Towards Understanding Pause Token Fine-Tuning Dynamics: A Mode Retention Perspective

Anonymous ACL submission

Abstract

Pause-token methods improve LLM reasoning by inserting special tokens into sequences. Prior work explains these gains through computational expressivity. However, there is relatively little investigation into the training dynamics of pause tokens. We explore how pause tokens reshape the training dynamics of fine-tuning. Two controlled pilots expose distinct asymmetries. On a synthetic continual-learning task, masked pauses overwrite a previously-learned distribution roughly 4× less at matched final adaptation (H1, **mode retention**); on a synthetic math-reasoning probe, the boundary-adjacent token comes to encode substantially more downstream-step information (H2, **non-myopic compression**). We formalize a training rule consistent with both — Masked Boundary Pause (MBP), pause tokens placed at reasoning-step boundaries with their loss masked. Across 1B-8B Qwen and Llama models, MBP consistently improves reasoning, achieving gains of up to 6 points on math and 2.5 points on code, while preserving general language understanding abilities. We further demonstrate that this mode-preserving strategy extend gains to GRPO. These results recast pause tokens as a training-dynamics intervention on the retention–adaptation trade-off, rather than merely an inference-time computation device.

1 Introduction

Recent progress in large language models (LLMs) has produced a wide range of training-time and test-time methods for improving reasoning: chain-of-thought prompting, test-time scaling, and reinforcement learning for reasoning achieve strong performance on complex tasks (Wei et al., 2023; Snell et al., 2024; Wang et al., 2023). However, these methods incur substantial costs in generation length, inference latency, sampling budget, or training compute. *Pause-* and *filler-token* methods stand out within this landscape (Goyal et al., 2024; Burtsev et al., 2021). They improve reasoning performance through a small and architecture-preserving change by inserting special tokens into sequences without modifying the model or the optimizer.

Pause-token gains have been attributed to forward-pass computation and expressivity (Pfau et al., 2024; London and Kanade, 2025), focal-loss amplification at hard tokens (Kim et al., 2025), and structural anchoring at sentence boundaries (Liu et al., 2026). These explanations center on effects at the inserted positions themselves. Yet much less is known about how pause training shapes the pre-trained distribution from which fine-tuning starts, even though this distribution is central to modern adaptation pipelines that increasingly warm-start from existing checkpoints (Gururangan et al., 2020; Roziere et al., 2023; Azerbayev et al., 2023).

We argue that this is precisely where an underexamined role of pause tokens is most consequential. Pause tokens also participate in the forward pass during fine-tuning, where they alter the context under which target tokens are predicted and reshape how gradient flows through the surrounding positions. (Wu et al., 2024; Pfau et al., 2024) These training-dynamics effects are present even when pause tokens are absent at inference and never appear as supervised targets, and they directly modulate the retention–adaptation trade-off central to fine-tuning. In a controlled continual-learning setting, we observe that loss-masked pauses reduce overwriting of a previously learned distribution by roughly $4\times$ at matched final adaptation, revealing a separation that neither inference-time nor low-confidence framings predict.

We investigate this through two controlled pilots that yield distinct empirical phenomena; a *retention asymmetry* on synthetic continual learning and an *information-packing asymmetry* on iGSM(Allen-Zhu and Li, 2024) probing, which we elevate to working hypotheses H1 (retention) and H2 (non-myopic compression). We then formalize a train-

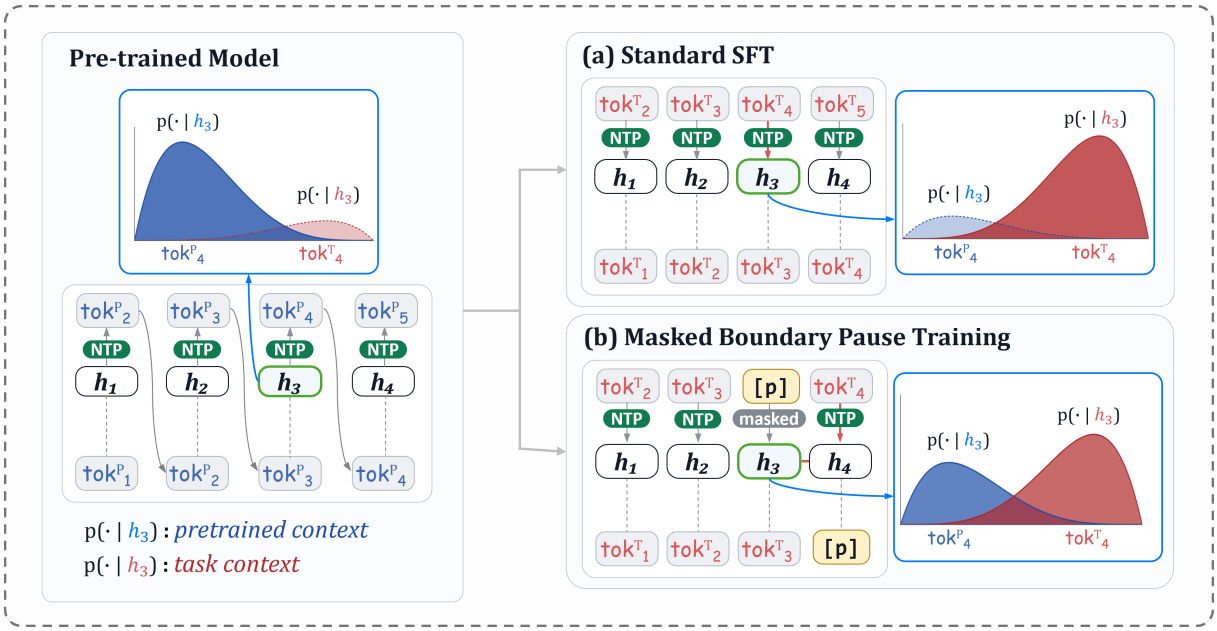

Figure 1: Comparison between standard fine-tuning and masked boundary pause fine-tuning. Standard SFT applies next-token prediction loss at every task-token position, allowing gradients from the target token to flow directly into the preceding context representation. This can strongly adapt the model to the task-data distribution, but may also erode the rich predictive distribution learned during pre-training. Masked boundary pause fine-tuning (MBP) instead inserts pause tokens at selected boundaries and masks their loss, thereby blocking the direct gradient path into the boundary context state. The model receives update signals only indirectly through neighboring token predictions, enabling task adaptation while better preserving the original pre-training distribution.

ing rule consistent with both hypotheses, **Masked Boundary Pause** (MBP), and test the hypotheses' scaled predictions across 1B–8B fine-tuning and GRPO. The headline finding is that *boundary placement and loss masking are jointly necessary* for the gains, with specific results reported in §5–§7.3.

In summary, our contributions are:

- Through controlled pilots, we identify two hypotheses: (H1) mode adaptation, and (H2) non-myopic compression.
- Motivated by these, we propose **Masked Boundary Pause (MBP)** fine-tuning, where pause tokens are inserted at sentence or reasoning-step boundaries while masking their cross-entropy loss.
- Across 1B–8B models, MBP improves math and code reasoning while preserving general capability; gains extend to GRPO. A four-way placement ablation shows that boundary placement and loss masking are jointly necessary.

2 Preliminaries

We formalize masked-pause fine-tuning under a single notation that subsequent sections re-use, and locate prior pause-token methods within the same framework.

2.1 Pause-token training

Formulation. Let (x, y) be a (prompt, response) pair drawn from a fine-tuning dataset $\mathcal{D}$, with response $y = (y_1, \ldots, y_N)$. Standard supervised fine-tuning (SFT) minimizes the per-token conditional cross-entropy of y given x; we recover it as the $g(x, y) = \emptyset$ case of the formulation below.

A pause-token training method is parameterized by a *position selector* g that returns insertion indices $g(x, y) = \{i_1 < i_2 < \cdots < i_M\} \subseteq \{0, \ldots, N\}$. With the conventions $i_0 = 0$ and $i_{M+1} = N$, these indices partition y into $M + 1$ contiguous segments

$$u_n = (y_{i_{n-1}+1}, \ldots, y_{i_n}), \quad n = 1, \ldots, M + 1, \tag{1}$$

with $y = u_1 \parallel u_2 \parallel \cdots \parallel u_{M+1}$. A distinguished pause token τ is inserted between adjacent segments, yielding the pause-augmented sequence

$$\tilde{y} = u_1 \parallel \tau \parallel u_2 \parallel \tau \parallel \cdots \parallel \tau \parallel u_{M+1} \tag{2}$$

of length $N + M$. Let $S_\tau = \{k : \tilde{y}_{k+1} = \tau\}$ be the set of prediction positions whose target is a pause token. Following prior work (Goyal et al., 2024;

Kim et al., 2025), we exclude S_τ from the loss: pause tokens appear in the autoregressive context but never as supervised outputs. The per-example training loss is

$$\mathcal{L}^g(\theta) = -\frac{1}{N} \sum_{k \notin S_\tau} \log p_\theta\big(\tilde{y}_{k+1} \mid x, \tilde{y}_{1:k}\big), \quad (3)$$

averaged over $(x, y) \sim \mathcal{D}$. Setting $g(x, y) = \emptyset$ recovers standard SFT.

Relation to existing work. Prior work differs in the position selector g: APPEND (Goyal et al., 2024; Pfau et al., 2024), DIT (Kim et al., 2025), and BOUNDARY at sentence endings (Liu et al., 2026). We additionally test RANDOM placement as an unstructured control. Of these, only Liu et al. (2026) trains pauses as predicted targets; the others adopt the masked treatment of Eq. 3. Our method, Masked Boundary Pause Training (§5), takes $g =$ BOUNDARY with S_τ masked, and Section 6.1 compares the four placements.

2.2 GRPO

Formulation. Group Relative Policy Optimization (GRPO) (Shao et al., 2024) optimizes a policy π_θ on prompts $q \in \mathcal{Q}$ by sampling G responses $\{y_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot \mid q)$ from the previous policy, scoring each with a reward $r_i = R(q, y_i)$, and computing the group-relative advantage $A_i = (r_i - \mu_r)/(\sigma_r + \epsilon)$ where μ_r, σ_r are the group's mean and standard deviation of rewards. With per-token policy ratio $\rho_{i,t}(\theta) = \pi_\theta(y_{i,t} \mid q, y_{i,<t})/\pi_{\theta_{\text{old}}}(y_{i,t} \mid q, y_{i,<t})$, GRPO maximizes

$$\begin{aligned} \mathcal{J}_{\text{GRPO}}(\theta) &= \mathbb{E}\left[\frac{1}{G}\sum_{i,t} c_{i,t}(\theta) - \beta D_{\text{KL}}\right], \\ c_{i,t}(\theta) &= \min\big(\rho_{i,t} A_i, \\ &\qquad \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon) A_i\big). \end{aligned} \quad (4)$$

where D_{KL} is a per-token KL penalty against a reference policy.

3 Pilot Experiments

The introduction posed a mechanistic question: what is masked-pause fine-tuning actually doing to a pretrained model's representations? We approach this in two controlled studies. The first asks whether masked pauses change *what gets overwritten* during adaptation (§3.1). The second asks *what gets encoded* at the boundary-adjacent token (§3.2). Each pilot yields a distinct empirical asymmetry that we crystallize into a working hypothesis.

