Learning Adaptive LLM Decoding

Anonymous Authors[1]

Abstract

Decoding from large language models (LLMs) typically relies on fixed sampling hyperparameters (e.g., temperature, top-p), despite substantial variation in task difficulty and uncertainty across prompts and individual decoding steps. We propose to learn adaptive decoding policies that dynamically select sampling strategies at inference time, conditioned on available compute resources. Rather than fine-tuning the language model itself, we introduce lightweight decoding adapters trained with reinforcement learning and verifiable terminal rewards (e.g. correctness on math and coding tasks). At the sequence level, we frame decoding as a contextual bandit problem: a policy selects a decoding strategy (e.g. greedy, top-k, min-p) for each prompt, conditioned on the prompt embedding and a parallel sampling budget. At the token level, we model decoding as a partially observable Markov decision process (POMDP), where a policy selects sampling actions at each token step based on internal model features and the remaining token budget. Experiments on the MATH and CodeContests benchmarks show that the learned adapters improve the accuracy–budget tradeoff: on MATH, the token-level adapter improves Pass@1 accuracy by up to 10.2% over the best static baseline under a fixed token budget, while the sequence-level adapter yields 2–3% gains under fixed parallel sampling. Ablation analyses support the contribution of both sequence- and token-level adaptation.

1. Introduction

Large language models (LLMs) have achieved remarkable performance across domains ranging from mathematical reasoning (Ren et al., 2025; Yang et al., 2024) to code generation (Chen et al., 2021; Princis et al., 2025) and scientific discovery (Boiko et al., 2023). Despite this progress, inference from LLMs remains a major computational bottleneck. A key source of inefficiency lies in decoding, the process by which a model samples output tokens from its predictive distribution. In current practice, decoding relies on fixed sampling hyperparameters such as temperature, top-k, and top-p, chosen statically for an entire model or dataset. This ignores the substantial heterogeneity across prompts, reasoning styles, and even individual tokens. In many cases, the optimal decoding strategy depends on latent features such as token-level uncertainty or problem structure, with recent analyses showing that uncertainty during reasoning is often concentrated at a small number of high-entropy tokens (Lin et al., 2024; Wang et al., 2025a).

Recent work has explored adaptive sampling and confidence-aware decoding, demonstrating that modulating stochasticity can substantially affect generation quality and reasoning performance (Nguyen et al., 2024; Zhang et al., 2024; Dhuliawala et al., 2024; Lin et al., 2024; Wang et al., 2025a). However, these approaches typically rely on static heuristics or offline-tuned parameters and do not incorporate decoding decisions directly into an end-to-end learning objective. In parallel, reinforcement learning with verifiable rewards (RLVR) (Guo et al., 2025) has improved reasoning performance by fine-tuning the model, but commonly treats decoding as a fixed generation setting rather than a learnable component (von Werra et al., 2020). This separation induces a train–test mismatch when models are trained under one decoding distribution and deployed under different inference-time constraints.

Motivated by these gaps, we explore a different axis of inference control: learning decoding-time policies that adapt sampling strategy based on model state and available compute budget. Our intuition draws on observations of "forking tokens" in reasoning (Wang et al., 2025a), where a small number of high-entropy decisions disproportionately influence multi-step outcomes. Rather than hand-designing entropy thresholds or performing explicit tree search, we allow different parts of a single trajectory to exhibit different degrees of stochasticity, learned directly from task reward.

To this end, we introduce *Learned Decoding Adapters*, a family of reinforcement-learning–based policies that modu-

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

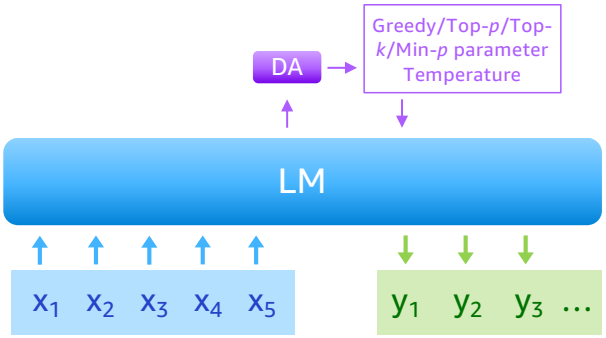
late decoding during inference while leaving the underlying language model fixed. At the sequence level, we formulate decoding strategy selection as a contextual bandit: the adapter selects a decoding configuration (e.g. greedy, top-k, top-p, or min-p) for each prompt, conditioned on prompt features and a parallel-sampling budget. The action space in this setting is constructed via a data-driven greedy selection procedure over candidate decoding strategies. At the token level, we treat decoding as a partially observable Markov decision process (POMDP): the adapter observes internal model representations and the remaining token budget, and selects a decoding action at each step. While the framework supports arbitrary decoding configurations, in our experiments we focus on temperature-based actions for token-level control, which provide a simple and interpretable axis for dynamically allocating stochasticity within a single generation trajectory.

We train both adapters using policy-gradient reinforcement learning (REINFORCE (Williams, 1992)) with verifiable terminal rewards, such as correctness checks on math and code problems. Experiments on the MATH and CodeContests benchmarks show that the learned adapters improve the accuracy-budget tradeoff. The token-level adapter improves Pass@1 accuracy by up to 10.2% under fixed token budgets, while the sequence-level adapter outperforms strong fixed strategies under limited parallel sampling. Ablation analyses support the complementary contributions of sequence-level and token-level adaptation.

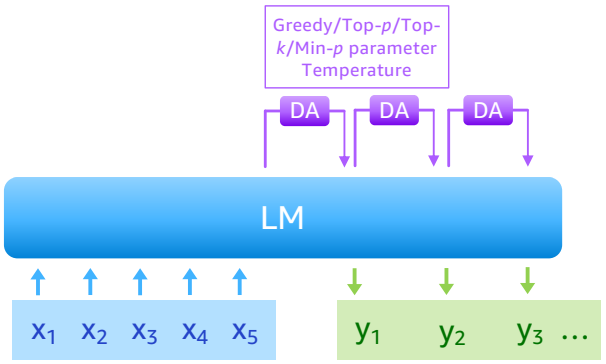
Contributions. We make the following contributions:

- We formulate decoding-time inference as a policy learning problem, introducing a unified reinforcement learning framework for both prompt-level and token-level adaptation under explicit compute budgets.
- We propose decoding adapters trained solely with online verifiable task rewards, without learned reward models, preference labels, or hand-designed decoding heuristics, while keeping the underlying language model fixed.
- We demonstrate empirical gains on mathematical and coding reasoning benchmarks under constrained compute, and analyze how the learned adapters allocate stochasticity to improve solution accuracy.

Figure 1 provides an overview of the proposed decoding adapter framework, illustrating both the sequence-level and token-level policies layered on top of a frozen language model under explicit compute budgets.



(a) **Sequence-level adapter**: a single DA predicts one decoding configuration that is applied throughout generation.



(b) **Token-level adapter**: the DA is invoked at each decoding step to select a (potentially different) decoding configuration per token.

Figure 1. Overview of the proposed decoding adapter (DA) for a frozen language model (LM). Blue blocks denote input tokens x_i; green blocks denote generated tokens y_i.

2. Methods

2.1. Preliminaries

We study inference-time control of decoding for a frozen large language model (LLM) f under explicit compute budgets. Our goal is to learn a lightweight *decoding adapter* that modulates how tokens are sampled from f while leaving all LLM parameters fixed.

Budgets. For each problem instance, let B denote a *parallel sampling budget*, quantified as the maximum number of full decoding trajectories that may be generated, and let b denote a *token budget*, quantified as the maximum number of decoding steps within a single trajectory. For token-level control, we also write $b_t = b - t$ for the remaining token budget at decoding step t.

Base model distribution and decoding actions. Let q denote an input prompt and $y_{<t}$ a partial generation. The frozen LLM produces next-token logits $z_t = f(q, y_{<t}) \in \mathbb{R}^{|\mathcal{V}|}$, inducing a base distribution $p_f(\cdot \mid q, y_{<t}) =$

$\mathrm{Softmax}(z_t)$. At each decision point (either once per sequence or once per token), the adapter selects an action from a discrete action space $S = \{a_1, \ldots, a_m\}$, where each action corresponds to a decoding configuration specified by sampling parameters such as `temperature`, `top_k`, `top_p`, and `min_p`. Each action $a \in S$ specifies a transformation T_a applied to the base distribution, yielding $p_{f,a}(\cdot \mid q, y_{<t}) \triangleq T_a(p_f(\cdot \mid q, y_{<t}))$, from which the next token is sampled.

Reward and policy. We assume access to a *verifiable terminal reward* function r, which indicates success or failure of the final generated output (e.g. correctness on math or coding problems). The adapter is parameterized by a policy $\pi_\theta(a \mid x)$, where the input x consists of features derived from the frozen LLM (e.g. prompt embeddings or hidden-state representations). In the *budget-aware* setting, we include the relevant budget in the policy input: $x = [e; B]$ or $x_t = [e_t; b_t]$, depending on whether decisions are made at the sequence or token level.

