Commit Locally, Exit Globally: Coordinating Adaptive Sampling and Early Exit in Diffusion Language Models

Anonymous authors
Paper under double-blind review

Abstract

Diffusion language models predict tokens in parallel but still require many iterative forward passes. Adaptive sampling accelerates local token commitment, while recent global stopping methods seek to end generation early. Our trajectory analysis shows that short answers can stabilize early in decoding, whereas reasoning outputs often require more steps to settle. A fixed progress threshold therefore struggles to avoid both redundant computation on short-answer tasks and premature termination on reasoning tasks. We introduce C^4, a training-free framework that combines local decoding acceleration with adaptive global early exit. **Confidence-Verified Early Exit** (CVEE) determines when to end generation, skipping the remaining steps once its stopping criterion is met. It combines confidence and temporal agreement to adapt stopping to the observed revision history. **Commit-Core-Then-Confirm** (CCTC) accelerates token commitment while generation continues, reducing the decoding steps needed to complete each block. It extends confidence-threshold parallel decoding with a boundary-anchored core and one-step confirmation of deferred predictions, constraining additional commitments to mitigate premature freezing under unresolved context. With CVEE calibrated once on LLaDA, the same numerical settings transfer across all 12 zero-shot tasks and to Dream without recalibration. Under this configuration, C^4 reduces decoding steps by 64–95% and achieves measured decoding-loop speedups of 2.6×–18.6×, with task performance close to full decoding. Component analyses show complementary contributions: early exit supplies most savings on short-answer tasks, while local acceleration contributes more on reasoning and code generation. These results demonstrate that C^4 effectively combines local token commitment and global early exit to accelerate diffusion decoding while largely preserving task performance.

1 Introduction

Diffusion language models (DLMs) use bidirectional context to predict masked tokens in parallel (Nie et al., 2025; Ye et al., 2025). In block-wise decoding, they generate a sequence one block at a time, resolving multiple token positions within each block through iterative parallel updates. This parallelism offers the potential for faster generation than the one-token-at-a-time decoding of autoregressive language models. Realizing this potential requires deciding both how quickly to resolve each block and how long to continue generation. Inference can therefore be accelerated at two scopes: *locally*, by committing tokens sooner and advancing to the next block, and *globally*, by terminating the sequence before its decoding budget is exhausted.

Specifically, tasks differ in where these opportunities arise: short answers can stabilize early, allowing later blocks to be skipped, whereas extended reasoning often benefits more from faster updates within blocks as predictions continue evolving. Decision criteria should therefore reflect both these dynamics and the action's scope: token-level confidence can guide local commitment, while global stopping requires evidence that the task-relevant prediction has stabilized.

Existing methods typically exploit one of these opportunities. At the local level, adaptive samplers such as Fast-dLLM (Wu et al., 2026), SlowFast Sampling (Wei et al., 2026), and KLASS (Kim et al., 2025) accelerate token commitment; WINO (Hong et al., 2026) additionally permits revocation through re-masking. By completing blocks faster, these methods reduce the cost of continued generation, but leave open when the remaining blocks can be skipped altogether. Recent work on global stopping begins to address this question: Prophet (Li et al., 2026) and SchED (Mohamed et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2026b) use region-level confidence or schedule-dependent rules to terminate generation early. Their savings, however, depend on how much computation remains at the stopping point, while preceding within-block updates still rely on the underlying sampler. When predictions settle late, global stopping alone therefore leaves limited room for acceleration, making local sampling efficiency especially important for reducing the cost of the blocks that must still be generated before termination.

Combining these complementary sources of savings calls for a stopping rule that adapts to observed decoding trajectories across tasks and backbones while keeping its numerical settings fixed. The trajectory itself, however, also depends on local sampling: committing tokens reshapes the context for subsequent predictions and thus the confidence and agreement signals available for stopping. Local acceleration therefore changes not only how much computation remains, but also the evidence used to decide whether that computation is still needed. This connection motivates designing local commitment and global stopping together around the same evolving generation.

We introduce C^4 around this division of responsibilities. *Confidence-Verified Early Exit (CVEE)* combines current confidence with temporal agreement, adapting its patience to the history of revisions. *Commit-Core-Then-Confirm (CCTC)* extends confidence-threshold parallel decoding with a boundary-anchored core and one-step confirmation of deferred predictions. It retains the reference decoder's scheduled commitments and operates alongside CVEE on the same trajectory. Together, the components target two sources of excess computation: repeated steps within a block and continued decoding after stabilization. Moreover, because CVEE adjusts its patience to observed revisions rather than a fixed decoding-progress threshold, it can accommodate different stabilization times without changing its numerical settings, reducing reliance on task-specific stopping calibration.

- We analyze stabilization timing across diffusion decoding trajectories, identifying how early-exit opportunities vary between short-answer and long-reasoning tasks. This analysis motivates confidence and temporal agreement as complementary signals for global stopping.
- We introduce C^4, a training-free framework that combines local acceleration and global termination with reduced reliance on task-specific calibration. Because local commitment reshapes the trajectory that stopping observes, their savings are not additive; CVEE and CCTC are designed to act on this shared trajectory.
- Across 12 zero-shot tasks on LLaDA and Dream, C^4 achieves 2.6×–18.6× measured decoding-loop speedups while largely preserving task performance. Our experiments show how its complementary components sustain savings across tasks: global early exit skips computation after early stabilization, while local commitment accelerates the blocks needed for extended reasoning and code generation.

2 Related Work

Diffusion language models. The diffusion framework traces to Sohl-Dickstein et al. (2015), with extensions to discrete data by Hoogeboom et al. (2021); Austin et al. (2021a); Campbell et al. (2022). Lou et al. (2024) and the masked-diffusion line (Shi et al., 2024; Sahoo et al., 2024; Ou et al., 2025; Zheng et al., 2025) establish the parameterizations underlying modern DLMs. LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025), the open-weight models we evaluate, support parallel prediction at masked positions with flexible inference budgets. Under block-wise decoding, this flexibility creates two opportunities to reduce computation: completing individual blocks in fewer steps and skipping the remaining sequence.

Adaptive sampling for DLMs. Adaptive sampling controls which token positions commit while decoding continues. Fast-dLLM (Wu et al., 2026), SlowFast Sampling (Wei et al., 2026), and KLASS (Kim et al., 2025) commit using token-level statistics, while concurrent work refines related signals: LESS (Mohamed et al., 2026a) combines confidence with stability, STDec (Chen et al., 2026) adapts thresholds using neighboring decoded states and temporal stability, TACG (Wang et al., 2026) uses EMA-logit trajectories, Swordsman (Zhang et al., 2026) re-partitions blocks along entropy shifts, R^2-dLLM (Du et al., 2026) exploits spatio-temporal redundancy through decoding rules and fine-tuning, and Kim et al. (2026) identify proximity bias in the denoising order. WINO (Hong et al., 2026) permits broad commitment followed by re-masking of disputed positions. CCTC builds on confidence-threshold parallel decoding by structuring additional commitments around a boundary-anchored core and confirming deferred predictions at the next step. It retains scheduled commitments and does not revoke committed tokens. This local control reduces within-block computation while leaving sequence termination to CVEE.

Early exit and sequence termination. Early exit has been studied at the layer, reasoning-step, and token levels. Gu et al. (2026) apply it to *depth* via hidden-state stabilization. S-GRPO (Dai et al., 2025) and DEER (Yang et al., 2026) study early exit in reasoning models. Related diffusion methods address revision and verification: CORE (Zhai et al., 2026) identifies tokens sensitive to masked-context perturbations, while BMC (Ruan et al., 2026) evaluates consistency through masking and reconstruction. For DLMs, Prophet (Li et al., 2026) and SchED (Mohamed et al., 2026b) stop using aggregate confidence signals over fixed monitoring regions or schedule-dependent thresholds to determine when generation should end. CVEE bases this decision on confidence and temporal agreement of the task-relevant prediction, with patience adapted to its revision history. This shifts the stopping criterion from decoding progress to observed prediction behavior, allowing numerical settings to be reused across the evaluated tasks and backbones. Together with CCTC, it addresses both the cost of ongoing generation and the point at which further decoding can be skipped.

3 Preliminaries

We describe the masked generation process and the block-wise schedule used in our evaluation, then distinguish the local commitment and global stopping decisions that control its cost.

Masked diffusion generation. A DLM generates a sequence x_0 of length L by reversing a discrete corruption process that progressively masks a clean sequence over forward time $u \in [0, 1]$,

$$q(x_u \mid x_0) = \prod_{i=1}^{L} q(x_u^i \mid x_0^i), \tag{1}$$

$$q(x_u^i \mid x_0^i) = \begin{cases} x_0^i, & \text{w.p. } 1-u, \quad \text{(a) token kept} \\ \texttt{[MASK]}, & \text{w.p. } u, \quad \text{(b) token masked} \end{cases} \tag{2}$$

so x_1 is fully masked and x_0 is the original sequence. The model $p_\theta(x_0 \mid x_u)$ learns to recover the clean sequence from masked inputs. Generation starts from masks and proceeds through T discrete decoding steps, indexed by $t \in \{1, \ldots, T\}$. At each step, the model predicts masked positions in parallel, and a remasking rule selects predictions to commit. The remaining positions stay masked for the next step. We use u for forward corruption time and t for the decoding-step index.

Block-wise decoding. We partition the generation region into N contiguous blocks $B_0, \ldots, B_{N-1}$. Blocks are processed from left to right, with parallel prediction inside the active block and $S = T/N$ scheduled steps per block. We use this semi-autoregressive schedule for both LLaDA-8B-Instruct (Nie et al., 2025) and Dream-7B-Instruct (Ye et al., 2025). At each step, low-confidence remasking commits the highest-confidence predictions among currently masked positions. The reference schedule sets the number of commitments so that all positions are filled within the block budget. Local acceleration can select additional positions to commit, allowing a completed block to advance before that budget is exhausted. Thus, the schedule specifies a decoding budget, while the commitment rule determines how many of those steps are actually needed to complete each block.

