AutoCRAT: Within-trajectory Joint Control of Stochasticity and Compute for LLM Reasoning

Anonymous ACL submission

Abstract

Large language models (LLMs) achieve strong reasoning performance, which depends critically on inference-time decisions. Yet these decisions are commonly handled by static, one-size-fits-all policies, limiting adaptation to diverse tasks and reasoning stages. Recent adaptive methods partially address this limitation, but they primarily adapt either decoding stochasticity (how the model explores) or reasoning compute (how long the model reasons) in isolation, leaving their interaction within a single reasoning trajectory unmodeled. To address this challenge, we shift toward a **within-trajectory joint control** view, and instantiate it in **AutoCRAT**, a decoder-side controller for frozen backbones. Using only signals available during decoding, **AutoCRAT** jointly adjusts *sampling stochasticity* and *reasoning budget* during generation. **AutoCRAT** operates over a discrete action space and updates control decisions only at semantic boundaries, improving stability while remaining responsive to the evolving reasoning process. Comprehensive evaluation across **6** benchmarks demonstrates that **AutoCRAT** **(I)** uses $13.8 \sim 52.7\%$ fewer inference tokens on average than recommended static configurations, **(II)** surpasses recommended static and adaptive single-axis baselines by $1.5 \sim 4.5\%$ in relative accuracy, and **(III)** enjoys strong cross-backbone transferability. Our work is openly accessible.

1 Introduction

Large language models (LLMs) have demonstrated strong performance on complex reasoning tasks such as mathematics and code generation (Chen et al., 2025b). In practice, their reasoning performance depends not only on the underlying capabilities but also heavily on inference-time configurations, including decoding stochasticity (e.g., temperature, top-p) and reasoning compute (e.g., reasoning depth and allocation of test-time compute), which are often manually pre-specified (Ji et al., 2025). Even when adjustments are available, they are primarily limited to per-request configurations fixed before generation begins, or rely on external human intervention, without a mechanism to adapt as the optimal configurations evolve within a single trajectory. For instance, early stages may require broader exploration, while later stages require consolidation and answer commitment. Consequently, such fixed configurations are often ill-suited to the varying needs across tasks and reasoning stages (Parashar et al., 2025).

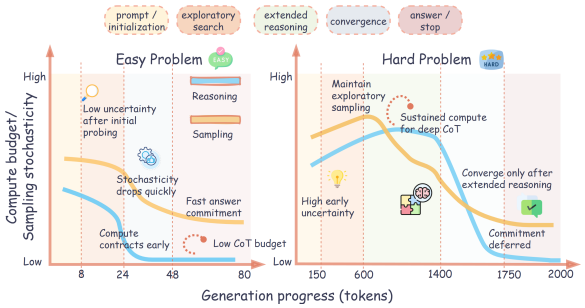

Figure 1: Existing methods adapt only part of the reasoning process, while the optimal workflow requires joint, within-trajectory coordination across stages.

Recent work has increasingly explored adaptive mechanisms for calibrating these factors during reasoning (Alomrani et al., 2025). One line of work adjusts decoding stochasticity through dynamic temperature scheduling or sampling adjustments (Zhu et al., 2024); another line adjusts reasoning compute through adaptive thinking length, dynamic budget allocation, early stopping, or self-correction (Pu et al., 2025; Wu et al., 2025b). However, prior studies suggest that these two dimensions interact and should be coordinated jointly (Wu et al., 2024), as demonstrated in Figure 1. Yet existing methods still lack a unified within-trajectory control perspective that jointly models their interaction.

To address this gap, we propose **AutoCRAT**, a decoder-side controller that performs joint within-

trajectory coordination. Technically, we build **AutoCRAT** on a two-dimensional stochasticity-compute control view. It simultaneously regulates sampling stochasticity and reasoning budget as the generation unfolds. **AutoCRAT** operates solely on signals observable during decoding and its own control state, without accessing hidden states of backbones, which supports transferability across models and interfaces (Zhang et al., 2026). To balance adaptivity and stability in this joint control setting, we introduce two designs: (i) control decisions are made over a discrete action space rather than continuous values, and (ii) are updated only at natural semantic boundaries (e.g., sentence endings, step markers), avoiding the instability of token-level switching while remaining more responsive than per-request configuration (Yu et al., 2025).

We conduct comprehensive evaluations on **6** widely adopted benchmarks, covering diverse use cases in code generation, mathematical reasoning, and challenging QA. Empirical results demonstrate that **AutoCRAT** is **(I) token-economical**, using $13.8 \sim 52.7\%$ fewer inference tokens on average than recommended static configurations; **(II) high-performing**, surpassing recommended static and adaptive baselines by $1.5 \sim 4.5\%$ in relative accuracy; **(III) transferable** across LLM-backbones; **(IV) insightful**, demonstrating the significance of a unified view of LLM reasoning.

Our contributions are summarized as follows:

❶ **Unified View.** We introduce a unified view of inference-time adaptation for LLM reasoning, arguing that stochasticity and compute should be coordinated within a trajectory rather than adjusted in isolation or only at the request level.

❷ **Practical Solution.** We propose **AutoCRAT**, a decoder-side controller for frozen backbones that jointly coordinates sampling stochasticity and reasoning budget. Its discrete action design and boundary-aware updates balance adaptivity, stability, and transferability.

❸ **Experimental Evaluation.** Extensive experiments on **6** benchmarks show that **AutoCRAT** achieves a substantially better accuracy–compute tradeoff than baselines, exhibits strong transferability, and provides insights into the dynamics of LLM reasoning control.

2 Related Work

Decoding Stochasticity Adaptation. Decoding stochasticity is a central determinant of LLM behavior because sampling controls the tradeoff between exploration and exploitation during generation. Classical studies showed that decoding choices crucially affect degeneration and sample quality (Holtzman et al., 2020; Shi et al., 2024). Later analyses further indicate that the effect of temperature on downstream reasoning is task-dependent rather than monotonic, making a single global configuration difficult to justify across diverse reasoning workloads (Renze, 2024; Salah and Muneer, 2026). Motivated by this sensitivity, recent work has moved beyond fixed per-request sampling rules toward adaptive stochasticity control. Representative directions include entropy-aware temperature adjustment (Zhang et al., 2024), confidence- or entropy-based candidate-set adaptation (Ravfogel et al., 2023), selective temperature control for multi-sample inference (Du et al., 2025; Troshin et al., 2025), and control-theoretic decoding in pre-logit space (Kanai et al., 2025).

Reasoning Compute Control. Reasoning compute is another major inference-time factor. Chain-of-thought (Wei et al., 2022), self-consistency (Wang et al., 2023), and tree-structured deliberation (Yao et al., 2023) established that allocating extra reasoning steps or sampled reasoning paths can substantially improve model performance. Recent models and scaling analyses further frame test-time compute as a primary lever for performance, especially on mathematics, code, and other reasoning-intensive domains (OpenAI, 2024; Snell et al., 2024). At the same time, a growing body of work from both academia and industry shows that longer reasoning is not uniformly better. Answers often converge before generation ends, and excessive deliberation can introduce redundancy or overthinking (Liu and Wang, 2025; Wang et al., 2026; OpenAI, 2025; Li et al., 2025). In response, existing methods study adaptive budget allocation (Lin et al., 2026), early stopping based on convergence or confidence signals (Yang et al., 2025b), and extra self-correction/refinement passes when additional reasoning appears useful (Madaan et al., 2023; Weng et al., 2023).

Toward Within-Trajectory Joint Control. While the above methods adapt a single inference-time dimension, recent work has begun to jointly adapt both. AdaReasoner (Wang et al., 2025a), for example, selects question-tailored reasoning instruction format, temperature, and reasoning steps, while task-level inference optimization methods such as EcoTune (Xu et al., 2025) jointly tune hyperparam-

eters like temperature and maximum output length. However, these methods still operate mainly at the request level. The controller chooses a configuration before generation, and the full rollout then follows that choice, so the trajectory itself remains largely uncontrolled. Recent analyses suggest that reasoning trajectories exhibit identifiable latent states and transition patterns rather than uniform behavior throughout a rollout (Sun et al., 2026), that temporal confidence and entropy evolution within a chain carry richer diagnostic information than static summaries (Zhu et al., 2026), and that reasoning models often signal answer readiness well before they actually stop (Sharma and Chopra, 2025). Other studies show that answer convergence and overthinking typically emerge only after partial progress has already been made, pointing to the importance of stage-sensitive intervention rather than one-shot configuration selection (Deng et al., 2023; Chen et al., 2025a). Therefore, within-trajectory control is not merely a finer implementation, but a different granularity of inference-time decision. These observations motivate inference-time controllers that operate within a single reasoning trajectory, dynamically coordinating sampling stochasticity and reasoning budget.