2.2. Sequence-Level: Contextual Bandits

In the sequence-level setting, the decoding adapter selects a single decoding configuration that is applied uniformly within each generation process. Because this decision is made once per prompt and determines the sampling dynamics for the entire rollout before a terminal reward is observed, the problem naturally admits a contextual bandit formulation.

Given a prompt embedding e and a parallel sampling budget B, the policy $\pi_\theta(a \mid x)$ with $x = [e; B]$ selects a decoding action $a \in S$. The language model then produces a complete output sequence under the sampling dynamics induced by a, after which a terminal reward $r(x, a)$ is observed. Including B in the context reflects that sequence-level decisions govern how compute is allocated across parallel rollouts: larger budgets may favor more exploratory strategies, smaller budgets favor conservative ones.

We train the adapter to maximize expected terminal reward with entropy regularization to encourage exploration:

$$\mathcal{J}_{\text{seq}}(\theta) = \mathbb{E}_{x\sim D,\, a\sim\pi_\theta(\cdot|x)}\big[r(x,a)\big] + \beta\, \mathbb{E}_{x\sim D}\big[\mathcal{H}(\pi_\theta(\cdot \mid x))\big]. \tag{1}$$

Here the terminal reward $r(x, a)$ corresponds to the evaluation metric induced by the inference procedure under budget B (e.g. Pass@k or best-of-B), aggregating outcomes across the available parallel samples. In practice, we optimize this objective using a Monte Carlo policy-gradient estimator with a variance-reducing baseline.

2.3. Token-Level: Reinforcement Learning

In the token-level setting, the decoding adapter selects actions at each decoding step, allowing the sampling strategy to vary within a single generation trajectory. Unlike the sequence-level case, decisions are made repeatedly over time and influence future states, making this setting naturally modeled as a partially observable Markov decision process (POMDP). Partial observability arises because the adapter does not observe the full environment state, only a compressed representation of the model's internal activations and the remaining budget.

At each step, the policy observes a compact state representation $x_t = [e_t; b_t]$, where e_t is derived from the frozen LLM's hidden-state embedding at step t and $b_t = b - t$ is the remaining token budget. The policy then samples $a_t \sim \pi_\theta(\cdot \mid x_t)$, which specifies the decoding configuration used to generate the next token. Including b_t reflects that token-level decisions govern how stochasticity is allocated *within* a trajectory: when substantial budget remains, exploratory sampling may be beneficial, while near the end more deterministic decoding may stabilize completion.

Given actions $\mathbf{a} = (a_1, \ldots, a_T)$, the objective is to maximize expected terminal reward:

$$\mathcal{J}_{\text{tok}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\mathbf{a}\sim\pi_\theta}\big[r(x^{(i)}, \mathbf{a})\big]. \tag{2}$$

We optimize using policy-gradient methods with entropy regularization and a variance-reducing baseline.

Training stability. Naive token-level REINFORCE led to high-variance gradients and unstable training. We apply two stabilizations: (i) we filter training prompts that produce extremely sparse or noisy reward signals, and (ii) we mask tokens whose next-token distribution is already highly concentrated (max probability exceeding 0.95), as these contribute little learning signal while significantly increasing gradient variance.

2.4. Selection of Action Space

While our framework allows for arbitrary decoding actions, in practice we restrict the action space to a finite set of representative sampling strategies. For the sequence-level adapter, we construct the action space using a data-driven selection procedure inspired by the coverage-based approach of AuPair (Mavalankar et al., 2025). Starting from a large candidate pool of decoding configurations spanning combinations of temperature, top-k, top-p, and min-p values, we greedily select a small subset $S \subseteq \mathcal{C}$ of size k that approximately maximizes $F(S) \triangleq \sum_{x\in\mathcal{D}_{\text{val}}} \max_{s\in S} R(x, s)$, a standard monotone submodular objective (Nemhauser et al., 1978). A full description of the procedure and candidate

pool is given in Appendix E.

For the token-level adapter, we adopt a more restricted action space focused on temperature-based decoding actions, while holding other sampling parameters fixed. We found temperature to be a particularly effective and interpretable axis for token-level control; dynamically varying truncation-based parameters such as top-k or top-p at the token level introduces additional complexity without yielding qualitatively different behavior in our setting.

3. Sequence-Level Experiments

We evaluate the *sequence-level* decoding adapter, which selects a single decoding strategy per prompt under an explicit parallel sampling budget, on two representative reasoning domains: mathematical problem solving and competitive programming. We set the max output length to 8092 for all sequence- and token-level experiments, except for AIME'25, where we extend to 38,912 as in (Yang et al., 2025).

3.1. Experimental Setup

Datasets. We use MATH (Hendrycks et al., 2021) and CodeContests (Li et al., 2022), both exhibiting substantial heterogeneity in problem structure and uncertainty.

Models. We primarily report results using Qwen3-4B (Yang et al., 2025). Additional results on Qwen2.5-Math-1.5B and Qwen3-8B are provided in Appendix D.

Decoding strategies and baselines. We greedily select a small action set separately for MATH, CodeContests, and a mixed math+code validation set (see Appendix E). We compare against two static baselines: (BEST), the single fixed strategy that achieves the highest validation performance, and (MIXED), a uniform mixture over the selected set. The latter uses the same action space as the adapter but without learning.

Prompt distributions and inference budgets. We evaluate under both standard prompting and mixed Chain-of-Thought (CoT) prompting. For CoT-mixed training, a fixed fraction of prompts is randomly augmented with a CoT template. We evaluate Pass@k with $k \in \{1, 2, 4, 8\}$. For budget-aware variants, the parallel sampling budget is encoded via a two-layer MLP and concatenated to the context embedding.

3.2. Main Results

We report mean accuracy with 95% confidence intervals aggregated over k=3 runs. Tables 1 (MATH) and 2 (CodeContests) summarize the sequence-level adapter versus static baselines. Across all settings, the learned adapter outperforms both BEST and MIXED, with gains increasing under larger parallel sampling budgets. Improved performance is not due to extended generation length (see Appendix).

Effect of budget conditioning and mixed-strategy training. Conditioning the policy on the parallel sampling budget consistently improves performance relative to budget-agnostic training. Training on a mixture of inference strategies further improves robustness to evaluation-time heterogeneity, with the largest gains observed under mixed CoT prompting.

Mixed-domain and out-of-domain generalization. A single sequence-level adapter trained jointly on math and coding data continues to improve over static baselines on both domains, though gains are smaller than in single-domain training (Appendix D). Evaluating a MATH-trained adapter on AIME'25 without additional tuning also yields gains over reported baselines, indicating that the learned strategy-selection policy captures transferable signals beyond the training distribution.

Policy behavior. Figure 2 illustrates how the sequence-level adapter allocates probability mass over decoding strategies during training. Rather than collapsing to a single strategy, the learned policy concentrates mass on a small subset of high-performing actions while maintaining non-zero probability on alternatives, consistent with a learned trade-off between exploitation and robustness. Action distributions are broadly consistent between CoT and non-CoT prompts and across inference metrics (Pass@1 vs. Pass@8), suggesting that selection is driven primarily by context and budget rather than prompt format or evaluation objective.

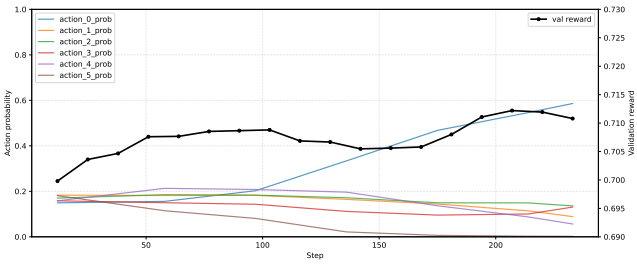


Figure 2. **Action distributions and validation reward on MATH** for sequence-level adapter strategies. The x-axis denotes training progress; the left y-axis reports action probability; the right y-axis reports validation reward.

These empirical observations align with the two structural properties we formalize in Appendix B: adaptation strictly improves over the best static strategy whenever the reward-maximizing action is prompt-dependent, and token-level policies subsume the sequence-level class.

Table 1. **MATH.** Comparison of static sampling baselines and the sequence-level adapter under different settings. For *CoT mix*, we apply CoT prompting to 30% of prompts (ratio = 0.3). Static sampling reports the best single strategy (BEST) and a fixed mixture (MIXED). The adapter is evaluated with a budget provided in feature set and without. Values are percentages (mean ± confidence interval).

Metric	Setting	Static sampling		Sequence-level adapter	
		Best	Mixed	w/o budget	w/ budget
Pass@1	TURN (Du et al., 2025)			72.72	
	w/o CoT	71.70 ± 1.25	71.20 ± 1.26	72.60 ± 1.24	72.90 ± 1.23
	Δ Abs. (Rel.)			↑ 0.90 (+1.26%)	↑1.20 (+1.67%)
	mix CoT	72.10 ± 1.24	71.93 ± 1.25	73.60 ± 1.22	**74.20** ± 1.21
	Δ Abs. (Rel.)			↑1.50 (+2.08%)	↑2.10 (+2.91%)
Pass@8	TURN (Du et al., 2025)			76.96	
	w/o CoT	76.70 ± 1.17	76.23 ± 1.18	78.30 ± 1.14	78.46 ± 1.14
	Δ Abs. (Rel.)			↑ 1.60 (+2.09%)	↑ 1.76 (+2.29%)
	mix CoT	77.10 ± 1.16	76.57 ± 1.17	78.80 ± 1.13	**79.80** ± 1.11
	Δ Abs. (Rel.)			↑1.70 (+2.20%)	↑ 2.70 (+3.50%)

Table 2. **CodeContests**: comparison of static sampling strategies vs. the **sequence**-level sampling adapter under different settings.

