence between draft and target logits from previous steps are informative signals for guiding SL decisions at the next step.
3) Within each input category, there exists an optimal CL for a given SL that maximizes speedup.

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.

Leveraging these insights, we propose an input- and step-adaptive tree-based speculative decoding system (AdaptSpec), which dynamically determines SL and CL for each input and each step to maximize the speedup of LLM inference. AdaptSpec consists of three components:

- **SL Determination.** A lightweight prediction head on the draft model predicts the SL for each step from the input prompt, request type, step index, and the acceptance ratios and draft–target logit divergence of a few (e.g., 3) previous steps.
- **CL Determination.** Given the SL, a binary search guided by a learned speedup estimator selects the CL, exploiting the unimodal relationship between CL and speedup.
- **Reinforcement Learning (RL) based Method.** Data from the above components trains an RL policy that jointly decides SL and CL; AdaptSpec switches to it once trained to reduce decision-making overhead and capture SL–CL interactions.

Our contributions are: (1) an experimental analysis identifying the input- and step-level factors that determine the speedup-optimal SL and CL (§3); (2) AdaptSpec, a system that dynamically adapts both parameters to input characteristics and decoding steps (§4); and (3) comprehensive experiments showing that AdaptSpec improves speedup by up to 98% over state-of-the-art methods with only 0.4% decision overhead (§5).

2 Background

Speculative decoding. Let M_t be the target model and M_d be the draft model. At each step, (1) M_d runs L steps to generate L draft tokens with their next-token probabilities, and (2) M_t runs a single forward pass over the L-token sequence to compute its own next-token probabilities and accepts each draft token whose probability matches within a threshold (e.g., 0.05 (Leviathan et al., 2023)). If a token is rejected, all subsequent draft tokens are discarded and M_d resumes from the last accepted token; M_t additionally appends one token of its own in the same forward pass, guaranteeing at least one new token per step.

Tree-based speculative decoding. Tree-based methods such as SpecInfer (Miao et al., 2024) have M_d use top-k sampling to generate k candidate tokens (the CL) at each position, forming a tree of width CL and depth SL. M_t verifies all root-to-leaf paths in a single forward pass, selecting at each position the candidate whose probability best matches its own, and assembles the accepted tokens into a merged sequence possibly spanning multiple branches. SpecInfer additionally applies confidence-guided verification, prioritizing high-confidence draft tokens and pruning low-probability branches to reduce verification latency. Because more candidates are proposed per position, tree-based decoding improves the acceptance ratio over single-path speculative decoding.

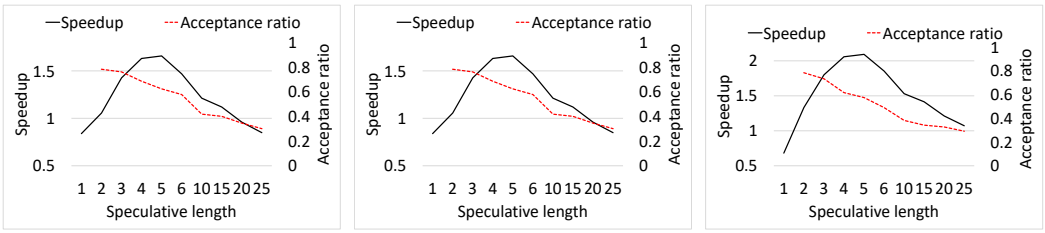
3 Motivation & Experimental Analysis

3.1 Experimental Settings

Machine settings. We ran experiments on a single machine with eight 80GB A100 GPUs interconnected via NVLink (112 GB/s maximum bidirectional bandwidth), representative of multi-GPU nodes in LLM-serving clusters.

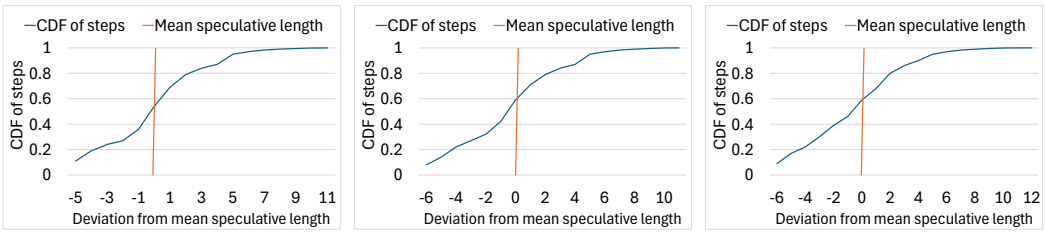
Dataset and request settings. We used the Alpaca (Taori et al., 2023a), ShareGPT (ShareGPT, 2023), BookCorpus (Kobayashi, 2018) and MTBench (Zheng et al., 2023a) traces. Alpaca and ShareGPT have 52K and 90K requests, with up to 500 and 2K prompt lengths, respectively. BookCorpus has 11K unpublished books with prompt lengths up to 100K tokens. MTBench comprises six request types (coding, humanities, reasoning, mathematics, extraction, and writing) with 80 requests each. We processed 2000 (Liu et al., 2024) requests from each of the first three datasets and all requests from MTBench.

Model settings. We used Llama-7B/Llama-33B, Llama-13B/Llama-33B, Flan-T5-small/Flan-T5-medium and Vicuna-68M/Vicuna-7B as draft/target pairs. The draft model runs on one GPU; the target model uses the remaining seven via Megatron-LM (Nvidia). We use Leviathan et al. (2023) as the vanilla method with a fixed SL of 5, with source code from Feifeibear (2023); SpecInfer (Miao



(a) Alpaca. **(b)** ShareGPT. **(c)** BookCorpus.

Figure 1: Speedup and acceptance ratio vs. SL for Llama-7B/Llama-33B (other pairs: Appendix F).



(a) Alpaca. **(b)** ShareGPT. **(c)** BookCorpus.

Figure 2: Deviation of step-wise oracle SL from the mean oracle SL for Llama-7B/Llama-33B (other pairs: Appendix F).

et al., 2024) code is from Golario (2024). Figures in this section show one representative model pair where space is limited; results for all pairs are in Appendix F and exhibit the same trends.

3.2 Speedup vs. Speculative Length (SL)

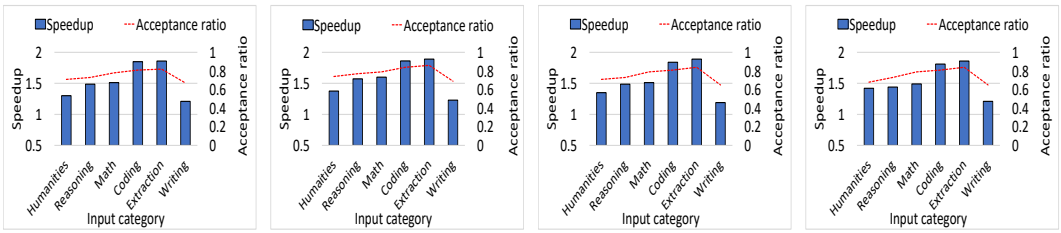
Figure 1 shows the impact of SL on speedup and acceptance ratio for the vanilla method. For all datasets and model pairs, as SL grows, the speedup first increases, peaks, and then gradually declines, while the acceptance ratio keeps decreasing. A smaller SL increases calls to the target model (logit computation and token verification), reducing speedup but yielding a higher acceptance ratio. A larger SL enhances potential speedup but lowers the acceptance ratio (Systems, 2023; Research, 2023; Team, 2024), leading to more discarded tokens and recomputation. This trade-off produces an optimal SL, ranging between 4 and 6, for each combination of model pair and dataset.

Different steps should also use different SLs because the draft model's prediction accuracy changes as the sequence progresses. Early steps benefit from shorter SLs due to limited context and because errors in early draft tokens propagate to subsequent ones (Li et al., 2025), while later steps can exploit longer SLs as the draft model gains context from accepted tokens. Input types (e.g., coding vs. writing) also influence token predictability. To measure this, we design an *Oracle* that varies the SL from 1 to 25 at each step for a prompt and selects the value that maximizes speedup. Figure 2 shows the CDF of steps versus the deviation of the step-wise oracle SL from the mean oracle SL. Across all model pairs, datasets, and steps, the deviation spans up to ± 10 tokens, confirming that a static SL is suboptimal and per-step adaptive selection is necessary.