3 Within-Trajectory Joint Control

Definition. We adopt the standard view of LLM reasoning as an autoregressive generation process that produces a token sequence $\mathbf{x} = (x_1, x_2, \ldots, x_T)$, where each token is sampled via

$$x_t \sim p_\theta(\cdot \mid x_{<t};\, \mathbf{c}_{s,t},\, \mathbf{c}_{c,t})\,, \tag{1}$$

with $\mathbf{c}_{s,t}$ and $\mathbf{c}_{c,t}$ denoting the control state at step t along two dimensions: decoding stochasticity ($\mathbf{c}_{s,t}$), controlling *how* the next token is drawn by parameterizing the shape of the sampling distribution, and reasoning compute ($\mathbf{c}_{c,t}$), controlling *whether* generation proceeds by determining the continuation condition $\mathbb{1}[t < \tau]$ with τ the stopping time.

Let $\mathbf{c}_t \triangleq (\mathbf{c}_{s,t},\, \mathbf{c}_{c,t})$ collect both components. A common inference-time practice is to operate at the *request level*: $\mathbf{c}_t$ is fixed before generation begins and remains constant throughout the entire rollout. We instead define **within-trajectory control** as a regime in which $\mathbf{c}_t$ is revised at a set of decision points $\mathcal{B} = \{t_1, t_2, \ldots\} \subset \{1, \ldots, T\}$ during generation, via

$$\mathbf{c}_{t_k} = f\big(x_{<t_k};\, \mathbf{c}_{t_{k-1}}\big)\,, \quad t_k \in \mathcal{B}, \tag{2}$$

while remaining unchanged between consecutive decision points. The control is *joint* in that the same controller f updates both components of $\mathbf{c}_{t_k}$ simultaneously at each decision point, rather than selecting them independently or fixing one while adapting the other.

Control space completeness. We argue that our two-dimensional decomposition of $\mathbf{c}_t$ is not arbitrary. In the within-trajectory setting of LLM inference, it exhausts all control degrees of freedom. At every step t, the only degrees of freedom available to a controller are intervention on the shape of the sampling distribution ($\mathbf{c}_{s,t}$) and continuation condition ($\mathbf{c}_{c,t}$). Any concrete control parameter, including temperature, top-p, repetition penalties, budget limits, or early-stopping signals, instantiates one of these two components. No third structural class exists within a single evolving trajectory.

We note that multi-trajectory methods such as beam search or best-of-n sampling do not constitute counterexamples (Freitag and Al-Onaizan, 2017; Wang et al., 2025b). They alter the inference topology by maintaining a set of concurrent trajectories, and therefore fall outside the *single-trajectory* scope that defines our setting. A detailed argument for this completeness claim is provided in Appendix A.

4 AutoCRAT

Figure 2 illustrates the architecture of our method. As **the first within-trajectory joint control framework**, **`AutoCRAT`** attaches to a frozen backbone without modifying its parameters or accessing its hidden states. A lightweight *control head* maps decoder-side observables and previous control states to two discrete outputs, a *sampling level* for sampling stochasticity and a *budget level* for reasoning budget, which are updated jointly at semantic boundaries as generation unfolds. Section 4.1 details the control head design and action space. Section 4.2 describes the boundary-aware update mechanism. Section 4.3 presents the training procedure.

4.1 Control Head and Action Space

The control head h_ϕ is the decision-making component of **`AutoCRAT`**, implementing the control update in Eq. (2). We denote the decoder-side observables at the decision point $t_k \in \mathcal{B}$ by $\mathbf{o}_{t_k}$, i.e., features derived from the decoding process without accessing backbone hidden states. The full specification is

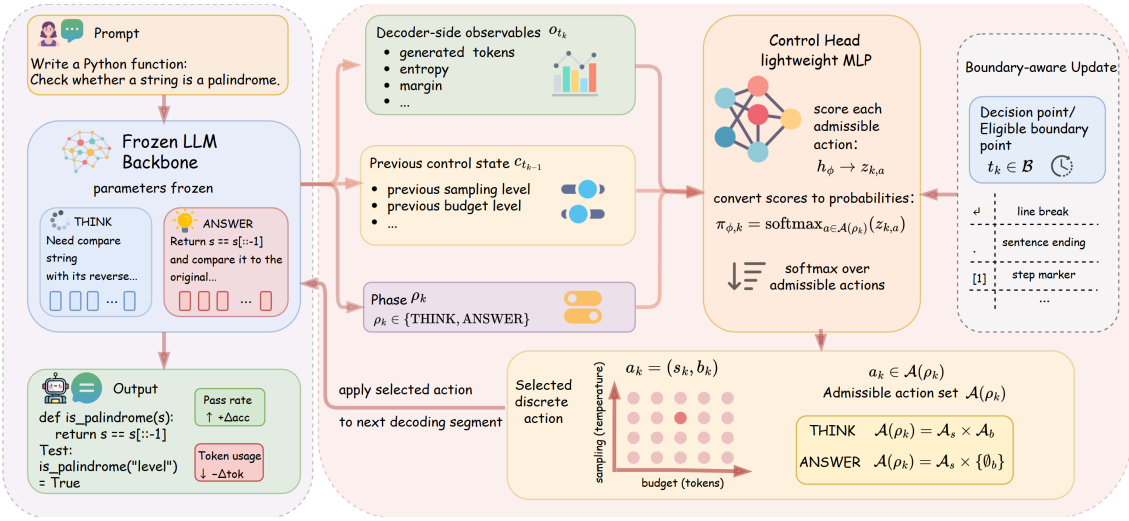

Figure 2: Overview of **AutoCRAT**.

provided in Appendix B. At each t_k, h_ϕ maps $\mathbf{o}_{t_k}$ and the previous control state $\mathbf{c}_{t_{k-1}}$ to a categorical policy $\pi_{\phi,k}$ over control actions:

$$h_\phi(\mathbf{o}_{t_k},\, \mathbf{c}_{t_{k-1}},\, \rho_k) = \pi_{\phi,k}, \tag{3}$$

where $\rho_k \in \{\text{THINK}, \text{ANSWER}\}$ denotes the current phase, and $\pi_{\phi,k}$ is a distribution over admissible control actions at t_k. $\mathbf{c}_{t_{k-1}}$ encodes the sampling and budget levels active at the preceding decision point, represented as concatenated one-hot vectors. An admissible action is written as $a_k = (s_k, b_k)$, where s_k is the *sampling level* instantiating $\mathbf{c}_{s,t_k}$, and b_k is the *budget level* instantiating $\mathbf{c}_{c,t_k}$. The action set is defined as:

$$\mathcal{A}(\rho_k) = \begin{cases} \mathcal{A}_s \times \mathcal{A}_b, & \rho_k = \text{THINK}, \\ \mathcal{A}_s \times \{\varnothing_b\}, & \rho_k = \text{ANSWER}, \end{cases} \tag{4}$$

where $\mathcal{A}_s$ and $\mathcal{A}_b$ denote the admissible sampling and budget levels, and $\varnothing_b$ denotes a no-op budget action. Since the answer phase is not intended to contain explicit reasoning traces, reasoning-budget control is frozen while the sampling level remains active.

The control head is implemented as a lightweight MLP g_ϕ that takes the concatenation of $\mathbf{o}_{t_k}$ and $\mathbf{c}_{t_{k-1}}$ as input and produces a score for each admissible action $a = (s, b) \in \mathcal{A}(\rho_k)$, followed by a softmax normalization:

$$\begin{aligned} \pi_{\phi,k} &= \text{Softmax}_{a\in\mathcal{A}(\rho_k)}(z_{k,a})\,, \\ z_{k,a} &= g_\phi\big([\mathbf{o}_{t_k};\, \mathbf{c}_{t_{k-1}}]\big)_a\,, \end{aligned} \tag{5}$$

where $z_{k,a}$ is the scalar score assigned to action a. The full pipeline is summarized in Algorithm 1.