Sequence-level stopping. Whereas local commitment advances the active block, a sequence-level stopping gate decides whether to end the entire generation, skipping all subsequent decoding steps. For example, Prophet (Li et al., 2026) computes the mean top-1/top-2 logit gap over a monitored region $\mathcal{R}$ to summarize the separation between the two leading predictions at each position,

$$\bar{g}_t = \frac{1}{|\mathcal{R}|} \sum_{i \in \mathcal{R}} \left(\ell_i^{(t),1} - \ell_i^{(t),2} \right), \tag{3}$$

where $\ell_i^{(t),1}$ and $\ell_i^{(t),2}$ are the two largest logits at position i and step t. We use a fixed task-format-specific definition of $\mathcal{R}$ in our evaluation, as detailed in Appendix A. Prophet fills all remaining masked positions once $\bar{g}_t$ crosses its published progress-dependent three-stage threshold; SchED replaces this with a progress-dependent decay schedule and a stability guard (Mohamed et al., 2026b). Local commitment and global stopping therefore act on the same evolving prediction but control different portions of the computation. In the following section, we examine how prediction stabilization informs stopping and develop the two controls within a shared decoder.

4 Local Commitment and Global Exit

C^4 reduces decoding cost through two decisions made along the same trajectory: which tokens to commit within the active block, and when to end the sequence. *CCTC* accelerates block completion while generation continues; *CVEE* skips the remaining steps when confidence and temporal agreement support termination (Figure 1). We first examine when predictions stabilize, then develop the global stopping rule and the local commitment mechanism.

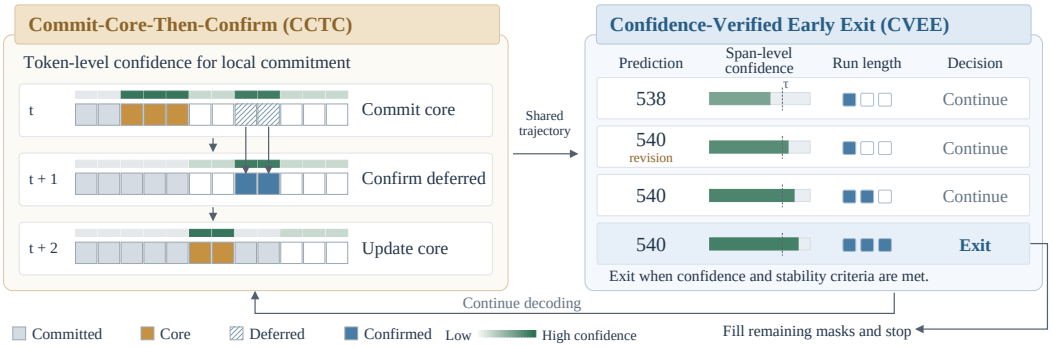

Figure 1: C^4 accelerates decoding at two complementary scales. CCTC uses token-level confidence to advance a boundary-anchored core and confirm deferred predictions as context develops. Along this shared trajectory, CVEE combines mean span-level confidence with temporal agreement to stop generation. Local commitment saves computation within blocks; global stopping removes later steps.

4.1 Stabilization Timing and the Opportunity for Early Exit

To determine when global stopping can save computation, we first examine when predictions settle during decoding. Full decoding trajectories let us compare this timing across tasks and establish the behavior that an adaptive stopping rule needs to accommodate.

Measuring stabilization along decoding trajectories. We measure when extracted answers stop changing in full LLaDA-8B-Instruct decodes drawn equally from MMLU, GSM8K, and MATH. Let $a_{j,r}$ denote the extracted candidate from trajectory j at normalized decoding progress r, and let a_j^{full} be its full-decoding answer. We define

$$s_j = \min\{s : a_{j,r} = a_j^{\text{full}} \ \forall r \geq s\}$$

as trajectory j's *stabilization progress*, and s_p as the progress by which a fraction p of trajectories have reached their own s_j. This retrospective measure identifies when the extracted answer reaches its final value, irrespective of whether that value is correct.

Different tasks settle at different stages. Using this measure, we find that $s_{0.9}$ ranges from 0.04 on short-answer tasks to 0.96 on long-reasoning ones. Early stabilization leaves substantial computation available to skip, whereas later stabilization leaves less room for early exit and makes within-block acceleration more important. Consequently, a fixed stopping point cannot accommodate both patterns: waiting longer forgoes savings on early-settling trajectories, while stopping sooner can interrupt predictions that are still changing. We therefore use observed confidence and temporal agreement to adapt the stopping decision to each trajectory.

4.2 Confidence-Verified Early Exit: Deciding When to Stop

The retrospective analysis above uses the final output to identify stabilization. During decoding, however, that output is not yet available. CVEE therefore combines current confidence with agreement across decoding steps and increases patience when predictions undergo more revisions. This lets stopping respond to the evolving output rather than to a fixed fraction of the decoding budget (Figure 2).

Tracking confidence and temporal agreement. To track the relevant prediction as its location changes, at step t a deterministic parser extracts and normalizes an answer from the current prediction $\hat{x}^{(t)}$, relocating its span as decoding proceeds. We denote this extracted value by a_t. CVEE applies to outputs with a consistently extractable answer unit. Conceptually, confidence describes the current prediction, while temporal agreement records its persistence across steps. Their combination provides the evidence for the stopping rule below.

Revision-adaptive patience. Let run_t be the number of successive same-value extractions (paused when extraction is unavailable); changes_t the number of changes in a_t; and c_t the mean current top-1 probability at positions in the eligible extracted span. CVEE fires at the first step satisfying

$$\begin{aligned} c_t &\geq \tau_{\text{CVEE}} && \text{(a) confidence gate} \\ \text{run}_t &\geq \max\left(p_{\min}, \lceil \gamma \cdot \text{changes}_t \rceil\right) && \text{(b) stability gate.} \end{aligned} \tag{4}$$

Here $p_{\min}$ sets minimum patience, while γ increases the required waiting period after revisions. Because this requirement follows the observed history, the gate has no explicit dependence on

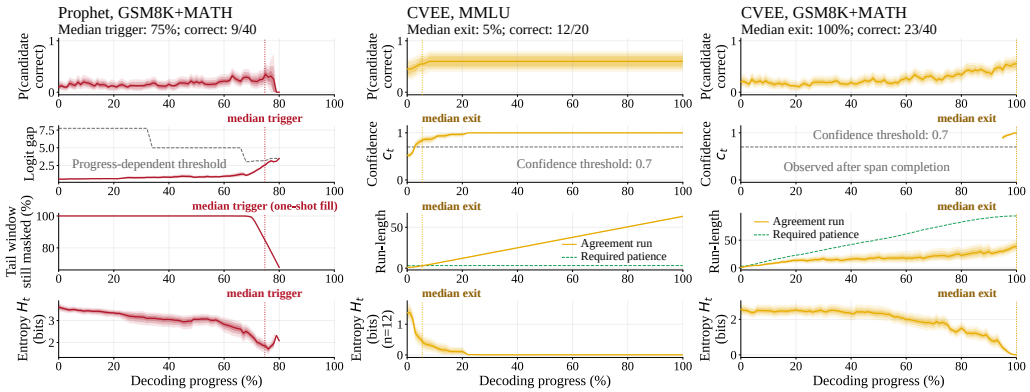

Figure 2: Stabilization timing shapes the opportunity for early exit. MMLU predictions settle early, leaving much of the decoding budget unused, whereas GSM8K and MATH predictions continue to change later in decoding. CVEE exits early on MMLU and waits longer on the reasoning tasks; Prophet can trigger while those predictions are still changing. These trajectories illustrate why stopping should track evolving predictions rather than elapsed decoding progress alone.

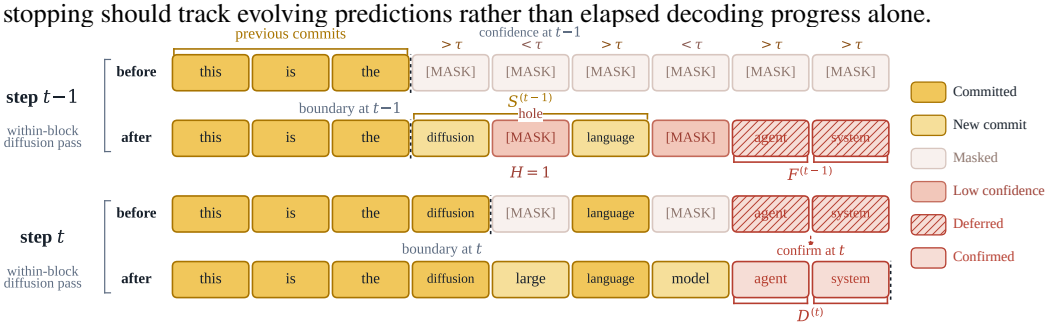

Figure 3: CCTC combines spatial structure with temporal confirmation. The boundary-anchored core advances through nearby high-confidence predictions, while proposals outside the core are revisited under updated context before confirmation. This couples parallel commitment to the evolving context: confidence identifies eligible positions, and position and persistence guide when to commit them as the surrounding context develops over successive decoding steps.

normalized decoding progress or a prescribed stopping step. Patience adapts to observed revisions, allowing the same numerical criterion to be used across decoding budgets and task regimes; we calibrate it once and freeze it across tasks and backbones (Section 5). The block schedule nevertheless determines observation eligibility: spans lie within active or previously decoded blocks. Because extraction retains committed tokens, which may differ from current top-1 predictions, CVEE is an empirical stopping criterion rather than a verification of the retained answer. Once the gate fires, the decoder fills remaining masks from current predictions. Appendix A details the updates.

4.3 Commit-Core-Then-Confirm: Selecting Parallel Commitments

While CVEE decides when to end generation, the preceding blocks still need to be decoded efficiently. CCTC addresses this remaining cost by building on Fast-dLLM's confidence-threshold parallel commitment rule (Wu et al., 2026). Specifically, it extends this selection rule with a boundary-anchored core and deferred confirmation: eligible positions outside the core are checked again under updated context (Figure 3). This spatial prior is motivated by the decoding order studied in Efficient-DLM (Fu et al., 2026) and blockwise prediction in DiffusionVL (Zeng et al., 2026). CCTC uses this prior to organize additional commitments within the active block while retaining parallel prediction and the reference decoder's scheduled updates.

Selecting eligible positions. To formalize this selection, let $E^{(t)}$ be the currently masked positions in active block B_n whose confidence is at least τ_{CCTC} at step t. CCTC partitions these positions into a core for immediate commitment and deferred proposals for confirmation under the next step's updated context. This separates two complementary checks: the core uses position to select immediate updates, whereas confirmation tests whether deferred predictions persist across steps.