> **Observation 1.** *There exists an optimal SL, which varies significantly across both steps and requests. (Figures 1, 2, and Appendix F)*

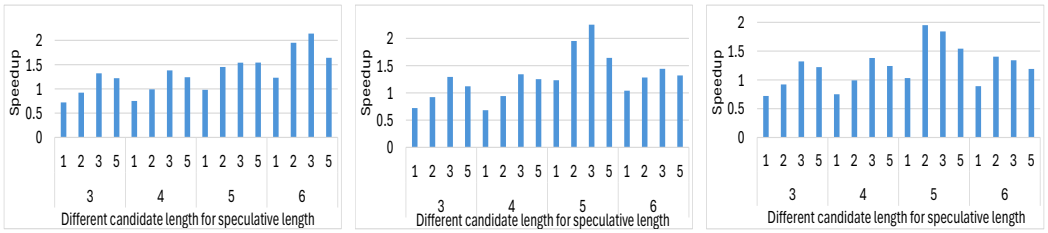
3.3 Impact of Inputs

Figure 3 shows the speedup and acceptance ratio of different MTBench input categories using speculative decoding with a static SL of 5 and CL of 1 (Leviathan et al., 2023). Extraction and coding achieve the highest speedup, followed by math, reasoning, humanities, and writing, for all model pairs. Structured tasks (e.g., arithmetic "What is 15 + 27?" $\rightarrow$ "42") enable the draft to align closely with the target, while open-ended tasks reduce acceptance.



(a) Llama-7B/33B. **(b)** Llama-13B/33B. **(c)** T5-small/medium. **(d)** Vicuna-68M/7B.

Figure 3: Performance of models on different categories of the MTBench dataset.



(a) Alpaca. **(b)** ShareGPT. **(c)** BookCorpus.

Figure 4: Speedup for different CLs at each SL for Llama-7B/Llama-33B (other pairs: Appendix F).

Observation 2. *The speedup and average acceptance ratio are influenced by the input categories. (Figure 3)*

3.4 Candidate Length (CL) vs. Speculative Length (SL)

For each SL $\in \{3, 4, 5, 6\}$, we vary CL $\in \{1, 2, 3, 5\}$ for SpecInfer with confidence-guided verification (Figure 4). For each SL there is an optimal CL: a wider tree raises the acceptance ratio but increases verification latency per step, so beyond a certain CL the speedup declines. This rise-then-fall pattern is stable across all SLs, model pairs, and datasets.

Observation 3. *For each model pair and input category, there exists an optimal CL for a given SL that maximizes speedup. (Figure 4 and Appendix F)*

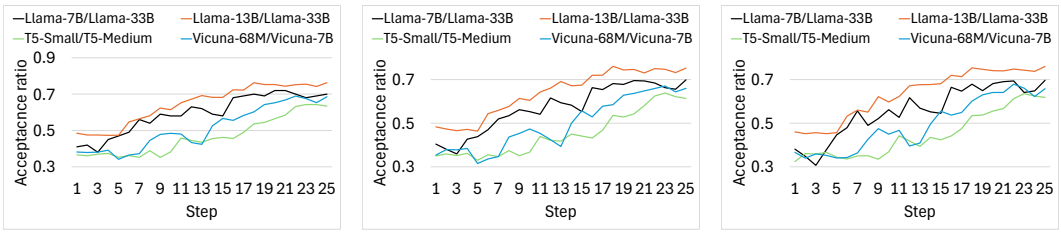
3.5 Acceptance Ratio and Logit Divergence as Signals

We measure the average acceptance ratio over 25 steps using a fixed SL of 5 and CL of 3, which gives the peak speedup in Figure 4. As shown in Figure 5, the acceptance ratio increases as the step progresses, so the step index is a critical temporal feature for predicting SL or CL. We further group, for each step i, the average acceptance ratio from steps $i-3$ to $i-1$ into bins and compute the average oracle SL at step i. As Figures 6a–6b show, a higher acceptance ratio in previous steps relates to a higher oracle SL in the current step.

Observation 4. *The acceptance ratio increases with the step index, and acceptance ratios from earlier steps can serve as signals for guiding SL decisions. (Figures 5 and 6)*

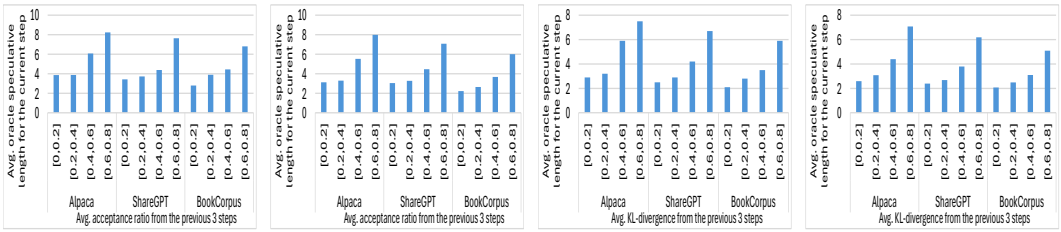
Similarly, we bin the average Kullback-Leibler (KL) divergence (Kullback & Leibler, 1951) between draft and target logits from steps $i-3$ to $i-1$. Lower divergence indicates closer draft–target agreement; as Figures 6c–6d show, it correlates with a higher oracle SL in the current step.

Observation 5. *The KL divergence between draft and target logits in previous steps can serve as a signal for guiding SL decisions. (Figure 6)*



(a) Alpaca. **(b)** ShareGPT. **(c)** BookCorpus.

Figure 5: Acceptance ratio vs. step for different datasets.



(a) Acc. ratio, Llama-7B/33B. **(b)** Acc. ratio, Vicuna-68M/7B. **(c)** KL div., Llama-7B/33B. **(d)** KL div., Vicuna-68M/7B.

Figure 6: Oracle SL vs. (a,b) acceptance ratio and (c,d) draft–target logit KL divergence of the previous three steps (all pairs: Appendix F).

4 Design of AdaptSpec

Objective. Let a request I consist of T steps. At each step $i \in \{1, \ldots, T\}$, the system observes the input features and selects a pair (H_i, W_i), where H_i and W_i denote the SL and CL at step i. Let $r(H, W)$ denote the speedup at a given step. The objective is to find $(H^*, W^*) = \arg\max_{H,W} \; r(H, W)$. Following Observations 1 and 2, we determine SL per step via ML prediction; we use binary search for CL given its stable unimodal behavior once SL is fixed (Observation 3); and we unify both with an RL agent that reduces overhead and captures their interactions.

4.1 SL Determination

AdaptSpec appends a lightweight linear prediction head to the draft model, which runs on the draft model's GPU while the target model uses the remaining GPUs. We use the draft model because it is lightweight and, as it typically belongs to the same family as the target model (Liu et al., 2024), its hidden states capture draft–target alignment—the key driver of optimal SL. The head reuses hidden states from the draft model's forward pass, requires no extra parameter storage, and adds less than 1% to decoding latency. Its input features are concatenated with the draft hidden state $\mathbf{h}_i$ of the current sequence into $\mathbf{x}_i^{\mathrm{SL}} = [\,\mathbf{h}_i;\ \mathbf{e}(T);\ i/i_{\max};\ \bar{A}_i;\ \bar{L}_i\,]$ (Appendix B):

- **Prompt (P):** Input text; its complexity affects the draft model's prediction accuracy and the SL (Observation 1).
- **Request Type (T):** Category of the prompt (coding, math, humanities, etc.), one-hot encoded ($\mathbf{e}(T)$, 6 categories), obtained before decoding via a lightweight zero-shot classifier (Schlör, 2023; Data Science Campus, 2024; Varangot-Reille et al., 2025) (Observation 2). On all 480 MTBench prompts, it reaches 93% accuracy at 38 ms, versus 81% for rule-based keyword detection and 89% for zero-shot DistilBERT (Appendix A.4); its cost is included in the 0.4% decision overhead (§5.2).
- **Step (i):** Normalized as $i/i_{\max}$; temporal context, since acceptance ratio increases with step index (Observation 4).
- **Acceptance Ratio (A_i):** Average over the previous n ($= 3$; zero-padded for $i \leq 3$) steps; higher values predict higher oracle SL (Observation 4).

- **Logit Divergence (L_i):** Average KL divergence over the previous m steps; lower divergence predicts higher oracle SL (Observation 5).

The head outputs $\hat{H}_i = \mathbf{w}^\top \mathbf{x}_i^{\mathrm{SL}} + b$, and the executed SL is $\mathrm{clip}(\mathrm{round}(\hat{H}_i), 1, 25)$. The draft model is fine-tuned with supervised learning using oracle SL labels, minimizing RMSE between predicted and oracle SL on a request-disjoint 80%/10%/10% split (details in Appendix A.1). Prediction is pipelined with the target model's verification (<0.01s runtime), so it does not block token generation: while the draft model generates tokens for step $i-1$, the predictor determines the SL for step i; with an average token generation time of 0.087s, the SL is ready within about two tokens.