Algorithm 1: AutoCRAT Control Workflow

Input: Prompt q, frozen backbone f_θ, control head g_ϕ, boundary detector $\mathcal{B}$, initial state (s_0, b_0)
Output: Final output $\hat{y}$ and boundary trace $\mathcal{M}$

// Initialize controller state and trace;
$\mathbf{x}_{\le t_0} \leftarrow q$, $\mathbf{c}_{t_0} \leftarrow (s_0, b_0)$, $\rho_0 \leftarrow$ THINK, $\mathcal{M} \leftarrow \emptyset$;
while *not terminated* **do**
  // Decode local segment under active control;
  $\mathbf{x}_{(t_k, t_{k+1}]} \sim f_\theta(\cdot \mid \mathbf{x}_{\le t_k}, \mathbf{c}_{t_k})$ ▷ Eq. (2);
  if $t_{k+1} \in \mathcal{B}$ **then**
    // Boundary-level control update;
    $\mathbf{o}_{t_{k+1}} \leftarrow \text{OBS}(\mathbf{x}_{\le t_{k+1}})$;
    $\mathcal{A}_{k+1} \leftarrow \mathcal{A}(\rho_{k+1})$ ▷ Eq. (4);
    $z_{k+1,a} \leftarrow g_\phi([\mathbf{o}_{t_{k+1}}; \mathbf{c}_{t_k}])_a$, $\forall a \in \mathcal{A}_{k+1}$ ▷ Eq. (5);
    $\pi_{\phi,k+1}(a) \leftarrow \dfrac{\exp(z_{k+1,a})}{\sum_{a'\in\mathcal{A}_{k+1}} \exp(z_{k+1,a'})}$, $\forall a \in \mathcal{A}_{k+1}$ ▷ Eq. (5);
    $a_{k+1} \sim \pi_{\phi,k+1}$, $\mathbf{c}_{t_{k+1}} \leftarrow a_{k+1}$, $\mathcal{M} \leftarrow \mathcal{M} \cup \{(t_{k+1}, \mathbf{o}_{t_{k+1}}, a_{k+1})\}$;
  if *budget exhausted* $\vee$ *answer complete* $\vee$ *EOS* **then**
    // Boundary-triggered stop/phase transition;
    break / phase-transition;
return $\hat{y}, \mathcal{M}$;

We use discrete levels rather than continuous values because the control head is driven by noisy decoder-side signals; discretization reduces high-frequency oscillation between consecutive boundaries and keeps each segment under a stable, interpretable configuration. All examples are initialized from a fixed balanced control state $(s_0, b_0) = (0.7, 1024)$. This avoids task-specific routing and leaves subsequent adaptation to the evolving trajectory state. The concrete level assignments and decoding parameters are detailed in Section 5.1. The two dimensions are instantiated as follows:

- ***Sampling level*** controls decoding stochasticity by selecting from a discrete set of sampling configurations. In the current instantiation of **AutoCRAT**, it is implemented via temperature only, as simultaneously adjusting multiple sampling parameters (e.g., `top_p`, `top_k`) introduces inter-parameter interactions that render the resulting sampling behavior difficult to predict or interpret, a practice explicitly discouraged by OpenAI and Anthropic (OpenAI, 2026; Anthropic, 2025). Notably, the sampling dimension is in principle compatible with other stochasticity controls. The current restriction ensures clean attribution of empirical gains to within-trajectory joint control, with each sampling level corresponding to a single, interpretable degree of stochasticity.
- ***Budget level*** imposes upper bounds on the total number of reasoning tokens, following prior work that frames token allocation as the primary lever for compute control (Han et al., 2025; Muennighoff et al., 2025). Specifically, the bound is absolute, and when the active budget level is updated to a value whose bound falls below the tokens already consumed, the remaining allowance is immediately treated as exhausted, without revising previously generated tokens. Similar to the sampling level, the budget level is in principle compatible with other reasoning compute controls such as step-level pruning.

4.2 Boundary-Aware Update

To balance responsiveness and stability, **AutoCRAT** updates control decisions only at eligible semantic boundary points $t_k \in \mathcal{B}$, which are specified in Appendix C. Between two consecutive decision points, the selected action (s_k, b_k) is held fixed for all generated tokens. At t_k, h_ϕ observes the updated $\mathbf{o}_{t_k}$ and $\mathbf{c}_{t_{k-1}}$, and may select a new action according to Eq. (3). This boundary-level granularity avoids high-frequency token-level switching while aligning updates with step-like transitions in reasoning traces (Liu and Wang, 2025).

Boundary-aware updates are especially important for the budget control. Since budget reductions directly change the upper bounds and shorten the remaining reasoning phase, applying them in the middle of an unfinished fragment can create abrupt reasoning-answer transitions and unstable outputs (Wu et al., 2025a). **AutoCRAT** instead applies a new budget level only after an eligible semantic boundary. If the lower budget makes the remaining reasoning allowance exhausted, reasoning is cut from that boundary onward. This keeps budget reductions interpretable by turning them into boundary-level reasoning termination. The controller may stop further reasoning, but only after the current local segment has been completed, avoiding arbitrary mid-fragment interruption.

4.3 Offline Training from Static Traces

We train the control head h_ϕ fully offline from static traces, decoupling controller learning from online exploration noise and keeping supervision stable under a shared data distribution. Supervision is constructed in two stages. First, we collect trajectories from a fixed discrete action grid and convert raw runs into canonical traces with standardized fields, including task outcome, token cost, and ordered boundary events. Each boundary event is aligned to a decision point t_k and represented as a boundary-level sample containing the local observable state and active control action. Second, we assign each trajectory a quality score that balances task correctness and normalized token cost:

$$r(\tau) = \mathrm{Acc}(\tau) - \lambda\, \widetilde{C}(\tau), \tag{6}$$

where τ denotes a trajectory, $\mathrm{Acc}(\tau)$ is task correctness, and $\widetilde{C}(\tau)$ is problem-level normalized token cost, i.e., z-score within static trajectories of the same problem. λ is a fixed scalar coefficient that governs the accuracy–efficiency tradeoff. A larger λ shifts the controller toward efficiency-oriented behavior, making the framework adaptable to different deployment requirements. These scores are then used to construct trajectory-level preference pairs, and boundary-level supervision samples.

Given boundary state samples, the control head is trained with two complementary signals: behavior cloning for distribution matching over actions, and preference learning for pairwise ranking between better and worse actions in comparable local contexts. The training objective is

$$\mathcal{L}_{\text{train}} = \mathcal{L}_{\text{BC}} + \beta \mathcal{L}_{\text{pref}}, \tag{7}$$

with

$$\mathcal{L}_{\text{BC}} = -\frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \sum_{a \in \mathcal{A}(\rho_k)} q_k(a) \log \pi_{\phi,k}(a), \tag{8}$$

$$\mathcal{L}_{\text{pref}} = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} w_p \log\big(1 + \exp(-\Delta z_p)\big), \tag{9}$$

where $\mathcal{K}$ is the set of boundary samples and $\mathcal{P}$ is the set of preferred action pairs. Both supervision

terms are derived from trajectory quality in Eq. (6): $q_k(a)$ is a reward-weighted soft target over actions under the same boundary context, while w_p is a reward-gap weight for matched preference pairs. For $p = (k, a^+, a^-)$, $\Delta z_p = z_{k,a^+} - z_{k,a^-}$. Full construction is provided in Appendix D.

For scale and transfer, we use a unified small-backbone pipeline for trace collection and control head training, then deploy the same controller on larger backbones without retraining. Under this offline formulation, the same trace pool can be reused to isolate transferable control dynamics from backbone-specific optimization.

5 Experiments

5.1 Experimental Setup

Benchmarks & Metrics. We evaluate **AutoCRAT** on **6** benchmarks covering **3** reasoning-intensive domains: **(1) math reasoning**, GSM8K (Cobbe et al., 2021) and MATH-500 (Lightman et al., 2024); **(2) challenging QA**, ARC-C (Clark et al., 2018) and GPQA-D (Rein et al., 2023); and **(3) code generation**, HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). For metrics, we report accuracy for non-code tasks and execution-based pass@1 under the provided unit tests for code tasks. For inference cost, we report the average number of completion tokens. The dataset statistics and split details are in Appendix E.

Backbones & Training Details. We leverage **4** open-source, instruct-tuned, and text-only models that natively produce explicit reasoning traces: Qwen3-4B-2504 (`Qwen3-4B`), Qwen3-8B (`Qwen3-8B`), Qwen3-30B-A3B (`Qwen3-30B`) (Yang et al., 2025a), and DeepSeek-R1-Distill-Llama-8B (`DSR1-8B`) (Guo et al., 2025). For Qwen3 series models, which expose a switchable thinking mode, we keep thinking mode enabled throughout all experiments. We collect static traces from `Qwen3-4B` on the pooled training splits of six benchmarks to train a single controller, then apply it to all backbones on held-out test splits without any benchmark- or backbone-specific retraining. This setup evaluates scale-up and cross-family transferability. The controller is trained with accuracy as the primary reward signal, rather than efficiency targets which lack a natural reference point. Training data details are provided in Appendix G.

Baselines & Implementation Details. **AutoCRAT** operates over 4 temperature levels for sampling $\{0, 0.3, 0.7, 1.0\}$ and 5 reasoning token levels for budget $\{0, 64, 256, 1024, 4096\}$. Top-$p$ is fixed at 0.95 across all configurations as recommended by the backbone providers, while other decoding parameters such as top-k, min-p, and max output length use the default settings. For all comparisons, we keep the prompting and single-sample evaluation protocol identical. We compare against **3** groups of baselines. ❶ **Static baselines** include BASE (temperature 0, corresponding to deterministic decoding) and RECOMMEND (temperature 0.6, following the provider recommendation of each backbone family), both without budget constraint. ❷ The **sampling-control baseline** includes EDT (Zhang et al., 2024), an entropy-based dynamic-temperature baseline. ❸ The **budget-control baseline** includes DEER (Yang et al., 2025b), a dynamic early-exit baseline that controls reasoning compute. Reproduction details for the two adaptive baselines are provided in Appendix F. We include the two baselines because they represent the two single-axis adaptation settings most directly comparable to our control space, avoiding additional variables such as task-level tuning, reranking, candidate selection, or auxiliary training procedures. This choice helps keep the comparison focused on control granularity and axis coordination. Existing request-level joint-control methods (Wang et al., 2025a; Xu et al., 2025) also introduce additional variables, therefore we further include a static joint-control grid over the complete action space in Section 5.3, which tests whether **AutoCRAT** improves over the best fixed joint configuration.