Metric	Setting	Static sampling		Sequence-level adapter	
		Best	Mixed	w/o budget	w/ budget
Pass@1	w/o CoT	11.43 ± 2.28	10.53 ± 2.19	14.53 ± 2.52	14.50 ± 2.52
	Δ Abs. (Rel.)			↑ 3.10 (+27.12%)	↑ 3.07 (+26.85%)
	mix CoT	13.97 ± 2.48	14.80 ± 2.54	17.06 ± 2.69	**19.70** ± 2.85
	Δ Abs. (Rel.)			↑ 2.26 (+15.27%)	↑ 4.90 (+33.11%)
Pass@8	w/o CoT	22.80 ± 3.00	22.23 ± 2.97	26.08 ± 3.14	23.10 ± 3.02
	Δ Abs. (Rel.)			↑ 3.28 (+14.39%)	↑ 0.30 (+1.32%)
	mix CoT	29.10 ± 3.25	25.63 ± 3.12	29.90 ± 3.28	**32.50** ± 3.35
	Δ Abs. (Rel.)			↑ 0.80 (+2.75%)	↑ 3.40 (+11.68%)

4. Token-Level Experiments

We evaluate the token-level decoding adapter, which selects a decoding action at each generation step, allowing stochasticity to vary within a single trajectory. As in the sequence-level setting, we compare against static sampling baselines. To isolate the effect of token-level adaptation, we restrict the action space to four temperature-based decoding strategies (Table 14).

Setup. We train token-level adapters under a fixed per-sequence token budget and consider two variants: one that conditions only on the usual model context, and one that additionally conditions on the remaining token budget. For static baselines, we evaluate greedy decoding and a fixed mixture over the same four temperature actions used by the adapter. Within this restricted temperature-only action space, greedy decoding achieves the strongest static performance on validation without chain-of-thought and therefore serves as the primary static baseline.

Main results. Results on MATH are summarized in Table 3. The token-level adapter substantially outperforms all static decoding strategies. Conditioning on the remaining token budget yields the strongest performance, improving Pass@1 by approximately 9–10% over the best static baseline. Even without explicit budget conditioning, the token-level adapter achieves consistent gains over static decoding. Notably, the magnitude of these improvements exceeds those observed for sequence-level adaptation under comparable evaluation settings, highlighting the value of fine-grained, within-trajectory control.

Is entropy alone sufficient? To test whether token-level adaptation can be reduced to entropy-based heuristics, we train an ablation in which the adapter observes only token entropy rather than the full contextual representation. This entropy-only policy reaches a validation reward of 72.03 (averaged over 3 seeds), comparable to static decoding baselines but far below the full token-level adapter, which exceeds 80% Pass@1. While entropy correlates with uncertainty, entropy alone is insufficient to recover the gains

Table 3. **Comparison of sampling strategies on Math**: performance comparison of **token**-level sampling adapter versus static sampling strategies. Here `budget` is the remaining token budget within a sequence. Values are percentages.

		Static sampling		Adapter			
Metric	**Setting**	Greedy	Mixed	w/o budget	Δ	budget	Δ
Pass@1	w/o CoT	71.33 ± 1.25	71.60 ± 1.25	78.28 ± 1.14	+6.68	80.82 ± 1.07	+9.49
	mix CoT	72.10 ± 1.24	72.67 ± 1.24	78.52 ± 1.14	+5.85	**82.33** ± 1.03	+10.23

achieved by learned token-level adaptation.

Policy behavior. We observe that lower-entropy tokens are more frequently collapsed to near-deterministic behavior under the learned policy, while higher-entropy tokens tend to retain greater stochasticity. Despite this trend, we did not identify simple qualitative rules or low-dimensional heuristics that fully explain the policy's behavior (see Appendix D.4), suggesting that the gains are not attributable to a small number of easily interpretable patterns.

5. Related Work

Adaptive Decoding Strategies. Sampling hyperparameters such as temperature, top-p, and top-k are typically fixed across inputs and decoding steps, despite substantial variation in prompt difficulty and uncertainty. Several recent methods seek to adapt these parameters dynamically. AutoDeco (Wang et al., 2025b) trains lightweight controller heads to select decoding parameters per token. Similarly, Dhuliawala et al. (2024) propose a latent preference optimization framework to adjust generation behavior on the fly. These methods typically rely on learned preferences or reward models. In contrast, our decoding adapters are trained solely with task-level correctness signals and condition explicitly on compute budgets.

Inference-Time Decision Making. Other work frames generation as a sequential decision process. Meta-RL approaches (Qu et al., 2025) optimize token-level compute use over multi-step reasoning. Entropy-aware objectives (Cui et al., 2025) preserve exploration during fine-tuning. Unlike these approaches, our decoding policies are trained on frozen models and operate only at inference time. Closely related is (Yan et al., 2025), which adapts reasoning depth based on uncertainty dynamics, and D-LLM (Jiang et al., 2024), which adjusts computation per token. Our method differs by learning *what decoding strategy to use*, rather than *how much compute to spend*.

Critical Tokens and Adaptive Sampling. Recent work identifies "forking" tokens (localized points of uncertainty with high downstream impact (Wang et al., 2025a)) and proposes to adapt generation around them. Our token-level adapter makes such adaptation learnable via reinforcement learning. While some methods (e.g., (Zhao et al., 2025; Fu et al., 2025)) post-process multiple samples to aggregate solutions or early-stop, we focus on decoding-time control without auxiliary reranking.

For additional background on sampling techniques and compute-aware inference, see Appendix A.

6. Conclusion

We framed decoding as a control problem and showed that lightweight policies, trained from scratch with sparse verifiable rewards, can adapt inference behavior under explicit compute budgets. By learning to select among decoding operators at the sequence and token levels, without modifying the underlying language model, our adapters consistently improve reasoning performance on mathematical and coding tasks. At the sequence level, budget-aware strategy selection yields consistent gains across MATH and CodeContests; at the token level, per-step adaptation delivers substantially larger improvements, with Pass@1 gains exceeding 10% over the best static baseline. A key finding is that token-level control is strictly more powerful than sequence-level control: the finer-grained action space enables within-trajectory adjustments that no fixed-per-prompt strategy can replicate.

Our evaluation focused on mathematical and coding reasoning using a single model family (Qwen), and further work is needed to assess whether the observed gains transfer to other domains, tasks, and architectures. Looking forward, the framework naturally extends to richer action spaces (e.g., jointly controlling temperature, top-p, and repetition penalties) and to joint training of the adapter alongside the base model, opening a path toward tighter integration of inference-time control with language model optimization.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Abdin, M., Aneja, J., Behl, H., Bubeck, S., Eldan, R., Gunasekar, S., Harrison, M., Hewett, R. J., Javaheripi, M., Kauffmann, P., et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., Joseph, N., Kadavath, S., Kernion, J., Conerly, T., El-Showk, S., Elhage, N., Hatfield-Dodds, Z., Hernandez, D., Hume, T., Johnston, S., Kravec, S., Lovitt, L., Nanda, N., Olsson, C., Amodei, D., Brown, T., Clark, J., McCandlish, S., Olah, C., Mann, B., and Kaplan, J. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Boiko, D. A., MacKnight, R., and Gomes, G. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023. doi: 10.1038/s41586-023-06792-0.

Borec, L., Sadler, P., and Schlangen, D. The unreasonable ineffectiveness of nucleus sampling on mitigating text memorization. *arXiv preprint arXiv:2408.16345*, 2024.

Chakraborty, S., Bhatt, S., Sehwag, U. M., Ghosal, S. S., Qiu, J., Wang, M., Manocha, D., Huang, F., Koppel, A., and Ganesh, S. Collab: Controlled decoding using mixture of agents for llm alignment. *arXiv preprint arXiv:2503.21720*, 2025.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Cui, G., Zhang, Y., Chen, J., Yuan, L., Wang, Z., Zuo, Y., Li, H., Fan, Y., Chen, H., Chen, W., et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

Dhuliawala, S., Kulikov, I., Yu, P., Celikyilmaz, A., Weston, J., Sukhbaatar, S., and Lanchantin, J. Adaptive decoding via latent preference optimization. *arXiv preprint arXiv:2411.09661*, 2024.

Du, W., Yang, Y., and Welleck, S. Optimizing temperature for language models with multi-sample inference. *arXiv preprint arXiv:2502.05234*, 2025.

Fu, Y., Wang, X., Tian, Y., and Zhao, J. Deep think with confidence. *arXiv preprint arXiv:2508.15260*, 2025.