4.2 CL Determination

Following Observation 3, speedup is unimodal in CL once SL is fixed—rising to a peak, then falling due to increasing verification overhead. We exploit this with a *binary search guided by a learned speedup estimator* $\hat{S}(W)$ over $[W_{\min}, W_{\max}]$ (empirically $W_{\min}$=1, $W_{\max}$=25). $\hat{S}$ is a random forest regressor, chosen for its ability to capture non-linear feature interactions and its robustness to overfitting, whose input is the queried CL W, the selected SL H_i, and the following features: the draft model's average probability of selected tokens over the previous m steps (higher confidence indicates better alignment); the acceptance ratio over the previous m (e.g., 3) steps, guiding how aggressively to expand width; the draft–target KL divergence; and the moving average of observed speedup from previous steps, capturing temporal trends in successful width settings. It is trained on $\approx 1.2\times10^6$ step records, each labeled with the resulting speedup, across all datasets and model pairs (training-split requests only, with W spanning [1, 25]) (hyperparameters in Appendix A.2).

At each iteration, we evaluate $\hat{S}(W_{\text{mid}}-1)$, $\hat{S}(W_{\text{mid}})$, and $\hat{S}(W_{\text{mid}}+1)$ with $W_{\text{mid}} = \lfloor (W_{\min} + W_{\max})/2 \rfloor$: if W_{mid} is a local maximum it is returned (within a step only W varies across these three queries, so they differ through the learned effect of W and its interaction with H_i); if speedup increases rightward, $W_{\min} \leftarrow W_{\text{mid}} + 1$; otherwise $W_{\max} \leftarrow W_{\text{mid}} - 1$. Because the speedup–CL relationship is unimodal, a local maximum exists in $[W_{\min}, W_{\max}]$ and this three-point evaluation suffices to identify the search direction; the search converges in $O(\log(W_{\max} - W_{\min}))$ steps (Algorithm 1, Appendix C).

4.3 Reinforcement Learning based Method

Separately deciding SL and CL incurs per-step overhead and ignores their interactions. We therefore use RL to jointly optimize both by training an Actor–Critic agent (Konda & Tsitsiklis, 2000) offline on data collected from the ML and binary-search components, and periodically refining it online. The system initially determines SL via ML and CL via binary search, and switches to the RL policy once trained. As Figure 7 shows, the actor outputs actions (SL and CL); the critic evaluates the expected state value to improve the policy.

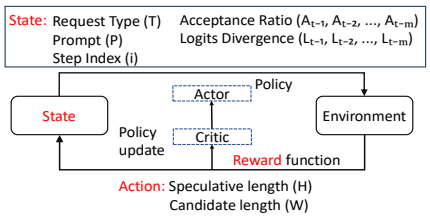


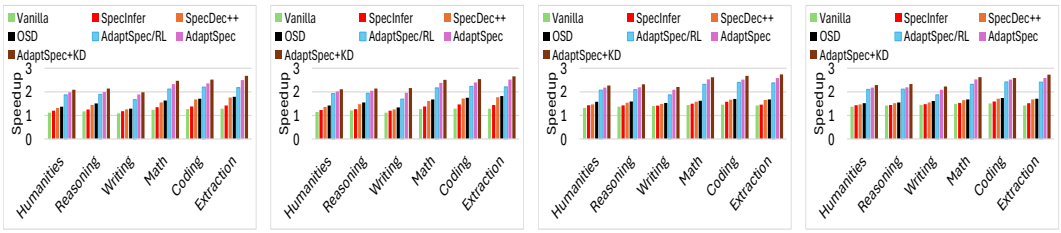
Figure 7: RL for SL and CL.

State Representation. The state $\mathbf{S_i} = \{i, H_i, W_i, P_i, V_{i-1}, D_{i-1}\}$, where i, H_i and W_i denote the step, SL and CL for step $i-1$ respectively; P_i is the current sequence; V is the acceptance ratio; and D is the logit divergence between draft and target models.

Action Space. $\mathbf{A_i} = \{H_i, W_i\}$. The actor outputs the mean $\boldsymbol{\mu}_\theta(\mathbf{S}_i) \in \mathbb{R}^2$ of a Gaussian policy $\pi_\theta = \mathcal{N}(\boldsymbol{\mu}_\theta, \sigma^2\mathbf{I})$; a sample $\mathbf{a}_i \sim \pi_\theta$ is rounded and clipped to the executed integer action. After execution of a step with $\mathbf{A_i}$, state $\mathbf{S_i}$ transits to $\mathbf{S_{i+1}}$.

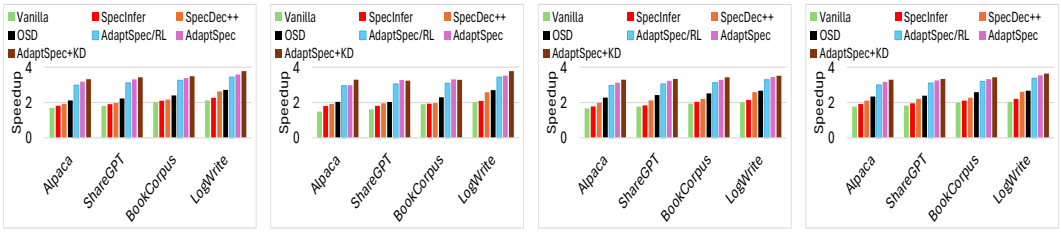
Reward Function. The total reward R is the cumulative speedup across all T steps: $R = \sum_{i=1}^{T} r(H_i, W_i)$, where $r(H_i, W_i)$ is the speedup of step i and an episode is one request. The RL agent maximizes the discounted return $\mathbb{E}[\sum_i \kappa^{i-1} r_i]$, the critic minimizes δ_i^2 with $\delta_i = r_i + \kappa V_\phi(\mathbf{S}_{i+1}) - V_\phi(\mathbf{S}_i)$, and the actor minimizes $-\delta_i \log \pi_\theta(\mathbf{a}_i|\mathbf{S}_i)$.

Implementation. The actor (4-layer MLP) outputs real-valued (H_i, W_i), rounded and clamped to valid ranges; the critic is a 3-layer MLP. The 133-dimensional state concatenates the step index,



(a) Llama-7B/33B. **(b)** Llama-13B/33B. **(c)** T5-small/medium. **(d)** Vicuna-68M/7B.

Figure 8: Overall speedup comparison for all models on the MTBench dataset.



(a) Llama-7B/33B. **(b)** Llama-13B/33B. **(c)** T5-small/medium. **(d)** Vicuna-68M/7B.

Figure 9: Overall speedup comparison for all models on the other datasets.

SL/CL, a 128-dimensional prompt embedding projected from the draft hidden state, the acceptance ratio, and the logit divergence. The policy is pre-trained offline on $\approx 2\times10^5$ transitions per model pair from the SL predictor and binary-search CL components, then retrained online every 500 new requests from the most recent transition buffer, matching continuously arriving request streams in serving. We used ε-greedy exploration with probability ε the executed action is drawn uniformly from the valid integer grid instead of from π_θ (Appendix B.3). Architecture and optimization details are in Appendix A.3.

5 Performance Evaluation

Settings and compared methods. We use the settings of §3, plus LongWriter (Bai et al., 2024) (32K-word outputs) to stress long-sequence generation. We compare AdaptSpec with Vanilla (Leviathan et al., 2023), SpecInfer (Miao et al., 2024), Online Speculative Decoding (OSD) (Liu et al., 2024), which uses knowledge distillation (KD) to align draft and target models dynamically (code from Zheng et al. (2023b)), and, on MTBench, SpecDec++ (Huang et al., 2025), which uses a trained acceptance-prediction head to adaptively select SL. AdaptSpec/RL denotes AdaptSpec with the ML and binary-search components only (no RL); AdaptSpec +KD adds periodic KD to AdaptSpec. Two ablations evaluate individual components: AdaptSpec/SL uses a fixed SL of 5 as in Vanilla instead of SL determination (§4.1), and AdaptSpec/CL uses a fixed CL of 3 instead of CL determination (§4.2). All learned components and oracle labels use only training-split requests, and MTBench and LongWriter are never used for training. During evaluation, the policy is retrained every 500 requests using only already-served requests, so no request is decoded by a policy trained on it.