5.2 Experimental Results

We structure the empirical analysis around **4** key **Obs**ervations, which systematically reveal the advantages of within-trajectory joint control and the design choices of **AutoCRAT**. The main results cover all **4** backbones and **6** benchmarks. The adaptive methods comparison and subsequent analyses are all conducted on `Qwen3-4B` to isolate effects from cross-backbone transfer, with the adaptive methods comparison spanning all **6** benchmarks and the joint control and ablation analyses over three representative benchmarks. All reported numbers are evaluated on held-out test splits.

Obs.❶ AutoCRAT improves the accuracy–compute tradeoff over static baselines on varying backbones without retraining. According to Table 1, **AutoCRAT** achieves a better accuracy–compute tradeoff than both static baselines across

Model	Setting	GSM8K		MATH-500		ARC-C		GPQA-D		HumanEval		MBPP		Avg.	
		Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.
Qwen3-4B	BASE	88.6	251	83.5	348	61.8	120	42.4	187	59.5	66	44.6	131	63.4	184
	RECOMMEND	92.0	2189	94.5	2765	68.4	1042	52.5	1276	67.2	2665	52.5	1708	71.2	1941
	AutoCRAT	**93.1**	**729**	**95.8**	**1333**	**70.2**	**382**	**53.8**	**1220**	**71.0**	**1138**	**55.4**	**1091**	**73.2**	**982**
Qwen3-8B	BASE	91.0	258	87.5	366	68.2	151	47.5	244	65.6	67	47.8	235	67.9	220
	RECOMMEND	95.8	2195	95.8	2692	76.4	1096	58.9	1492	71.8	2503	56.3	1715	75.8	1949
	AutoCRAT	**96.8**	**562**	**97.0**	**1088**	**77.8**	**296**	**60.1**	**1375**	**75.6**	**864**	**59.5**	**1096**	**77.8**	**880**
DSR1-8B	BASE	68.4	546	76.0	972	63.2	573	42.4	771	48.1	995	40.5	781	56.4	773
	RECOMMEND	78.3	878	88.3	1484	66.2	643	48.1	1131	51.1	2978	46.0	1546	63.0	1443
	AutoCRAT	**79.8**	**658**	**90.0**	**1243**	65.9	**604**	**49.4**	**1109**	**55.0**	**2195**	**50.7**	**1437**	**65.1**	**1208**
Qwen3-30B	BASE	93.4	310	91.0	462	75.6	210	53.2	305	69.5	92	54.8	280	72.9	277
	RECOMMEND	97.4	2520	97.5	3150	82.0	1180	65.2	1660	76.3	2840	62.2	1880	80.1	2205
	AutoCRAT	**98.2**	**710**	**98.0**	**1260**	**83.5**	**340**	**66.5**	**1450**	**79.4**	**1010**	**64.8**	**1220**	**81.7**	**998**

Table 1: Main results across backbones and benchmarks. Acc. denotes accuracy for non-code tasks and pass@1 for code tasks, reported as percentages. Tok. denotes average completion tokens.

Setting	GSM8K		MATH-500		ARC-C		GPQA-D		HumanEval		MBPP		Avg.	
	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.
RECOMMEND	92.0	2189	94.5	2765	68.4	1042	52.5	1276	67.2	2665	52.5	1708	71.2$_{\downarrow 2.7\%}$	1941$_{\uparrow 97.7\%}$
EDT	92.3	2105	95.0	2638	69.0	1018	53.2	1238	68.7	2520	54.3	1629	72.1$_{\downarrow 1.5\%}$	1858$_{\uparrow 89.2\%}$
DEER	91.7	846	93.5	1450	67.1	336	51.3	1262	64.9	920	50.7	906	69.9$_{\downarrow 4.5\%}$	953$_{\downarrow 2.9\%}$
AutoCRAT	**93.1**	**729**	**95.8**	**1333**	**70.2**	**382**	**53.8**	**1220**	**71.0**	**1138**	**55.4**	**1091**	**73.2**	**982**

Table 2: Comparison with existing adaptive baselines. ↓ marks the relative accuracy drop. ↑ marks baseline token overhead over **AutoCRAT**, and ↓ marks baseline token reduction below **AutoCRAT**.

benchmarks, as any fixed configuration cannot be optimal across problems of varying difficulty and reasoning stages. Compared with RECOMMEND, **AutoCRAT** improves average accuracy by 2.2–4.0% while reducing token usage by 13.8%–52.7% across all four backbones. The controller trained on Qwen3-4B retains an overall advantage on all backbones. As the control head operates solely on decoder-side observables that are structurally comparable across architectures, the learned policy captures transferable reasoning dynamics rather than backbone-specific patterns. Table 3 further shows statistically consistent gains of **AutoCRAT** over RECOMMEND across backbones.

Obs.❷ AutoCRAT outperforms existing adaptive methods across benchmarks. As shown in Table 2, compared to EDT and DEER, **AutoCRAT** achieves higher average accuracy, while the token trend depends on the single-axis controller. Specifically, **AutoCRAT** outperforms EDT by 1.5% in relative accuracy while using 47.1% fewer tokens. Compared with DEER, DEER trails **AutoCRAT** by 4.5% in relative accuracy while using 2.9% fewer tokens on average. This tradeoff suggests that single-axis compute control can over-compress reasoning, whereas coordinating stochasticity and budget within a trajectory better preserves task performance.

Backbone	Wins	ΔAcc.	Avg. Tok. Red.
Qwen3-4B	6/6	+3.1% [+1.9%, +4.5%]	46.6%
Qwen3-8B	6/6	+2.9% [+1.6%, +4.1%]	52.7%
DSR1-8B	5/6	+4.0% [+1.5%, +5.6%]	13.8%
Qwen3-30B	6/6	+2.2% [+1.1%, +3.4%]	52.5%

Table 3: Sample-level bootstrap summary against RECOMMEND. ΔAcc. reports the relative accuracy gain normalized by RECOMMEND accuracy, with a 95% confidence interval estimated by bootstrapping test examples within each benchmark and then averaging benchmark-level gains. Wins report the number of benchmarks where **AutoCRAT** improves accuracy. Avg. Tok. Red. reports the average relative token reduction.

5.3 Joint Control Analysis

To verify that the gains of **AutoCRAT** are indeed attributable to within-trajectory joint control, we compare against a static grid over the complete action space, which includes all possible fixed joint configurations. The oracle serves as an upper fixed-configuration reference.

	Oracle Config		Oracle		AutoCRAT	
	Temp.	Budget	Acc.	Tok.	Acc.	Tok.
GSM8K	0.7	1024	92.5	1008	93.1$_{\uparrow 0.6\%}$	729$_{\downarrow 27.7\%}$
ARC-C	0.7	1024	69.4	812	70.2$_{\uparrow 1.2\%}$	382$_{\downarrow 53.0\%}$
HumanEval	0.7	4096	68.7	2150	71.0$_{\uparrow 3.3\%}$	1138$_{\downarrow 47.1\%}$
Avg.	–	–	76.9	1323	78.1$_{\uparrow 1.6\%}$	750$_{\downarrow 43.4\%}$

Table 4: Within-trajectory joint control versus oracle fixed joint control. Oracle selects the highest-accuracy fixed temperature–budget pair from the same action grid for each benchmark. ↑ marks the relative accuracy gain of **AutoCRAT** over Oracle, and ↓ marks the token reduction relative to Oracle.

Setting	GSM8K		ARC-C		HumanEval	
	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.
BASE	88.6	251	61.8	120	59.5	66
RECOMMEND	92.0	2189	68.4	1042	67.2	2665
Sampling-only	92.2	1714	68.8	824	68.7	2192
Budget-only	91.9	668	67.2	316	66.4	987
Token-level	92.6	1019	69.1	451	68.7	1484
AutoCRAT	**93.1**	**729**	**70.2**	**382**	**71.0**	**1138**

Table 5: Ablation results on Qwen3-4B over 3 representative benchmarks. Sampling-only and Budget-only adapt a single control axis, and Token-level updates the controller at every token rather than at semantic boundaries.