3.1 Modular continual learning: a retention asymmetry

Our first pilot asks whether masked pauses change what gets overwritten during adaptation. We use the simplest setting that can support a clean before/after comparison, a synthetic continual-learning task with two disjoint regimes. Detailed experimental settings are provided in Appendix A.

Setup. Sequences are generated by a *Modular Linear Chain* (MLC),

$$x_{t+1} = (a_t\, x_t + b_t) \bmod p, \quad (5)$$

where a *regime* is defined by a length-L coefficient sequence $\{(a_t, b_t)\}_{t=1}^L$. We sample two disjoint coefficient sets to define regimes D_1 and D_2, and train a small GPT-style Transformer sequentially: Phase 1 trains on D_1 until accuracy exceeds 0.99; Phase 2 forks from a single Phase-1 checkpoint and continues on D_2 under two matched conditions sharing architecture, optimizer, batch size, learning rate, and update budget:

- NO PAUSE: standard next-token prediction on raw D_2 sequences.
- PAUSE: one pause token inserted after every content token; pause-target positions are excluded from the loss.

Results. Figure 2 shows the Phase-2 loss curves on both regimes. Both conditions adapt to D_2 to the same $\sim 10^{-7}$ floor, with NO PAUSE marginally faster in descent but indistinguishable at convergence. The conditions diverge sharply in how much they overwrite D_1: NO PAUSE plateaus near 6 nats of held-out D_1 loss while PAUSE plateaus near 1.5 — a $\sim 4\times$ retention gap, consistent across seeds. The gap occurs at *matched* D_2 fit and is not explained by tokens-processed per step.

Hypothesis. Masked-pause training matches the new regime's final fit while preserving the previous regime substantially better:

> **H1 (mode retention).** *Masked-pause training reduces overwriting of previously-learned distributions during adaptation, relative to standard fine-tuning at matched final performance.*

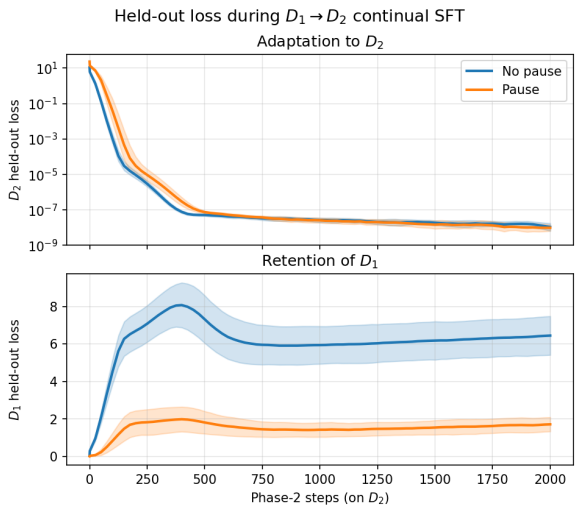

Figure 2: Phase-2 dynamics from a D_1-pretrained backbone (n=8 seeds, $\pm 1\sigma$). **Top:** D_2 adaptation (held-out loss, log scale). **Bottom:** D_1 retention (held-out loss, linear scale). Both conditions reach the same D_2 floor; PAUSE retains $D_1 \sim 4\times$ better than NO PAUSE.

3.2 iGSM step probing: an information-packing asymmetry

The MLC pilot shows masked pauses change adaptation dynamics, but not what the representation absorbs. To localize the mechanism, we probe what the *boundary-adjacent token* — the period immediately preceding an inserted [PAUSE] — encodes after training. We use iGSM (Allen-Zhu and Li, 2024), a synthetic mathematical reasoning dataset where each reasoning step concludes with a deterministic integer answer, providing the clean per-step probe targets this analysis requires.

Setup. We supervise-fine-tune Qwen3-0.6B-Base (Yang et al., 2025) on 3K iGSM problems under two conditions — NO PAUSE and PAUSE (one [PAUSE] inserted at each step boundary, loss masked) — and compare against the pretrained BASE checkpoint (n=3 seeds each). For each held-out problem we extract the final-layer hidden state at the boundary-adjacent period of every reasoning step and fit two linear probes: one for the integer answer of the *current* step, one for the *next* step.

Results. Figure 3 shows masked-pause SFT yielding the highest decodability on both targets: current-step accuracy roughly doubles over no pause (0.282 vs. 0.133, and clearly above the base 0.182), and next-step accuracy is highest as well (0.104 vs. 0.062 for no pause and 0.075 for base). The boundary-adjacent token under masked-pause SFT carries substantially more information about both the just-finished and the upcoming reasoning step. The pattern is consistent across seeds.

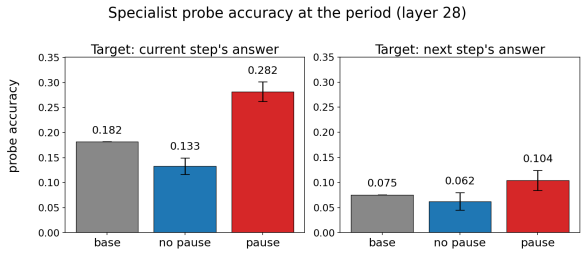

Figure 3: Linear-probe accuracy on the boundary-adjacent token's final hidden state (n=3 seeds, $\pm 1\sigma$). **Left:** current step's integer answer. **Right:** next step's integer answer.

Hypothesis. Under ordinary SFT, the period token is subject to next token prediction loss; under masked-pause SFT, that prediction is masked. The period's representation is therefore freed from immediate next-token specialization and can instead absorb gradient from downstream positions:

> **H2 (non-myopic compression).** *Masked pauses redirect gradient flow such that boundary-adjacent tokens encode more downstream information than under ordinary SFT.*

3.3 From hypotheses to predictions

Motivated by H1 and H2, the rest of the paper tests three predictions that follow if they hold at scale on real reasoning tasks:

- **RQ1 (scaling).** Does masked boundary pause fine-tuning at 1B–8B improve math and code reasoning while preserving general capability (MMLU (Hendrycks et al., 2020), GPQA (Rein et al., 2023), BBH (Suzgun et al., 2022)) better than standard fine-tuning? H1 predicts this trade-off.
- **RQ2 (placement).** Does pause placement affect performance at matched pause-token budget? If so, which strategy is optimal? H1 and H2 both act locally at the inserted positions, predicting that placement matters and that boundary insertion outperforms random, prompt-append (Goyal et al., 2024), and low-confidence (Kim et al., 2025) alternatives.
- **RQ3 (RL exploration).** Do GRPO rollouts under MBP retain wider behavioral support than standard GRPO — more per-prompt reward variance, slower distribution sharpening, and improved convergence? H1 predicts so.

Sections 5–7.3 formalize MBP and answer these questions via 1B–8B math/code SFT with OOD-retention measurement, and GRPO experiments.

4 A retention certificate for masked-pause SFT

H1 admits a first-order retention certificate. The intuition: old D_1 predictions on no-pause prefixes survive D_2 fine-tuning when their top-token margins outlast the logit movement induced by adaptation. Bounding this movement uniformly over a protected prefix set converts retention into a comparison between the certified drifts of two procedures. Let $\Phi(r)$ denote the fraction of protected D_1 prefixes whose original margin exceeds r.

Theorem 4.1 (Retention certificate, informal)**.** *In the local linearized regime around the D_1-trained parameters and matched to the same D_2 fitting level, let δ_O be the certified protected drift of ordinary SFT on no-pause D_1 logits. Masked-pause SFT has certified protected drift at most $\beta_{\text{ctx}}\delta_O$, where β_{ctx} measures how separated the pause-perturbed prefix is from the ordinary prefix in the Jacobian sense. Consequently,*

$$\text{Retain}(W_{\text{SFT}}) \geq \Phi(2\delta_O),$$
$$\text{Retain}(W_{\text{pause}}) \geq \Phi(2\beta_{\text{ctx}}\delta_O).$$

Since Φ is non-increasing, whenever $\beta_{\text{ctx}} < 1$, masked-pause SFT inherits a stronger retention guarantee than ordinary SFT.

The condition $\beta_{\text{ctx}} < 1$ has a geometric reading: masked-pause SFT fits D_2 with less drift on D_1 no-pause logits whenever the pause-perturbed state is sufficiently separated from the ordinary state. This separation is structural rather than architectural — a property of *where* pauses are inserted, which the boundary selector of MBP encodes by design. The full setup, lemma statements (margin stability, protected drift bounds), and proofs are in Appendix B, C.

5 Method

Motivated by H1 and H2 in §3, we formalize the masked pause training strategy that are predicted to exhibit optimal performance.

5.1 Masked Boundary Pause Training

MBP specializes the position selector g from §2 to a *boundary selector* g_{Boundary} that produce segments u_n of Eq. 1 that are *semantic units*: sentences, reasoning steps, or code lines.

Boundary placement is the advantageous choice for both hypotheses:

- **H1 (retention).** Semantic transitions are where fine-tuning's structural learning concentrates (Wang et al., 2025; Lin et al., 2024a). A pause embedding at these positions absorbs that learning, preserving the pretrained distribution that would otherwise drift (§3.1).
- **H2 (non-myopic compression).** Loss-masked positions encourage implicit summarization at the preceding token (§3.2), so semantic boundaries are the natural placement: there the compression encapsulates a complete semantic unit, rather than being forced to predict the next token mid-sentence.

The MBP training loss is Eq. 3 with $g = g_{\text{Boundary}}$, and all placement baselines in §6.1 use the same pause-token budget M so that comparisons isolate the role of g.

5.2 Extension to GRPO

MBP extends to GRPO without changing rollouts or rewards: responses are sampled and scored on the clean policy, while pauses appear only when computing the policy update. Given a sampled response y_i, we form $\tilde{y}_i$ (Eq. 2) and apply the GRPO objective at non-pause positions, with policy ratio and KL penalty computed under the pause-perturbed prefix $(q, \tilde{y}_{i,<j})$ rather than the clean prefix $(q, y_{i,<j})$. Pauses thus remain outside the action space, perturbing only the policy update's context — the mechanism through which we test RQ3 (§3.3): whether retention of the pretrained distribution improves GRPO's exploration.

6 Experiment Setup

We test the predictions of §3.3 in two settings: downstream math/code supervised fine-tuning (§6.1, RQ1 and RQ2) and GRPO-based reasoning training (§6.2, RQ3). Full hyperparameters and data details are in Appendix D.

6.1 Downstream Math/Code SFT

Setup. We fine-tune Qwen3 (1.7B, 4B, 8B-Base)(Yang et al., 2025) and Llama-3.1-8B-Instruct (Grattafiori et al., 2024) on math data (83K curated DeepMath-103K (He et al., 2025) subset) and code data (100K OpenCodeInstruct subset (Ahmad et al., 2025)), inserting pauses only on the response side. We apply lora for 8B models. We evaluate reasoning on math (GSM8K, MATH, AMC23,

AIME24, Minerva) (Cobbe et al., 2021; Hendrycks et al., 2021; Lewkowycz et al., 2022) and code (HumanEval(+), MBPP(+)) benchmarks; general-capability retention on MMLU, GPQA, HellaSwag, BBH, PIQA. We report pass@4.

Baselines (matched pause-token budget). SFT (no pauses); APPEND: pauses at the prompt-response boundary (Goyal et al., 2024); RANDOM: pauses at random response positions; DIT: pauses before low-likelihood target tokens (Kim et al., 2025); MBP (ours): pauses at reasoning-step or code-line boundaries. All pause variants are loss-masked. Matching the budget isolates placement from token count.