5.1 Overall Performance Comparison and Ablation Study

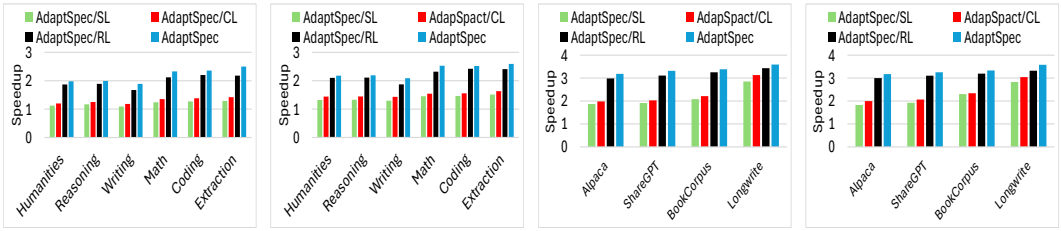
Figures 8 and 9 show average speedup across all request categories and datasets, and Table 1 summarizes the gains. Even without KD, AdaptSpec outperforms Vanilla by 66–98%, SpecInfer by 59–78%, and OSD by 46–51% across all task categories and model pairs; AdaptSpec +KD adds a further 8–17%. AdaptSpec outperforms SpecDec++ by 54–67%, while SpecDec++ achieves speedups between SpecInfer and OSD. Unlike the static configurations of SpecInfer and Vanilla, AdaptSpec dynamically adjusts decoding based on input characteristics and real-time feedback. OSD relies on static response-length prediction without step-level dynamics, and SpecDec++ only adapts SL

Table 1: Speedup improvement of AdaptSpec over baselines.

Model pair	Vanilla	SpecInfer	SpecDec++	OSD
Llama-7B/33B	70–95%	61–74%	56–65%	48–51%
Llama-13B/33B	67–94%	59–73%	54–63%	46–49%
T5-small/med	66–96%	59–75%	54–64%	47–50%
Vicuna-68M/7B	69–98%	61–78%	57–67%	49–51%

Table 2: Mean speedup $\pm$ std. over 3 runs (Alpaca; seeds 0, 1, 2).

Method	**Llama-7B/33B**	**Llama-13B/33B**	**T5-small/med**	**Vicuna-68M/7B**
Vanilla	1.21 ± 0.02	1.19 ± 0.02	1.18 ± 0.02	1.20 ± 0.02
SpecInfer	1.43 ± 0.03	1.41 ± 0.03	1.40 ± 0.02	1.42 ± 0.03
OSD	1.49 ± 0.03	1.47 ± 0.03	1.46 ± 0.02	1.48 ± 0.03
AdaptSpec/RL	1.88 ± 0.04	1.85 ± 0.04	1.83 ± 0.03	1.86 ± 0.04
AdaptSpec	2.11 ± 0.04	2.07 ± 0.04	2.06 ± 0.03	2.09 ± 0.04
AdaptSpec +KD	2.42 ± 0.05	2.38 ± 0.05	2.36 ± 0.04	2.40 ± 0.05



(a) Llama-7B/33B, MT-Bench. **(b)** Vicuna-68M/7B, MT-Bench. **(c)** Llama-7B/33B, others. **(d)** Vicuna-68M/7B, others.

Figure 10: Ablation studies on MTBench (a,b) and on the Alpaca, ShareGPT, BookCorpus, and LongWriter datasets (c,d). All model pairs: Appendix F.

while ignoring input conditions. AdaptSpec achieves 12–19% higher speedup than AdaptSpec/RL, confirming the value of joint RL-based SL/CL selection. Higher speedups for structured tasks (extraction, coding) are consistent with §3. Because serving cost scales with GPU time, these gains translate into fewer GPU-hours for a fixed request volume, or higher throughput from a fixed GPU allocation.

Figure 10 shows the contribution of individual components. Compared to AdaptSpec, AdaptSpec/SL and AdaptSpec/CL achieve 33–44% and 29–42% lower speedup across all datasets and model pairs, respectively. AdaptSpec/SL uses a fixed SL, causing either excessive recomputation or draft underutilization; AdaptSpec/CL uses a fixed tree structure, causing suboptimal branching. Both confirm that each component contributes independently to the overall speedup.

5.2 Time Overhead and Accuracy

Figure 11 shows the decision overhead as a fraction of end-to-end (E2E) latency, averaged across datasets; OSD and AdaptSpec +KD are excluded since KD adds no online overhead. Vanilla has zero overhead; SpecInfer adds 1.2% of E2E via static tree parameters. AdaptSpec incurs only **0.4%** of E2E latency (2.8% for AdaptSpec/RL), i.e., RL reduces decision overhead by $3.22\times$. AdaptSpec/SL and AdaptSpec/CL cut overhead by $1.13\times$ and 88%, respectively, but at the cost of reduced speedup. All methods achieve 100% accuracy (Appendix F): every speculative token is verified by the same target model under identical decoding settings, so outputs are identical to standard decoding.

5.3 Sensitivity Testing and Statistical Significance

We vary the number of previous steps used to average logit divergence and acceptance ratio—two key features for SL and CL prediction—from 1 to 6. In both cases (Figures 12 and 13), performance peaks at 3–4 previous steps, after which the benefit plateaus due to the diminishing relevance of older steps. Compared with the oracle, AdaptSpec's SL and CL deviate by 2.9 and 2.4 on average, matching the oracle at 35% and 39% of steps (Appendix F); AdaptSpec still surpasses prior methods, underscoring the value of dynamic SL and CL prediction.

All results are means over 3 independent runs (seeds 0, 1, 2). Table 2 reports mean speedup $\pm$ standard deviation on Alpaca. Standard deviations are below 3% of the mean in every cell, so the gains are stable and not artifacts of random initialization. The main source of run-to-run variance is RL

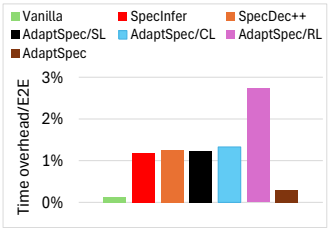


Figure 11: Time overhead.

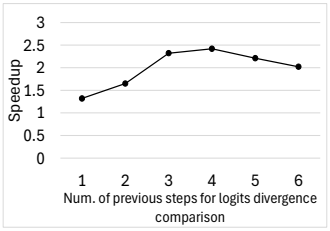


Figure 12: Sensitivity to the logit-divergence window.

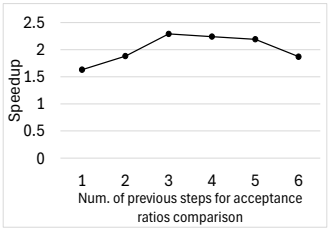


Figure 13: Sensitivity to the acceptance-ratio window.

policy initialization and ε-greedy exploration; the SL predictor and random forest are deterministic given fixed seeds. Compute costs are reported in Appendix D.

6 Related Work

Speculative decoding accelerates autoregressive generation by using a lightweight draft model to propose candidate tokens verified by a larger target model, while preserving the target distribution (Chen et al., 2023; Timor et al., 2025). Recent improvements include on-the-fly adaptation of SL (Liu et al., 2025), confidence-based early stopping (Agrawal et al., 2024), in-context reference overlaps (Yang et al., 2023), fusing drafting into the target model (Fu et al., 2025), and relaxing acceptance criteria based on token interchangeability in latent space (Jang et al., 2025). Token-tree methods explore candidate paths in parallel at the cost of higher verification overhead (Cai et al., 2024; Spector & Ré, 2023). Multi-draft approaches rely on multiple draft models (Hu et al., 2025; Kim et al., 2026), whereas multi-candidate methods generate several proposals in a single forward pass (Miao et al., 2024; Yang et al., 2025; Lu et al., 2024; Zhang et al., 2025; Wu et al., 2025). EAGLE-2 introduces a context-aware dynamic draft tree (Y. Li & Zhang, 2024), AdaEagle predicts optimal draft length using a lightweight controller (Zhang et al., 2024), and Opt-tree optimizes expected accepted length per step (Wang et al., 2025). SpecDec++ formulates SL selection as an MDP and trains an acceptance-prediction head to terminate drafting when rejection probability exceeds a learned threshold (Huang et al., 2025). From a systems perspective, SSD parallelizes draft and verification stages (Kumar et al., 2026), MagicDec alleviates KV-cache pressure under large batch sizes (Sadhukhan et al., 2025), and LongSpec and ML-SpecQD improve efficiency through KV-cache reuse, context-aware drafting, and quantized draft models (Yang et al., 2026; Georganas et al., 2025). In contrast, AdaptSpec jointly optimizes both SL and CL at each decoding step considering request input type and other factors, dimensions that prior work does not explicitly consider, while remaining deployable within the standard multi-GPU draft/target layout (Nvidia).

7 Conclusion

We presented AdaptSpec, an adaptive framework that considers both input characteristics and the decoding step index to optimize SL and CL for accelerating LLM inference. We find that request input, decoding step index, and other factors significantly influence the speedup-optimal SL and CL. AdaptSpec uses these factors in ML and binary search to adjust SL and CL dynamically, then transitions to an RL-based policy once trained, achieving up to 98% higher speedup than state-of-the-art methods across diverse input types, datasets, and models. We will release the source code upon acceptance.