Obs.❸ Within-trajectory joint control improves over oracle fixed joint control. As shown in Table 4, even when the fixed baseline is allowed to choose the highest-accuracy temperature–budget pair separately for each benchmark, **AutoCRAT** still improves the average accuracy by 1.6%, and uses 43.4% fewer tokens. The gains are not caused by a one-shot configuration choice. Figure 3 shows that **AutoCRAT** actively revises its control state within trajectories, with average switching counts ranging from 2.34 on ARC-C to 4.17 on MBPP. The benchmark-level ranges further show that the switching behavior is task dependent: ARC-C has the lowest and tightest interval (2.26–2.52), while code tasks require more frequent and more variable control updates, reaching 3.34–4.01 on HumanEval and 3.77–4.48 on MBPP. This pattern indicates that different domains affect both the absolute number of control updates and their variability. Appendix H provides an intuitive view through representative controlled trajectories, showing how **AutoCRAT** works at different reasoning stages.

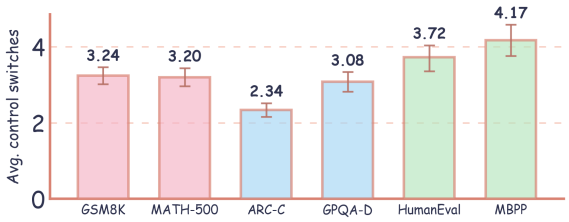

Figure 3: Average control-switching counts across six benchmarks.

5.4 Ablation Studies

We conduct ablation studies to analyze the impact of different components of **AutoCRAT**.

Obs.❹ Both joint control and boundary-aware updates are crucial design choices. As shown in Table 5, **AutoCRAT** outperforms both sampling-only and budget-only variants in accuracy while achieving a better token-accuracy tradeoff. Compared with the sampling-only variant, **AutoCRAT** improves accuracy by 1.0–3.3% across three benchmarks while reducing token usage by an average of 52.5%, confirming that budget control is not redundant given adaptive sampling. Compared with the budget-only variant, **AutoCRAT** achieves 1.3–6.9% accuracy gains at only a marginal token overhead, demonstrating that sampling control recovers task performance at a modest extra cost. These results confirm that the two dimensions interact and their joint optimum cannot be recovered by either alone. Boundary-aware updates also prove their importance. Token-level switching yields lower accuracy and higher token use across all benchmarks, as frequent updates on noisy local signals disrupt reasoning fragments mid-completion, whereas semantic boundaries provide a more stable observation window over complete local units. Together, the studies show the complementary value of joint control and boundary-aware updates.

6 Conclusion

In this paper, we shift the view of inference-time LLM reasoning control from isolated or per-request adaptation to within-trajectory joint control of decoding stochasticity and reasoning compute. Building on this concept, we propose **AutoCRAT**, which dynamically controls both dimensions in a single trajectory, and achieves a substantially better accuracy–compute tradeoff than baselines. We believe that **AutoCRAT** paves the way toward more comprehensive, principled and effective inference-time control for LLM reasoning.

Limitations

The central claim of this work is the value of within-trajectory joint control of decoding stochasticity and reasoning compute. To ensure that empirical gains are attributable to this view rather than to sophisticated optimization, we deliberately simplify two aspects of the instantiation. The action space is restricted to temperature and token-budget limits, and the controller is trained from offline static traces rather than online reinforcement learning. Both choices trade expressive power and training signal richness for cleaner attribution. Other parameters and actions such as top-p, rollback, and reasoning steps, as well as online training, are left for future work.

For design tradeoffs, **AutoCRAT** relies solely on decoder-side observables and control-state variables. While this reduces coupling to specific model architectures, it also limits the access to finer-grained internal reasoning signals. Furthermore, boundary detection for update relies on lightweight structural heuristics, which may be less robust in tasks with weak or irregular discourse structure. Finally, we do not perform per-backbone retraining or online reinforcement learning. While these strategies may further improve absolute performance, they introduce backbone-specific optimization and are outside our transferability-focused scope.

The controller also introduces a non-zero per-boundary inference overhead, as the control head is invoked once at every eligible boundary. In our setting this overhead remains modest, since the control head is a lightweight MLP over low-dimensional decoder-side features and the number of boundary invocations per trajectory remains bounded by a small constant in our experiments, well below the number of generated tokens. Nevertheless, on latency-sensitive deployments or with very long trajectories the cumulative overhead becomes non-trivial, and tighter co-design between the controller and the decoding loop is a worthwhile direction.

Ultimately, our experiments are currently limited to text-based reasoning on English benchmarks. The behavior of **AutoCRAT** or similar frameworks in broader interactive environments, including multilingual generalization, multimodal reasoning, and agentic workflows remains to be established.

Ethics Statement

This work uses publicly available benchmarks and open models within their applicable licenses, terms, and intended research settings. It does not involve human-subject experiments, private data collection, or sensitive personal data annotation. Therefore, the ethical considerations of this study are limited and mainly concern compliance with dataset and model usage policies, together with responsible deployment to reduce the risk of misuse. Our evaluation is currently limited to English text benchmarks, and broader generalization is not assessed in this work.

References

Mohammad Ali Alomrani, Yingxue Zhang, Derek Li, Qianyi Sun, Soumyasundar Pal, Zhanguang Zhang, Yaochen Hu, Rohan Deepak Ajwani, Antonios Valkanas, Raika Karimi, Peng Cheng, Yunzhou Wang, Pengyi Liao, Hanrui Huang, Bin Wang, Jianye Hao, and Mark Coates. 2025. Reasoning on a budget: A survey of adaptive and controllable test-time compute in llms. *arXiv preprint arXiv:2507.02076*.

Anthropic. 2025. Claude api release notes. https://docs.anthropic.com/en/release-notes/api. Accessed: 2026-04-24.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

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

Qiguang Chen, Dengyun Peng, Jinhao Liu, HuiKang Su, Jiannan Guan, Libo Qin, and Wanxiang Che. 2025a. Aware first, think less: Dynamic boundary self-awareness drives extreme reasoning efficiency in large language models. *arXiv preprint arXiv:2508.11582*.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. 2025b. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind

Tafjord. 2018. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Shumin Deng, Ningyu Zhang, Nay Oo, and Bryan Hooi. 2023. Towards a unified view of answer calibration for multi-step reasoning. *arXiv preprint arXiv:2311.09101*.

Weihua Du, Yiming Yang, and Sean Welleck. 2025. Optimizing temperature for language models with multi-sample inference. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 14648–14668. PMLR.

Markus Freitag and Yaser Al-Onaizan. 2017. Beam search strategies for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 56–60, Vancouver, Canada. Association for Computational Linguistics.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. Token-budget-aware llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24842–24855, Vienna, Austria. Association for Computational Linguistics.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*.

Shengran Hu, Cong Lu, and Jeff Clune. 2024. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*.

Yixin Ji, Juntao Li, Yang Xiang, Hai Ye, Kaixin Wu, Kai Yao, Jia Xu, Linjian Mo, and Min Zhang. 2025. A survey of test-time compute: From intuitive inference to deliberate reasoning. *arXiv preprint arXiv:2501.02497*.

Sekitoshi Kanai, Tsukasa Yoshida, Hiroshi Takahashi, Haru Kuroki, and Kazumune Hashimoto. 2025. Test-time alignment of LLMs via sampling-based optimal control in pre-logit space. *arXiv preprint arXiv:2510.26219*.

Zhiyuan Li, Yi Chang, and Yuan Wu. 2025. THINK-Bench: Evaluating thinking efficiency and chain-of-thought quality of large reasoning models. *arXiv preprint arXiv:2505.22113*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *Proceedings of the Twelfth International Conference on Learning Representations*.

Junhong Lin, Xinyue Zeng, Jie Zhu, Song Wang, Julian Shun, Jun Wu, and Dawei Zhou. 2026. Plan and budget: Effective and efficient test-time scaling on reasoning large language models. *Preprint*, arXiv:2505.16122.

Xin Liu and Lu Wang. 2025. Answer convergence as a signal for early stopping in reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17896–17907, Suzhou, China. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20275–20321, Suzhou, China. Association for Computational Linguistics.

OpenAI. 2024. Learning to reason with LLMs. *OpenAI Blog*.

OpenAI. 2025. Introducing GPT-5 for developers. *OpenAI Blog*.

OpenAI. 2026. Responses api reference. `https://platform.openai.com/docs/api-reference/responses/create`. Accessed: 2026-04-24.

Shubham Parashar, Blake Olson, Sambhav Khurana, Eric Li, Hongyi Ling, James Caverlee, and Shuiwang Ji. 2025. Inference-time computations for llm reasoning and planning: A benchmark and insights. *arXiv preprint arXiv:2502.12521*.

Xiao Pu, Michael Saxon, Wenyue Hua, and William Yang Wang. 2025. THOUGHTTERMINATOR: Benchmarking, calibrating, and mitigating overthinking in reasoning models. *arXiv preprint arXiv:2504.13367*.