6.2 GRPO Setup

We compare STANDARD GRPO against MBP GRPO on Qwen3-1.7B and 4B, training on 5K-problem subsets of DeepMath-103K (He et al., 2025) and SimpleRLZoo (Zeng et al., 2025). We report pass@1 on the math benchmarks above, and track two training-dynamics metrics: convergence AUC (area under the training reward curve, measuring how quickly reward accumulates), and mixed-group ratio (proportion of rollout groups containing both correct and incorrect rollouts).

7 Results and Analysis

We answer RQ1 (§7.1), RQ2 (§7.2), and RQ3 (§7.3) in order, along with retention-related ablations. Across these analyses, the results support the predictions of H1 and H2.

7.1 Scaling and retention (RQ1)

Reasoning gains scale across model sizes and families. Tables 1 and 2 show that MBP outperforms standard SFT on every model variant we tested. Math gains over SFT range from +1.7 to +6.3 points across Qwen3 (1.7B-8B-Base) and Llama (3.2-1B/3B-Instruct, 3.1-8B-Instruct); code gains range from +0.6 to +2.5 across Qwen3. The improvement is consistent in sign across every model variant, showing that pause training is not a model-specific phenomenon.

General capability is preserved. Table 3 reports MMLU, GPQA, HellaSwag, BBH, and PIQA on Qwen3-1.7B-Base after math and code SFT. Standard SFT is at best neutral on general capability after math SFT and degrades it after code SFT. By contrast, MBP improves the general-benchmark average over base in both settings (+4.2 after math, +2.5 after code), beating SFT by +3.1 and +4.8, respectively. The MBP-vs-SFT gap on general benchmarks is larger than the gap on the trained-domain benchmarks themselves. This suggests that MBP does not trade reasoning gains for retention loss, but improves both simultaneously.

These findings support H1 at scale: MBP preserves the pretrained distribution while still acquiring new-task reasoning.

Model	Method	GSM8K	MATH500	MATH	AMC23	MINERVA	Avg.
Qwen3-1.7B Base	base	65.13	54.20	54.82	35.00	17.65	45.36
	SFT	82.41	**63.40**	62.98	27.50	20.59	51.38
	step	**83.02**	61.00	**64.56**	**37.50**	**23.16**	**53.85**
Qwen3-4B Base	base	82.56	66.00	67.64	37.50	28.68	56.48
	SFT	**91.36**	69.20	71.24	45.00	29.78	61.32
	step	90.07	**73.40**	**72.32**	**50.00**	**30.51**	**63.26**
Qwen3-8B Base	base	88.40	65.40	65.86	42.50	26.84	57.80
	SFT	**92.95**	71.60	72.54	52.50	32.72	64.46
	step	92.57	**75.00**	**72.84**	**55.00**	**35.29**	**66.14**
Llama3.2-1B Instruct	base	38.97	23.80	25.70	15.00	7.72	22.24
	SFT	50.72	28.00	38.48	10.00	8.09	27.06
	step	**51.63**	**31.80**	**41.66**	**27.50**	**14.34**	**33.39**
Llama3.2-3B Instruct	base	70.43	40.00	43.22	10.00	17.65	36.26
	SFT	76.80	46.40	55.02	27.50	13.97	43.94
	step	**77.03**	**48.20**	**55.92**	**30.00**	**19.85**	**46.20**
Llama3.1-8B Instruct	base	77.48	46.40	48.62	26.50	**22.43**	44.29
	SFT	84.84	53.20	56.28	17.50	21.69	46.70
	step	**86.28**	**58.00**	**61.02**	**27.50**	**22.43**	**51.05**

Table 1: Math-domain results. Average denotes the macro-average over GSM8K, MATH500, MATH, AMC23, and MINERVA.

Model	Method	MBPP	MBPP+	HumanEval	HumanEval+	Avg.
Qwen3-1.7B Base	base	69.05	59.26	58.54	53.05	59.98
	SFT	69.58	**62.43**	67.07	61.59	65.17
	step	**73.02**	**62.43**	**70.12**	**65.24**	**67.70**
Qwen3-4B Base	base	74.34	64.81	76.83	72.56	72.14
	SFT	78.04	**67.99**	79.27	73.78	74.77
	step	**79.10**	65.08	**82.93**	**78.05**	**76.29**
Qwen3-8B Base	base	82.80	72.49	**88.41**	**84.15**	81.96
	SFT	86.51	75.13	85.98	81.71	82.33
	step	**87.30**	**75.66**	86.59	82.32	**82.97**
Llama3.1-8B Instruct	base	67.46	59.52	67.68	62.20	64.22
	SFT	68.25	**60.05**	68.29	**64.02**	65.15
	step	**68.52**	**60.05**	**69.51**	**64.02**	**65.53**

Table 2: Code-domain results. Average denotes the macro-average over MBPP, MBPP+, HumanEval, and HumanEval+.

7.2 Placement and masking (RQ2)

H1 and H2 predict that boundary placement activates both mechanisms: placements without alignment to semantic transitions (append, random) or correlated with hard tokens rather than transitions (DIT) should fail to do so. We test this on Qwen3-

Trained Domain	Method	MMLU	GPQA	HellaSwag	BBH	PIQA	Avg.
Math	base	62.95	**27.46**	38.69	38.84	62.13	46.01
	sft	62.32	24.78	40.39	42.06	66.10	47.13
	step	**63.93**	27.23	**48.32**	**44.13**	**67.63**	**50.25**
Code	base	62.95	**29.91**	38.00	38.84	62.13	46.37
	sft	60.19	23.88	36.44	39.15	60.66	44.06
	step	**63.78**	25.67	**44.01**	**42.88**	**68.01**	**48.87**

Table 3: General-domain results on Qwen3-1.7B-Base. Average denotes the macro-average over MMLU, GPQA, HellaSwag, BBH, and PIQA.

Method	Math					Avg.
	GSM8K	MATH500	MATH	AMC23	MINERVA	
base	65.13	54.20	54.82	35.00	17.65	45.36
sft	82.41	**63.40**	62.98	27.50	20.59	51.38
append	79.68	59.20	60.96	30.00	22.79	50.53
random	81.35	60.80	63.26	20.00	**23.16**	49.71
dit	68.01	57.20	56.38	30.00	20.22	46.36
step	**83.02**	61.00	**64.56**	**37.50**	**23.16**	**53.85**
step_pred	80.97	62.20	62.64	27.50	**23.16**	51.29

Method	Code				Avg.
	MBPP	MBPP+	HumanEval	HumanEval+	
base	69.05	59.26	58.54	53.05	59.98
sft	69.58	**62.43**	67.07	61.59	65.17
append	69.84	60.58	70.73	**65.24**	66.60
random	70.37	61.11	67.07	62.80	65.34
dit	69.84	60.58	**71.95**	**65.24**	66.90
step	**73.02**	**62.43**	70.12	**65.24**	**67.70**
step_pred	70.63	58.99	67.68	62.20	64.88

Table 4: Method ablation results on Qwen3-1.7B-Base. The upper block reports math benchmarks, and the lower block reports code benchmarks.

1.7B-Base at matched pause-token budget.

Boundary placement is the only consistent winner. Table 4 reveals a 7.5-point spread across the four placements on math (from DIT's -5.0 to MBP's $+2.5$ relative to SFT), despite identical pause-token budgets. Of the four, only MBP is reliably positive on both math and code domains. The other placements fluctuate: Append and DIT modestly help code ($+1.4$, $+1.7$) but hurt math (-0.9, -5.0). Random is near-flat on code ($+0.2$) and clearly negative on math (-1.7). The inconsistency suggests that random, append, and low-confidence positions activate H1 and H2's mechanisms only when an inserted pause happens to coincide with a meaningful boundary, whereas BOUNDARY placement targets these positions by construction and delivers reliable gains across domains.

7.3 Extend to GRPO exploration (RQ3)

If MBP preserves the pretrained distribution (H1), GRPO rollouts should retain wider behavioral support, yielding a richer gradient signal. We test this on Qwen3-1.7B-Base and Qwen3-4B-Base.

Faster convergence, higher final accuracy. Table 5 shows MBP-GRPO improves final math accuracy by $+1.0$ (Qwen3-1.7B) and $+1.1$ (Qwen3-4B) over standard GRPO, with a more pronounced gain in convergence AUC ($+1.4$ pp and $+1.2$ pp).

Early rollout diversity drives the gain. Figure 4 reveals the mechanism. From the earliest steps, MBP-GRPO's mixed-group ratio spikes substantially above standard GRPO, generating more useful gradient updates when the advantage signal is most valuable. This translates immediately into the reward and accuracy curves. MBP-GRPO pulls ahead from the first steps and stays ahead throughout training. (Later, MBP's mixed-group ratio falls below standard's, consistent with earlier convergence on solved prompts.) The pattern matches H1 in that a retained pretrained distribution provides more diverse rollouts when they most help, and the early advantage compounds.

7.4 Retention analyses

We close with two analyses testing H1's retention mechanism directly: a decomposition of post-training correctness, and an ablation on loss-masking.

MBP preserves base-correct examples while solving new ones. BCP (Base-Correct Preservation) is the fraction of base-correct examples still correct after fine-tuning; NSR (New-Solve Rate) is the fraction of base-wrong examples that become correct. Table 6 shows MBP improves both on Qwen3-1.7B-Base: BCP $89.18 \rightarrow 90.00$ and NSR $38.20 \rightarrow 41.11$. MBP does not trade preservation for adaptation; it improves both, matching H1's prediction.

Masking is essential at boundary placement. H1 predicts masking matters: training pauses as targets would shift the inference distribution and contaminate retention. We test this by training with the same boundary placement but predicting pause tokens as targets (STEP_PRED in Table 4). Switching from masked to predicted costs -2.6 points on math and -2.8 on code on Qwen3-1.7B-Base; on code, predicted boundary placement even falls

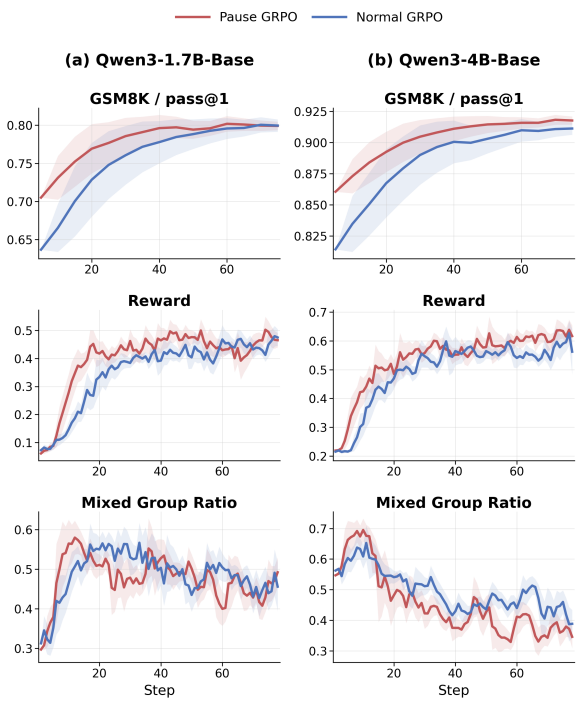

Figure 4: **Training dynamics of MBP-GRPO.** Rows correspond to Qwen3-1.7B-Base (top) and Qwen3-4B-Base (bottom), while columns show GSM8K validation pass@1 (left), training reward (middle), and mixed-group ratio (right). Red denotes MBP-GRPO and blue denotes standard GRPO. MBP-GRPO exhibits a higher mixed-group ratio in the early stage of training, providing more useful update signals and leading to faster convergence than standard GRPO.

below standard SFT, confirming that masking is load-bearing.