Grubisic, D., Seeker, V., Synnaeve, G., Leather, H., Mellor-Crummey, J., and Cummins, C. Priority sampling of large language models for compilers. In *Proceedings of the 4th Workshop on Machine Learning and Systems*, pp. 91–97, 2024.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

He, K., Liu, M., Wang, C., Li, Z., Wang, Y., Peng, X., and Zheng, Z. Adadec: Uncertainty-guided adaptive decoding for llm-based code generation. *arXiv preprint arXiv:2506.08980*, 2025.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

Huang, J., Parthasarathi, P., Rezagholizadeh, M., and Chandar, S. Context-aware assistant selection for improved inference acceleration with large language models. *arXiv preprint arXiv:2408.08470*, 2024.

Jiang, Y., Wang, H., Xie, L., Zhao, H., Qian, H., Lui, J., et al. D-llm: A token adaptive computing resource allocation strategy for large language models. *Advances in Neural Information Processing Systems*, 37:1725–1749, 2024.

Li, B., Wang, Y., Lochab, A., Grama, A., and Zhang, R. Cascade reward sampling for efficient decoding-time alignment. *arXiv preprint arXiv:2406.16306*, 2024.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Lago, A. D., Hubert, T., Choy, P., de Masson d'Autume, C., Babuschkin, I., Chen, X., Huang, P.-S., Welbl, J., Gowal, S., Cherepanov, A., Molloy, J., Mankowitz, D. J., Robson, E. S., Kohli, P., de Freitas, N., Kavukcuoglu, K., and Vinyals, O. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://www.science.org/doi/abs/10.1126/science.abq1158.

Lin, Z., Liang, T., Xu, J., Lin, Q., Wang, X., Luo, R., Shi, C., Li, S., Yang, Y., and Tu, Z. Critical tokens matter: Token-level contrastive estimation enhances llm's reasoning capability. *arXiv preprint arXiv:2411.19943*, 2024.

Liu, X., Lei, B., Zhang, R., and Xu, D. D. Adaptive draft-verification for efficient large language model decoding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24668–24676, 2025.

Liu, Z., Zhou, Z., Wang, Y., Yang, C., and Qiao, Y. Inference-time language model alignment via integrated value guidance. *arXiv preprint arXiv:2409.17819*, 2024.

Mavalankar, A., Mansoor, H., Marinho, Z., Samsikova, M., and Schaul, T. Aupair: Golden example pairs for code repair. *arXiv preprint arXiv:2502.18487*, 2025.

Mudgal, S., Lee, J., Ganapathy, H., Li, Y., Wang, T., Huang, Y., Chen, Z., Cheng, H.-T., Collins, M., Strohman, T., et al. Controlled decoding from language models. *arXiv preprint arXiv:2310.17022*, 2023.

Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, 1978.

Nguyen, M., Baker, A., Neo, C., Roush, A., Kirsch, A., and Shwartz-Ziv, R. Turning up the heat: Min-p sampling for creative and coherent llm outputs. *arXiv preprint arXiv:2407.01082*, 2024.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

Princis, H., Sharma, A., and David, C. Treecoder: Systematic exploration and optimisation of decoding and constraints for llm code generation. *arXiv preprint arXiv:2511.22277*, 2025.

Qu, Y., Yang, M. Y., Setlur, A., Tunstall, L., Beeching, E. E., Salakhutdinov, R., and Kumar, A. Optimizing test-time compute via meta reinforcement fine-tuning. *arXiv preprint arXiv:2503.07572*, 2025.

Ren, Z., Shao, Z., Song, J., Xin, H., Wang, H., Zhao, W., Zhang, L., Fu, Z., Zhu, Q., Yang, D., et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Schaeffer, R., Kazdan, J., and Denisov-Blanch, Y. Turning down the heat: A critical analysis of min-p sampling in language models. *arXiv preprint arXiv:2506.13681*, 2025.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

The Ray Team. Ray train: Scalable model training, 2025. URL https://docs.ray.io/en/latest/train/train.html.

von Werra, L., Belkada, Y., Tunstall, L., Beeching, E., Thrush, T., Lambert, N., Huang, S., Rasul, K., and Gallouédec, Q. TRL: Transformers Reinforcement Learning, 2020. URL https://github.com/huggingface/trl.

Wang, S., Yu, L., Gao, C., Zheng, C., Liu, S., Lu, R., Dang, K., Chen, X., Yang, J., Zhang, Z., et al. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*, 2025a.

Wang, Z., Ma, D., Huang, X., Cai, D., Lan, T., Xu, J., Mi, H., Tang, X., and Wang, Y. The end of manual decoding: Towards truly end-to-end language models. *arXiv preprint arXiv:2510.26697*, 2025b.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Yan, H., Xu, F., Xu, R., Li, Y., Zhang, J., Luo, H., Wu, X., Tuan, L. A., Zhao, H., Lin, Q., et al. Mur: Momentum uncertainty guided reasoning for large language models. *arXiv preprint arXiv:2507.14958*, 2025.

Yang, A., Zhang, B., Hui, B., Gao, B., Yu, B., Li, C., Liu, D., Tu, J., Zhou, J., Lin, J., Lu, K., Xue, M., Lin, R., Liu, T., Ren, X., and Zhang, Z. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zhang, S., Bao, Y., and Huang, S. Edt: Improving large language models' generation by entropy-based dynamic temperature sampling. *arXiv preprint arXiv:2403.14541*, 2024.

Zhao, W., Aggarwal, P., Saha, S., Celikyilmaz, A., Weston, J., and Kulikov, I. The majority is not always right: Rl training for solution aggregation. *arXiv preprint arXiv:2509.06870*, 2025.

A. Expanded Related Work

Large language models (LLMs) have made significant progress in reasoning, code generation, and open-ended tasks, but their decoding behavior at inference time remains a blunt instrument: hyperparameters like temperature or top-p are typically fixed per model or dataset. Recent work has begun to explore decoding-time adaptation, dynamic compute allocation, and reward-supervised control. We review these directions below, emphasizing where our approach of learning decoding strategies under budget constraints via verifiable rewards differs.

A.1. Sampling Strategies and Their Limitations

Sampling from the softmax distribution is commonly modified to balance diversity and precision. Top-k, top-p (nucleus), temperature scaling, and min-p sampling are widely used, but typically tuned offline and fixed across prompts. Nucleus sampling Holtzman et al. (2019) avoids low-probability tails, but subsequent work shows its effect on memorization is limited. Borec et al. (2024) find that increasing the nucleus reduces memorization modestly but does not eliminate it. Schaeffer et al. (2025) critique the empirical basis for min-p sampling, arguing that evaluation artifacts obscure its true behavior. Grubisic et al. (2024) propose Priority Sampling, a deterministic variant that avoids duplicate or incoherent outputs caused by stochastic sampling at high temperatures.

These methods reflect the need to modulate decoding behavior, but treat decoding hyperparameters as static knobs. They provide no mechanism for input- or token-specific control, and do not incorporate performance feedback.

A.2. Adaptive Decoding Architectures

Several recent works propose models that adapt decoding behavior based on context. AutoDeco Wang et al. (2025b) adds lightweight controller heads to predict temperature and top-p at each generation step, trained end-to-end with preference modeling. Dhuliawala et al. (2024) introduce a latent preference optimization (LPO) framework where an LLM selects a discrete preference vector to modulate generation style based on prompt features or learned guidance. Du et al. (2025) improve decoding quality by sampling multiple outputs at different temperatures and selecting the best via reranking, optimizing temperature allocation at the example level.

Learned inference-time controllers have also emerged as a flexible way to steer frozen LLMs without fine-tuning. Dhuliawala et al. (2024)'s AdaptiveDecoder head selects sampling temperature token-by-token using preference-trained RL, outperforming fixed-temperature decoding across tasks. Our approach is similar in spirit, but differs by conditioning on compute budgets and using task-verifiable correctness as reward. Collab (Chakraborty et al., 2025) takes this further by learning a token-level Q-learning policy that selects among expert LLMs mid-generation, effectively creating a mixture-of-agents decoding strategy with substantial quality gains. Cascade Reward Sampling (CARDS) (Li et al., 2024) guides a frozen model using a learned reward model that scores and filters sampled segments, avoiding low-reward continuations. Controlled decoding methods further bias token choice using learned value functions: Mudgal et al. (2023) optimize a prefix-level scorer to guide greedy decoding toward high-reward outcomes, while Integrated Value Guidance (Liu et al., 2024) blends implicit and explicit value heads to align frozen models at inference time.

While these controllers improve alignment or output quality, they typically rely on trained reward models, preference supervision, or post-hoc reranking. In contrast, we train decoding policies tabula rasa from correctness-based rewards, using no learned value models and keeping the LLM frozen. Our approach highlights decoding-time stochasticity as a learnable control surface, and shows that generalization to unseen compute budgets is possible with minimal assumptions.

Moreover, we introduce budget-aware decoding: our policies explicitly condition on token or sampling budget and are trained across a range of compute regimes. This enables generalization to unseen inference-time constraints, a dimension not addressed by prior adaptive decoders.