Boundary-anchored commitment within a block. The core favors additional commitments near the resolved left boundary while predictions remain parallel. More precisely, the gap budget H specifies

how many unresolved low-confidence positions it may cross:

$$h_i^{(t)} = \left|\{\, j \in B_n : j < i,\ x_j^{(t)} = \texttt{[MASK]},\ j \notin E^{(t)} \,\}\right|, \qquad S^{(t)} = \{\, i \in E^{(t)} : h_i^{(t)} \le H \,\}. \tag{5}$$

Here $h_i^{(t)}$ counts unresolved holes before i. Already committed positions do not count, so the core remains anchored at the resolved boundary. Eligible positions outside the core are **deferred** unless confirmed or selected by the scheduled rule; their predictions are recorded for the next step. The boundary constraint, however, applies only to the core: confirmed deferrals and scheduled commits can bypass it, so the full update is not strictly left to right.

Confirming proposals under updated context. To reconsider these deferred proposals after the context changes, let $F^{(t-1)} = E^{(t-1)} \setminus \left(S^{(t-1)} \cup D^{(t-1)}\right)$ be the proposal buffer from $t-1$, with recorded predictions $\tilde{x}_i^{(t-1)}$. Positions already committed by the schedule are excluded by the mask condition below. At step t, confirmation selects

$$D^{(t)} = \{\, i \in F^{(t-1)} \cap E^{(t)} : x_i^{(t)} = \texttt{[MASK]},\ \hat{x}_i^{(t)} = \tilde{x}_i^{(t-1)} \,\}, \tag{6}$$

so this branch commits a deferred position only if it still clears τ_{CCTC} and still predicts the recorded token under the next step's updated context. If a prediction changes or its confidence falls, however, the position receives no commit from this branch. It may still commit through the core or scheduled top-k rule. If it remains above the threshold without entering the core, it is deferred again with its new prediction, and otherwise it leaves the deferral buffer. The proposal buffer is empty at block entry and never carries across blocks. Because confirmation reuses the next scheduled forward pass, it requires no additional model evaluation; committed tokens remain fixed. Combining these branches with the reference schedule gives the complete update. We accelerate all blocks, $\mathcal{A} = \{0, \ldots, N-1\}$, and let $\mathcal{K}^{(t)}$ denote the reference decoder's scheduled top-k selection. Every masked $i \in B_n$, $n \in \mathcal{A}$, updates as

$$x_i^{(t+1)} = \begin{cases} \hat{x}_i^{(t)},\ i \in S^{(t)} \text{ (a) core} & \hat{x}_i^{(t)},\ i \in D^{(t)} \text{ (b) confirmed deferral} \\ \hat{x}_i^{(t)},\ i \in \mathcal{K}^{(t)} \text{ (c) scheduled top-}k & x_i^{(t)}, \text{ otherwise (d) stays masked.} \end{cases} \tag{7}$$

CCTC constrains only the additional threshold-driven commits in branches (a) and (b), while the reference decoder's scheduled top-k branch (c) is retained unchanged to guarantee progress. Once all positions in a block have committed, the block-completion condition $R_n(x) = \bigwedge_{i \in B_n} [x_i \neq \texttt{[MASK]}]$ holds, and the remaining steps allocated to that block are skipped.

Together, these decisions determine both the pace and duration of decoding: CCTC shortens the path through blocks that must be generated, while CVEE determines how far along that path decoding needs to proceed. Committed tokens become context for later predictions, so CVEE observes the trajectory produced by local acceleration. The resulting decoder combines parallel block completion with revision-adaptive termination, without additional training or extra forward passes for confirmation.

5 Experiments

Benchmarks and hardware. We evaluate LLaDA-8B-Instruct (Nie et al., 2025) and Dream-7B-Instruct (Ye et al., 2025) with bf16 and greedy decoding; main-table timing uses a single RTX 4090. The 12 tasks comprise five *short-answer* benchmarks (MMLU (Hendrycks et al., 2021a), ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), and PIQA (Bisk et al., 2020)), five *long-reasoning* benchmarks (GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021b), SVAMP (Patel et al., 2021), ASDiv (Miao et al., 2020), and GSM-Hard (Gao et al., 2023)), and two *code-generation* benchmarks graded by hidden tests (HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021b)).

Hyperparameters and calibration. C^4 uses five gate and commitment hyperparameters in the main comparison. CVEE uses γ=2.0, τ_{CVEE}=0.7, and $p_{\min}$=3; CCTC uses τ_{CCTC}=0.8 and gap budget H=1. The three CVEE parameters are calibrated once on a shared 120-example LLaDA pool and then frozen across all 12 tasks and transferred unchanged to Dream. The two CCTC parameters are fixed *a priori*. All reported configurations are evaluated with live decoding. Appendix A details the prompting protocol, baseline configurations, and task-specific extraction rules.

Evaluation metrics. Accuracy follows task-specific answer extraction, or execution grading for code. Step is the number of executed decoding steps, and Saving is $1 - \text{Step}/T_{\text{full}}$. Speedup is method TPS divided by the corresponding full-decoding baseline TPS. Table 1 highlights drops exceeding 2.0 points as a visual aid, independent of calibration. Score differences are finite-sample point estimates, not equivalence claims. Accuracy and step counts use the task-specific evaluation sets; throughput uses a fixed 20-example timing subset with three repeats (Appendix B).