8 Related Work

Pause and Dummy Tokens. Prior works have utilized pause tokens to enhance LLM reasoning by delaying generation and increasing theoretical computational expressivity (Kim et al., 2025; Goyal et al., 2024). Building on this, recent approaches have explored dynamic token insertion at strategic points, like low-confidence positions or sentence boundaries, to better guide cognitive processing (Liu et al., 2026; Herel and Mikolov, 2024). Unlike these prior works that primarily view token insertion as a mechanism for increasing computational bandwidth or providing direct generation cues, our work uniquely isolates the role of pause tokens as training-time context perturbations.

Post-Training for Reasoning and Alignment. Contemporary post-training paradigms for LLMs aim to align behavior and enhance complex problem-solving, typically progressing from SFT to reinforcement learning frameworks such as GRPO (Wei et al., 2022; Chung et al., 2022; Shao et al., 2024). A critical challenge across these stages is the trade-off between reasoning adaptation and mode retention, as fine-tuning on a new reasoning task often degrades the model's previously established language fluency and factual knowledge (Luo et al., 2025; Zheng et al., 2025; Lin et al., 2024b). Our work is complementary to these post-training paradigms. We show that a lightweight token-insertion strategy can improve both SFT and GRPO when applied at semantic boundaries and masked from direct prediction, while mitigating the adaptation-retention trade-off.

Table 5: GRPO accuracy and normalized AUC.

Method	GSM8K	M500	MATH	AIME24	AMC23	Minerva	Avg.	AUC
Qwen3-1.7B-Base								
STANDARD	80.52	61.20	59.68	3.33	37.50	20.96	43.87	0.366
MBP	**81.35**	**61.80**	**61.24**	3.33	**40.00**	**21.32**	**44.84**	**0.414**
Qwen3-4B-Base								
STANDARD	**92.19**	74.40	74.30	13.33	50.00	**30.15**	55.73	0.505
MBP	91.58	**76.00**	**75.26**	**16.67**	**57.50**	23.90	**56.82**	**0.550**

Table 6: BCP(Base-Correct Preservation), NSR(New-Solve Rate), and final accuracy of Standard-SFT and MBP on Qwen3-1.7B-Base.

Metric	Method	GSM8K	MATH500	MATH	Overall
BCP ↑	STANDARD	92.08	**88.56**	88.33	89.18
	MBP	**93.36**	**88.56**	**89.09**	**90.00**
NSR ↑	STANDARD	63.04	**35.37**	33.42	38.20
	MBP	**66.74**	31.88	**36.83**	**41.11**
Acc. ↑	STANDARD	81.96	**64.20**	63.52	67.14
	MBP	**84.08**	62.60	**65.48**	**68.87**

9 Conclusions

In this work, we extend the study of pause tokens to their effect on fine-tuning dynamics. Through controlled pilot experiments, we identify two hypotheses, mode retention (H1) and non-myopic compression (H2), which naturally lead to Masked Boundary Pause Training. Through this lens, we show improvements in math and code reasoning while preserving general capability on unrelated benchmarks. We further show that boundary placement reliably outperforms SFT across model scales, as our hypotheses predict, and that retention also helps exploration in RL training. We hope this training-dynamics perspective, a shift from computational expressivity to train-time mode retention and non-myopic compression, supports further research across domains and adaptation regimes.

Limitations

Our current study focuses on full fine-tuning and evaluates retention primarily through per-example correctness, using BCP and NSR to measure whether base-correct examples remain correct and base-wrong examples become solved. Additionally, initial LoRA experiments show more moderate gains, suggesting that MBP's interaction with parameter-efficient fine-tuning requires further study. Future work should examine retention at finer token- or distribution-level granularity and systematically analyze MBP under parameter-efficient adaptation.

References

Wasi Uddin Ahmad, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Vahid Noroozi, Somshubra Majumdar, and Boris Ginsburg. 2025. Opencodeinstruct: A large-scale instruction tuning dataset for code llms.

Zeyuan Allen-Zhu and Yuanzhi Li. 2024. Physics of language models: Part 2.1, grade-school math and the hidden reasoning process. *Preprint*, arXiv:2407.20311.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*.

Mikhail S. Burtsev, Yuri Kuratov, Anton Peganov, and Grigory V. Sapunov. 2021. Memory transformer. *Preprint*, arXiv:2006.11527.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, and 16 others. 2022. Scaling instruction-finetuned language models. *Preprint*, arXiv:2210.11416.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. 2024. Think before you speak: Training language models with pause tokens. *Preprint*, arXiv:2310.02226.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 8342–8360.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *Preprint*, arXiv:2504.11456.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

David Herel and Tomas Mikolov. 2024. Thinking tokens for language modeling. *Preprint*, arXiv:2405.08644.

Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. 2024. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*.

Eunki Kim, Sangryul Kim, and James Thorne. 2025. Learning to insert [pause] tokens for better reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, page 23760–23777. Association for Computational Linguistics.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, and 1 others. 2022. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Bill Yuchen Lin, Abhilasha Ravichander, Ximing Lu, Nouha Dziri, Melanie Sclar, Khyathi Chandu, Chandra Bhagavatula, and Yejin Choi. 2024a. The unlocking spell on base LLMs: Rethinking alignment via in-context learning. In *The Twelfth International Conference on Learning Representations*.

Yong Lin, Hangyu Lin, Wei Xiong, Shizhe Diao, Jianmeng Liu, Jipeng Zhang, Rui Pan, Haoxiang Wang, Wenbin Hu, Hanning Zhang, Hanze Dong, Renjie Pi,

Han Zhao, Nan Jiang, Heng Ji, Yuan Yao, and Tong Zhang. 2024b. Mitigating the alignment tax of rlhf. *Preprint*, arXiv:2309.06256.

Zhichen Liu, Yongyuan Li, and Yang Xu. 2026. Think in sentences: Explicit sentence boundaries enhance language model's capabilities. *Preprint*, arXiv:2604.10135.

Charles London and Varun Kanade. 2025. Pause tokens strictly increase the expressivity of constant-depth transformers. *Preprint*, arXiv:2505.21024.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2025. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *Preprint*, arXiv:2308.08747.

Jacob Pfau, William Merrill, and Samuel R Bowman. 2024. Let's think dot by dot: Hidden computation in transformer language models. *arXiv preprint arXiv:2404.15758*.

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*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

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

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Yu Zhang, and 1 others. 2025. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *Preprint*, arXiv:2506.01939.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. *Preprint*, arXiv:2203.11171.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned language models are zero-shot learners. *Preprint*, arXiv:2109.01652.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Wilson Wu, John Xavier Morris, and Lionel Levine. 2024. Do language models plan ahead for future tokens? In *First Conference on Language Modeling*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. 2025. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *Preprint*, arXiv:2503.18892.

Junhao Zheng, Xidi Cai, Shengjie Qiu, and Qianli Ma. 2025. Spurious forgetting in continual learning of language models. *Preprint*, arXiv:2501.13453.

A Pilot Test Settings

A.1 MLC continual learning pilot

Task and data. Sequences are generated by the Modular Linear Chain (MLC) recurrence $x_{t+1} = (a_t\, x_t + b_t) \bmod p$ with prime $p = 37$ and sequence length $L = 8$. A *regime* is a fixed length-L coefficient sequence $\{(a_t, b_t)\}_{t=1}^{L}$ with each a_t, b_t drawn uniformly from $\{0, \ldots, p-1\}$. We sample two disjoint coefficient sets to define regimes D_1 and D_2. Vocabulary size is $p + 2$ (digits, BOS, and one pause token).

Model and optimizer. A 2-layer, 4-head GPT-style Transformer with hidden size $d = 128$ and learned positional embeddings. Trained with AdamW (learning rate 3×10^{-4}, weight decay 0.01), batch size 64.

Two-phase training. *Phase 1*: train on D_1 with standard next-token prediction until D_1 accuracy exceeds 0.99 (approximately 2,000 optimizer steps). *Phase 2*: fork from a single Phase-1 checkpoint and continue training on D_2 under two matched conditions (NO PAUSE and PAUSE), each run for an additional 2,000 steps. The PAUSE condition inserts one pause token after every content token, doubling sequence length; pause-target positions are excluded from the cross-entropy loss. Both conditions share architecture, optimizer, batch size, learning rate, and update budget. We report $n = 8$ seeds with shaded bands at $\pm 1\sigma$.

Evaluation. We track held-out loss on D_2 (adaptation) and D_1 (retention) every 5 Phase-2 steps. D_2 loss is computed on pause-augmented sequences for PAUSE and bare sequences for NO PAUSE; D_1 loss is bare for both conditions (matching the no-pause inference distribution).

A.2 iGSM probing pilot

Base model and data. Qwen3-0.6B-Base, fine-tuned on 3,000 problems sampled from the iGSM generator (Allen-Zhu and Li, 2024). We use the `max_op`=15, `max_edge`=20, `perm_level`=5 difficulty setting.

SFT. Two conditions, both trained for 2 epochs with learning rate 1×10^{-5}, batch size 32, AdamW optimizer, and bf16 precision: NO PAUSE (standard next-token prediction on bare iGSM solutions) and PAUSE (one `[PAUSE]` token inserted at every reasoning-step boundary, pause-target positions excluded from the loss). The BASE condition is the pretrained checkpoint with no fine-tuning. We report $n = 3$ seeds.

Probe training. For each held-out problem, we extract the final-layer hidden state at the period token immediately preceding each reasoning step boundary (the same character position across all three protocols). We train two separate linear probes on these representations: one predicting the integer answer of the *current* step, one predicting the integer answer of the *next* step. Probes are trained with a by-problem 70/30 split, with each problem's boundary positions assigned to either train or test based on the problem split.

B Theoretical analysis for masked-pause SFT

B.1 A first-order view of masked-pause mode retention

Old D_1 behaviours survive D_2 fine-tuning as long as their logits do not move enough to erase their old margins. We make this precise on a protected set of D_1 no-pause prefixes and then show that masked-pause SFT yields a smaller certified protected logit drift than ordinary SFT. The result is a first-order sufficient condition within the local linearized model around W_1; it certifies preservation of the D_1 top token on selected no-pause prefixes, not preservation of the full D_1 predictive distribution.

Margin stability. Whether an old top token survives depends only on whether its margin outlasts the logit movement at that prefix; bounding the movement uniformly over $\mathcal{C}_1$ converts this into a retention guarantee.

Lemma B.1 (Margin stability)**.** *Let W_1 be the shared parameters after D_1 training and $\mathcal{C}_1$ a set of D_1 no-pause prefixes whose top-token behaviour we protect. For $c \in \mathcal{C}_1$, let $a(c) = \arg\max_v z_{W_1}(c)_v$ be the D_1 top token and $\gamma(c)$ its top-vs-runner-up margin at W_1 — the amount a competing logit must close to flip the prediction at c. Define the margin survival function, the retention rate, and the protected logit drift*

$$\Phi(r) = \tfrac{1}{|\mathcal{C}_1|}\big|\{c \in \mathcal{C}_1 : \gamma(c) > r\}\big|,$$

$$\mathrm{Retain}(W) = \tfrac{1}{|\mathcal{C}_1|}\big|\{c \in \mathcal{C}_1 : \arg\max_v z_W(c)_v = a(c)\}\big|,$$

$$\Delta_1(W) = \max_{c \in \mathcal{C}_1} \|z_W(c) - z_{W_1}(c)\|_\infty,$$

with Φ non-increasing in r. Then

$$\Delta_1(W) \le \delta \implies \mathrm{Retain}(W) \ge \Phi(2\delta).$$

The factor of two appears because the old top logit may move down by δ while a competitor may move up by δ. Comparing the certified retention of two adaptation procedures therefore reduces to comparing their certified protected logit drifts on $\mathcal{C}_1$.