Shauli Ravfogel, Yoav Goldberg, and Jacob Goldberger. 2023. Conformal nucleus sampling. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 27–34, Toronto, Canada. Association for Computational Linguistics.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. GPQA: A graduate-level google-proof Q&A benchmark. *arXiv preprint arXiv:2311.12022*.

Matthew Renze. 2024. The effect of sampling temperature on problem solving in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7346–7356, Miami, Florida, USA. Association for Computational Linguistics.

Jon Saad-Falcon, Adrian Gamarra Lafuente, Shlok Natarajan, Nahum Maru, Hristo Todorov, Etash Guha, E. Kelly Buchanan, Mayee Chen, Neel Guha, Christopher Ré, and Azalia Mirhoseini. 2025. Archon: An architecture search framework for inference-time techniques. *Preprint*, arXiv:2409.15254.

Mousa Salah and Amgad Muneer. 2026. Temperature-dependent performance of prompting strategies in extended reasoning large language models. *arXiv preprint arXiv:2604.08563*.

Aman Sharma and Paras Chopra. 2025. Think just enough: Sequence-level entropy as a confidence signal for LLM reasoning. *arXiv preprint arXiv:2510.08146*.

Chufan Shi, Haoran Yang, Deng Cai, Zhisong Zhang, Yifan Wang, Yujiu Yang, and Wai Lam. 2024. A thorough examination of decoding methods in the era of llms. *arXiv preprint arXiv:2402.06925*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Lihao Sun, Hang Dong, Bo Qiao, Qingwei Lin, Dongmei Zhang, and Saravan Rajmohan. 2026. LLM reasoning as trajectories: Step-specific representation geometry and correctness signals. *arXiv preprint arXiv:2604.05655*.

Sergey Troshin, Wafaa Mohammed, Yan Meng, Christof Monz, Antske Fokkens, and Vlad Niculae. 2025. Control the temperature: Selective sampling for diverse and high-quality LLM outputs. *arXiv preprint arXiv:2510.01218*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*.

Xiangqi Wang, Yue Huang, Yanbo Wang, Xiaonan Luo, Kehan Guo, Yujun Zhou, and Xiangliang Zhang. 2025a. AdaReasoner: Adaptive reasoning enables more flexible thinking in large language models. *arXiv preprint arXiv:2505.17312*.

Xinyan Wang, Xiaogeng Liu, and Chaowei Xiao. 2026. ROM: Real-time overthinking mitigation via streaming detection and intervention. *arXiv preprint arXiv:2603.22016*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*.

Yiming Wang, Pei Zhang, Siyuan Huang, Baosong Yang, Zhuosheng Zhang, Fei Huang, and Rui Wang. 2025b. Sampling-efficient test-time scaling: Self-estimating the best-of-n sampling in early decoding. *arXiv preprint arXiv:2503.01422*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. 2023. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2550–2575, Singapore. Association for Computational Linguistics.

Tsung-Han Wu, Mihran Miroyan, David M. Chan, Trevor Darrell, Narges Norouzi, and Joseph E. Gonzalez. 2025a. Are large reasoning models interruptible? *arXiv preprint arXiv:2510.11713*.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.

Yuyang Wu, Yifei Wang, Ziyu Ye, Tianqi Du, Stefanie Jegelka, and Yisen Wang. 2025b. When more is less: Understanding chain-of-thought length in llms. *arXiv preprint arXiv:2502.07266*.

Yuebin Xu, Zhiyi Chen, and Zeyi Wen. 2025. Ecotune: Token-efficient multi-fidelity hyperparameter optimization for large language model inference. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7735–7745, Suzhou, China. Association for Computational Linguistics.

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

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

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. 2025b. Dynamic early exit in reasoning models. *Preprint*, arXiv:2504.15895.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Sheldon Yu, Yuxin Xiong, Junda Wu, Xintong Li, Tong Yu, Xiang Chen, Ritwik Sinha, Jingbo Shang, and Julian McAuley. 2025. Explainable chain-of-thought reasoning: An empirical analysis on state-aware reasoning dynamics. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 16660–16667, Suzhou, China. Association for Computational Linguistics.

Qingjie Zhang, Yujia Fu, Yang Wang, Liu Yan, Tao Wei, Ke Xu, Minlie Huang, and Han Qiu. 2026. Stop before you fail: Operational capability boundaries for mitigating unproductive reasoning in large reasoning models. *Preprint*, arXiv:2509.24711.

Shimao Zhang, Yu Bao, and Shujian Huang. 2024. EDT: Improving large language models' generation by entropy-based dynamic temperature sampling. *arXiv preprint arXiv:2403.14541*.

Chenghua Zhu, Siyan Wu, Xiangkang Zeng, Zishan Xu, Zhaolu Kang, Yifu Guo, Yuquan Lu, Junduan Huang, and Guojing Zhou. 2026. EDIS: Diagnosing LLM reasoning via entropy dynamics. *arXiv preprint arXiv:2602.01288*.

Wenhong Zhu, Hongkun Hao, Zhiwei He, Yiming Ai, and Rui Wang. 2024. Improving open-ended text generation via adaptive decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 62386–62404. PMLR.

A Detail of Control Space Completeness

This section provides a formal argument for the control-space completeness claim in Section 3, which asserts that the two-dimensional decomposition $\mathbf{c}_t = (\mathbf{c}_{s,t}, \mathbf{c}_{c,t})$ exhausts all control degrees of freedom within a single LLM inference trajectory. Appendix A.1 & A.2 formalize the admissible intervention space. Appendix A.3 states a completeness proposition. Appendix A.4 explains why potential counterexamples do not violate the claim.

A.1 Scope of the Argument

The completeness claim is relative to a precisely defined setting. We consider **single-trajectory autoregressive decoding** (subsuming standard LLM reasoning and generation) with a **frozen backbone**. Under this setting, the controller is constrained as follows:

❶ not modify model parameters;

❷ not maintain or branch into multiple parallel candidate trajectories;

❸ not revise or retract previously generated tokens (generation is strictly forward);

❹ not introduce information from outside the current trajectory, including external tools, auxiliary verifiers, or reward models.

We emphasize that completeness here refers to *completeness of control degrees of freedom within this setting*. Methods that operate outside these constraints are not counterexamples to our claim.

A.2 Admissible Interventions

In standard autoregressive generation, the system at step t, given the token history $x_{<t}$, faces exactly two structural questions:

- **Q1 (Continuation):** Should generation proceed, or should it stop?
- **Q2 (Sampling):** If generation proceeds, from what distribution should the next token x_t be drawn?

These two questions admit the following formal objects of control. The continuation decision is parameterized by a stopping condition $\mathbb{1}[t < \tau]$, where τ denotes a possibly random stopping time. A controller controls τ dynamically. The sampling decision is parameterized by the (possibly controller-modified) next-token distribution

$$\tilde{p}(x_t \mid x_{<t}, \mathbf{c}_{s,t}) = \mathcal{T}[p_\theta(\cdot \mid x_{<t}); \mathbf{c}_{s,t}], \quad (10)$$

where $\mathcal{T}[\,\cdot\,]$ denotes any transformation of the base model's next-token distribution, instantiated for example by temperature scaling, nucleus truncation, logit biasing, or constrained decoding masks.

These two objects align precisely with $\mathbf{c}_{c,t}$ (continuation control) and $\mathbf{c}_{s,t}$ (distribution shaping). Throughout the paper we use decoding stochasticity control as a convenient shorthand for $\mathbf{c}_{s,t}$. It refers to interventions that shape the conditional next-token distribution, including but not limited to changes in randomness level. Intuitively, $\mathbf{c}_{s,t}$ controls how the trajectory explores at each step, while $\mathbf{c}_{c,t}$ controls how far the trajectory extends. Together, these two dimensions completely characterize the degrees of freedom of a single trajectory.

A.3 Proposition

Proposition 1 (Control Space Completeness)**.** *Under the single-trajectory setting, any admissible inference-time intervention that alters the rollout distribution does so through one of the following two pathways, or their combination:*

(i) ***Distribution shaping*** $(\mathbf{c}_{s,t})$*: modification of the conditional next-step token-selection distribution* $\tilde{p}(x_t \mid x_{<t}, \mathbf{c}_{s,t})$*;*

(ii) ***Continuation control*** $(\mathbf{c}_{c,t})$*: modification of the stopping condition* $\mathbb{1}[t < \tau]$*.*

No third intervention class exists whose effect on the rollout distribution cannot be reduced to, or expressed as a combination of, (i) and (ii). Certain implementations may couple both dimensions, as discussed in Appendix A.4.

We establish the proposition in three steps.