A.3. Inference-Time Decision Making and Compute Allocation

Beyond decoding parameters, recent work explores when and where to allocate compute during generation. Qu et al. (2025) propose meta-RL to optimize token-level compute under regret-based objectives. Cui et al. (2025) design entropy-aware policy updates that avoid overconfident collapse in RL fine-tuning.

Several systems use uncertainty signals to trigger additional reasoning. MUR (Yan et al. (2025)) monitors the momentum of token entropy to decide when to continue or halt reasoning. Jiang et al. (2024) learn to skip transformer layers per token,

reducing inference cost without quality degradation.

These methods focus on *when* and *how much* to compute, but not *how to sample*. Our work is orthogonal and complementary: given a token or prompt, we learn how to sample the next generation step, trading off exploration and determinism under budget.

A.4. Critical Tokens and Token-Level Adaptation

Multiple studies suggest that reasoning failures often hinge on a small set of high-impact tokens. Wang et al. (2025a) identify "forking" tokens with high entropy and high downstream influence. Abdin et al. (2024) define "pivotal tokens" as those with strong local gradients with respect to outcome success.

These findings motivate adaptive sampling that varies across a sequence. AdaDec (He et al., 2025) uses uncertainty spikes to trigger beam-style reranking at critical steps. Unlike such heuristics, our token-level adapter is trained via reinforcement learning to make sampling decisions per step. By observing internal model states and remaining token budget, the adapter learns to allocate stochasticity where it matters.

A.5. Aggregation and Speculative Decoding

Several methods post-process multiple samples to select better outputs. AggLM Zhao et al. (2025) learns a solution aggregator via RL, outperforming simple voting on complex reasoning. Fu et al. (2025) use token probabilities to filter low-quality reasoning paths mid-generation. Speculative decoding (Huang et al., 2024; Liu et al., 2025) accelerates generation via draft-verification, and some variants use contextual bandits to choose draft strategies.

These methods improve sample efficiency or response quality, but require generating multiple candidates and selecting among them. Our approach is distinct: we aim to improve *per-sample* quality by learning how to generate, not just how to rerank. Our adapters operate during generation, not after it.

A.6. Fine tuning LLMs with Reinforcement Learning

Reinforcement learning has emerged as a powerful tool for aligning language models with task objectives or human preferences. In reinforcement learning from human feedback (RLHF), language models are fine-tuned to maximize a learned reward model derived from human preference comparisons, as in InstructGPT (Ouyang et al., 2022) and Anthropic's helpful-harmless framework (Bai et al., 2022). Reinforcement learning from verifiable rewards (RLVR) replaces noisy preference models with automated correctness checks, enabling reward supervision from ground-truth signals in domains such as math and code. Group-Relative Policy Optimization (GRPO) (Shao et al., 2024; Guo et al., 2025) further improves efficiency by using group-normalized Monte Carlo advantages, and has shown strong performance on reasoning tasks. These approaches treat the language model itself as the policy, optimized via PPO-style methods. In contrast, our approach treats the language model as the *environment* and learns a lightweight decoding-time policy that interacts with its sampling behavior. This tabula rasa RL framing enables adaptive inference under compute budgets, without modifying the base model.

A.7. Positioning

Across these lines of work, few methods explicitly frame decoding as a policy learning problem, or incorporate budget constraints as part of the learning signal. Our formulation unifies these axes: we learn decoding-time policies at both the sequence and token level, trained under verifiable rewards and explicit compute budgets, without modifying the base model.

We view this as a complementary direction to prior work: rather than designing or searching over decoding strategies, we make decoding strategy *learnable*, enabling flexible and efficient adaptation across tasks, budgets, and token-level uncertainties.

B. Theoretical Motivation

We establish two structural properties that motivate the adaptive approach: (i) sequence-level adaptation is beneficial whenever the best decoding strategy depends on the prompt, and (ii) token-level control is strictly richer than sequence-level control.

Proposition B.1 (Sequence-level adaptation under prompt heterogeneity)**.** *Let X be a prompt random variable and $A = \{1, \dots, m\}$ a finite set of decoding strategies, with $R(x, a) \in [0, 1]$ the expected terminal reward. Define the best static value $V^*_{\text{stat}} = \max_{a \in A} \mathbb{E}[R(X, a)]$ and the oracle adaptive value $V^*_{\text{adapt}} = \mathbb{E}[\max_{a \in A} R(X, a)]$. Then $V^*_{\text{adapt}} - V^*_{\text{stat}} \geq 0$, with strict inequality whenever the identity of the reward-maximizing action depends on x on a set of positive probability.*

Proof. For each x, $\max_a R(x, a) \geq R(x, \bar{a})$ for any fixed $\bar{a}$. Taking expectations and maximizing over $\bar{a}$ gives $\mathbb{E}[\max_a R(X, a)] \geq \max_a \mathbb{E}[R(X, a)]$. Strictness holds when no single action is pointwise optimal almost surely. □

Proposition B.2 (Token-level control is strictly richer)**.** *Let Π_{seq} be the class of sequence-level policies and Π_{tok} the class of token-level policies. Then $\Pi_{\text{seq}} \subseteq \Pi_{\text{tok}}$, and $\sup_{\pi \in \Pi_{\text{tok}}} J(\pi) \geq \sup_{\pi \in \Pi_{\text{seq}}} J(\pi)$.*

Proof. Any sequence-level policy is represented as a token-level policy that selects the same action at every step regardless of state. □

C. Future Directions

Our work suggests several directions for extending adaptive decoding beyond the setting studied here.

Richer decoding policies. We restrict actions to a small discrete set of decoding operators, but decoding can more generally be viewed as a learnable mapping from the base model distribution to an induced sampling distribution. Moving to more expressive, continuously parameterized decoding transformations would fundamentally change the learning problem, introducing new challenges in optimization and sample efficiency.

Expanded action spaces. The framework naturally supports decoding controls beyond sampling hyperparameters. For example, incorporating early stopping or adaptive termination would allow joint control over stochasticity and generation length under fixed compute budgets. More broadly, decoding adapters could allocate sampling effort within structured inference procedures, such as tree search or other multi-trajectory methods, where actions govern both branching and budget allocation.

Joint training with the base model. We treat the language model as a frozen environment and learn only a lightweight decoding policy. An important direction is to study joint or alternating training of the adapter and the underlying model, which may reduce the burden on the decoding policy by shaping representations to better support adaptive inference.

D. Additional experiments

D.1. Additional Models

In the following, we provide additional experiment with two other models: Qwen3-8B and Qwen2.5-Math-1.5B.

Table 4. **Comparison of sampling strategies on MATH.** against the proposed **sequence**-level sampling adapter, using Qwen3-8B. For the 'w/ CoT mix' configuration, a CoT ratio of 0.3 is applied. All reported values are percentages.

		Static sampling		**Sequence-level adapter**	
Metric	**Setting**	Best	Mixed	w/o budget	w/ budget
Pass@1	w/o CoT	72.86 ± 1.23	72.38 ± 1.24	73.20 ± 1.23	73.67 ± 1.22
	mix CoT	73.02 ± 1.23	72.92 ± 1.23	73.80 ± 1.22	74.07 ± 1.21
Pass@8	w/o CoT	78.38 ±1.14	77.10 ±1.16	78.20 ± 1.14	78.80 ± 1.13
	mix CoT	78.98 ± 1.13	78.58 ± 1.14	79.00 ± 1.13	**79.13** ± 1.13

Table 5. **Comparison of sampling strategies on CodeContests.**: vs. the **sequence**-level sampling adapter under two inference settings (without/with a mix of CoT prompts), using Qwen3-8B. For 'w/ CoT mix', we use a CoT ratio of 0.3.

		Static sampling		Sequence-level adapter	
Metric	**Setting**	Best	Mixed	w/o budget	w/ budget
Pass@1	w/o CoT	12.80 ± 2.39	12.00 ± 2.33	18.80 ± 2.80	17.90 ± 2.74
	mix CoT	19.70 ± 2.85	14.50 ± 2.52	22.20 ± 2.97	23.10 ± 3.02
Pass@8	w/o CoT	23.10 ± 3.02	22.23 ±2.98	26.50 ± 3.16	27.40 ± 3.19
	mix CoT	29.10 ± 3.25	23.90 ± 3.05	32.50 ± 3.35	**34.20** ± 3.40

Table 6. **Comparison of sampling strategies on MATH.** the **sequence**-level sampling adapter under two inference settings (without/with a mix of CoT prompts). 0.3.

		Static sampling		Sequence-level adapter	
Metric	**Setting**	Best	Mixed	Just 4 actions	w/ mix of inference strategy
Pass@1	w/o CoT	61.52 ± 1.35	56.42 ± 1.37	61.52 ± 1.35	62.62 ± 1.34
	mix CoT	62.10 ± 1.34	59.00 ± 1.36	62.76 ± 1.34	**63.20** ± 1.34
Pass@8	w/o CoT	61.52 ± 1.35	66.74 ± 1.31	68.25 ± 1.29	69.66 ± 1.27
	mix CoT	63.00 ± 1.34	67.10 ± 1.30	69.32 ± 1.28	**70.88** ± 1.26

D.2. Generalization

In Table 7, we present the evaluation results on CodeContests of a sequence adapter trained on MATH to validate the out-of-domain generalization ability. We also evaluate the MATH-trained sequence-level adapter on AIME'25 without additional tuning in Table 8; since AIME'25 is small, we run 30 random seeds.