Task	Variant	LLaDA-8B-Instruct Acc (%)	Avg. Step	Saving	TPS	Speedup	Dream-7B-Instruct Acc (%)	Avg. Step	Saving	TPS	Speedup
General / short-answer tasks (single-token or short-span answers)											
MMLU (n=200, sequence length=64) (block=16, 4 blocks)	Baseline	64.0	64.0	0.0%	30.4	1.00×	71.5	64.0	0.0%	28.2	1.00×
	Prophet (Li et al., 2026)	62.5 (-1.5)	8.2 (-55.8)	87.1%	202.2	6.65×	71.5 (+0.0)	14.9 (-49.2)	76.8%	109.8	3.89×
	SchED (Mohamed et al., 2026b)	61.5 (-2.5)	31.0 (-33.0)	51.5%	53.3	1.75×	69.0 (-2.5)	13.0 (-51.1)	79.8%	178.7	6.34×
	SlowFast (Wei et al., 2026)	63.0 (-1.0)	17.9 (-46.1)	72.0%	130.4	4.29×	74.5 (+3.0)	9.6 (-54.4)	85.0%	196.9	6.98×
	KLASS (Kim et al., 2026)	62.5 (-1.5)	29.2 (-34.8)	54.4%	51.7	1.70×	72.5 (+1.0)	11.3 (-52.7)	82.3%	129.0	4.57×
	WINO (Hong et al., 2026)	61.0 (-3.0)	16.8 (-47.2)	73.8%	100.4	3.30×	–	–	–	–	–
	C^4	64.0 (+0.0)	4.8 (-59.2)	92.5%	490.4	16.13×	70.0 (-1.5)	4.5 (-59.5)	93.0%	389.7	13.82×
ARC-C (n=200, sequence length=64) (block=16, 4 blocks)	Baseline	86.5	64.0	0.0%	33.8	1.00×	87.5	64.0	0.0%	32.6	1.00×
	Prophet (Li et al., 2026)	86.0 (-0.5)	7.5 (-56.5)	88.3%	295.3	8.74×	86.0 (-1.5)	14.0 (-50.0)	78.2%	138.3	4.24×
	SchED (Mohamed et al., 2026b)	85.0 (-1.5)	41.1 (-22.9)	35.8%	57.5	1.70×	88.5 (+1.0)	11.0 (-53.0)	82.8%	215.1	6.60×
	SlowFast (Wei et al., 2026)	85.0 (-1.5)	23.7 (-40.3)	63.0%	107.3	3.17×	87.0 (-0.5)	10.4 (-53.6)	83.8%	219.4	6.73×
	KLASS (Kim et al., 2025)	87.0 (+0.5)	37.5 (-26.5)	41.5%	53.7	1.59×	88.5 (+1.0)	11.6 (-52.4)	81.9%	158.9	4.87×
	WINO (Hong et al., 2026)	84.0 (-2.5)	20.1 (-43.9)	68.6%	76.9	2.28×	–	–	–	–	–
	C^4	86.5 (+0.0)	3.9 (-60.1)	94.0%	483.3	14.30×	88.5 (+1.0)	4.2 (-59.8)	93.5%	565.7	17.35×
HellaSwag (n=200, sequence length=64) (block=16, 4 blocks)	Baseline	75.5	64.0	0.0%	20.8	1.00×	75.5	64.0	0.0%	22.1	1.00×
	Prophet (Li et al., 2026)	75.0 (-0.5)	6.0 (-58.0)	90.6%	223.4	10.74×	75.0 (-0.5)	16.2 (-47.8)	74.7%	82.0	3.71×
	SchED (Mohamed et al., 2026b)	75.5 (+0.0)	27.7 (-36.3)	56.7%	46.4	2.23×	75.0 (-0.5)	19.0 (-45.0)	70.3%	63.0	2.85×
	SlowFast (Wei et al., 2026)	77.0 (+1.5)	10.9 (-53.1)	83.0%	134.2	6.45×	72.5 (-3.0)	9.0 (-55.0)	85.9%	160.0	7.24×
	KLASS (Kim et al., 2025)	75.0 (-0.5)	19.4 (-44.6)	69.6%	56.5	2.72×	69.5 (-6.0)	11.3 (-52.7)	82.3%	94.0	4.25×
	WINO (Hong et al., 2026)	73.0 (-2.5)	12.9 (-51.1)	79.8%	79.0	3.80×	–	–	–	–	–
	C^4	75.5 (+0.0)	3.7 (-60.3)	94.3%	348.6	16.76×	75.5 (+0.0)	4.4 (-59.6)	93.1%	297.8	13.48×
WinoGrande (n=200, sequence length=64) (block=16, 4 blocks)	Baseline	75.5	64.0	0.0%	36.1	1.00×	71.5	64.0	0.0%	35.5	1.00×
	Prophet (Li et al., 2026)	76.0 (+0.5)	5.9 (-58.1)	90.8%	383.9	10.63×	69.5 (-2.0)	15.3 (-48.7)	76.1%	140.6	3.96×
	SchED (Mohamed et al., 2026b)	75.5 (+0.0)	5.8 (-58.2)	91.0%	447.1	12.39×	71.0 (-0.5)	2.1 (-61.9)	96.7%	1034.4	29.14×
	SlowFast (Wei et al., 2026)	77.5 (+2.0)	8.5 (-55.5)	86.7%	285.5	7.91×	71.0 (-0.5)	8.0 (-56.0)	87.5%	279.6	7.88×
	KLASS (Kim et al., 2025)	76.5 (+1.0)	13.3 (-50.7)	79.2%	160.0	4.43×	72.5 (+1.0)	7.6 (-56.4)	88.2%	236.8	6.67×
	WINO (Hong et al., 2026)	77.0 (+1.5)	10.1 (-53.9)	84.2%	176.1	4.88×	–	–	–	–	–
	C^4	75.5 (+0.0)	3.4 (-60.6)	94.6%	672.8	18.64×	71.5 (+0.0)	4.2 (-59.8)	93.4%	561.4	15.81×
PIQA (n=200, sequence length=64) (block=16, 4 blocks)	Baseline	81.5	64.0	0.0%	35.2	1.00×	87.5	64.0	0.0%	32.5	1.00×
	Prophet (Li et al., 2026)	82.0 (+0.5)	10.8 (-53.2)	83.1%	245.7	6.98×	87.5 (+0.0)	18.9 (-45.2)	70.5%	104.0	3.20×
	SchED (Mohamed et al., 2026b)	81.0 (-0.5)	24.5 (-39.5)	61.7%	98.6	2.80×	87.0 (-0.5)	17.1 (-46.9)	73.3%	129.1	3.97×
	SlowFast (Wei et al., 2026)	84.0 (+2.5)	10.0 (-54.0)	84.4%	205.2	5.83×	89.5 (+2.0)	9.1 (-55.0)	85.9%	209.2	6.44×
	KLASS (Kim et al., 2025)	81.5 (+0.0)	18.6 (-45.4)	70.9%	106.3	3.02×	91.5 (+4.0)	10.6 (-53.4)	83.4%	158.0	4.86×
	WINO (Hong et al., 2026)	81.0 (-0.5)	11.4 (-52.6)	82.2%	187.7	5.33×	–	–	–	–	–
	C^4	82.0 (+0.5)	3.6 (-60.4)	94.4%	546.5	15.53×	88.0 (+0.5)	3.5 (-60.5)	94.5%	575.7	17.71×
Long-reasoning tasks (multi-step CoT)											
GSM8K (n=100, sequence length=256) (block=32, 8 blocks)	Baseline	66.0	256.0	0.0%	19.0	1.00×	85.0	256.0	0.0%	21.2	1.00×
	Prophet (Li et al., 2026)	38.0 (-28.0)	185.0 (-71.0)	27.7%	22.6	1.19×	16.0 (-69.0)	76.1 (-179.9)	70.3%	67.0	3.16×
	SchED (Mohamed et al., 2026b)	59.0 (-7.0)	224.3 (-31.7)	12.4%	21.1	1.11×	18.0 (-67.0)	76.9 (-179.1)	70.0%	64.9	3.06×
	SlowFast (Wei et al., 2026)	41.0 (-25.0)	70.9 (-185.1)	72.3%	68.0	3.58×	20.0 (-65.0)	89.2 (-166.8)	65.1%	59.6	2.81×
	KLASS (Kim et al., 2025)	65.0 (-1.0)	105.5 (-150.5)	58.8%	34.5	1.82×	33.0 (-52.0)	96.5 (-159.5)	62.3%	37.2	1.75×
	WINO (Hong et al., 2026)	72.0 (+6.0)	49.2 (-206.8)	80.8%	76.6	4.03×	–	–	–	–	–
	C^4	64.0 (-2.0)	62.5 (-193.5)	75.6%	67.5	3.55×	85.0 (+0.0)	54.5 (-201.5)	78.7%	102.1	4.82×
MATH (n=100, sequence length=256) (block=32, 8 blocks)	Baseline	32.0	256.0	0.0%	17.8	1.00×	42.0	256.0	0.0%	19.4	1.00×
	Prophet (Li et al., 2026)	19.0 (-13.0)	192.7 (-63.3)	24.7%	21.4	1.20×	15.0 (-27.0)	164.9 (-91.1)	35.6%	26.7	1.38×
	SchED (Mohamed et al., 2026b)	26.0 (-6.0)	233.5 (-22.5)	8.8%	19.3	1.08×	33.0 (-9.0)	199.0 (-57.0)	22.3%	23.3	1.20×
	SlowFast (Wei et al., 2026)	24.0 (-8.0)	100.0 (-156.0)	60.9%	45.5	2.56×	9.0 (-33.0)	98.9 (-157.1)	61.4%	51.9	2.68×
	KLASS (Kim et al., 2025)	34.0 (+2.0)	133.9 (-122.1)	47.7%	26.1	1.47×	4.0 (-38.0)	98.7 (-157.3)	61.4%	40.3	2.08×
	WINO (Hong et al., 2026)	35.0 (+3.0)	81.7 (-174.3)	68.1%	42.8	2.40×	–	–	–	–	–
	C^4	33.0 (+1.0)	92.5 (-163.5)	63.9%	46.7	2.62×	42.0 (+0.0)	84.6 (-171.5)	67.0%	53.5	2.76×
SVAMP (n=100, sequence length=256) (block=32, 8 blocks)	Baseline	81.0	256.0	0.0%	19.3	1.00×	76.0	256.0	0.0%	21.4	1.00×
	Prophet (Li et al., 2026)	40.0 (-41.0)	160.1 (-96.0)	37.5%	27.5	1.42×	29.0 (-47.0)	48.8 (-207.2)	80.9%	76.1	3.56×
	SchED (Mohamed et al., 2026b)	70.0 (-11.0)	196.0 (-60.0)	23.4%	26.9	1.39×	20.0 (-56.0)	42.9 (-213.1)	83.2%	75.8	3.54×
	SlowFast (Wei et al., 2026)	46.0 (-35.0)	81.7 (-174.3)	68.1%	63.4	3.28×	67.0 (-9.0)	49.9 (-206.1)	80.5%	137.0	6.40×
	KLASS (Kim et al., 2025)	79.0 (-2.0)	96.4 (-159.6)	62.4%	39.9	2.07×	71.0 (-5.0)	58.1 (-197.9)	77.3%	71.0	3.32×
	WINO (Hong et al., 2026)	85.0 (+4.0)	45.6 (-210.4)	82.2%	81.0	4.20×	–	–	–	–	–
	C^4	81.0 (+0.0)	58.8 (-197.2)	77.0%	82.8	4.29×	75.0 (-1.0)	42.8 (-213.2)	83.3%	113.0	5.28×
ASDiv (n=100, sequence length=256) (block=32, 8 blocks)	Baseline	68.0	256.0	0.0%	19.4	1.00×	78.0	256.0	0.0%	21.4	1.00×
	Prophet (Li et al., 2026)	37.0 (-31.0)	163.7 (-92.3)	36.1%	29.5	1.52×	25.0 (-53.0)	54.8 (-201.3)	78.6%	122.5	5.72×
	SchED (Mohamed et al., 2026b)	60.0 (-8.0)	197.0 (-59.0)	23.0%	28.6	1.47×	17.0 (-61.0)	53.3 (-202.7)	79.2%	101.1	4.72×
	SlowFast (Wei et al., 2026)	34.0 (-34.0)	82.4 (-173.7)	67.8%	64.9	3.35×	52.0 (-26.0)	57.9 (-198.1)	77.4%	144.0	6.73×
	KLASS (Kim et al., 2025)	66.0 (-2.0)	100.8 (-155.2)	60.6%	40.7	2.10×	58.0 (-20.0)	67.2 (-188.8)	73.7%	65.4	3.06×
	WINO (Hong et al., 2026)	60.0 (-8.0)	48.2 (-207.8)	81.2%	84.5	4.36×	–	–	–	–	–
	C^4	71.0 (+3.0)	60.9 (-195.1)	76.2%	86.9	4.48×	76.0 (-2.0)	45.8 (-210.2)	82.1%	135.1	6.31×
GSM-Hard (n=100, sequence length=256) (block=32, 8 blocks)	Baseline	35.0	256.0	0.0%	18.8	1.00×	42.0	256.0	0.0%	20.6	1.00×
	Prophet (Li et al., 2026)	18.0 (-17.0)	194.1 (-61.9)	24.2%	21.9	1.16×	9.0 (-33.0)	103.9 (-152.1)	59.4%	45.7	2.22×
	SchED (Mohamed et al., 2026b)	31.0 (-4.0)	237.4 (-18.6)	7.3%	19.7	1.05×	10.0 (-32.0)	123.5 (-132.6)	51.8%	41.6	2.02×
	SlowFast (Wei et al., 2026)	21.0 (-14.0)	70.0 (-186.0)	72.6%	70.2	3.73×	11.0 (-31.0)	96.8 (-159.2)	62.2%	62.6	3.04×
	KLASS (Kim et al., 2025)	35.0 (+0.0)	109.7 (-146.3)	57.1%	34.0	1.81×	10.0 (-32.0)	111.7 (-144.3)	56.4%	35.1	1.70×
	WINO (Hong et al., 2026)	34.0 (-1.0)	51.2 (-204.8)	80.0%	77.3	4.11×	–	–	–	–	–
	C^4	37.0 (+2.0)	64.4 (-191.6)	74.8%	67.6	3.60×	45.0 (+3.0)	64.9 (-191.1)	74.7%	82.2	3.99×
Code generation (whole-program answer, execution-graded)											
HumanEval (n=164, sequence length=384) (block=32, 12 blocks)	Baseline	40.2	384.0	0.0%	13.8	1.00×	45.1	384.0	0.0%	14.5	1.00×
	Prophet (Li et al., 2026)	3.0 (-37.2)	124.0 (-260.0)	67.7%	30.9	2.24×	0.6 (-44.5)	29.3 (-354.7)	92.4%	144.2	9.94×
	SchED (Mohamed et al., 2026b)	12.8 (-27.4)	175.7 (-208.3)	54.3%	33.1	2.40×	1.8 (-43.3)	39.3 (-344.7)	89.8%	142.9	9.86×
	SlowFast (Wei et al., 2026)	19.5 (-20.7)	36.3 (-347.7)	90.5%	213.1	15.44×	23.2 (-22.0)	36.4 (-347.6)	90.5%	205.6	14.18×
	KLASS (Kim et al., 2025)	40.9 (+0.6)	92.6 (-291.4)	75.9%	45.4	3.29×	25.6 (-19.5)	103.5 (-280.5)	73.1%	34.7	2.39×
	WINO (Hong et al., 2026)	37.8 (-2.4)	54.2 (-329.8)	85.9%	77.0	5.58×	–	–	–	–	–
	C^4	42.1 (+1.8)	39.4 (-344.6)	89.7%	120.0	8.70×	43.3 (-1.8)	29.2 (-354.8)	92.4%	146.0	10.07×
MBPP (n=257, sequence length=384) (block=32, 12 blocks)	Baseline	45.9	384.0	0.0%	15.4	1.00×	56.0	384.0	0.0%	15.6	1.00×
	Prophet (Li et al., 2026)	1.9 (-44.0)	32.1 (-351.9)	91.6%	73.2	4.75×	0.8 (-55.3)	1.4 (-382.6)	99.6%	433.7	27.80×
	SchED (Mohamed et al., 2026b)	27.6 (-18.3)	66.6 (-317.4)	82.6%	101.3	6.58×	1.9 (-54.1)	16.3 (-367.7)	95.8%	414.1	26.54×
	SlowFast (Wei et al., 2026)	41.2 (-4.7)	21.6 (-362.4)	94.4%	362.0	23.51×	46.7 (-9.3)	30.0 (-354.0)	92.2%	274.1	17.57×
	KLASS (Kim et al., 2025)	46.5 (+0.6)	69.2 (-314.8)	82.0%	66.8	4.34×	53.0 (-3.0)	93.1 (-290.9)	75.8%	42.3	2.71×
	WINO (Hong et al., 2026)	47.1 (+1.2)	54.4 (-329.6)	85.8%	88.0	5.71×	–	–	–	–	–
	C^4	45.9 (+0.0)	33.3 (-350.7)	91.3%	170.8	11.09×	56.0 (+0.0)	25.5 (-358.5)	93.4%	209.3	13.42×