Limitations and Future Work. AdaptSpec's predictors rely on input and request-agnostic signals (acceptance ratio, draft–target KL divergence, step index), which may not fully capture higher-level semantic structure or task complexity; richer signals such as context entropy or other model-internal statistics could improve prediction accuracy. AdaptSpec also assumes the draft and target models belong to the same family. This matters most for the SL predictor, whose linear head is trained on draft-model hidden states that encode family-specific alignment; the CL estimator and RL policy use family-agnostic features, and transferring only these two components across families degrades speedup by less than 5%. Broader cross-family deployment may require retraining or recalibrating the SL predictor.

AI USE STATEMENT

Generative AI tools were used to improve the clarity, grammar, and presentation of portions of the manuscript. The authors reviewed and revised all AI-assisted text and take full responsibility for the final content.

ETHICS STATEMENT

AdaptSpec reduces LLM inference latency and GPU-hour consumption, lowering the cost and barrier to deploying language models in real-time applications. A potential negative consequence is that faster generation may facilitate the large-scale production of harmful content, underscoring the importance of responsible deployment practices and effective safeguards. All datasets and models are used under their licenses (Appendix E).

REPRODUCIBILITY STATEMENT

Experimental settings are given in §3 and §5. Full training and implementation details for the SL predictor, CL estimator, and RL policy are in Appendix A, the model inputs and RL formulation in Appendix B, the CL search procedure in Appendix C, compute costs in Appendix D, and asset licenses in Appendix E. All results are averaged over three seeds. We will release the source code upon acceptance.

REFERENCES

Sudhanshu Agrawal, Wonseok Jeon, and Mingu Lee. Adaedl: Early draft stopping for speculative decoding of large language models via an entropy-based lower bound on token acceptance probability. In *ENLSP*, 2024. URL https://api.semanticscholar.org/CorpusID:273549756.

Yushi Bai, Jiajie Zhang, Xin Lv, Linzhi Zheng, Siqi Zhu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongWriter: Unleashing 10,000+ Word Generation from Long Context LLMs. In *Proc. of ICLR*, 2024.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, L. Sifre, and John M. Jumper. Accelerating large language model decoding with speculative sampling. *ArXiv*, abs/2302.01318, 2023. URL https://api.semanticscholar.org/CorpusID:256503945.

Data Science Campus. ClassifAI – Exploring the use of Large Language Models (LLMs) to assign free text to commonly used classifications, July 2024. https://datasciencecampus.ons.gov.uk/classifai-exploring-the-use-of-large-language-models-llms-to-assign-free-text-to-commonly-used-classifications/.

Feifeibear. Llmspeculativesampling, 2023. URL https://github.com/feifeibear/LLMSpeculativeSampling/.

Nikhil Bhendawade Irina Belousova Qichen Fu, Henry Mason, Mohammad Rastegari, and Mahyar Najibi. Speculative streaming: Fast llm inference without auxiliary models. In *Proc. of ICLR*, 2025.

Evangelos Georganas, Dhiraj D. Kalamkar, Alexander Kozlov, and Alexander Heinecke. Ml-specqd: Multi-level speculative decoding with quantized drafts. *ArXiv*, abs/2503.13565, 2025.

Golario. specinfer-ae, 2024. URL https://github.com/goliaro/specinfer-ae.

Zhengmian Hu, Tong Zheng, Vignesh Viswanathan, Ziyi Chen, Ryan A. Rossi, Yihan Wu, Dinesh Manocha, and Heng Huang. Towards optimal multi-draft speculative decoding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=9KxnxWOBA5.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. SpecDec++: Boosting Speculative Decoding via Adaptive Candidate Lengths. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Y131N9fUbU.

Doohyuk Jang, Sihwan Park, June Yong Yang, Yeonsung Jung, Jihun Yun, Souvik Kundu, Sung-Yub Kim, and Eunho Yang. Lantern: Accelerating visual autoregressive models with relaxed speculative decoding. In *Proc. of ICLR*, 2025.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. Speculative decoding with big little decoder. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 2024.

Taehyeon Kim, Hojung Jung, and Se-Young Yun. Multi-drafter speculative decoding with alignment feedback. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 32532–32573, 2026.

Sosuke Kobayashi. Homemade bookcorpus. https://github.com/soskek/bookcorpus, 2018.

Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In *SIAM Journal on Control and Optimization*. MIT Press, 2000.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

Tanishq Kumar, Tri Dao, and Avner May. Speculative speculative decoding. In *Proc. of ICLR*, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 19274–19286, 2023.

Jinze Li, Yixing Xu, Haiduo Huang, Xuanwu Yin, Dong Li, Edith C. H. Ngai, and Emad Barsoum. Gumiho: A hybrid architecture to prioritize early tokens in speculative decoding. In *Proc. of ICML*, 2025.

Jiesong Liu, Brian Park, and Xipeng Shen. A drop-in solution for on-the-fly adaptation of speculative decoding in large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9778–9794, 2025.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Xiao-Fan Lu, Yixiao Zeng, Feiyang Ma, Zixu Yu, and Marco Levorato. Improving multi-candidate speculative decoding. In *ENLSP*, 2024. URL https://api.semanticscholar.org/CorpusID:272693854.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949, 2024.

Nvidia. Megatron-LM. https://github.com/NVIDIA/Megatron-LM.

Google Research. Looking back at speculative decoding. *Google Research Blog*, 2023. URL https://research.google/blog/looking-back-at-speculative-decoding/.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian En-Hsu Yen, Avner May, Tianqi Chen, and Beidi Chen. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *Proc. of ICLR*, 2025.

Daniel Schlör. Prompt category classification (part 1), July 2023. https://blog.pulze.ai/prompt-category-classification/.

ShareGPT. sharegpt-english. https://huggingface.co/datasets/theblackcat102/sharegpt-english, 2023.

Benjamin Spector and Chris Ré. Accelerating llm inference with staged speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning (ICML 2023)*, 2023.

SambaNova Systems. Speculative decoding in sambastudio. *SambaNova Documentation*, 2023. URL https://docs.sambanova.ai/sambastudio/latest/spec-decoding.html.

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

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, et al. Stanford alpaca: An instruction-following llama model. *GitHub repository*, 2023b. URL https://github.com/tatsu-lab/stanford_alpaca.

PyTorch Team. A hitchhiker's guide to speculative decoding. *PyTorch Blog*, 2024. URL https://pytorch.org/blog/hitchhikers-guide-speculative-decoding/.

Nadav Timor, Jonathan Mamou, Daniel Korat, Moshe Berchansky, Gaurav Jain, Oren Pereg, Moshe Wasserblat, and David Harel. Accelerating LLM inference with lossless speculative decoding algorithms for heterogeneous vocabularies. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=vQubr1uBUw.

Clovis Varangot-Reille, Christophe Bouvard, Mathieu Ciancone, Antoine Gourru, Marion Schaeffer, and François Jacquenet. Doing more with less: A survey on routing strategies for resource optimisation in large language model-based systems. *J. Artif. Intell. Res.*, 86, 2025. URL https://api.semanticscholar.org/CorpusID:276094662.

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

Yize Wu, KE GAO, Ling Li, and Yanjun Wu. Easyspec: Layer-parallel speculative decoding for efficient multi-gpu utilization. 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. 158879–158903. Curran Associates, Inc., 2025. doi: 10.52202/085713-5307. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/e90edb290360871d352c47886c68c0d3-Paper-Conference.pdf.

C. Zhang Y. Li, F. Wei and H. Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. In *Proc. of Conference on Empirical Methods in Natural Language Processing*, 2024.

Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. Inference with reference: Lossless acceleration of large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL 2023)*. Association for Computational Linguistics, 2023. URL https://arxiv.org/pdf/2304.04487.

Penghui Yang, Cunxiao Du, Fengzhuo Zhang, Haonan Wang, Tianyu Pang, Chao Du, and Bo An. LongSpec: Long-context lossless speculative decoding with efficient drafting and verification. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, July 2026.

Sen Yang, Shujian Huang, Xinyu Dai, and Jiajun Chen. Multi-candidate speculative decoding. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pp. 335–348. Springer, 2025.

Kaiqi Zhang, Jing Zhao, and Rui Chen. Koala: Enhancing speculative decoding for llm via multi-layer draft heads with adversarial learning. In *2025 28th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 231–236, 2025. doi: 10.1109/CSCWD64889.2025.11033265.

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. *arXiv preprint arXiv:2412.18910*, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Haotong Zhang, Joseph Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of NeurIPS*, 2023a.

Siyuan Zheng, Xiaoxuan Liu, Dawei Zhang, Lianmin Liu, Ion Stoica, and Joseph E. Gonzalez. Official code for "online speculative decoding". `https://github.com/liuxiaoxuanpku/osd`, 2023b.