Table 7. Coding: performance comparison of **sequence**-level adapter trained on MATH train versus baseline numbers.

		Static sampling		Adapter
Metric	**Setting**	Best	Mixed	w/o budget
Pass@1	w/o CoT	11.43 ± 2.28	10.53 ± 2.20	12.80 ± 2.39
	mix CoT	13.97 ± 2.48	14.80 ± 2.54	17.90 ± 2.74

Table 8. AIME'25 performance comparison of a **sequence**-level adapter trained on MATH-train vs. numbers reported in the Qwen3-4B technical report (Yang et al., 2025).

Metric	**Setting**	**Reported**	**Adapter** (w/ budget)
Pass@1	non-thinking	19.1	20.1 ± 2.62
	thinking	65.6	71.1 ± 2.96

D.3. Mixed-Domain Training

Table 9 reports results when training a single sequence-level adapter jointly on math and coding data (with balanced sampling during training) and evaluating on each domain's original validation distribution. The adapter continues to improve over static baselines on both domains, though gains are smaller than in single-domain training, consistent with the increased diversity of the joint task distribution.

D.4. Qualitative Results

This appendix provides qualitative evidence about how the learned adapter behaves at the token level. Our goal here is *not* to reverse-engineer a complete, human-interpretable rule for the policy, but to (1) sanity-check that the adapter meaningfully

Table 9. Mixed training of math and coding: comparison of static sampling strategies vs. the **sequence**-level sampling adapter. We use **Code** to refer to CodeContests in this table.

Metric	Setting	Static sampling				Sequence-level adapter			
		Best		Mixed		w/o budget		w/ budget	
		MATH	**Code**	**MATH**	**Code**	**MATH**	**Code**	**MATH**	**Code**
Pass@1	w/o CoT	72.00 ± 3.18	11.80 ± 2.39	71.10 ± 3.17	10.30 ± 2.18	73.55 ± 3.12	13.70 ± 2.46	**74.20** ± 3.13	13.70 ± 2.46
	Δ Abs. (Rel.)					↑ 1.55 (2.15%)	↑ 1.90 (16.10%)	↑ 2.20 (3.06%)	↑ 1.90 (16.10%)
	mix CoT	73.80 ± 3.15	17.10 ± 2.69	73.30 ± 3.17	14.50 ± 2.52	74.80 ± 3.11	**18.50**± 2.89	74.70 ± 3.11	18.20 ± 2.76
	Δ Abs. (Rel.)					↑1.00 (1.36%)	↑ 1.40 (8.19%)	↑ 0.90 (1.22%)	↑ 1.10 (6.43%)
Pass@8	w/o CoT	76.30 ± 3.04	14.50 ± 2.52	75.70 ± 3.07	14.50 ± 2.52	77.70 ± 2.97	17.90 ± 2.74	78.00 ± 2.96	19.70 ± 2.85
	Δ Abs. (Rel.)					↑ 0.70 (0.92%)	↑ 4.50 (31.03%)	↑ 1.70 (2.23%)	↑ 5.20 (35.86%)
	mix CoT	76.70 ± 2.98	22.20 ± 2.97	77.10 ± 2.98	18.80 ± 2.80	77.70 ± 2.93	**24.80** ± 3.09	**78.10** ± 2.96	23.10 ± 3.02
	Δ Abs. (Rel.)					↑ 0.60 (0.78%)	↑ 2.60 (11.71%)	↑ 1.00 (1.30%)	↑ 0.90 (4.05%)

changes token-level stochasticity relative to static decoding, and (2) probe whether a small set of simple heuristics (e.g., entropy-thresholding) could plausibly explain the gains. Overall, we observe consistent trends (e.g., differential treatment of low- vs. high-entropy tokens), but we do not find a single low-dimensional qualitative pattern that fully characterizes the learned behavior.

Entropy modulation. Figure 3 compares token entropy distributions under static decoding and the learned token-level adapter. A consistent qualitative trend is that the adapter collapses *low-entropy* tokens to near-deterministic behavior more often, while *high-entropy* tokens retain stochasticity more frequently. Importantly, this trend should be interpreted as a correlational signature of the learned policy rather than a complete explanation of its success: in the main paper we show that an entropy-only observation is insufficient to match the full adapter's performance.

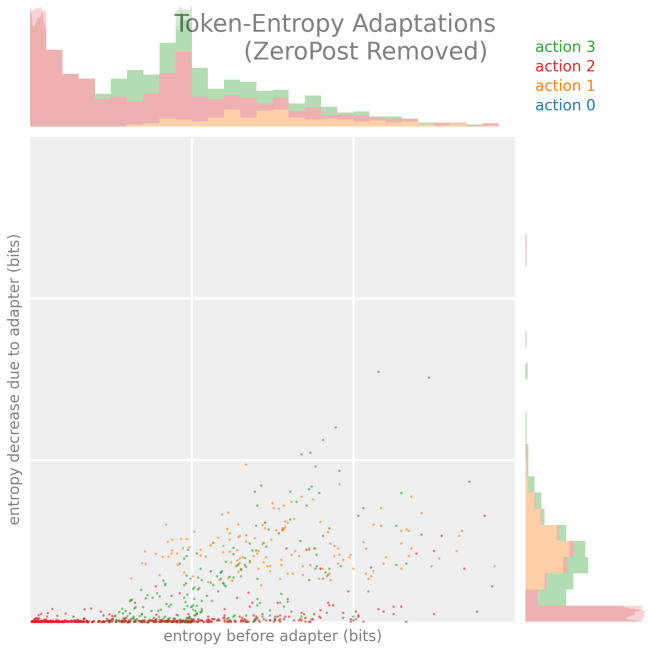


Figure 3. **Entropy modulation under token-level control.** Token entropy distributions for a representative static strategy versus the learned token-level adapter, using the temperature-only action set from Table 14. Action 0 is greedy; larger action indices correspond to higher temperature. Qualitatively, the adapter more often preserves stochasticity on higher-entropy tokens, while collapsing many low-entropy tokens to near-deterministic behavior.

Generation length statistics. Figure 4 reports generation-length distributions for a representative static strategy and for the learned sequence- and token-level adapters. We include this as a coarse behavioral check: large performance gains can

(a) Static decoding (representative strategy). *(b)* Sequence-level adapter.

Figure 4. **Generation length distributions.** Generation length statistics for a representative static strategy and the learned adapters. Length shifts are present but not extreme, suggesting the token-level gains are not explained solely by systematic truncation/verbosity changes.

sometimes be confounded by systematic changes in generation length (e.g., consistently shorter outputs). Here, lengths shift only moderately, suggesting the gains are not solely a trivial artifact of length.

Example generation (action traces). To make token-level control concrete, we show an example generation with tokens colored by the *current* selected action (Section D.4) and the *next* action (Section D.4). Black indicates tokens with very low entropy, where the choice of decoding action rarely changes the sampled token; other colors correspond to the four actions in Table 14. While these examples illustrate that the policy switches actions within a single trajectory, we did not find a simple, global rule (e.g., a single entropy threshold) that explains when switches occur across prompts.

Example Generation (Current Action)

To determine how many seconds it takes for the plane to reach an altitude of 12,000 feet, we need to analyze the pattern of its climbing. The plane climbs 100 feet in the first second, 200 feet in the second second, 300 feet in the third second, and so on. This forms an arithmetic series where the first term \(a_1\) is 100 feet and the common difference \(d\) is also 100 feet.

The total altitude climbed after \(n\) seconds is the sum of the first\(n\)terms of this arithmetic series. The sum \(S_n\)ofthefirst\(n\)terms of an arithmetic series is given by the formula: \[S_n=\frac{n}{2}(2a_1+(n-1)d)\]

Substituting the given values\(a_1=100\) and \(d=100\), we get: \[S_n = \frac{n}{2}(2\cdot100+(n-1)\cdot100) = \frac{n}{2}(200+100(n-1)) = \frac{n}{2}(100n+100)=50n(n+1)\]

We need to find \(n\) such that\(S_n=12000\): \[50n(n+1)=12000\] \[n(n+1)=\frac{12000}{50}=240\]

Now, we need to solve the quadratic equation \(n\^2+n-240 = 0\). We can use the quadratic formula\(n=\frac{-b\pm\sqrt{b\^2-4ac}}{2a}\), where \(a=1\),\(b = 1\),and\(c=-240\): \[n=\frac{-1\pm\sqrt{1\^2-4\cdot1\cdot(-240)}}{2\cdot1} = \frac{-1\pm\sqrt{1+960}}{2} = \frac{-1\pm\sqrt{961}}{2} = \frac{-1\pm31}{2}\]

This gives us two solutions:
\[n=\frac{30}{2}=15\]
\[n=\frac{-32}{2}=-16\]

Since \(n\) must be a positive integer, we discard the negative solution. Therefore, \(n=15\).