Table 1: Zero-shot evaluation across 12 tasks on LLaDA-8B-Instruct and Dream-7B-Instruct. Parentheses report accuracy and step changes from full decoding. TPS denotes tokens per second; speedup is the method/baseline TPS ratio. Red marks accuracy drops exceeding 2.0 percentage points.

5.1 Performance across Task Regimes

Across the 12 tasks in Table 1, C^4 reduces decoding cost on both backbones while largely preserving task performance. To understand where the acceleration arises, we relate these savings to the timing of stabilization. The dominant source changes across tasks: early-settling predictions leave later blocks to skip, whereas extended generation requires faster completion of the blocks that remain necessary. Figure 4 relates stopping to stabilization on individual trajectories; the component ablations in Section 5.3 separate the two sources of savings.

Early stabilization favors global stopping. On short-answer tasks, predictions often settle within the first block. CVEE can then skip subsequent blocks, making global termination the main source of savings. The MMLU trajectories in Figure 4 place stopping close to stabilization, illustrating how a short useful decoding trajectory can coexist with a much larger prescribed generation budget.

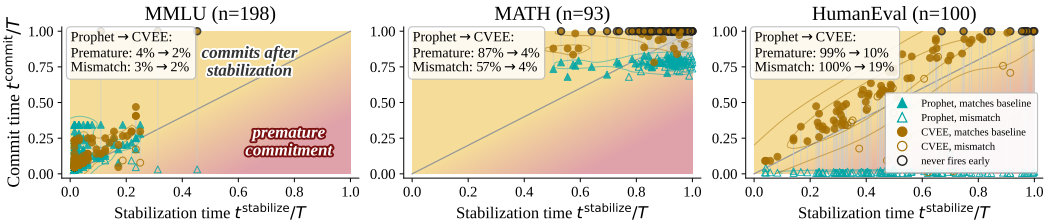

Figure 4: Exit–stabilization phase diagram. Each faint line pairs Prophet and CVEE on the same LLaDA-8B-Instruct trajectory. On MMLU, both often trigger near answer stabilization; on MATH and HumanEval, Prophet frequently triggers before the final answer has stabilized, whereas CVEE waits substantially longer. The comparison measures alignment between stopping and answer stabilization.

Continued generation favors local acceleration. For long-reasoning tasks, however, predictions often stabilize near the decoding horizon, leaving fewer steps for global stopping to remove. CCTC instead reduces the work inside blocks that still need to execute. The Dream GSM8K ablation makes this distinction clear: CVEE alone saves few steps, while adding CCTC removes most of the within-block work. Consequently, local sampling quality becomes especially important in this regime. For example, WINO outperforms C^4 on LLaDA GSM8K in both accuracy (72.0% versus 64.0%) and speedup (4.03× versus 3.55×), showing that the strongest local sampler can retain an advantage when early-exit opportunities are scarce.

Code generation links local updates to program completion. Code generation adds a further dependency: correctness is determined by execution of the complete program, so commitments throughout the output contribute to the evaluated result. The HumanEval ablations show that accelerating the final block provides further savings. At the same time, its later stabilization in Figure 4 highlights the need to distinguish faster token commitment from readiness to terminate: CVEE waits longer than Prophet while the program continues evolving.

Beyond the overall savings, we examine how commitment settings and locations affect this balance. Threshold sweeps show a broad region of stable accuracy on short-answer tasks, while reasoning and code are more sensitive to aggressive commitment (Appendix F). Paired controls further show that protecting the monitored span changes exit timing only modestly on MMLU and GSM8K, with identical graded answers on both backbones; protecting the entire block causes larger changes (Appendix C). Together, the results connect the benefits of global stopping to the trajectory produced by local updates, motivating the sampler-composition experiments that follow.

5.2 Early Exit across Sampling Trajectories

The preceding results link useful early exit to the trajectory produced by local updates. To examine this connection with a different sampler, we pair CVEE with WINO (Hong et al., 2026), retaining WINO's local update rule and using CVEE only for sequence termination (Table 2). This separates the stopping rule from CCTC and tests it on a different sampling trajectory.

Table 2: CVEE combined with WINO. CVEE provides substantial additional step reduction on short-answer tasks, whereas on long-reasoning and code tasks, preserving accuracy leaves fewer opportunities to terminate the already shortened trajectory.

	WINO		+ CVEE		Conservative CVEE	
Task	Acc	Step	Acc	Step	Acc	Step
MMLU	61.0	16.8	61.5	4.1	–	–
HellaSwag	73.0	12.9	73.0	3.9	–	–
GSM8K	72.0	49.2	36.0	43.6	70.0	48.5
SVAMP	85.0	45.6	62.0	41.7	83.0	45.1
HumanEval	37.8	54.2	14.6	29.7	36.6	50.1

When local acceleration leaves room to stop. On MMLU and HellaSwag, CVEE removes a further 70–76% of WINO's steps with nearly unchanged accuracy. Local sampling has shortened generation, yet predictions still settle before the remaining scheduled updates finish. Global stopping captures this residual saving.

When the remaining steps still matter. On GSM8K, SVAMP, and HumanEval, however, the same exit setting substantially lowers accuracy. A more conservative setting recovers most of the score, but brings computation close to WINO's original step count. Thus, the apparent opportunity to terminate sooner largely vanishes when task performance is recovered. These trajectories leave less redundant work for CVEE to remove after local acceleration.

Taken together, these composition results show that savings from sampling and stopping are not simply additive. A stopping rule observes predictions shaped by the local update mechanism, so its useful operating point depends on that trajectory. Appendix E examines the corresponding confidence

Task	Variant	LLaDA-8B-Instruct Acc (%)	Step	Saving	TPS	Speedup	Dream-7B-Instruct Acc (%)	Step	Saving	TPS	Speedup
General / short-answer tasks											
MMLU	Baseline	64.0	64.0	0.0%	30.4	1.00×	71.5	64.0	0.0%	28.2	1.00×
(n=200, sequence length=64)	+ confidence gate only	64.0	3.7	94.1%	636.8	20.95×	70.5	2.9	95.5%	277.2	9.83×
(block=16, 4 blocks)	+ stability gate only	62.0	4.0	93.7%	525.7	17.29×	70.0	3.7	94.2%	417.0	14.79×
	+ CVEE (joint gate)	64.0	7.0	89.1%	452.7	14.89×	69.5	8.0	87.5%	208.9	7.41×
	+ CCTC, core only	63.5	5.2	91.9%	496.1	16.32×	70.5	5.2	91.9%	323.3	11.46×
	+ CCTC, core then confirm	64.0	5.0	92.1%	491.1	16.15×	70.0	4.7	92.6%	386.4	13.70×
	C^4: CCTC on every block	64.0	4.8	92.5%	490.4	16.13×	70.0	4.5	93.0%	389.7	13.82×
Long-reasoning tasks (multi-step CoT)											
GSM8K	Baseline	66.0	256.0	0.0%	19.0	1.00×	85.0	256.0	0.0%	21.2	1.00×
(n=100, sequence length=256)	+ confidence gate only	61.0	59.2	76.9%	20.7	1.09×	83.0	53.8	79.0%	21.3	1.00×
(block=32, 8 blocks)	+ stability gate only	33.0	24.5	90.4%	50.0	2.63×	83.0	53.2	79.2%	23.3	1.10×
	+ CVEE (joint gate)	66.0	250.0	2.3%	19.2	1.01×	85.0	256.0	0.0%	21.1	1.00×
	+ CCTC, core only	66.0	85.2	66.7%	51.8	2.73×	85.0	85.8	66.5%	63.7	3.00×
	+ CCTC, core then confirm	66.0	83.9	67.2%	52.0	2.74×	85.0	84.3	67.1%	64.6	3.05×
	C^4: CCTC on every block	64.0	62.5	75.6%	67.5	3.55×	85.0	54.5	78.7%	102.1	4.82×
Code generation (whole-program answer)											
HumanEval	Baseline	40.2	384.0	0.0%	13.8	1.00×	45.1	384.0	0.0%	14.5	1.00×
(n=164, sequence length=384)	+ confidence gate only	4.9	9.7	97.5%	67.6	4.90×	5.5	9.5	97.5%	196.5	13.55×
(block=32, 12 blocks)	+ stability gate only	42.1	39.4	89.7%	24.8	1.80×	43.3	29.2	92.4%	43.9	3.03×
	+ CVEE (joint gate)	37.2	212.4	44.7%	24.8	1.80×	42.7	109.2	71.6%	39.3	2.71×
	+ CCTC, core only	40.9	50.0	87.0%	90.8	6.58×	43.3	36.6	90.5%	116.5	8.03×
	+ CCTC, core then confirm	42.1	48.8	87.3%	92.6	6.71×	43.3	35.8	90.7%	119.0	8.21×
	C^4: CCTC on every block	42.1	39.4	89.7%	120.0	8.70×	43.3	29.2	92.4%	146.0	10.07×

Table 3: Complementary savings from early exit and local commitment. Early stabilization makes CVEE the main source of savings on MMLU. On GSM8K and HumanEval, CCTC further reduces computation within the blocks that generation still needs. Highlighted rows show C^4.

and agreement signals. We next study how CVEE and CCTC contribute within C^4 and how their savings change with the decoding budget.