What masked-pause SFT changes. Ordinary SFT predicts the next D_2 target y_{i+1} from the no-pause prefix b_i; masked-pause SFT predicts the same target from the pause-perturbed prefix $b_i \circ p$ and masks the loss for the pause token p itself. Masking is essential: were p a training target, b_i would again receive a direct supervised gradient, restoring the interference channel masked-pause SFT removes. Four scalars control the resulting first-order drifts on $\mathcal{C}_1$. The interference exposure I_O is large when ordinary prefixes b_i push the parameters in directions that strongly move protected D_1 logits. Decomposing the Jacobian of $b_i \circ p$ into a part inherited from the ordinary boundary Jacobian and a pause-local residual yields a worst-case carry coefficient λ and a worst-case pause-local overlap ν with $\mathcal{C}_1$. Finally, $\kappa = M_P/M_O$ is the ratio of cumulative normalized shared-update masses the two procedures need to reach the same D_2 fitting level (matched D_2 validation loss in our experiments).

Lemma B.2 (Protected drift bounds)**.** *In the local linearized regime around W_1 with $I_O > 0$, the protected logit drifts of the ordinary and masked-pause endpoints satisfy*

$$\Delta_1(\widehat{W}_O) \le M_O\, I_O,$$
$$\Delta_1(\widehat{W}_P) \le M_P\big(\lambda I_O + \nu\big).$$

Equivalently, defining $\beta_{\mathrm{ctx}} := \kappa\big(\lambda + \nu/I_O\big)$, the masked-pause bound can be written as

$$\Delta_1(\widehat{W}_P) \;\le\; \beta_{\mathrm{ctx}}\, M_O I_O.$$

Both bounds share the ordinary scale $M_O I_O$, and in this certificate β_{ctx} is the factor multiplying the masked-pause drift bound. Feeding these bounds through Lemma B.1 turns the comparison into a retention guarantee.

Theorem B.3 (First-order context-perturbation retention certificate)**.** *Assume the local small-step regime around W_1, with ordinary and masked-pause SFT using the same normal D_2 targets, reaching the same D_2 fitting level, and $I_O > 0$. Combining the drift bounds of Lemma B.2 with the margin-stability bound of Lemma B.1,*

$$\mathrm{Retain}(\widehat{W}_O) \;\ge\; \Phi(2\, M_O I_O),$$
$$\mathrm{Retain}(\widehat{W}_P) \;\ge\; \Phi(2\beta_{\mathrm{ctx}}\, M_O I_O).$$

Masked-pause SFT yields the stronger certificate whenever $\beta_{\mathrm{ctx}} < 1$.

The condition $\beta_{\mathrm{ctx}} < 1$ admits a geometric reading: masked-pause SFT fits D_2 with less first-order drift on D_1 no-pause logits whenever the pause state $b_i \circ p$ is sufficiently separated from the ordinary boundary b_i in the Jacobian sense captured by λ and ν. This separation is structural rather than architectural — a property of the pause insertion policy, which our empirical preference for reasoning-step boundaries effectively encodes. The full setup, lemmas, proofs, and the role of masking and pause-insertion position are in Appendix B.

C Theoretical analysis for masked-pause SFT - Proofs

This appendix contains the full setup, lemmas, and proofs supporting Theorem B.3. The analysis is a local first-order sufficient-condition statement on a protected no-pause prefix set $\mathcal{C}_1$. It does *not* claim preservation of the full next-token distribution, of D_1 multimodality, or of the realized retained fraction; it compares certified lower bounds within the local linearized model around W_1.

C.1 Setup and notation

Let $\mathcal{V}$ be the vocabulary, W the ordinary shared model parameters, and e_p the pause embedding. We write $\circ$ for token-sequence concatenation, so $b_i \circ p$ denotes the prefix obtained by appending a single pause token p to b_i. For a no-pause prefix c, let $z_W(c) \in \mathbb{R}^{|\mathcal{V}|}$ be the next-token logit vector; for a prefix containing p, write $z_{W,e_p}(s)$ when the dependence on e_p should be explicit.

Let W_1 be the shared parameters after D_1 training and let $\mathcal{C}_1$ be the protected set of D_1 no-pause prefixes. For each $c \in \mathcal{C}_1$, the D_1 top token, its margin, and the margin survival function are

$$a(c) = \arg\max_{v \in \mathcal{V}} z_{W_1}(c)_v,$$
$$\gamma(c) = z_{W_1}(c)_{a(c)} - \max_{v \ne a(c)} z_{W_1}(c)_v,$$
$$\Phi(r) = \tfrac{1}{|\mathcal{C}_1|}\big|\{c \in \mathcal{C}_1 : \gamma(c) > r\}\big|.$$

Old-mode retention and protected no-pause logit drift are

$$\mathrm{Retain}(W) = \tfrac{1}{|\mathcal{C}_1|}\sum_{c\in\mathcal{C}_1} \mathbf{1}\big[\arg\max_v z_W(c)_v = a(c)\big],$$
$$\Delta_1(W) = \max_{c\in\mathcal{C}_1}\|z_W(c) - z_{W_1}(c)\|_\infty.$$

Because every prefix in $\mathcal{C}_1$ is a no-pause prefix, both quantities depend only on the shared parameters W; in particular, $\Delta_1(\widehat{W}_P)$ and $\mathrm{Retain}(\widehat{W}_P)$ do not directly involve the learned pause embedding $\widehat{e}_P$.

For a matrix B, the operator norm induced by ℓ_∞ is

$$\|B\|_{\infty\to\infty} = \sup_{u\neq 0}\frac{\|Bu\|_\infty}{\|u\|_\infty} = \max_j \sum_k |B_{jk}|,$$

and satisfies $\|XY\|_{\infty\to\infty} \le \|X\|_{\infty\to\infty}\|Y\|_{\infty\to\infty}$ whenever XY is defined.

C.2 Direct supervised-prefix relocation

Let $b_i = (x, y_1, \ldots, y_i)$ be a no-pause D_2 reasoning boundary and $\ell(z, y)$ the softmax cross-entropy loss for target y. The two methods give the per-step direct losses

$$\ell_i^O(W) = \ell\big(z_W(b_i),\, y_{i+1}\big),$$
$$\ell_i^P(W, e_p) = \ell\big(z_{W,e_p}(b_i \circ p),\, y_{i+1}\big),$$

with the pause-token prediction loss masked out in the second case. Thus the direct supervised prefix for the normal target y_{i+1} moves from b_i to $b_i \circ p$. Earlier tokens of b_i can still receive indirect gradient through the pause position; those contributions are folded into the pause-state Jacobian below.

We denote the resulting (multi)sets of supervised prefixes by

$$\mathcal{S}_O = \{b_i : \text{loss predicts a normal } D_2 \text{ token from } b_i\},$$
$$\mathcal{S}_P = \{b_i \circ p : \text{loss predicts the same normal token from } b_i \circ p\},$$

read as multisets if a prefix is supervised more than once.

C.3 First-order interference and Jacobians

For a prefix s, define the start-of-adaptation shared-parameter logit Jacobian

$$J_1(s) = \begin{cases} \left.\dfrac{\partial z_W(s)}{\partial W}\right|_{W_1} & s \text{ no-pause}, \\ \left.\dfrac{\partial z_{W,e_p^{(0)}}(s)}{\partial W}\right|_{W_1} & s \text{ contains } p, \end{cases}$$

where $e_p^{(0)}$ is the pause embedding at the start of D_2 adaptation; in the pause case, $e_p^{(0)}$ is held fixed and the derivative is taken with respect to W only. For $c \in \mathcal{C}_1$ and a supervised prefix s, the first-order interference matrix and exposure are

$$K_1(c, s) = J_1(c)J_1(s)^\top,$$
$$I_1(\mathcal{C}_1,) = \max_{c\in\mathcal{C}_1}\sum_{s\in}\|K_1(c, s)\|_{\infty\to\infty}.$$

The operator-norm choice matches the ℓ_∞ logit-drift quantity Δ_1. We write $I_O = I_1(\mathcal{C}_1, \mathcal{S}_O)$. Operationally, I_O is large when ordinary D_2 supervised prefixes share gradient directions with protected D_1 prefixes.

C.4 Pause-state Jacobian decomposition

For each paired boundary, write $J_i^O = J_1(b_i)$ and $J_i^P = J_1(b_i \circ p)$. Let P_i^O be the orthogonal projection in parameter space onto the row space of J_i^O, acting by right-multiplication, so that $J_i^P P_i^O$ projects each row of J_i^P onto $\mathrm{row}(J_i^O)$. Define

$$J_i^{\mathrm{carry}} = J_i^P P_i^O,$$
$$J_i^{\mathrm{loc}} = J_i^P(I - P_i^O),$$

so $J_i^P = J_i^{\mathrm{carry}} + J_i^{\mathrm{loc}}$. Because every row of J_i^{carry} lies in $\mathrm{row}(J_i^O)$, there exists A_i with $J_i^{\mathrm{carry}} = A_i J_i^O$, and we take the canonical minimum-norm choice

$$A_i = J_i^P(J_i^O)^+,$$

where $(\cdot)^+$ is the Moore–Penrose pseudoinverse. If $J_i^O = 0$, set $A_i = 0$. The carry and pause-local interference summaries are

$$\lambda = \max_i \|A_i^\top\|_{\infty\to\infty},$$
$$\nu = \max_{c\in\mathcal{C}_1}\sum_i \|J_1(c)(J_i^{\mathrm{loc}})^\top\|_{\infty\to\infty}.$$

Operationally, λ is small when the pause state does not reuse the same gradient-sensitive directions

as the ordinary boundary, and ν is small when the pause-local residual does not align with the gradient-sensitive directions of protected D_1 prefixes. Whether λ and ν are small for a given architecture and insertion policy is a separate structural question; see Section C.9.

C.5 Training-regime quantities

Remark C.1 (Local small-step regime). The D_2 adaptation trajectory is analyzed in a neighbourhood of W_1 in which the linearization $z_W(s) \approx z_{W_1}(s) + J_1(s)(W - W_1)$ is accurate; for the full nonlinear model, the same statements hold up to a standard local linearization error in Δ_1.

For $A \in \{O, P\}$, let $w_t^A(s)$ be the normalized weight of supervised prefix s in the minibatch loss at step t and η_t the learning rate. Define

$$m_A(s) = \sum_t \eta_t w_t^A(s),$$

$$M_A = \max_s m_A(s), \qquad \kappa = \frac{M_P}{M_O}.$$

M_A is the largest cumulative normalized update mass assigned to any single supervised prefix along the trajectory. Throughout this appendix, "the same D_2 fitting level" refers to matched D_2 validation loss (or task accuracy, when that is the empirical comparison metric used in the main paper); $\kappa > 1$ then quantifies the additional shared-parameter update mass masked-pause SFT requires, if any, to reach that level.