A Full Training and Implementation Details

A.1 SL Predictor (Linear Head on Draft Model)

- **Base model.** A single linear layer appended to the final hidden state of M_d. Hidden-state dimensions: 4096 (Llama-7B), 5120 (Llama-13B), 512 (Flan-T5-small), 2048 (Vicuna-68M).
- **Input encoding.** Request type: one-hot over the six MTBench categories. Prompt text: tokenized with the draft model's native tokenizer and truncated to 512 tokens.
- **Training.** Supervised learning on oracle SL labels, minimizing RMSE between predicted and oracle SL, using AdamW (learning rate 5×10^{-5}, weight decay 0.01) for 10 epochs with linear warm-up over the first 5% of steps followed by cosine decay, batch size 32, on one A100.
- **Data split.** 80%/10%/10% train/validation/test split (request-disjoint, seed 42) for each of Alpaca, ShareGPT, and BookCorpus.
- **Warm-up.** The linear head is used from step 1; the RL policy replaces it once trained (Appendix A.3).

A.2 CL Estimator (Random Forest Regression)

- **Algorithm.** Scikit-learn `RandomForestRegressor` with 42 trees, maximum depth 8, and a minimum of 4 samples per leaf, all selected via 5-fold cross-validation on the Alpaca training split.
- **Training data.** $\approx 1.2 \times 10^6$ step records collected across all datasets and model pairs; each instance is one speculative decoding step labeled with the resulting speedup.
- **Train/test split.** Same 80/10/10 split as Appendix A.1, seed 42.
- **Search range and convergence threshold.** $W_{\min}=1$, $W_{\max}=25$, $\epsilon=1$.

A.3 Reinforcement Learning (Actor–Critic)

- **Architecture.** Actor: 4-layer MLP (133→256→256→128→output) producing real-valued (H_i, W_i), rounded and clamped to valid ranges. Critic: 3-layer MLP (133→256→128→1).
- **State encoding.** The 133-dimensional state concatenates step index (1), SL/CL (2), a prompt embedding projected from the draft hidden state (128), acceptance ratio (1), and logit divergence (1).
- **Optimization.** Adam with learning rate 10^{-3} for the actor and 5×10^{-4} for the critic; discount factor $\kappa=0.9$; ε-greedy exploration (initial $\varepsilon=0.9$, decay 0.995/episode, minimum 0.05); replay batch size 16 transitions per gradient update.
- **Training data.** Pre-trained offline on $\approx 2 \times 10^5$ transitions per model pair collected from the SL predictor and binary-search CL components; retrained online every 500 new requests from the most recent transition buffer.
- **Random seeds.** All results are averages over 3 independent runs with seeds $\{0, 1, 2\}$.

A.4 Request-Type Classifier

Table 3 compares the three request-type classifiers we evaluated on all 480 MTBench prompts. We use the lightweight zero-shot classifier (Schlör, 2023; Data Science Campus, 2024; Varangot-Reille et al., 2025).

Table 3: Request-type classification latency and accuracy (mean over 480 MTBench prompts).

Classifier	Latency (ms)	Accuracy (%)
Rule-based keyword detection	<1	81
Zero-shot LLM (DistilBERT)	12	89
Lightweight zero-shot	38	93

B Model Inputs and RL Formulation

B.1 SL Predictor Input

For step i, the SL head receives the concatenation

$$\mathbf{x}_i^{\mathrm{SL}} = [\,\mathbf{h}_i\ (d_h);\ \mathbf{e}(T)\ (6);\ i/i_{\max}\ (1);\ \bar{A}_i\ (1);\ \bar{L}_i\ (1)\,] \in \mathbb{R}^{d_h+9},$$

where $\mathbf{h}_i$ is the final hidden state of M_d at the last position of the current sequence (d_h as in Appendix A.1), $\mathbf{e}(T)$ is the one-hot request type from the classifier, $i_{\max}$=25, and $\bar{A}_i$ and $\bar{L}_i$ are the acceptance ratio and draft–target KL divergence averaged over steps $i-3$ to $i-1$ (zero for missing steps). Scalar features are standardized using training-split statistics. The output $\hat{H}_i = \mathbf{w}^\top \mathbf{x}_i^{\mathrm{SL}} + b$ is regressed onto the oracle SL (RMSE), then rounded and clipped to $[1, 25]$ at inference.

B.2 CL Estimator Input

$\hat{S}(W) = f_{\mathrm{RF}}(\mathbf{x}_i^{\mathrm{CL}}(W))$ with

$$\mathbf{x}_i^{\mathrm{CL}}(W) = [\,W;\ H_i;\ \bar{c}_i;\ \bar{A}_i;\ \bar{D}_i;\ \bar{s}_i\,] \in \mathbb{R}^6,$$

where W is the queried CL, H_i the SL selected for step i, $\bar{c}_i$ the draft model's average probability of selected tokens, $\bar{A}_i$ the acceptance ratio, $\bar{D}_i$ the KL divergence (all averaged over the previous m=3 steps), and $\bar{s}_i$ the moving average of observed speedup over all previous steps. Each training record is one executed step, with its executed (H, W) and its measured speedup as label.

B.3 Actor–Critic Formulation

An episode is one request comprising T decoding steps. At each step i, the policy observes the current state and selects both speculative length (SL) and candidate length (CL).

State. Let $\mathbf{P}_i \in \mathbb{R}^{128}$ be the projection of the draft-model hidden state for the current sequence, V_{i-1} the acceptance ratio observed at the previous step, and D_{i-1} the corresponding draft–target logit divergence. The state is

$$\mathbf{S}_i = [\,i;\ H_{i-1};\ W_{i-1};\ \mathbf{P}_i;\ V_{i-1};\ D_{i-1}\,] \in \mathbb{R}^{133}.$$

Here H_{i-1} and W_{i-1} are the previously executed SL and CL. The feedback and previous actions are initialized to zero when $i = 1$. The SL predictor and CL estimator use their own three-step averages.

Action. The actor outputs $\boldsymbol{\mu}_\theta(\mathbf{S}_i) \in \mathbb{R}^2$ for a Gaussian policy $\pi_\theta(\cdot \mid \mathbf{S}_i) = \mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{S}_i), \sigma^2\mathbf{I})$. A continuous sample $\mathbf{a}_i = (a_{i,1}, a_{i,2})$ is mapped to the executed integer action

$$(H_i, W_i) = \big(\mathrm{clip}(\mathrm{round}(a_{i,1}), 1, 25),\ \mathrm{clip}(\mathrm{round}(a_{i,2}), 1, 25)\big).$$

Here σ denotes the policy's action standard deviation. The policy-gradient update uses the log-probability of the continuous sample.

Reward and objective. Let n_i be the number of tokens committed at step i, τ_t the target-only latency per token, and ℓ_i the measured step latency including drafting, verification, and decision overhead. The step-level speedup reward is $r_i = \tau_t n_i / \ell_i$. The total undiscounted reward is $R = \sum_{i=1}^{T} r_i$; with discount $\kappa = 0.9$, the actor–critic agent maximizes

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{i=1}^{T} \kappa^{i-1} r_i\right].$$

This sum of step-level rewards is the training objective; request-level end-to-end speedup is measured separately in the evaluation.

Actor–critic update. Let $d_i = 1$ at the terminal step and zero otherwise. The one-step temporal-difference error is

$$\delta_i = r_i + \kappa(1 - d_i)V_\phi(\mathbf{S}_{i+1}) - V_\phi(\mathbf{S}_i).$$

The critic minimizes δ_i^2 and the actor minimizes $-\delta_i \log \pi_\theta(\mathbf{a}_i \mid \mathbf{S}_i)$, using mini-batches of 16 transitions.

Exploration. During ε-greedy exploration, with probability ε the executed action is drawn uniformly from $\{1, \ldots, 25\}^2$; otherwise it is sampled from the Gaussian policy and mapped to integers as above. The schedule is $\varepsilon_0 = 0.9$, multiplication by 0.995 per request, and a floor of 0.05 during offline pre-training.

C Binary Search for CL

Algorithm 1 gives the full CL search procedure of §4.2.