5.3 Component Contributions and Scaling

Having examined how CVEE behaves on a different sampling trajectory, we now separate the contributions of the stopping signals and commitment branches within C^4. We then consider how these savings change with generation length, block length, and caching. Table 3 separates the component contributions; Appendix G examines how generation budgets, block lengths, and caching affect the resulting accuracy–efficiency trade-offs.

Confidence and agreement play complementary roles. The individual CVEE gates can stop quickly without consistently retaining task performance (Table 3). Confidence alone performs poorly on HumanEval, while stability alone sharply reduces accuracy on LLaDA GSM8K. By combining the signals, the joint gate retains full-decoding GSM8K accuracy on both backbones, but consumes nearly the full budget. The joint gate therefore leaves most reasoning acceleration to local commitment rather than obtaining it through earlier termination.

Structured commitment supplies the remaining acceleration. With the joint stopping gate in place, adding the CCTC core substantially reduces steps on GSM8K and HumanEval. Building on this reduction, one-step confirmation provides a smaller, consistent further reduction across the reported tasks and backbones; on HumanEval, it also improves the LLaDA point estimate while leaving Dream accuracy unchanged. Furthermore, extending CCTC to the final block adds savings, with a score trade-off on LLaDA GSM8K. The per-block profiles show the resulting distribution of work: short-answer savings largely remove later blocks, whereas reasoning and code savings reduce updates within blocks that remain active (Appendix F).

Scaling and caching. These component-level gains also depend on the decoding configuration. As the generation budget grows, C^4's executed steps grow more slowly, increasing the fraction of computation saved. Block length instead changes the scope of parallel updates and the prediction trajectory. Caching can further reduce runtime in some configurations, with task-performance trade-offs. Appendix G reports the full results.

6 Conclusion

We presented C^4, a training-free framework that reduces diffusion decoding cost through local token commitment and adaptive global stopping. Its central design principle is to match each decision to the evidence it needs: CCTC structures parallel commitments while generation continues, and CVEE uses confidence and revision history to determine when to stop. The experiments show why both controls matter. Early stabilization leaves entire blocks to skip, whereas extended reasoning and code generation benefit more from reducing the work inside blocks that still need to run. Reusing one numerical configuration across the evaluated tasks and both backbones further demonstrates the practical value of adapting decisions to prediction behavior without task-specific recalibration. Future work will extend this approach to multimodal models and longer generations, and combine adaptive stopping with refinement that preserves the coherence and completeness of the full response.

AI USE STATEMENT

Generative AI tools assisted with code implementation and debugging, result aggregation, figure preparation, manuscript drafting and editing, and literature and bibliographic checks. They also supported consistency checks across the method description, reported results, and released code, informing revisions to empirical claims. The authors reviewed and verified the resulting code, analyses, references, and text and take full responsibility for the final content.

REFERENCES

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 17981–17993, 2021a. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/958c530554f78bcd8e97125b70e6973d-Abstract.html.

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

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 28266–28279, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/b5b528767aa35f5b1a60fe0aaeca0563-Abstract.html.

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, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Yuzhe Chen, Jiale Cao, Xuyang Liu, Jin Xie, Aiping Yang, and Yanwei Pang. STDec: Spatio-temporal stability guided decoding for dLLMs. *arXiv preprint arXiv:2604.06330*, 2026.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Muzhi Dai, Chenxu Yang, and Qingyi Si. S-GRPO: Early exit via reinforcement learning in reasoning models. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*, 2025. URL https://openreview.net/forum?id=wNMK5o0Vfg.

Zhenbang Du, Kejing Xia, Xinrui Zhong, Yonggan Fu, Nicolai Oswald, Binfei Ji, Brucek Khailany, Pavlo Molchanov, and Yingyan Lin. R^2-dLLM: Accelerating diffusion large language models via spatio-temporal redundancy reduction. *arXiv preprint arXiv:2604.18995*, 2026.

Yonggan Fu, Lexington Whalen, Zhifan Ye, Xin Dong, Shizhe Diao, Jingyu Liu, Chengyue Wu, Hao Zhang, Enze Xie, Song Han, Maksim Khadkevich, Jan Kautz, Yingyan Celine Lin, and Pavlo Molchanov. Efficient-DLM: From autoregressive to diffusion language models, and beyond in speed. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=ynRnOqI2W8.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Jiawei Gu, Ziyue Qiao, and Xiao Luo. Detecting the semantic fixed point: A geometric framework for efficient inference. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=DACN5xM4h7.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations (ICLR)*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021b. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Feng Hong, Geng Yu, Yushi Ye, Haicheng Huang, Huangjie Zheng, Ya Zhang, Yanfeng Wang, and Jiangchao Yao. Wide-in, narrow-out: Revokable decoding for efficient and effective DLLMs. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/435cce71b4007699041dfffa4f034079-Abstract-Conference.html.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 12454–12465, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/67d96d458abdef21792e6d8e590244e7-Abstract.html.

Jiyeon Kim, Sungik Choi, Yongrae Jo, Moontae Lee, and Minjoon Seo. Early decisions matter: Proximity bias and initial trajectory shaping in non-autoregressive diffusion language models. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/61766.

Seo Hyun Kim, Sunwoo Hong, Hojung Jung, Youngrok Park, and Se-Young Yun. KLASS: KL-guided fast inference in masked diffusion models. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/8514c1f12dcdd061ac8521feb78f28ca-Abstract-Conference.html. arXiv:2511.05664.

Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Yi Liang, Soroush Vosoughi, and Shiwei Liu. Diffusion language model knows the answer before it decodes. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=g88nt4ieTG.

Shufan Li, Konstantinos Kallidromitis, Hritik Bansal, Akash Gokul, Yusuke Kato, Kazuki Kozuka, Jason Kuen, Zhe Lin, Kai-Wei Chang, and Aditya Grover. LaViDa: A large diffusion language model for multimodal understanding. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 105101–105134. Curran Associates, Inc., 2025. doi: 10.52202/085713-3511. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/975affbe7b5f3b55fba62247b6877b1c-Paper-Conference.pdf.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32819–32848, 2024. URL https://proceedings.mlr.press/v235/lou24a.html.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020. URL https://aclanthology.org/2020.acl-main.92/.

Amr Mohamed, Guokan Shang, and Michalis Vazirgiannis. LESS is more: Mutual-stability sampling for diffusion language models. *arXiv preprint arXiv:2606.16908*, 2026a. URL https://arxiv.org/abs/2606.16908.

Amr Mohamed, Yang Zhang, Michalis Vazirgiannis, and Guokan Shang. Fast-decoding diffusion language models via progress-aware confidence schedules. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 35793–35807, 2026b. URL https://aclanthology.org/2026.findings-acl.1782/.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*, 2025. URL https://openreview.net/forum?id=KnqiC0znVF. arXiv:2502.09992.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=sMyXP8Tanm.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2021. URL https://aclanthology.org/2021.naacl-main.168/.

Jiaoyang Ruan, Xin Gao, Yinda Chen, Hengyu Zeng, Liang Du, Guanghao Li, Jie Fu, and Jian Pu. Reasoning on the manifold: Bidirectional consistency for self-verification in diffusion language models. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=CUVBdw2tKy. arXiv:2604.16565.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/eb0b13cc515724ab8015bc978fdde0ad-Abstract-Conference.html.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial Winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and generalized masked diffusion for discrete data. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/bad233b9849f019aead5e5cc60cef70f-Abstract-Conference.html.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning (ICML)*, pp. 2256–2265, 2015. URL https://proceedings.mlr.press/v37/sohl-dickstein15.html.

Ye Tian, Ling Yang, Jiongfan Yang, Anran Wang, Yu Tian, Jiani Zheng, Haochen Wang, Zhiyang Teng, Zhuochen Wang, Yinjie Wang, Yunhai Tong, Mengdi Wang, and Xiangtai Li. MMaDA-Parallel: Multimodal large diffusion language models for thinking-aware editing and generation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=mkQAd11ovn.

Chengcheng Wang, Tingzhang Luo, Wenhao Li, Jianyuan Guo, and Chang Xu. TACG: Trajectory-aware commit gating for diffusion language model decoding. *arXiv preprint arXiv:2607.03236*, 2026.

Qingyan Wei, Yaojie Zhang, Zhiyuan Liu, Puyu Zeng, Yuxuan Wang, Biqing Qi, Dongrui Liu, and Linfeng Zhang. Accelerating diffusion large language models with SlowFast sampling: The three golden principles. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=Uh17FiwF4q. arXiv:2506.10848.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=3Z3Is6hnOT. arXiv:2505.22618.

Andrea Wynn, Metod Jazbec, Charith Peris, Rinat Khaziev, Anqi Liu, Daniel Khashabi, and Eric Nalisnick. Controlling the risk of corrupted contexts for language models via early-exiting. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=8bUDbWMo5v. arXiv:2510.02480.