C.6 Additional lemmas

(Lemmas B.1 and B.2 are stated in the main text; their proofs are included below.)

Lemma C.2 (Direct-loss relocation)**.** *Write $J_W(s) = \partial z_W(s)/\partial W$ and $J_{W|e_p}(s) = \partial z_{W,e_p}(s)/\partial W$ (with e_p held fixed). Then*

$$\nabla_W \ell_i^O(W) = J_W(b_i)^\top r_i^O,$$

$$\nabla_W \ell_i^P(W, e_p) = J_{W|e_p}(b_i \circ p)^\top r_i^P,$$

with $r_i^O = \nabla_z \ell(z_W(b_i), y_{i+1})$ and $r_i^P = \nabla_z \ell(z_{W,e_p}(b_i \circ p), y_{i+1})$. There is no direct loss term of the form $\ell(z_W(b_i), p)$, because the pause-token prediction loss is masked. Under the local-linearized convention, J_W and $J_{W|e_p}$ reduce to J_1 at the start of D_2 adaptation.

Lemma C.3 (Interference separation)**.** *Under the canonical decomposition of Section C.4,*

$$I_1(\mathcal{C}_1, \mathcal{S}_P) \;\le\; \lambda\, I_O + \nu.$$

If $I_O > 0$, equivalently $I_1(\mathcal{C}_1, \mathcal{S}_P) \le (\lambda + \nu/I_O)\, I_O = \alpha_{\text{ctx}}\, I_O$.

Lemma C.4 (Interference controls drift)**.** *In the local linearized regime, any D_2 adaptation procedure with supervised multiset and normalized update masses $m(s)$ satisfies*

$$\begin{aligned}\Delta_1(W) &\le \max_{c \in \mathcal{C}_1} \sum_{s \in} m(s) \|K_1(c, s)\|_{\infty \to \infty} \\ &\le M\, I_1(\mathcal{C}_1,),\end{aligned}$$

where $M = \max_s m(s)$.

C.7 Proofs

Proof of Lemma B.1 (margin stability). Fix $c \in \mathcal{C}_1$ with $\gamma(c) > 2\delta$ and any $v \neq a(c)$. Since each logit moves by at most δ,

$$\begin{aligned} & z_W(c)_{a(c)} - z_W(c)_v \\ & \quad \ge \big(z_{W_1}(c)_{a(c)} - z_{W_1}(c)_v\big) - 2\delta \\ & \quad \ge \gamma(c) - 2\delta \;>\; 0. \end{aligned}$$

Thus $a(c)$ remains the top token under W, and the fraction of c with $\gamma(c) > 2\delta$ lower-bounds $\mathrm{Retain}(W)$. □

Proof of Lemma C.2. Ordinary SFT predicts y_{i+1} from b_i, so its CE gradient is attached to the output state at b_i; the chain rule gives the stated form. Masked-pause SFT inserts p after b_i, masks the would-be loss for predicting p from b_i, and predicts y_{i+1} from the pause-state output position $b_i \circ p$; again the chain rule gives the stated form. Indirect gradient on earlier tokens flows through $J_{W|e_p}(b_i \circ p)$ and is handled by the carry/local decomposition. □

Proof of Lemma C.3. Fix $c \in \mathcal{C}_1$. Using $J_i^P = A_i J_i^O + J_i^{\text{loc}}$,

$$\begin{aligned} K_1(c, b_i \circ p) &= J_1(c)(J_i^P)^\top \\ &= K_1(c, b_i)\, A_i^\top + J_1(c)(J_i^{\text{loc}})^\top. \end{aligned}$$

By submultiplicativity of $\|\cdot\|_{\infty \to \infty}$,

$$\begin{aligned} \|K_1(c, b_i \circ p)\|_{\infty \to \infty} \le \lambda \|K_1(c, b_i)\|_{\infty \to \infty} \\ + \|J_1(c)(J_i^{\text{loc}})^\top\|_{\infty \to \infty}. \end{aligned}$$

Summing over i and taking $\max_{c \in \mathcal{C}_1}$ on both sides, with $\max(A(c) + B(c)) \le \max A(c) + \max B(c)$, gives $I_1(\mathcal{C}_1, \mathcal{S}_P) \le \lambda I_O + \nu$. □

Proof of Lemma C.4. In the linearized regime, the step is $\Delta W_t = -\eta_t \sum_s w_t(s) J_1(s)^\top r_t(s)$, where the softmax CE residual $r_t(s) = q_t(s) - e_{y_s}$ satisfies $\|r_t(s)\|_\infty \le 1$ uniformly along the trajectory. For $c \in \mathcal{C}_1$, summing first-order logit changes,

$$\begin{aligned}\|\Delta z(c)\|_\infty &\le \sum_t \eta_t \sum_s w_t(s) \\ &\qquad \|K_1(c,s)\|_{\infty\to\infty} \|r_t(s)\|_\infty \\ &\le \sum_s m(s) \|K_1(c,s)\|_{\infty\to\infty}.\end{aligned}$$

Taking $\max_{c\in\mathcal{C}_1}$ and then $M = \max_s m(s)$ gives both inequalities. □

Proof of Lemma B.2. Applying Lemma C.4 to the ordinary supervised multiset $\mathcal{S}_O$ gives $\Delta_1(\widehat{W}_O) \le M_O\, I_1(\mathcal{C}_1, \mathcal{S}_O) = M_O I_O$. For masked-pause SFT, Lemma C.4 on $\mathcal{S}_P$ followed by Lemma C.3 gives

$$\begin{aligned}\Delta_1(\widehat{W}_P) &\le M_P\, I_1(\mathcal{C}_1, \mathcal{S}_P) \\ &\le M_P\big(\lambda I_O + \nu\big).\end{aligned}$$

When $I_O > 0$, factoring $M_O I_O$ out of the last bound gives

$$\begin{aligned}M_P\big(\lambda I_O + \nu\big) &= \frac{M_P}{M_O}\left(\lambda + \frac{\nu}{I_O}\right) M_O I_O \\ &= \beta_{\text{ctx}}\, M_O I_O. \qquad \square\end{aligned}$$

Proof of Theorem B.3. By Lemma B.2, $\Delta_1(\widehat{W}_O) \le M_O I_O$ and $\Delta_1(\widehat{W}_P) \le \beta_{\text{ctx}}\, M_O I_O$. Applying Lemma B.1 with $\delta = M_O I_O$ and $\delta = \beta_{\text{ctx}}\, M_O I_O$ respectively yields

$$\begin{aligned}&\text{Retain}(\widehat{W}_O) \ge \Phi(2\, M_O I_O), \\ &\text{Retain}(\widehat{W}_P) \ge \Phi(2\beta_{\text{ctx}}\, M_O I_O).\end{aligned}$$

Since Φ is non-increasing and $I_O > 0$, the condition $\beta_{\text{ctx}} < 1$ gives $\beta_{\text{ctx}}\, M_O I_O < M_O I_O$, so the masked-pause certificate dominates the ordinary-SFT one. The dominance is strict whenever a nonzero fraction of $\mathcal{C}_1$ has $\gamma(c) \in (2\beta_{\text{ctx}}\, M_O I_O,\, 2\, M_O I_O]$. □

C.8 Why masking the pause-token loss matters

If the pause-token prediction loss is not masked, each inserted pause introduces an extra direct loss $\ell_i^{\text{pause}}(W) = \ell(z_W(b_i), p)$ whose gradient is attached to the original no-pause boundary b_i. This re-introduces a supervised multiset $\mathcal{S}_{\text{pause}} = \{b_i : \text{loss predicts } p \text{ from } b_i\}$ and changes the drift certificate to

$$\begin{aligned}\Delta_1(\widehat{W}_P^{\text{unmask}}) &\le M_{\text{norm}}(\lambda I_O + \nu) \\ &\qquad + M_{\text{pause}}\, I_{\text{pause}},\end{aligned}$$

where $I_{\text{pause}} = I_1(\mathcal{C}_1, \mathcal{S}_{\text{pause}})$. Since $\mathcal{S}_{\text{pause}}$ consists of the original no-pause boundaries, this added term can erase the benefit of context perturbation. Masking removes this loss and leaves Theorem B.3 clean.

C.9 Pause insertion position

An insertion policy A determines $\mathcal{S}_P^A$ and therefore changes $\lambda(A)$, $\nu(A)$, and $\kappa(A) = M_P^A/M_O$, giving $\beta_{\text{ctx}}(A) = \kappa(A)(\lambda(A) + \nu(A)/I_O)$. The theorem applies as soon as $\beta_{\text{ctx}}(A) < 1$. The remark below collects the qualitative considerations; it is not part of the formal theorem.

Remark C.5 (Why reasoning-step boundaries are natural insertion points). (i) At a reasoning-step boundary the previous step has ended, so the pause state $b_i \circ p$ is less pressured to copy a mid-phrase token representation, which tends to reduce the carry coefficient $\lambda(A)$. (ii) The pause-local residual at a complete boundary need not preserve ordinary syntactic-continuation features tied to D_1, which tends to reduce $\nu(A)$. (iii) Predicting the next reasoning step from a boundary-plus-pause prefix is still a natural next-token task, so the fitting-budget ratio $\kappa(A)$ stays moderate. Insertion that is too sparse leaves most supervised prefixes close to $\mathcal{S}_O$; insertion at unnatural dense positions can force the pause state to copy the immediately preceding prefix, increasing $\lambda(A)$, $\nu(A)$, or $\kappa(A)$.

D Experimental Settings

D.1 Hyperparameter settings

We report the hyperparameter settings used for each training stage. Across experiments, we use FusedAdam as the optimizer, CosineAnnealingLR as the learning-rate scheduler, a warmup ratio of 0.03, and bf16 precision for training, while adapting the learning rate and batch configuration to each training objective and model family.

For SFT on mathematical reasoning data, we train each model for 3 epochs with a batch size of 32. The learning rate is selected depending on the model family: 5×10^{-6} for Qwen models, 1×10^{-5} for Llama3.2, and 5×10^{-5} for Llama3.1. The full set of hyperparameters for math SFT is provided in Table 7.

Hyperparameter	SFT on math
Learning rate	Qwen: 5×10^{-6}; Llama3.2: 1×10^{-5}; Llama3.1: 5×10^{-5}
Epochs	3
Batch size	32
Optimizer	FusedAdam
Scheduler	CosineAnnealingLR
Warmup ratio	0.03
Training dtype	bf16

Table 7: SFT hyperparameters for mathematical reasoning training.

Hyperparameter	SFT on code
Learning rate	Qwen3-1.7B-Base: 5×10^{-7}; Qwen3-4B-Base: 5×10^{-6}; Qwen3-8B-Base: 5×10^{-5}; Llama3.1-8B-Instruct: 1×10^{-8}
Epochs	3
Batch size	32
Optimizer	FusedAdam
Scheduler	CosineAnnealingLR
Warmup ratio	0.03
Training dtype	bf16

Table 8: SFT hyperparameters for code generation training.

For SFT on code generation data, we also train for 3 epochs with a batch size of 32, using the same optimizer, scheduler, warmup ratio, and training precision as in math SFT. The learning rate is model-specific: 5×10^{-7} for Qwen3-1.7B-Base, 5×10^{-6} for Qwen3-4B-Base, 5×10^{-5} for Qwen3-8B-Base, and 1×10^{-8} for Llama3.1-8B-Instruct. The complete configuration for code SFT is summarized in Table 8.