To verify, we calculate the sum of the first 15 terms of the series: \[S_{15} = 50\cdot15\cdot16 = 12000\]

Thus, it takes the plane \(\boxed{15}\

Example Generation (Next Action)

To determine how many seconds it takes for the plane to reach an altitude of 12,000 feet, we need to analyze the pattern of its climbing. The plane climbs 100 feet in the first second, 200 feet in the second second, 300 feet in thethirdsecond, and so on. This

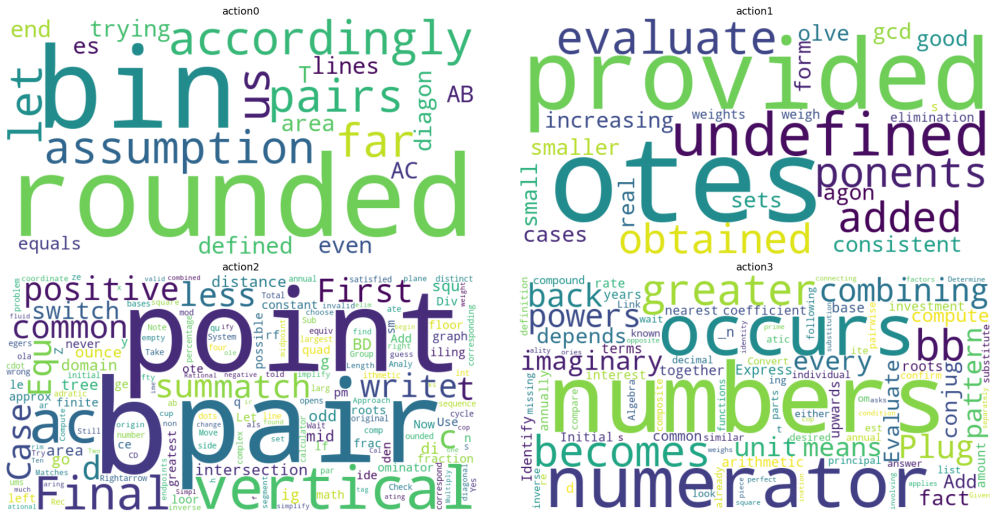


Figure 5. **Word clouds by action.** Most frequent tokens under each temperature-based action from Table 14. This is a lightweight diagnostic and does not by itself explain the policy's performance improvements.

forms an arithmetic series where the first term \(a_1\) is 100 feet and the common difference \(d\) is also 100 feet.

The total altitude climbed after\(n\) seconds is the sum of the first \(n\) terms of this arithmetic series. The sum \(S_n\) of the first \(n\) terms of an arithmetic series is given by the formula: \[S_n=\frac{n}{2}(2a_1+(n-1)d)\]

Substituting the given values \(a_1=100\) and \(d = 100\), we get: \[S_n = \frac{n}{2}(2\cdot100+(n-1)\cdot100) = \frac{n}{2}(200+100(n-1))=\frac{n}{2}(100n+100) = 50n(n+1)\]

We need to find \(n\) such that \(S_n = 12000\): \[50n(n+1)=12000\] \[n(n+1) = \frac{12000}{50} = 240\]

Now, we need to solve the quadratic equation
\(n\^2+n-240 = 0\).
We can use the quadratic formula
\(n = \frac{-b\pm\sqrt{b\^ 2-4ac}}{2a}\),
where \(a = 1\),\(b=1\), and\(c=-240\): \[n = \frac{-1\pm\sqrt{1\^2-4\cdot1\cdot(-240)}}{2 \cdot 1} = \frac{-1\pm\sqrt{1+960}}{2}=\frac{-1\pm\sqrt{961}}{2} = \frac{-1\pm31}{2}\]

This gives us two solutions:
\[n=\frac{30}{2}=15\]
\[n = \frac{-32}{2} = -16\]

Since\(n\) must be a positive integer, we discard the negative solution. Therefore, \(n=15\).

To verify, we calculate the sum of the first 15 terms of the series: \[S_{15} = 50 \cdot 15 \cdot 16 = 12000\]

Thus, it takes the plane \(\boxed{15}\

Token content (word clouds). Figure 5 visualizes frequent tokens under each action. We view this as an exploratory diagnostic rather than strong evidence: word clouds can highlight obvious stylistic differences, but are insensitive to context and can be dominated by generic tokens. In our runs, these visualizations did not reveal a clean, human-interpretable separation between actions beyond coarse stylistic effects.

Limitations. These qualitative analyses are intended to provide intuition and sanity checks, not a definitive mechanistic account of the learned policy. In particular, the entropy trend in Fig. 3 is consistent with adaptive modulation of stochasticity, but the main paper's entropy-only ablation indicates that richer contextual information is required to recover the full gains.

E. Action Space Selection

This appendix describes the procedure used to construct a finite action space for the decoding adapters, along with empirical details of the selection process and the final strategies used in our experiments.

Candidate action pool. Let $\mathcal{S}$ denote a finite set of candidate decoding strategies. Each strategy $s \in \mathcal{S}$ corresponds to a fixed configuration of decoding hyperparameters (e.g., temperature, top-k, top-p, min-p). In our experiments, $\mathcal{S}$ is formed by the Cartesian product of the parameter ranges shown in Table 10, yielding a total of $5 \times 4 \times 3 \times 3 = 180$ candidate strategies. This pool is designed to span a broad range of decoding behaviors, from near-greedy to highly stochastic sampling.

Table 10. Sweep ranges of as the action space for our sampling adapter.

Parameter	Value
Temperature	$[0.3, 0.5, 0.75, 1.0, 1.25]$
Top-k	$[5, 10, 50,$ "off"$]$
Top-p	$[0.9, 0.95,$ "off"$]$
Min-p	$[0.1, 0.2,$ "off"$]$

Reward model. Let $D = \{x_i\}_{i=1}^{N}$ denote a validation dataset. For each input x_i and strategy s, we compute a verifiable terminal reward $r(x_i, s) \in \{0, 1\}$ indicating task success (e.g., correctness of the generated solution). We define the empirical performance of a strategy as

$$Q(s) = \frac{1}{N} \sum_{i=1}^{N} r(x_i, s).$$

Objective. Our goal is not to identify a single best decoding strategy, but rather to select a small subset $S \subseteq \mathcal{S}$ such that, for each input, at least one strategy in S performs well. Accordingly, we consider the following coverage-based objective:

$$F(S) = \frac{1}{N} \sum_{i=1}^{N} \max_{s \in S} r(x_i, s),$$

which corresponds to the expected performance of a best-of-S decoder.

Greedy selection. Directly optimizing $F(S)$ is combinatorial. Following prior work on coverage-based selection (Mavalankar et al., 2025), we construct S using a greedy procedure: starting from $S_0 = \emptyset$, at each step we add

$$s^* = \arg \max_{s \in \mathcal{S} \setminus S_k} \left(F(S_k \cup \{s\}) - F(S_k) \right),$$

until a desired cardinality $|S| = K$ is reached. In practice, K is chosen to balance expressivity of the action space against training stability of the downstream policy.

Empirical selection behavior. Figure 6 compares greedy coverage-based selection against a baseline that selects the top-K strategies by average performance $Q(s)$. Greedy selection consistently achieves higher best-of-K accuracy, indicating that it favors complementary strategies that succeed on different subsets of inputs rather than redundant high-average strategies. This behavior motivates the use of greedy coverage as a principled method for constructing a compact but expressive action space.

Selected strategies. Tables 13 and 14 report the final sets of decoding strategies selected by the greedy procedure for the sequence-level and token-level experiments, respectively. At the sequence level, the selected strategies span a range of sampling behaviors and include both stochastic and near-greedy decoding. At the token level, the selected actions differ in temperature, reflecting the exploratory observation that stochasticity is the dominant axis along which complementary decoding behaviors emerge.

Below we provide the best set of strategies for math, coding, and the mixture of math and coding evaluation sets.

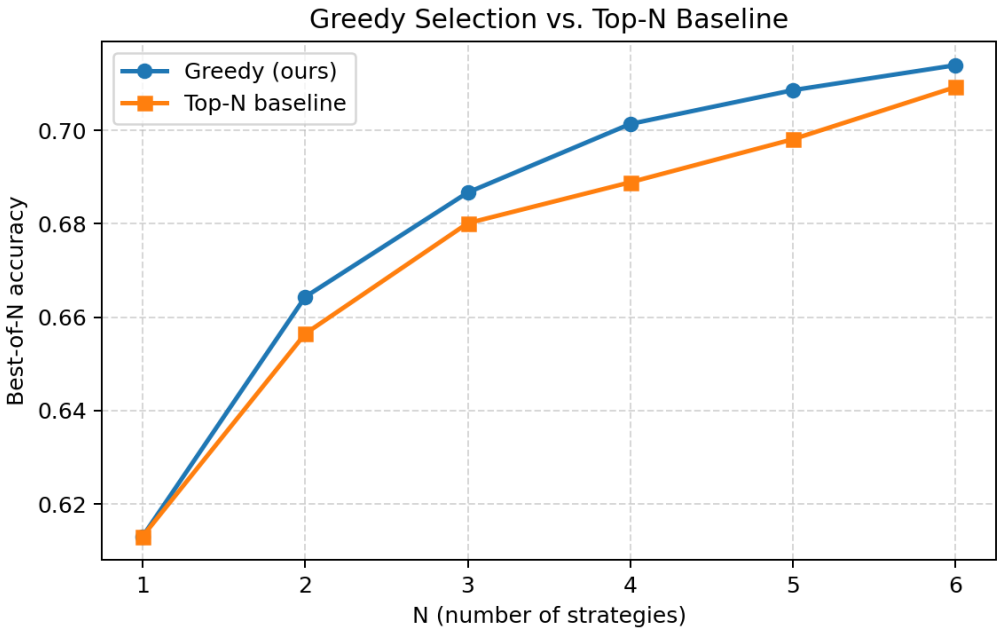


Figure 6. Best of N accuracy vs number of selected actions for greedy selection (our approach) vs the baseline of taking the top N actions.