Yangxinyu Xie, Tao Wang, Soham Mallick, Yan Sun, Georgy Noarov, Mengxin Yu, Tanwi Mallick, and Edgar Dobriban. Statistical early stopping for reasoning models. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=TJshRZDdyW. arXiv:2602.13935.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. Dynamic early exit in reasoning models. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=NpU7ZXafRi.

Ling Yang, Ye Tian, Bowen Li, Xinchen Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. MMaDA: Multimodal large diffusion language models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 138867–138907. Curran Associates, Inc., 2025. doi: 10.52202/085713-4636. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/caa934a507a952698d54efb24845fc4b-Paper-Conference.pdf.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7B: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025. URL https://arxiv.org/abs/2508.15487.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019. URL https://aclanthology.org/P19-1472/.

Lunbin Zeng, Jingfeng Yao, Bencheng Liao, Hongyuan Tao, Wenyu Liu, and Xinggang Wang. DiffusionVL: Translating any autoregressive models into diffusion vision language models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2026. URL https://arxiv.org/abs/2512.15713.

Kevin Zhai, Sabbir Mollah, Zhenyi Wang, and Mubarak Shah. CORE: Context-robust remasking for diffusion language models. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=bmKHxLWkz9. arXiv:2602.04096.

Yu Zhang, Xinchen Li, Jialei Zhou, Hongnan Ma, Zhongwei Wan, Yiwei Shi, Duoqian Miao, Qi Zhang, and Longbing Cao. Swordsman: Entropy-driven adaptive block partition for efficient diffusion language models. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/65444. arXiv:2602.04399.

Kaiwen Zheng, Yongxin Chen, Hanzi Mao, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=CTC7CmirNr.

Appendix Overview

The appendices document the evaluation protocol and provide additional analyses of decoding behavior.

- **A** Experimental settings and implementation details.
- **B** Decoding-loop timing protocol.
- **C** Commitment location and stopping behavior.
- **D** Stopping decisions on real decoding trajectories.
- **E** CVEE with WINO.
- **F** Per-block computation and threshold sensitivity.
- **G** Generation length, block length, and caching.
- **H** Text quality after early exit.
- **I** Limitations and future directions.

A Experimental Setup and Decoding Settings

Prompting and evaluation protocol. All tasks use zero-shot prompts. Reasoning prompts add "Let's think step by step," without demonstrations or additional answer-format instructions. Dream-7B-Instruct uses its official one-position logit shift for next-token indexing; otherwise, its evaluation protocol matches LLaDA-8B-Instruct. Prophet uses its published stopping rule within the same prompting and evaluation pipeline.

Decoding parameters. Table 4 lists the evaluation sizes, decoding budgets, and task-specific settings. Across tasks and backbones, CVEE uses confidence threshold τ_{CVEE}=0.7, revision multiplier γ=2.0, and minimum patience $p_{\min}$=3. CCTC uses confidence threshold τ_{CCTC}=0.8 and gap budget H=1. The extraction mode follows the task's output structure, as described below. The trajectories in Figure 2 are excluded from CVEE calibration.

Task	n	Len.	Steps	Block	search_mode	Prophet ($\tau_{\text{high}}/\tau_{\text{mid}}/\tau_{\text{low}}$)	SchED (mode/patience)	KLASS ($\tau/\epsilon_{\text{KL}}$)	WINO (τ/τ_{back})
MMLU	200	64	64	16	first5	7.5/5.0/2.5	cosine / 0	0.7/0.015	0.5/0.9
ARC-C	200	64	64	16	first5	7.5/5.0/2.5	cosine / 0	0.7/0.015	0.5/0.9
HellaSwag	200	64	64	16	first5	7.5/5.0/2.5	cosine / 0	0.7/0.015	0.5/0.9
WinoGrande	200	64	64	16	first5	7.5/5.0/2.5	cosine / 0	0.7/0.015	0.5/0.9
PIQA	200	64	64	16	first5	7.5/5.0/2.5	cosine / 0	0.7/0.015	0.5/0.9
GSM8K	100	256	256	32	last	8.0/5.0/3.5	linear / 4	0.6/0.015	0.6/0.9
MATH	100	256	256	32	last	8.0/5.0/3.5	linear / 4	0.6/0.015	0.7/0.9
SVAMP	100	256	256	32	last	8.0/5.0/3.5	linear / 4	0.6/0.015	0.6/0.9
ASDiv	100	256	256	32	last	8.0/5.0/3.5	linear / 4	0.6/0.015	0.6/0.9
GSM-Hard	100	256	256	32	last	8.0/5.0/3.5	linear / 4	0.6/0.015	0.6/0.9
HumanEval	164	384	384	32	code	7.5/5.0/4.5	cosine / 4	0.9/0.01	0.7/0.9
MBPP	257	384	384	32	code	7.5/5.0/4.5	cosine / 4	0.7/0.01	0.8/0.9

Table 4: Task settings used in Table 1. n is the number of evaluated examples, Len. is the generation length, Steps is the full decoding budget, and Block is the block length. `search_mode` is determined by the expected output format. KLASS code settings are for LLaDA; Dream values appear below.

Baseline settings. Prophet uses its published staged confidence thresholds, with a separate schedule for code generation. SchED combines its calibrated decay schedule and patience with Prophet's high and low thresholds. Its code experiments use the long-form stability guard, following the available configuration for extended generation; SchED provides no dedicated code setting.

KLASS uses calibrated settings for each task family. For code generation, the released $(\tau, \epsilon_{\text{KL}})$ values are $(0.9, 0.01)$ and $(0.7, 0.01)$ for LLaDA on HumanEval and MBPP, respectively; Dream uses $(0.8, 0.001)$ and $(0.9, 0.001)$. WINO uses published task thresholds where available and the nearest task-family setting otherwise, with τ_{back}=0.9. Fast-dLLM combines DualCache with threshold 0.9.

Locating the monitored span. At each step, a deterministic parser locates the span used by CVEE. The `search_mode` setting connects this extraction to the task's output structure. For reasoning tasks (GSM8K, MATH, SVAMP, ASDiv, and GSM-Hard), `last` searches the final 30% of the generation (`tail_frac=` 0.3). For multiple-choice tasks (MMLU, ARC-C, HellaSwag, WinoGrande, and PIQA), `first5` extracts the first answer letter and locates its matching token sequence within the

first five output positions. For HumanEval and MBPP, `code` tracks the function body using Python indentation to identify its end. Character spans are mapped directly to token indices to preserve multi-line code boundaries.

Confidence and observation updates. CVEE observes spans within active or previously decoded blocks; positions beyond the active block receive confidence $-\infty$. Code stability counters update when span confidence is finite. Missing extractions pause the counters, so the agreement run counts successive available observations, which can be separated by decoding steps with no extraction.

Each forward pass supplies the top-1 probabilities $p_i^{\max}(t) = \max_v p_\theta(v \mid x^{(t)})_i$. CVEE averages these over the eligible span A_t to obtain $c_t = |A_t|^{-1} \sum_{i \in A_t} p_i^{\max}(t)$. The extracted text combines current predictions at masked positions with retained tokens at committed positions. At a committed position, the current top-1 prediction can differ from the retained token, so its confidence does not independently verify that token. When CVEE exits, the current predictions fill all remaining masks in the sequence. This final fill reuses the existing forward pass.

B Throughput Measurement

Timing scope. The RTX 4090 measurements cover the complete decoding loop, including model forward passes, token updates, and method-specific operations such as extraction and stopping checks. Before and after each timed decode, `torch.cuda.synchronize()` ensures that GPU work is complete at the timing boundaries. Elapsed host time is measured with `time.time()`. This procedure applies to Baseline, Prophet Li et al. (2026), SlowFast Sampling (Wei et al., 2026), SchED (Mohamed et al., 2026b), KLASS (Kim et al., 2025), WINO (Hong et al., 2026), and C^4 in the main experiments and ablations.

Evaluation runs. Each method decodes the same fixed set of 20 evaluation examples after warming up on two additional examples. Three complete runs over the timing set yield the mean throughput reported in the paper. Runs use batch size 1, `bf16`, and greedy decoding on an idle RTX 4090. Table 5 summarizes the environment and variation across runs.

Setting	Value
GPU	NVIDIA GeForce RTX 4090
Batch size	1
Precision	`bf16`
PyTorch / CUDA	2.5.1+cu121 / 12.1
Synchronization	`torch.cuda.synchronize()` before and after each timed decode
Timing	Host wall-clock time with `time.time()`
Examples / warmup / repeats	20 / 2 / 3
LLaDA timing CV	0.00–2.98% (mean 0.14%)
Dream timing CV	0.00–0.48% (mean 0.08%)

Table 5: RTX 4090 timing protocol. CV denotes the coefficient of variation across the three repeated timing runs for each measured configuration.

Reading the efficiency metrics. Step counts the executed decoding updates, TPS measures decoding throughput, and Speedup is the ratio of method TPS to the corresponding full-decoding baseline TPS. Together, these metrics describe how much decoding work is removed and how that reduction translates into elapsed time, including the cost of each method's token selection and stopping operations.

C Commitment Location and Stopping Behavior

Local updates shape both the context for future predictions and the observations available to CVEE. We vary where CCTC may commit while retaining the reference top-k schedule, separating changes in the monitored span from changes in its surrounding context.

Effect of Protecting the Candidate Span. We first prevent CCTC from making additional commits inside the candidate span observed by CVEE. Table 6 reports four diagnostics. *Frozen* is the fraction of the accepted span already committed when CVEE fires, and *By CCTC* is the fraction committed specifically by CCTC rather than by the reference decoder's scheduled top-k rule. *Recon. tok.* and

Recon. exact re-mask the accepted span on a copy of the current buffer and measure whether one additional forward pass reconstructs the same tokens or the entire span, respectively. *Same ans.* is the fraction of examples whose graded answer remains unchanged when CCTC is barred from committing inside the candidate span.

	LLaDA-8B-Instruct					Dream-7B-Instruct				
Task	Frozen	By CCTC	Recon. tok.	Recon. exact	Same ans.	Frozen	By CCTC	Recon. tok.	Recon. exact	Same ans.
MMLU	68.1%	37.7%	97.9%	97.9%	100.0%	74.6%	50.3%	97.9%	97.9%	100.0%
GSM8K	11.5%	7.7%	100.0%	100.0%	100.0%	0.0%	0.0%	100.0%	100.0%	100.0%
HumanEval	99.9%	86.3%	33.8%	6.8%	59.1%	99.7%	75.1%	32.2%	12.5%	72.0%

Table 6: Effect of protecting the candidate span from CCTC commits. For MMLU and GSM8K, removing CCTC's additional commits inside the candidate span leaves every graded answer unchanged. HumanEval differs because the candidate is the generated program itself.