For GRPO training, we use a learning rate of 5×10^{-7}, train for 1 episode, and use a batch size of 128. For each prompt, we sample 8 rollouts and apply a KL coefficient of 0.001. The optimizer, scheduler, warmup ratio, and training dtype are kept consistent with the SFT settings. The GRPO hyperparameters are listed in Table 9.

D.2 Computation resources

All training experiments in this paper were conducted with OpenRLHF on 2× NVIDIA RTX PRO 6000 and 2× NVIDIA RTX A100 GPUs.

D.3 Evaluation setup

For mathematical reasoning benchmarks, we report pass@1 accuracy under a deterministic decoding setting with temperature 0.0. For code generation

Hyperparameter	GRPO
Learning rate	5×10^{-7}
Episodes	1
Batch size	128
Rollouts per prompt	8
Optimizer	FusedAdam
Scheduler	CosineAnnealingLR
Warmup ratio	0.03
Training dtype	bf16
KL coefficient	0.001

Table 9: GRPO hyperparameters.

benchmarks, we report pass@1 using the EvalPlus package, and a prediction is counted as correct only when it passes all test cases. For general benchmarks, we report pass@4 accuracy with temperature 0.7 using the LightEval package.

E Implementation Details

We implemented most components using OpenRLHF(Hu et al., 2024), and used the default OpenRLHF settings for hyperparameters not explicitly specified.

E.1 Objective function

Let x denote the prompt and $y = (y_1, \ldots, y_T)$ denote the target response tokens. For a prefix s, write $p_\theta(\cdot \mid s)$ for the model distribution and $\ell_\theta(s, a) = -\log p_\theta(a \mid s)$ for the next-token cross-entropy loss. Standard SFT optimizes

$$\mathcal{L}_{\text{SFT}} = \frac{1}{T}\sum_{t=1}^{T} \ell_\theta(x \circ y_{<t}, y_t).$$

For pause-based training, let $u = (u_1, \ldots, u_L)$ be the sequence obtained after inserting pause tokens into y, where each u_j is either a normal target token or the pause token p. Masked-pause SFT uses the same next-token CE loss, but removes the loss terms whose target is p:

$$m_j = \mathbf{1}[u_j \neq p],$$
$$\mathcal{L}_{\text{mask}} = \frac{1}{\sum_j m_j}\sum_{j=1}^{L} m_j\, \ell_\theta(x \circ u_{<j}, u_j).$$

Thus the model is trained to predict normal answer tokens from prefixes that may contain pauses, while it is not directly trained to emit the pause token itself. The pause-predicted variant keeps the same inserted sequence u, but does not mask pause

targets:

$$\mathcal{L}_{\text{pred}} = \frac{1}{L} \sum_{j=1}^{L} \ell_\theta(x \circ u_{<j}, u_j).$$

The distinction is important: masked-pause SFT changes the context used for predicting ordinary tokens, whereas pause-predicted SFT additionally adds direct supervision for predicting p.

E.2 Examples of Methods

The following examples illustrate the actual data formats used for different pause-token placement strategies, including MBP (Masked Boundary Pause), Random, Append, and DIT. Examples for math data are shown in Table 10, while examples for code data are provided in Table 11 and Table 12.

F Additional Results

This section reports the full SFT evaluation results that are not shown in the main text. We include mathematical reasoning, code generation, and general benchmark results for Qwen3-1.7B-Base, Qwen3-4B-Base, Qwen3-8B-Base, Llama3.2-1B-Instruct, Llama3.2-3B-Instruct, and Llama3.1-8B-Instruct. The tables compare the base models, standard SFT, and the pause-insertion variants Pause_append, Pause_random, Pause_dit, Pause_step, and Pause_step_predicted where available. The full results for math and general language understanding tasks are shown in Table 13, while the full results for code and general language understanding tasks are presented in Table 14.

Method	Math example
MBP	**Question.** Round each number to the nearest tenth and add the results: 543.55, 25.23, and 299.98. **Answer.** <|PAUSE|>First, let's round each number to the nearest tenth: <|PAUSE|>1. 543.55 rounds to 543.6 (since the hundredths place is 5). <|PAUSE|>2. 25.23 rounds to 25.2 (since the hundredths place is less than 5). <|PAUSE|>3. 299.98 rounds to 300.0 (since the hundredths place is 8). <|PAUSE|>Now, let's add the rounded numbers: <|PAUSE|>543.6 + 25.2 + 300.0 = 868.8. <|PAUSE|>So, the final answer is $\boxed{868.8}$.
append	**Question.** Round each number to the nearest tenth and add the results: 543.55, 25.23, and 299.98.<|PAUSE|><|PAUSE|><|PAUSE|><|PAUSE|><|PAUSE|><|PAUSE|><|PAUSE|><|PAUSE|><|PAUSE|> **Answer.** First, let's round each number to the nearest tenth: 1. 543.55 rounds to 543.6 (since the hundredths place is 5). 2. 25.23 rounds to 25.2 (since the hundredths place is less than 5). 3. 299.98 rounds to 300.0 (since the hundredths place is 8). Now, let's add the rounded numbers: 543.6 + 25.2 + 300.0 = 868.8. So, the final answer is $\boxed{868.8}$.
random	**Question.** Round each number to the nearest tenth and add the results: 543.55, 25.23, and 299.98. **Answer.** First, let's round each number to the nearest tenth: 1. 543.55 rounds to 5<|PAUSE|>4<|PAUSE|>3.6 (since the hundredths place is 5). 2. 25.23 rounds to 25.<|PAUSE|>2 (since the hundred<|PAUSE|>ths place is less than 5). 3. 299.98 rounds to 3<|PAUSE|>00.0 (since the hundredths place is 8). Now, let's add the rounded numbers: 543.6 + 25.2 + 300.0 = 868.8. So, the final<|PAUSE|> answer is \boxed<|PAUSE|>{868.8}.
dit	**Question.** Round each number to the nearest tenth and add the results: 543.55, 25.23, and 299.98. **Answer.** <|PAUSE|>First,<|PAUSE|> let's round each number to the nearest tenth: <|PAUSE|>1. 543.55 rounds to 543.6 (since the hundredths place is 5<|PAUSE|>). 2. 25.23 rounds to 25.2 (since the hundredths place is<|PAUSE|> less than 5). 3. 299.98 rounds to 300.0 (since the hundredths place is 8). Now, let's add the rounded numbers: 543.6 + 25.2 + 300.0 = 868.8. So, the<|PAUSE|> final answer is<|PAUSE|> $\boxed{868.8}$.

Table 10: Examples of pause-token placement for the math task.

Method	Code example

MBP	**Question.** Ninja Atsushi stands guard atop the Ninja Building, which consists of two adjacent buildings of equal height. Each building has a series of walls of varying heights. Atsushi must jump between buildings to patrol the rooftops. Given the heights of the walls for both buildings, determine the minimum height Atsushi must jump for each wall position. Input: two lists of integers, `heights_building_1` and `heights_building_2`, with the same length. Output: a list of integers where each element is the minimum height Atsushi must jump for each wall position. Constraints: the common list length is between 1 and 10^5, and each wall height is between 1 and 10^9. Sample input: `heights_building_1 = [3, 6, 2]`, `heights_building_2 = [5, 4, 8]`. Sample output: `[5, 6, 8]`.
	Answer.
	`<|PAUSE|>def min_jumps(heights_building_1,`
	`  heights_building_2):`
	`  <|PAUSE|>Calculate the minimum height Atsushi must jump`
	`  for each wall position between two buildings.`
	`  <|PAUSE|>:param heights_building_1: list of integers representing`
	`  the heights of walls in the first building.`
	`  <|PAUSE|>:param heights_building_2: list of integers representing`
	`  the heights of walls in the second building.`
	`  <|PAUSE|>:return: list of integers representing`
	`  the minimum jump heights for each wall position.`
	`  <|PAUSE|>if len(heights_building_1) != len(heights_building_2):`
	`  <|PAUSE|>raise ValueError("Both buildings must have the same number of walls.").`
	`  <|PAUSE|>return [`
	`  max(heights_building_1[i], heights_building_2[i])`
	`  for i in range(len(heights_building_1))`
	`  ].`

append	**Question.** Ninja Atsushi stands guard atop the Ninja Building, which consists of two adjacent buildings of equal height. Each building has a series of walls of varying heights. Atsushi must jump between buildings to patrol the rooftops. Given the heights of the walls for both buildings, determine the minimum height Atsushi must jump for each wall position. Input: two lists of integers, `heights_building_1` and `heights_building_2`, with the same length. Output: a list of integers where each element is the minimum height Atsushi must jump for each wall position. Constraints: the common list length is between 1 and 10^5, and each wall height is between 1 and 10^9. Sample input: `heights_building_1 = [3, 6, 2]`, `heights_building_2 = [5, 4, 8]`. Sample output: `[5, 6, 8]`.<|PAUSE|><|PAUSE|> <|PAUSE|><|PAUSE|><|PAUSE|><|PAUSE|><|PAUSE|><|PAUSE|><|PAUSE|>
	<|PAUSE|><|PAUSE|>
	Answer.
	`def min_jumps(heights_building_1,`
	`  heights_building_2):`
	`  Calculate the minimum height Atsushi must jump`
	`  for each wall position between two buildings.`
	`  :param heights_building_1: list of integers representing`
	`  the heights of walls in the first building.`
	`  :param heights_building_2: list of integers representing`
	`  the heights of walls in the second building.`
	`  :return: list of integers representing`
	`  the minimum jump heights for each wall position.`
	`  if len(heights_building_1) != len(heights_building_2):`
	`  raise ValueError("Both buildings must have the same number of walls.").`
	`  return [`
	`  max(heights_building_1[i], heights_building_2[i])`
	`  for i in range(len(heights_building_1))`
	`  ].`

Table 11: Examples of pause-token placement for the code task.