Algorithm 1 Binary Search for Optimal CL at Decoding Step i

Input: $W_{\min}$, $W_{\max}$, speedup estimator $\hat{S}(\cdot)$, convergence threshold ϵ
Output: Optimal candidate length W_{mid} at convergence

while $|W_{\max} - W_{\min}| > \epsilon$ **do**
  $W_{\text{mid}} \leftarrow \lfloor (W_{\min} + W_{\max})/2 \rfloor$ $s_- \leftarrow \hat{S}(W_{\text{mid}}-1)$; $s_0 \leftarrow \hat{S}(W_{\text{mid}})$; $s_+ \leftarrow \hat{S}(W_{\text{mid}}+1)$ **if** $s_0 > s_-$ ***and*** $s_0 > s_+$ **then**
    return W_{mid} `// local maximum --- stop`
  end if
  if $s_- < s_0 < s_+$ **then**
    $W_{\min} \leftarrow W_{\text{mid}} + 1$ `// increasing rightward --- shift right`
  else
    $W_{\max} \leftarrow W_{\text{mid}} - 1$ `// non-increasing rightward --- shift left`
  end if
end while
return W_{mid}

D Compute Resources and Training Cost

Table 4 reports the approximate wall-clock time and GPU-hours for different operations and training. The full experimental suite required approximately 1,600 A100 GPU-hours, with oracle SL label collection adding ≈300 and preliminary experiments ≈400 GPU-hours, for a total of 2,300 A100 GPU-hours.

Table 4: Wall-clock time per experimental run (Alpaca, 2,000 requests).

Operation	Time (h)	GPU-hours
Vanilla / SpecInfer / OSD	≈ 1.5	12
AdaptSpec (no RL training)	≈ 1.6	12.8
SL predictor fine-tuning	≈ 2.0	2 (1 GPU)
CL estimator training	< 0.5	< 0.5 (CPU)
RL offline pre-training	≈ 4.0	4 (1 GPU)
Oracle SL label collection	≈ 18	144

E Dataset and Asset Licenses

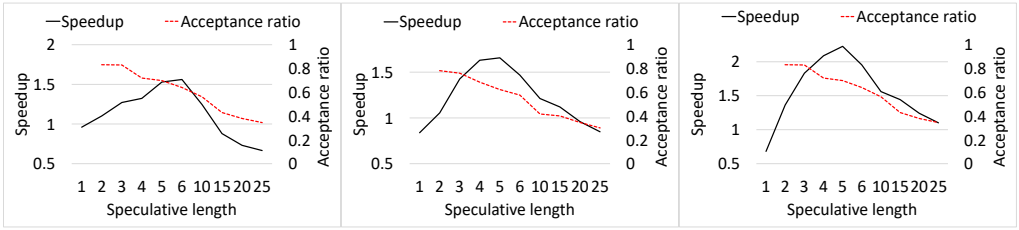
Table 5 lists all external assets used in this paper with their licenses. No modifications were made to any dataset. All assets are used in accordance with their respective licenses for non-commercial academic research.

F Additional Results for All Model Pairs

This section contains the per-model-pair results summarized in the main text.

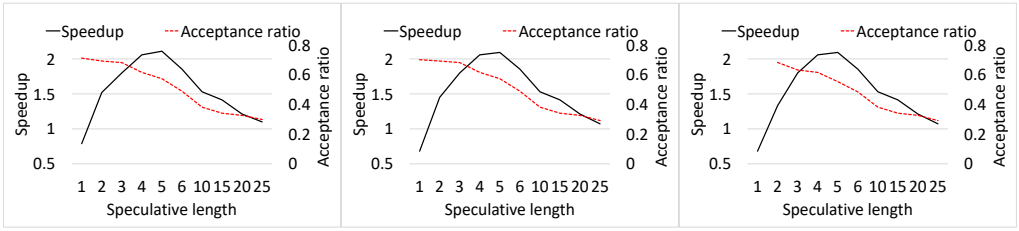
Table 5: Licenses for all external assets used in this work.

Asset	Type	License	Reference / URL
Alpaca	Dataset	CC BY 4.0	Taori et al. (2023a); `github.com/tatsu-lab/stanford_alpaca`
ShareGPT	Dataset	CC BY 4.0	ShareGPT (2023); `huggingface.co/datasets/theblackcat102/sharegpt-english`
BookCorpus	Dataset	MIT	Kobayashi (2018); `github.com/soskek/bookcorpus`
MTBench	Dataset	Apache 2.0	Zheng et al. (2023a); `github.com/lm-sys/FastChat`
LongWriter	Dataset	Apache 2.0	Bai et al. (2024)
Llama-7B / 13B / 33B	Model	LLaMA Community License	Meta AI
Flan-T5-small / medium	Model	Apache 2.0	Google via HuggingFace Hub
Vicuna-68M / 7B	Model	LLaMA Community License	LMSYS via HuggingFace Hub
Megatron-LM	Code	BSD 3-Clause	Nvidia; `github.com/NVIDIA/Megatron-LM`
Vanilla spec. decoding code	Code	MIT	Feifeibear (2023)
SpecInfer code	Code	Apache 2.0	Golario (2024)
OSD code	Code	MIT	Zheng et al. (2023b); `github.com/liuxiaoxuanpku/osd`
SpecDec++ code	Code	MIT	Huang et al. (2025); `github.com/Kaffaljidhmah2/SpecDec_pp`



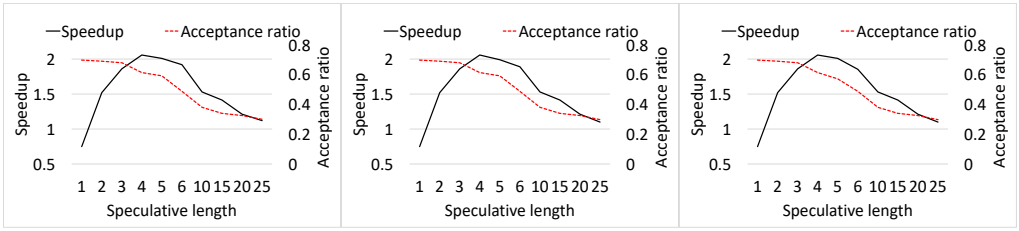
(a) Alpaca. **(b)** ShareGPT. **(c)** BookCorpus.

Figure 14: Speedup and acceptance ratio vs. SL for Llama-13B/Llama-33B.



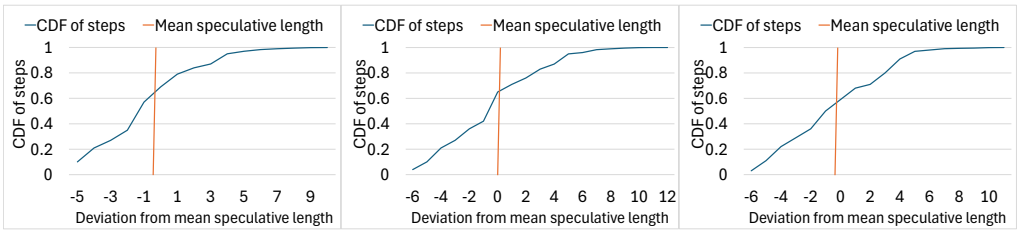
(a) Alpaca. **(b)** ShareGPT. **(c)** BookCorpus.

Figure 15: Speedup and acceptance ratio vs. SL for T5-small/T5-medium.



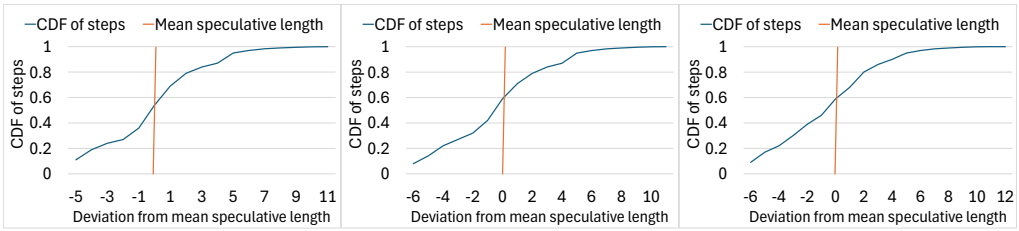
(a) Alpaca. **(b)** ShareGPT. **(c)** BookCorpus.

Figure 16: Speedup and acceptance ratio vs. SL for Vicuna-68M/Vicuna-7B.



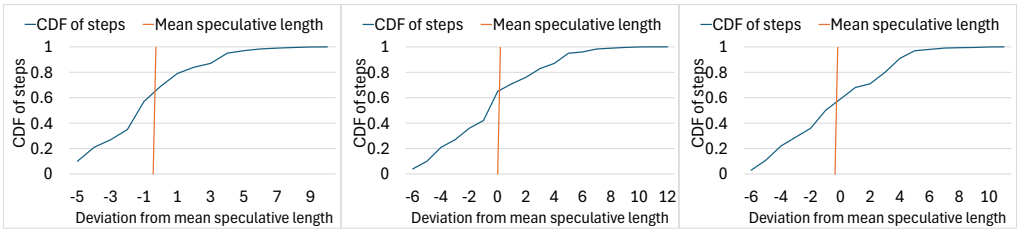
(a) Alpaca. **(b)** ShareGPT. **(c)** BookCorpus.

Figure 17: Deviation of step-wise oracle SL from the mean oracle SL of Llama-13B/Llama-33B.