Table 11. Selected decoding strategies for **math** tasks.

Strategy	**Parameters**				
	do_sample	temperature	top_k	top_p	min_p
1	true	0.75	–	0.9	0.1
2	true	1.0	–	0.9	–
3	true	1.25	10	0.9	–
4	true	1.0	5	0.9	–
5	true	1.25	50	0.9	–
6	true	1.25	–	–	–

Table 12. Selected decoding strategies for **coding** tasks.

Strategy	**Parameters**				
	do_sample	temperature	top_k	top_p	min_p
1	true	1.25	–	0.95	0.1
2	true	1.25	–	0.95	0.2
3	true	1.0	–	0.95	0.2
4	true	1.25	50	0.9	0.1
5	true	1.25	5	0.9	–
6	true	0.75	5	–	0.1

Discussion. Although the candidate pool includes combinations of multiple sampling parameters, the greedy procedure tends to select strategies that differ primarily along a single axis of stochasticity. This observation motivates our use of temperature-based actions in the token-level experiments, while retaining a more general action space at the sequence level. We emphasize that this selection procedure is used only to define a finite action space prior to training; during policy learning, the adapter receives only task-level rewards and does not observe per-strategy performance estimates.

Table 13. Selected decoding strategies for a mix of **math and coding** tasks.

Strategy	Parameters				
	do_sample	temperature	top_k	top_p	min_p
0	true	0.5	–	–	–
1	true	1.0	–	–	–
2	true	1.25	–	–	–
3	true	0.75	10	–	–
4	true	0.75	10	0.95	0.1
5	false	0.0	–	–	–

Table 14. Selected decoding strategies for token-level **math** tasks.

Strategy	Parameters	
	do_sample	temperature
0	false	0
1	true	0.5
2	true	1.0
3	true	1.25

F. Implementation Details

F.1. Hyperparameters

In this section, we provide details on the implementation of our adapters.

Table 15. Parameters of **token**-level sampling adapter.

Parameter	Value
Architecture	MLP
Number of Layers	3
Activation	SiLU
Dropout	0.1
Optimizer	Adam
format	bf16
Learning rate scheduler	exponential

We provide pseudocode for our sequence-level (Algorithm 1) and token-level (Algorithm 2) Adapters below. For Sequence-Level Adapters, We use vLLM engine to fastly generate rollout. For token-Level, We use Ray Train (The Ray Team, 2025) and distributed data parallelism (DDP). We use past_key_values as an incremental KV-cache returned by the model each step, pass it back into the next forward for speed, and whenever some sequences finish early we "shrink" the cache to only the still-active batch rows via index_select so it stays aligned with active_idx.

F.2. Prompts for Qwen3 Models

MATH.

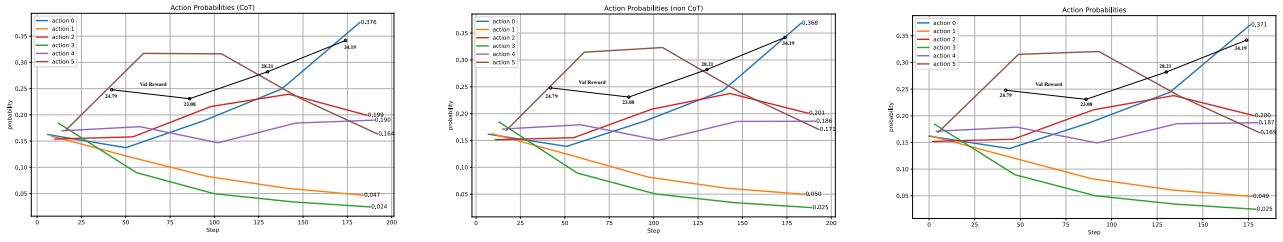
- **Without thinking:** "Provide the final answer within \\boxed{}. /nothink"
- **With thinking:** "Provide the final answer within \\boxed{}. /think"

CodeContests.

- **Without thinking:** "You are a coding assistant. /nothink"
- **With thinking:** "You are a coding assistant. Please reason step by step. /think"

Algorithm 1 Sequence-Level Decoding Adapter

function SEQFORWARD($\mathbf{e}, \mathbf{a}, T$)
  if a $\neq \varnothing$ **then**
    z $\leftarrow$ [e $\|$ EmbedAux(a)]
  else
    z $\leftarrow$ e
  $\boldsymbol{\ell} \leftarrow \mathrm{MLP}(\mathrm{z})$ ▷ logits over strategies
  $\mathbf{p} \leftarrow \mathrm{softmax}(\boldsymbol{\ell}/T)$
  return $\mathbf{p}$
function SEQSELECTSTRATEGY($\mathbf{e}, \mathbf{a}, T$, deterministic)
  $\mathbf{p} \leftarrow$ SEQFORWARD($\mathbf{e}, \mathbf{a}, T$)
  if deterministic **then**
    $s \leftarrow \arg\max \mathbf{p}$
  else
    $s \sim \mathrm{Categorical}(\mathbf{p})$
  return $s, \mathbf{p}$

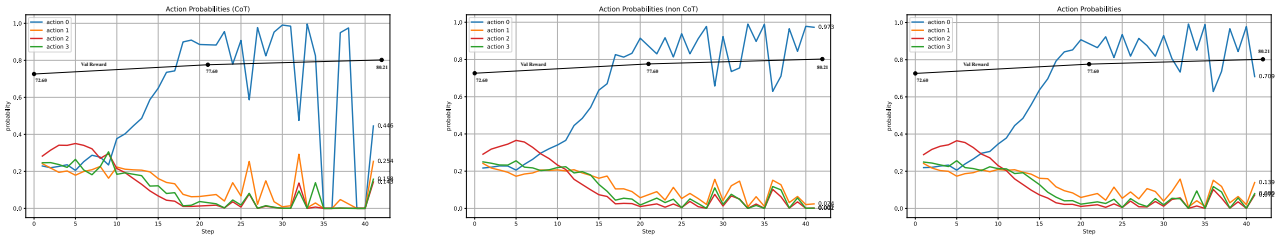


(a) CoT action distribution *(b)* Non-CoT action distribution *(c)* Overall action distribution

Figure 7. **Action distributions and val reward on CodeContests** for sampling strategies with sequence-level adapter. The actions are defined in table 12. Axes are same as in Figure 2.

Algorithm 2 Token-Level Decoding Adapter

function TOKFORWARD($\mathbf{x}_t, T$)
  $\boldsymbol{\ell}_t \leftarrow \mathrm{MLP}(\mathbf{x}_t)$ ▷ logits over strategies
  $\mathbf{p}_t \leftarrow \mathrm{softmax}(\boldsymbol{\ell}_t/T)$
  return $\mathbf{p}_t$
function TOKSELECTSTRATEGY($\mathbf{x}_t, T$, deterministic)
  $\mathbf{p}_t \leftarrow$ TOKFORWARD($\mathbf{x}_t, T$)
  if deterministic **then**
    $s_t \leftarrow \arg\max \mathbf{p}_t$
  else
    $s_t \sim \mathrm{Categorical}(\mathbf{p}_t)$
  return $s_t, \mathbf{p}_t$
function TOKFEATURES(LLM, prefix, t)
  $\mathbf{h}_t \leftarrow \mathrm{LastHidden}(\mathrm{LLM}(\mathrm{prefix}))$
  return $\mathbf{x}_t \leftarrow [\,\mathbf{h}_t \,\|\, \phi(t)\,]$ ▷ optional aux/budget features
function TOKSTEP(LLM, prefix, transforms, T, deterministic)
  $\mathbf{x}_t \leftarrow$ TOKFEATURES(LLM, prefix, t)
  $(s_t, \mathbf{p}_t) \leftarrow$ TOKSELECTSTRATEGY($\mathbf{x}_t, T$, deterministic)
  $y_t \leftarrow \mathrm{SampleToken}(\mathrm{LLM}, \mathrm{prefix}; \mathrm{transforms}[s_t])$
  return $y_t, s_t, \mathbf{p}_t$



(a) CoT action distribution *(b)* Non-CoT action distribution *(c)* Overall action distribution

Figure 8. **Action distributions and val reward on MATH** for sampling strategies with the **token**-level adapter. Actions are defined in table 11. Axes match fig. 2. Note that a CoT action probability of 0 indicates that no CoT prompt was sampled in that batch.