Two-component decomposition. In a forward-only, single-trajectory autoregressive process, the per-step evolution at step t is governed by a conditional transition law K_t. At this granularity, a transition only decides whether the trajectory continues and, if it continues, which next token is produced. Given the token history $x_{<t}$ and the current control state $\mathbf{c}_t$, this law can be decomposed into two components:

$$K_t \equiv \Big(\underbrace{P(\text{continue} \mid x_{<t}, \mathbf{c}_t)}_{\text{continuation component}}, \underbrace{\tilde{p}(x_t \mid \text{continue}, x_{<t}, \mathbf{c}_t)}_{\text{sampling component}} \Big). \quad (11)$$

We separate these two components by structural role: the first governs whether the trajectory con-

tinues, and the second governs which token is produced if it does. The same concrete control implementation may affect one or both components. This decomposition is not an empirical observation; it is a direct structural consequence of the autoregressive generation mechanism under the forward-only, no-branching, no-rollback constraints of our setting. Any controller that influences the trajectory must do so by modifying at least one component of K_t.

Pathway reduction. Any concrete control parameter operates through one of two pathways:

- **Pathway (i)** — affecting the sampling component: temperature, top-p, top-k, repetition penalties, logit biases, grammar masks, and entropy-regularized rescaling all instantiate the transformation $\mathcal{T}[\,\cdot\,]$ on the base logits. They differ in their parameterization but share the same role: shaping $\tilde{p}(x_t \mid x_{<t})$.
- **Pathway (ii)** — affecting the continuation component: maximum token budget, dynamic budget allocation, convergence-based early stopping, answer-readiness gating, and confidence-triggered termination all instantiate modifications to τ. They differ in their triggering logic but share the same role: controlling whether generation proceeds.

Mechanisms in this setting instantiate one or both of these pathways, corresponding to $\mathbf{c}_{s,t}$ and $\mathbf{c}_{c,t}$ respectively.

Exhaustiveness. Under this kernel definition, if an intervention leaves both components of K_t unchanged for every reachable history, then every step uses the same transition law as the original rollout. The induced rollout distribution is therefore unchanged by recursive composition over t. Such an intervention has no observable effect on produced tokens or continuation decisions, so it is not a separate control degree of freedom in this setting.

A.4 Counterexample Handling

We now address four categories of inference-time methods that may appear to challenge the proposition. They are either outside the scope, or can be reduced to one or both of the two dimensions.

(a) Beam search, best-of-n, and tree-structured search. These methods maintain a set of concurrent candidate trajectories, operating over a multi-trajectory structure rather than a single evolving sequence. Their defining characteristic is therefore a change in the structure of inference itself, not a modification of control within a single trajectory.

(b) Rollback, backtracking, and self-refinement with token revision. Methods that allow a system to retract previously generated tokens and rewrite earlier segments of the trajectory violate the forward-only, single-trajectory assumption (Constraint 3 in Appendix A.1). Once rollback is permitted, the trajectory is no longer a monotonically growing sequence and the single-trajectory framework no longer applies.

(c) Hidden-state steering and activation-level intervention. Methods that directly read or modify the backbone's intermediate hidden states introduce a deeper access channel into the generation process. However, in any transformer-based LLM architecture (Vaswani et al., 2017), such interventions ultimately influence the trajectory through the next-token logit distribution. A modified hidden state must pass through the remaining layers and the unembedding matrix before it affects which token is sampled. The effect therefore still terminates at $\tilde{p}(x_t \mid x_{<t})$, placing hidden-state steering within Pathway (i). If the intervention also changes cached states and later steps, those effects appear as changes to the corresponding future $\tilde{p}(x_t \mid x_{<t})$ terms; the cross-step effect is a composition of stepwise transition changes rather than a new control dimension.

(d) EOS token biasing. Mechanistically, increasing the logit of the end-of-sequence token to encourage earlier termination is a modification of the next-token distribution (Pathway i). Functionally, it can serve as an indirect mechanism for continuation control (Pathway ii). It is therefore a coupled instance of distribution shaping and continuation control, which is already covered by the proposition rather than a counterexample.

B Decoder-side Observable List

This section summarizes the used decoder-side observables in Table 6.

C Boundary Detection Details

Table 7 summarizes the boundary cues and termination events used by **AutoCRAT**. Semantic boundaries trigger regular control updates, and termination events end either the reasoning phase or the entire generation. Notably, BUDGET_EXHAUSTED

Category	Included Features
Uncertainty features	Entropy, margin, top-1 probability, top-2 probability, top-k probability mass, EOS probability, EOS rank.
Progress and budget features	Generated-token progress ratio, remaining reasoning-budget ratio, segment progress ratio.
Structural boundary features	Boundary-kind indicators, answer-zone flag, think-zone flag.
Control-state features	Joint sampling-budget one-hot in think phase, sampling-level one-hot in answer phase.

Table 6: Feature categories used by **AutoCRAT** at boundary-level decision points.

Cue / event	Definition	Examples
Semantic boundary:		
Line break	Line- or paragraph-level separation	Newline, blank line
Sentence ending	Sentence-final punctuation	Period, question mark, exclamation mark
Step marker	Explicit reasoning-step delimiter	Numbered steps, `Step 1:`, `Reasoning:`
Termination event:		
Natural EOS	Backbone-generated end-of-sequence event	EOS token
Answer completion	Task-format or final-answer closure	Final-answer marker, code block or JSON closure
Budget exhausted	Exhaustion of the active reasoning budget	Remaining reasoning allowance reaches zero

Table 7: Boundary cues and termination events used by **AutoCRAT**. All cues are detected from generated text and decoding metadata.

only terminates the reasoning phase and transitions to the answer phase, instead of stopping generation. To avoid overly dense updates, a cue triggers a decision point only after a minimum number of 6 tokens from the previous one.

D Training Algorithm

This section presents the offline training workflow. Algorithm 2 summarizes the full pipeline, from static-trace scoring and supervision construction, to joint optimization of behavior-cloning and preference losses. The controller is trained once using pooled static traces collected from the training splits of all 6 benchmarks. For all experiments, we use the same trained controller without per-benchmark retraining. We define the boundary context key κ_k by the same problem id, phase, boundary ordinal, and boundary cue type. CTX returns this key; MATCHPAIRS forms preference pairs only within the same key; and TRACEPAIR returns the higher- and lower-reward trajectories in such a pair.

E Dataset Statistics

Building upon established methodologies in workflow automation (Saad-Falcon et al., 2025; Hu et al., 2024), we divide each dataset into training and test sets using a TRAIN:TEST ratio of 1:4. The dataset statistics are included in Table 8. For evaluation, we adopt the official pipelines from the benchmarks. For non-code tasks, we report exact-match accuracy after task-specific answer extraction and normalization. For the mathematical tasks with symbolic answers in the non-code tasks, we additionally use the math grader when normalization alone is insufficient. For code tasks, we extract the final submitted program and report execution-based pass@1 under the provided unit tests.

Domain	Dataset	#Train	#Test	Metric
Code Generation	HumanEval	33	131	pass@1
	MBPP	86	341	pass@1
Math Reasoning	GSM8K	264	1055	Acc
	MATH-500	100	400	Acc
Challenging QA	ARC-Challenge	458	1833	Acc
	GPQA-Diamond	40	158	Acc

Table 8: Dataset Statistics.

F Adaptive Baseline Reproduction Details

This section describes how we reproduce the two adaptive baselines used in Table 2, EDT (Zhang et al., 2024) and DEER (Yang et al., 2025b). For tasks covered by the released implementations, we follow the original hyperparameter settings and do not perform per-benchmark calibration. For DEER, the public repository did not release the code-generation implementation at the time of our experiments; therefore, on HumanEval and MBPP we use a paper-faithful reimplementation of the

Algorithm 2: Offline Training Workflow

Input: Static trajectories $\mathcal{T}$, coefficients $\lambda, \alpha, \gamma, \beta$, learning rate η
Output: Trained controller parameters ϕ