Method	Code example

random	**Question.** Ninja Atsushi stands guard atop the Ninja Building, which consists of two adjacent buildings of equal height. Each building has a series of walls of varying heights. Atsushi must jump between buildings to patrol the rooftops. Given the heights of the walls for both buildings, determine the minimum height Atsushi must jump for each wall position. Input: two lists of integers, `heights_building_1` and `heights_building_2`, with the same length. Output: a list of integers where each element is the minimum height Atsushi must jump for each wall position. Constraints: the common list length is between 1 and 10^5, and each wall height is between 1 and 10^9. Sample input: `heights_building_1 = [3, 6, 2]`, `heights_building_2 = [5, 4, 8]`. Sample output: `[5, 6, 8]`.
	Answer.
	`def min_jumps(heights_building<|PAUSE|>_1,`
	`<|PAUSE|> heights_building_2):`
	`Calculate the minimum height Atsushi must jump`
	`for each wall position between two buildings.`
	`:param heights_building_1: list of integers representing`
	`the heights of walls in the first building.`
	`<|PAUSE|>:param heights_building_2: list of integers representing`
	`the heights of walls in the second building.`
	`:return: list of integers representing`
	`the minimum jump heights for each wall position<|PAUSE|>.`
	`if len<|PAUSE|>(height<|PAUSE|>s_building_1) != len(heights_building_2):`
	`raise ValueError("Both buildings must have the same number of walls.").`
	`return [`
	`max(heights_building_1[i], heights_building_2<|PAUSE|>[i<|PAUSE|>])`
	`for i in range<|PAUSE|>(len(heights_building_1<|PAUSE|>))`
	`].`

dit	**Question.** Ninja Atsushi stands guard atop the Ninja Building, which consists of two adjacent buildings of equal height. Each building has a series of walls of varying heights. Atsushi must jump between buildings to patrol the rooftops. Given the heights of the walls for both buildings, determine the minimum height Atsushi must jump for each wall position. Input: two lists of integers, `heights_building_1` and `heights_building_2`, with the same length. Output: a list of integers where each element is the minimum height Atsushi must jump for each wall position. Constraints: the common list length is between 1 and 10^5, and each wall height is between 1 and 10^9. Sample input: `heights_building_1 = [3, 6, 2]`, `heights_building_2 = [5, 4, 8]`. Sample output: `[5, 6, 8]`.
	Answer.
	`<|PAUSE|>def min<|PAUSE|>_jumps(`
	`heights_building_1, heights_building_2):`
	`Calculate the minimum height Atsushi must jump`
	`for each wall position<|PAUSE|> between two buildings.`
	`<|PAUSE|>:param heights_building_1: list of integers representing`
	`the heights of<|PAUSE|> walls in the first building.`
	`:param heights_building_2: list of integers representing`
	`the heights of walls in the second building.`
	`:return: list of integers representing the minimum<|PAUSE|>`
	`jump heights for each wall position.`
	`<|PAUSE|>if len(heights_building_1) != len(heights_building_2):`
	`raise ValueError("Both buildings must have the same number of walls<|PAUSE|>.").`
	`<|PAUSE|>return [`
	`max(heights_building_1[i], heights_building_2[i])`
	`for i in range(len(heights_building_1))`
	`].<|PAUSE|>`

Table 12: Examples of pause-token placement for the code task, continued.

Method	GSM8K	MATH500	MATH	AIME24	AMC23	MINERVA	MMLU	GPQA	HellaSwag	BBH	PIQA	Average
Qwen3-1.7B-Base												
Base	65.13	54.20	54.82	0.00	35.00	17.65	62.95	**27.46**	38.69	38.84	62.13	41.53
SFT	82.41	**63.40**	62.98	0.00	27.50	20.59	62.32	24.78	40.39	42.06	66.10	44.78
Pause_append	79.68	59.20	60.96	0.00	30.00	22.79	62.93	25.67	41.53	42.62	66.59	44.72
Pause_random	81.35	60.80	63.26	0.00	20.00	**23.16**	63.64	23.66	**48.73**	43.96	**69.64**	45.29
Pause_dit	68.01	57.20	56.38	**6.67**	30.00	20.22	62.90	25.67	44.75	42.66	69.31	43.98
Pause_step	**83.02**	61.00	**64.56**	3.33	**37.50**	**23.16**	**63.93**	27.23	48.32	44.13	67.63	**47.62**
Pause_step_predicted	80.97	62.20	62.64	**6.67**	27.50	**23.16**	63.53	24.55	45.86	**44.17**	66.70	46.18
Qwen3-4B-Base												
Base	82.56	66.00	67.64	6.67	37.50	28.68	73.05	**30.80**	63.69	**56.64**	73.29	53.32
SFT	91.36	69.20	71.24	6.67	45.00	29.78	73.33	29.91	75.53	54.64	77.91	56.78
Pause_append	89.92	67.00	68.62	**10.00**	42.50	28.68	73.35	29.02	**76.47**	54.94	78.45	56.27
Pause_random	**91.43**	70.40	71.06	**10.00**	42.50	29.04	73.57	28.79	76.14	54.89	77.64	56.86
Pause_dit	87.19	63.00	64.00	6.67	40.00	29.04	**73.60**	30.36	70.35	54.49	79.43	54.38
Pause_step	90.07	**73.40**	**72.32**	**10.00**	**50.00**	**30.51**	73.54	26.79	76.04	54.55	**79.65**	**57.90**
Pause_step_predicted	90.98	69.20	70.14	6.67	40.00	26.84	73.03	26.12	76.07	54.44	75.08	55.32
Qwen3-8B-Base												
Base	88.40	65.40	65.86	**13.33**	42.50	26.84	77.79	29.46	70.82	58.48	78.24	56.10
SFT	92.95	71.60	72.54	6.67	52.50	32.72	78.21	27.90	74.80	58.56	85.09	59.41
Pause_append	93.25	**75.40**	73.00	10.00	**60.00**	33.82	78.20	**35.04**	**76.45**	58.49	84.17	**61.62**
Pause_random	93.03	73.80	72.26	6.67	47.50	34.56	**78.53**	22.32	74.40	**58.82**	84.98	58.81
Pause_dit	**93.93**	74.40	**73.54**	6.67	52.50	**35.66**	78.00	26.12	73.64	58.14	**85.53**	59.83
Pause_step	92.57	75.00	72.84	3.33	55.00	35.29	78.25	23.66	75.52	57.28	85.36	59.46
Llama3.2-1B-Instruct												
Base	38.97	23.80	25.70	0.00	15.00	7.72	**42.34**	27.90	**30.63**	32.43	**61.26**	27.80
SFT	50.72	28.00	38.48	0.00	10.00	8.09	42.09	**28.79**	28.54	31.77	58.92	29.58
Pause_append	45.87	28.20	33.62	0.00	15.00	6.62	41.34	25.00	29.03	32.33	58.11	28.65
Pause_random	51.93	28.60	38.94	**3.33**	10.00	7.72	39.70	25.45	26.91	31.49	58.65	29.34
Pause_dit	43.75	21.80	29.12	0.00	17.50	6.25	39.74	24.33	27.62	**33.07**	56.86	27.28
Pause_step	51.63	**31.80**	**41.66**	0.00	**27.50**	**14.34**	40.81	27.68	28.43	32.16	55.82	**31.98**
Pause_step_predicted	**52.16**	30.60	39.64	0.00	2.50	6.99	40.90	26.12	27.35	32.38	55.11	28.52
Llama3.2-3B-Instruct												
Base	70.43	40.00	43.22	0.00	10.00	17.65	**59.33**	27.90	**56.28**	**43.06**	**76.77**	40.42
SFT	76.80	46.40	55.02	3.33	27.50	13.97	55.99	28.57	49.78	41.09	73.50	42.90
Pause_append	73.77	44.60	52.26	**6.67**	25.00	15.44	56.13	29.24	50.76	40.78	74.05	42.61
Pause_random	75.97	47.80	54.16	3.33	27.50	13.24	56.62	**29.69**	51.42	41.15	74.10	43.18
Pause_dit	70.36	38.00	43.00	3.33	17.50	12.13	57.60	28.12	52.39	42.41	74.05	39.90
Pause_step	**77.03**	**48.20**	**55.92**	3.33	**30.00**	**19.85**	56.46	25.89	50.04	41.56	73.72	**43.82**
Pause_step_predicted	76.19	47.40	54.58	3.33	20.00	15.81	55.64	25.22	49.93	40.73	73.78	42.06
Llama3.1-8B-Instruct												
Base	77.48	46.40	48.62	**6.67**	26.50	22.43	**67.56**	28.79	**73.30**	**49.42**	**82.05**	**48.11**
SFT	84.84	53.20	56.28	3.33	17.50	21.69	59.49	24.33	49.52	46.72	74.65	44.69
Pause_append	84.46	49.80	53.08	3.33	20.00	20.59	60.84	26.56	50.24	48.26	73.67	44.62
Pause_random	85.90	52.80	56.44	3.33	20.00	20.59	59.79	29.91	49.93	46.44	73.72	45.35
Pause_dit	77.41	44.00	47.14	3.33	20.00	**22.79**	58.51	**31.92**	52.53	45.89	75.46	43.54
Pause_step	**86.28**	**58.00**	**61.02**	**6.67**	**27.50**	22.43	61.30	29.91	50.78	48.13	72.25	47.66

Table 13: Additional SFT results on mathematical reasoning and general benchmarks. All entries report accuracy. Average denotes the macro-average over all reported benchmark columns.

Method	MBPP	MBPP+	HumanEval	HumanEval+	MMLU	GPQA	HellaSwag	BBH	PIQA	Average
Qwen3-1.7B-Base										
Base	69.05	59.26	58.54	53.05	62.95	27.46	38.69	38.84	62.13	52.22
SFT	69.58	**62.43**	67.07	61.59	60.19	23.88	36.44	39.15	60.66	53.44
Pause_append	69.84	60.58	70.73	**65.24**	61.53	**27.90**	36.58	40.65	64.09	55.24
Pause_random	70.37	61.11	67.07	62.80	**64.03**	24.78	48.36	42.50	**68.77**	56.64
Pause_dit	69.84	60.58	**71.95**	**65.24**	63.71	25.45	**50.40**	**43.09**	67.41	**57.52**
Pause_step	**73.02**	**62.43**	70.12	**65.24**	63.78	25.67	44.01	42.88	68.01	57.24
Qwen3-4B-Base										
Base	74.34	64.81	76.83	72.56	73.05	30.80	63.69	56.64	73.29	65.11
SFT	78.04	67.99	79.27	73.78	74.19	28.12	75.35	**56.85**	78.45	68.00
Pause_append	**80.42**	**68.25**	80.49	74.39	74.43	**34.15**	**77.15**	53.38	75.63	**68.70**
Pause_random	78.84	67.20	82.32	**78.66**	74.37	28.12	76.08	53.22	78.29	68.57
Pause_dit	75.93	66.93	81.10	76.22	74.36	28.12	76.38	53.43	76.77	67.69
Pause_step	79.10	65.08	**82.93**	78.05	**74.56**	29.24	74.72	54.94	**78.89**	68.61
Qwen3-8B-Base										
Base	82.80	72.49	**88.41**	**84.15**	77.79	29.46	70.82	58.48	78.24	71.40
SFT	86.51	75.13	85.98	81.71	77.96	29.24	75.33	61.25	83.30	72.93
Pause_append	85.19	72.75	83.54	78.05	77.89	23.88	79.01	**61.39**	82.97	71.63
Pause_random	86.24	75.13	84.76	81.10	**78.09**	28.12	**79.63**	59.85	**85.75**	73.19
Pause_dit	84.92	74.07	87.20	81.10	77.75	**32.14**	76.98	60.58	83.62	73.15
Pause_step	**87.30**	**75.66**	86.59	82.32	78.02	29.24	78.16	61.31	83.03	**73.51**
Llama3.1-8B-Instruct										
Base	67.46	59.52	67.68	62.20	**67.56**	28.79	73.30	49.42	82.05	62.00
SFT	68.25	60.05	68.29	**64.02**	67.47	26.56	73.63	49.55	81.88	62.19
Pause_append	69.58	**60.85**	67.68	62.20	67.44	25.45	73.87	49.47	**82.15**	62.08
Pause_random	69.31	60.05	68.90	63.41	67.37	25.45	73.60	**49.76**	82.10	62.22
Pause_dit	**70.11**	60.58	68.29	**64.02**	67.42	**29.24**	73.75	49.58	82.10	**62.79**
Pause_step	68.52	60.05	**69.51**	**64.02**	67.41	28.35	**73.93**	49.36	82.10	62.58

Table 14: Additional SFT results on code generation and general benchmarks. Code benchmark entries report pass@1, while general benchmark entries report accuracy. Average denotes the macro-average over all reported benchmark columns.