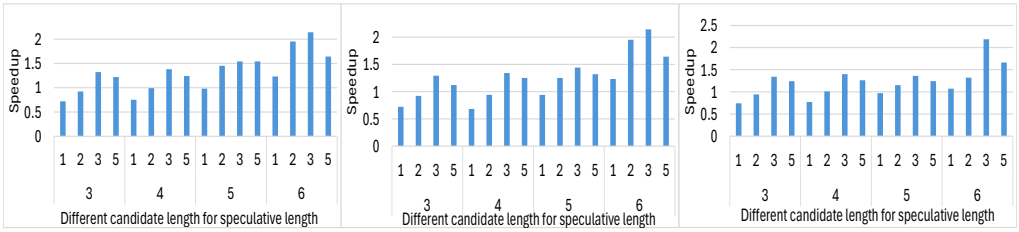
(a) Alpaca. **(b)** ShareGPT. **(c)** BookCorpus.

Figure 18: Deviation of step-wise oracle SL from the mean oracle SL of T5-small/T5-medium.



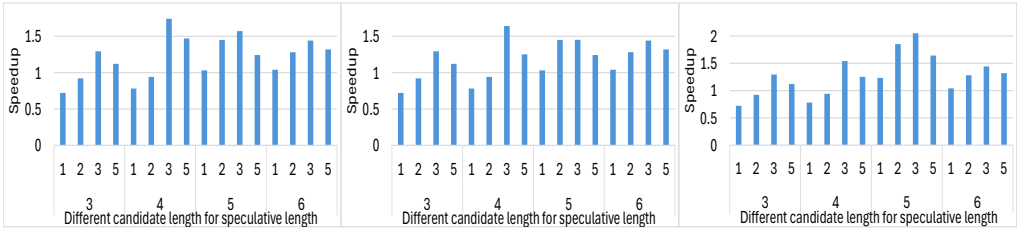
(a) Alpaca. **(b)** ShareGPT. **(c)** BookCorpus.

Figure 19: Deviation of step-wise oracle SL from the mean oracle SL of Vicuna-68M/Vicuna-7B.



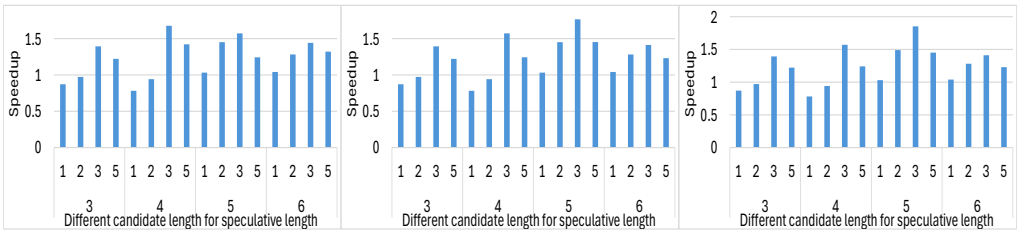
(a) Alpaca. **(b)** ShareGPT. **(c)** BookCorpus.

Figure 20: Speedup for different CLs at each SL for Llama-13B/Llama-33B.



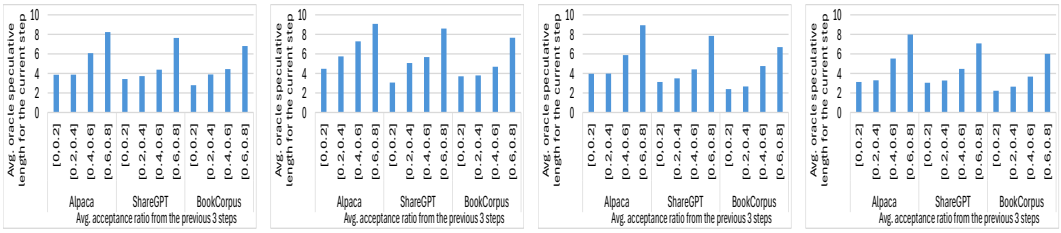
(a) Alpaca. **(b)** ShareGPT. **(c)** BookCorpus.

Figure 21: Speedup for different CLs at each SL for T5-small/T5-medium.



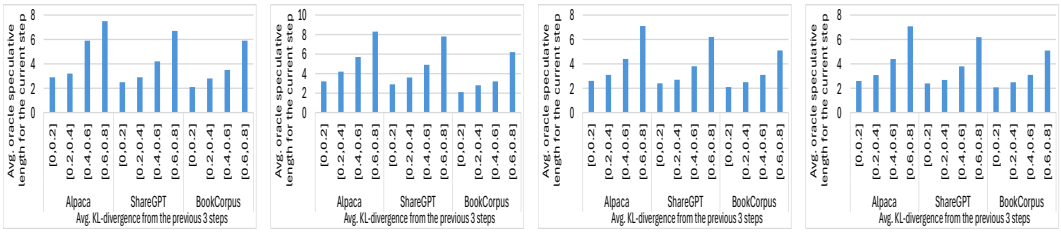
(a) Alpaca. **(b)** ShareGPT. **(c)** BookCorpus.

Figure 22: Speedup for different CLs at each SL for Vicuna-68M/Vicuna-7B.



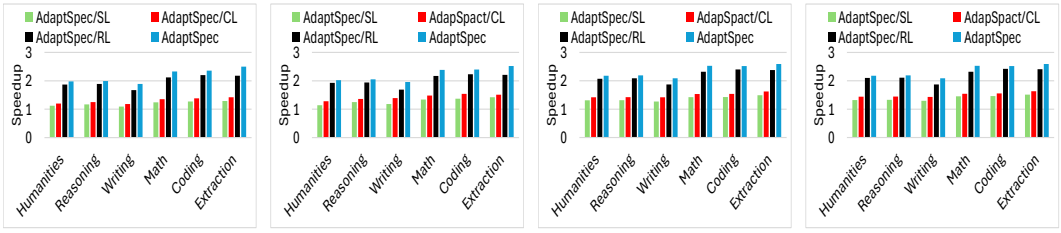
(a) Llama-7B/33B. **(b)** Llama-13B/33B. **(c)** T5-small/medium. **(d)** Vicuna-68M/7B.

Figure 23: Oracle SL vs. acceptance ratio of previous three steps, all model pairs.



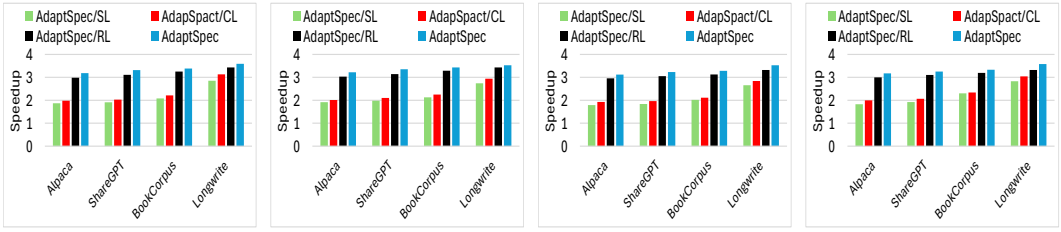
(a) Llama-7B/33B. **(b)** Llama-13B/33B. **(c)** T5-small/medium. **(d)** Vicuna-68M/7B.

Figure 24: Oracle SL vs. logit divergence of previous three steps, all model pairs.



(a) Llama-7B/33B. **(b)** Llama-13B/33B. **(c)** T5-small/medium. **(d)** Vicuna-68M/7B.

Figure 25: Ablation studies for all models on the MTBench dataset.



(a) Llama-7B/33B. **(b)** Llama-13B/33B. **(c)** T5-small/medium. **(d)** Vicuna-68M/7B.

Figure 26: Ablation studies for all models on the Alpaca, ShareGPT, BookCorpus, and LongWriter datasets.

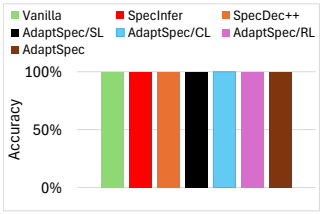
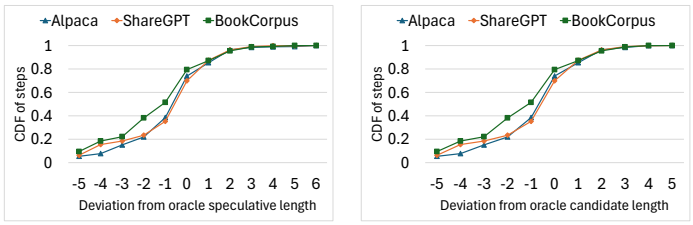


Figure 27: Accuracy of all methods.



(a) SL. **(b)** CL.

Figure 28: Variation of SL and CL from the oracle for AdaptSpec.