// Canonicalize traces and assign trajectory quality;
foreach $\tau \in \mathcal{T}$ **do**
  $r(\tau) \leftarrow \mathrm{Acc}(\tau) - \lambda \widetilde{C}(\tau)$ ▷ Eq. (6);
// Build boundary samples and BC soft targets from $r(\tau)$;
$\mathcal{K} \leftarrow$ CollectBoundarySamples($\mathcal{T}$);
foreach $k \in \mathcal{K}$ **do**
  foreach $a \in \mathcal{A}(\rho_k)$ **do**
    // Match traces that share boundary context κ_k and choose action a;
    $\mathcal{T}_{k,a} \leftarrow \{\tau \in \mathcal{T} \mid \mathrm{Ctx}(\tau, k) = \kappa_k,\ \mathrm{Act}(\tau, k) = a\}$;
    $q_{k,a} \leftarrow \sum_{\tau \in \mathcal{T}_{k,a}} \exp(\alpha r(\tau))$;
  $q_{k,a} \leftarrow q_{k,a} / \sum_{a' \in \mathcal{A}(\rho_k)} q_{k,a'},\ \forall a \in \mathcal{A}(\rho_k)$;
// Build preference pairs and pair weights from reward gaps;
$\mathcal{P} \leftarrow$ MatchPairs($\mathcal{T}, \mathcal{K}$);
foreach $p = (k, a^+, a^-) \in \mathcal{P}$ **do**
  $(\tau^+, \tau^-) \leftarrow \mathrm{TracePair}(p),\ r(\tau^+) > r(\tau^-)$;
  $w_p \leftarrow \sigma\big(\gamma(r(\tau^+) - r(\tau^-))\big)$;
initialize ϕ;
// Joint optimization of BC and preference objectives;
for $e \leftarrow 1$ **to** E **do**
  foreach *mini-batch* $\mathcal{D} \subset \mathcal{K}$ **do**
    $z_{k,a} \leftarrow g_\phi([\mathbf{o}_{t_k}; \mathbf{c}_{t_{k-1}}])_a,\ \forall a \in \mathcal{A}_k, \forall k \in \mathcal{D}$ ▷ Eq. (5);
    $\mathcal{A}_k \leftarrow \mathcal{A}(\rho_k),\ \pi_{k,a} \leftarrow \mathrm{Softmax}_{a \in \mathcal{A}_k}(z_{k,a}),\ \forall k \in \mathcal{D}$ ▷ Eq. (5);
    $\mathcal{L}_{\mathrm{BC}}(\mathcal{D}) \leftarrow -\frac{1}{|\mathcal{D}|} \sum_{k \in \mathcal{D}} \sum_{a \in \mathcal{A}_k} q_{k,a} \log \pi_{k,a}$ ▷ Eq. (8);
  foreach *mini-batch* $\mathcal{U} \subset \mathcal{P}$ **do**
    $\Delta z_p \leftarrow z_{k,a^+} - z_{k,a^-},\ \forall p = (k, a^+, a^-) \in \mathcal{U}$;
    $\mathcal{L}_{\mathrm{pref}}(\mathcal{U}) \leftarrow \frac{1}{|\mathcal{U}|} \sum_{p \in \mathcal{U}} w_p \log(1 + \exp(-\Delta z_p))$ ▷ Eq. (9);
  // Use mini-batch averaged $\mathcal{L}_{\mathrm{BC}}$ and $\mathcal{L}_{\mathrm{pref}}$ in this epoch step;
  $\mathcal{L}_{\mathrm{train}} \leftarrow \mathcal{L}_{\mathrm{BC}} + \beta \mathcal{L}_{\mathrm{pref}}$ ▷ Eq. (7);
  $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_{\mathrm{train}}$;
return ϕ;

same early-exit rule. We keep the original stopping threshold unchanged and do not tune it on code benchmarks. The stop parser is adapted only to recognize task-specific answer boundaries for code completion, while the early-exit decision rule itself is unchanged.

For both baselines, final answers, including code completions, are evaluated with the same task-specific extraction and scoring scripts used for **AutoCRAT**. Completion tokens are counted with the same tokenizer from the corresponding backbone. This keeps the comparison focused on whether a single-axis controller can match a within-trajectory joint controller under a shared evaluation pipeline.

G Training Details

In this section, Table 9 lists the controller training settings used in our experiments.

Setting	Value
Backbone	`Qwen3-4B-2504`
Action grid	4×5 configurations
Sampling levels	0, 0.3, 0.7, 1.0
Budget levels	0, 64, 256, 1024, 4096
Initial control state	$(0.7, 1024)$
Top-p	0.95
Minimum boundary interval	6 tokens
Overall budget	16384 tokens
Reward cost coefficient λ	0.2
Target softmax temperature	0.25
MLP hidden dimension	256
Dropout	0.1
Optimizer	AdamW
Learning rate	3×10^{-4}
Weight decay	10^{-4}
Pairwise loss weight	0.5
Controller training epochs	140

Table 9: Fixed trace-collection and controller-training settings.

H Case Study of Controlled Trajectory

This section provides two qualitative case studies that make the within-trajectory control concrete. Both examples use the `Qwen3-8B` backbone, and are selected from the original benchmarks. For readability, we report condensed boundary-level trajectories rather than raw token streams. The first example is a direct mathematical conversion from MATH-500, representing a case where the reasoning path is short and converges quickly. The second is an algorithmic code-generation problem from MBPP, representing a case where the controller must navigate genuine uncertainty before the solution structure becomes clear.

H.1 Example of Early Consolidation

The instance `test/precalculus/807.json` from MATH-500 asks: "Convert the point $(0, 3)$ in rectangular coordinates to polar coordinates. Enter your answer in the form (r, θ), where $r > 0$ and $0 \le \theta < 2\pi$." The correct answer is $\left(3, \frac{\pi}{2}\right)$.

This problem has a simple structure: once the controller observes that the prompt only requires a direct coordinate conversion, the useful reasoning path is short and deterministic. A fixed large-

Stage	Local observation	Temperature	Budget	Condensed generated segment
t_0 Think	New problem; math prior, no evidence yet about difficulty.	0.7	1024	Identify the task as rectangular-to-polar conversion and recall $r = \sqrt{x^2 + y^2}$.
t_1 Think	Formula matched; entropy drops after substituting $x = 0, y = 3$.	0.3	256	Compute $r = \sqrt{0^2 + 3^2} = 3$. Since the point lies on the positive y-axis, $\theta = \pi/2$.
t_2 Think	Both required fields are determined; additional reasoning is unnecessary.	0.3	0	Close the reasoning phase after checking that $\theta = \pi/2$ satisfies $0 \leq \theta < 2\pi$.
t_3 Answer	Reasoning has ended; budget control is frozen, but answer formatting remains active.	0	$\varnothing_b$	Output $\boxed{\left(3, \frac{\pi}{2}\right)}$ in the requested tuple format.

Table 10: Condensed controlled trajectory for the MATH-500 polar-coordinate example on `Qwen3-8B`. This example contains three control updates.

Stage	Local observation	Temperature	Budget	Condensed generated segment
t_0 Think	Code task detected; prompt underspecifies whether pairs are pre-sorted.	0.7	1024	Parse the required relation: pair j can precede pair i when $a_i > b_j$. Consider DP over pair positions.
t_1 Think	Competing strategies appear: greedy sorting is tempting but not guaranteed by the prompt.	1.0	4096	Compare solution templates and select the safer recurrence: `mcl[i]` is the best chain ending at `i`.
t_2 Think	Recurrence is fixed; implementation should be stable and syntax-sensitive.	0.3	1024	Generate the nested-loop function, initialize all chain lengths to 1, and update `mcl[i] = mcl[j] + 1`.
t_3 Think	Code structure and tests are resolved; remaining work is submission formatting.	0.3	0	End the reasoning phase after preserving the `Pair.a`/`Pair.b` interface and the required function signature.
t_4 Answer	Reasoning budget is frozen; sampling controls deterministic code-body emission.	0.3	$\varnothing_b$	Emit the `Pair` class and the loop structure, keeping the generated code syntactically complete.
t_5 Answer	Code body is complete; only final submission closure remains.	0	$\varnothing_b$	Return `max(mcl)` after the loops and stop after the submitted program.

Table 11: Condensed controlled trajectory for the MBPP longest-chain code example on `Qwen3-8B`. The trajectory contains five control updates.

budget setting can still solve the problem, but it tends to spend unnecessary tokens restating the polar-coordinate formula and checking quadrants. The control process of **AutoCRAT** is demonstrated in Table 10. The case illustrates why dynamic budget reduction matters even on easy examples. The first boundary keeps enough flexibility to recognize the task type, while the later boundaries become deterministic and short once the local state is resolved. This prevents overthinking without relying on a globally small budget that would be detrimental for harder math problems.

H.2 Example of Exploratory-Consolidation

The instance `task_id=601` from MBPP asks the model to "write a function to find the longest chain which can be formed from the given set of pairs." A valid solution defines a pair (a, b) and returns the maximum number of pairs that can be chained so that the next pair's first element is larger than the previous pair's second element. The representative tests include chains such as `[(5,24),(15,25),(27,40),(50,60)] -> 3`.

In this example, the model typically infers the algorithmic invariant, handles ordering assumptions, and produces executable Python. A low-budget deterministic rollout often jumps directly to code and risks missing the dynamic-programming recurrence, while an unconstrained high-budget rollout may spend many tokens discussing alternative greedy and sorting variants. As shown in Table 11, **AutoCRAT** allocates more compute during algorithm selection, then reduces stochasticity during implementation. After reasoning ends, the budget action is frozen, but the temperature remains controlled across two answer-stage boundaries. This trajectory highlights the interaction between the two controlled axes. Furthermore, the answer phase demonstrates that temperature control remains meaningful even after budget actions

are frozen, as deterministic emission is preferred for code body but a small allowance is retained during the transition to final closure.