On MMLU and GSM8K, span protection changes stopping time slightly while preserving every graded output on both backbones. High reconstruction agreement provides a second view of the same behavior: the selected values remain supported by the current context after remasking. These observations distinguish the timing of acceptance from the identity of the accepted output in this control; they do not establish independence between commitment and stopping.

Effect of Protecting the Entire Candidate Block. Protecting only the candidate span still allows CCTC to commit neighboring positions in the same block, which could indirectly affect the candidate trajectory. We therefore use a stricter control: from the first step at which a candidate is extractable until CVEE fires, CCTC makes no additional commits anywhere in the block containing that candidate. All other blocks retain the deployed rule.

Block protection delays exit more than span protection, indicating that updates to neighboring positions help the monitored prediction settle. Most graded outputs remain unchanged, while the few correctness changes occur in both directions (four gains and two losses; paired sign-test $p = 0.69$). The contrast between these controls highlights context development as part of the interaction: local acceleration can advance stopping even when it does not directly commit the monitored positions.

Effect on Whole-Program Answers. For HumanEval, the monitored unit covers the generated function, so protecting it suppresses local acceleration over most of the program. The resulting increase in decoding cost is therefore an effect of changing the generation process itself. LLaDA accuracy also decreases under this control, while Dream accuracy stays unchanged.

Reconstruction behaves differently here as well. Rebuilding a long program in one parallel pass requires recovering many interdependent tokens, and agreement remains low on both the deployed and protected trajectories. The probe is thus sensitive to reconstruction difficulty as well as to the state of the original generation. Whole-program protection and reconstruction cannot separate local commitment from global stopping as cleanly as the short-span controls.

D Prediction Evolution and Stopping Decisions

Figure 5 follows two trajectories through the signals used by CVEE. Reading each column from top to bottom connects changes in the extracted prediction to confidence and accumulated agreement. The examples complement the task-level patterns in Figure 2 by showing how stopping decisions emerge over successive steps.

Early stabilization. On MMLU, an early revision from D to A is followed by rising confidence and repeated agreement. Confidence crosses its threshold before the required run is complete; CVEE exits at step 5 of 64 once both conditions hold. Most of the generation budget can therefore be skipped after a short period of stable predictions.

Later stabilization. On GSM8K, the extracted value changes repeatedly before settling on 21. These revisions increase the required patience, while confidence becomes available when the span is eligible. CVEE exits at step 178 of 256 as the agreement run reaches that requirement. Prophet's exit at step 139 is marked on the same trajectory, placing both stopping decisions in the context of the evolving predictions. The longer revision history gives temporal agreement a larger role in determining when this trajectory ends.

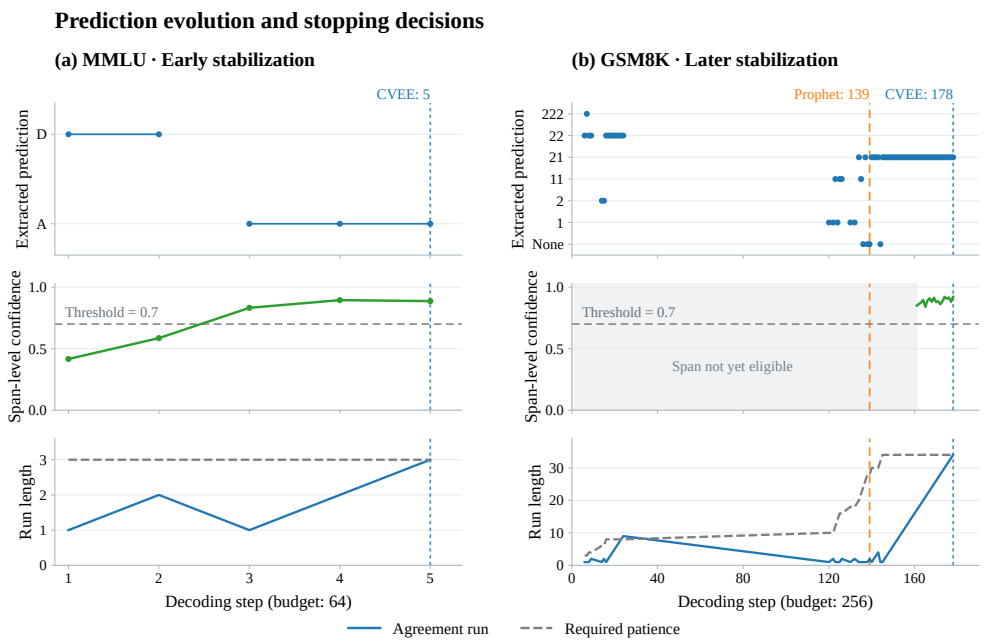

Figure 5: Early and late stabilization along individual decoding trajectories. Each column follows extracted predictions, span-level confidence, and temporal agreement. MMLU settles early; GSM8K undergoes later revisions that increase the required patience. Vertical markers locate CVEE's exit and, on the same GSM8K trajectory, Prophet's earlier exit. Prediction values are categorical; the shaded region marks an ineligible span.

From stopping decisions to generated text. Figures 6 and 7 show the corresponding output-level behavior. All displayed methods produce the correct numerical values, while their final sentences differ in fluency and completion. Badges mark answer presence in the literal text. The released extractor can match trailing `..` and return an empty string, so these badges are assigned by inspecting the displayed outputs. Appendix H examines one-shot-fill quality across tasks for C^4 and the baselines.

D.1 Consistency in Reasoning and Code

Reasoning and code generation require dependencies to remain consistent across the output. An intermediate quantity must carry through to the final calculation, while a computed condition must lead to the requested program behavior. Figures 9 and 10 illustrate these dependencies through selected GSM8K and HumanEval examples from LLaDA-8B-Instruct.

Prompt: Liam and Mitchell own competing lemonade stands across the street from one another. When Liam bragged that he had made $63 one weekend, Mitchell laughed and told Liam he had sold 21 lemonades at $4 apiece the same weekend. How many more dollars did Mitchell make selling lemonade that weekend than Liam?

Baseline (256/256 steps) **Answer present**

```
To determine how many more dollars Mitchell made compared to Liam, we need to
follow these steps:
1.  Calculate Mitchell's earnings:
Mitchell sold 21 lemonades.  Each lemonade was sold for $4.  Therefore:
                              21 x 4 = 84 dollars
2.  Determine the difference:
Liam made $63.  Mitchell made $84.
                             84 - 63 = 21 dollars
So, Mitchell made $21 more than Liam.
```

Prophet (139/256 steps) **Answer present**

```
To determine how many more dollars Mitchell made compared to Liam, we need to
follow these steps:
1.  Calculate Mitchell's earnings:
Mitchell sold 21 lemonades.  Each lemonade was sold for $4.  Therefore:
                              21 x 4 = 84 dollars
2.  Determine the difference:
Liam made $63.  Mitchell made $84.
                             84 - 63 = 21 dollars
So, Mitchell made $21 more than than..
```

(correct numerical answer present; repeated words and malformed punctuation)

C^4 (178/256 steps) **Answer present**

```
To determine how many more dollars Mitchell made compared to Liam, we need to
follow these steps:
1.  Calculate Mitchell's earnings:
Mitchell sold 21 lemonades.  Each lemonade was sold for $4.  Therefore:
                              21 x 4 = 84 dollars
2.  Determine the difference:
Liam made $63.  Mitchell made $84.
                             84 - 63 = 21 dollars
So, Mitchell made $21 more dollars than Liam.
```

(committed here; budget left unspent)

Figure 6: Literal decoded text at each method's stopping step. Real GSM8K trajectory with ground-truth answer 21, corresponding to Figure 5. All outputs state the correct answer, highlighted in blue. Badges indicate answer presence in the displayed text, not automated evaluator scores; red text marks malformed wording or punctuation.

SVAMP (ground truth = 15)

Baseline (256/256 steps) **Answer present**

Sure, let's break it down step by step:

1. Initial number of cakes:
The baker initially made 155 cakes.
2. Cakes bought by the friend:
The friend bought 140 cakes from the baker.

To find out how many cakes the baker still has, we need to subtract the number of cakes bought by the friend from the initial number of cakes.

155 - 140 = 15

So, the baker still has 15 cakes.

Prophet (92/256 steps) **Answer present**

Sure, let's break it down step by step:

1. Initial number of cakes:
The baker initially made 155 cakes.
2. Cakes bought by the friend:
The friend bought 140 cakes from the baker.

To find out how many cakes the baker still has, we need to subtract the number of cakes bought by the friend from the initial number of cakes.

155 - 140 = 15

SoSo, the would still still 15 cakes cakes.

(correct numerical answer present; repeated words and malformed final sentence)

C^4 (256/256 steps) **Answer present**

Sure, let's break it down step by step:

1. Initial number of cakes:
The baker initially made 155 cakes.
2. Cakes bought by the friend:
The friend bought 140 cakes from the baker.

To find out how many cakes the baker still has, we need to subtract the number of cakes bought by the friend from the initial number of cakes.

155 - 140 = 15

So, the baker still has 15 cakes.

(CVEE uses the full budget; the displayed answer and final sentence are correct)

Figure 7: Representative SVAMP trajectory. All three outputs state the correct answer, 15. Prophet stops earlier with a malformed final sentence; CVEE uses the full budget on this example. This comparison illustrates text quality rather than a difference in numerical-answer correctness.

PIQA (ground truth = A)

Baseline (64/64 steps) ✓ **Correct**

```
A. Pile your dirty clothing items into a zip it compressor bag and place bottom
of your suitcase.
```

Prophet (16/64 steps) ✓ **Correct**

```
A. Pile packing dirty clothing items into zip it compressor bag and place in the
bottom of your suitcase.
```

(the extracted letter is still correct, but the one-shot fill drops and reorders words downstream of the answer token itself)

C^4 (3/64 steps) ✓ **Correct**

```
A. Pile your dirty clothing items into a zip it compressor bag and place in the
bottom of your suitcase.
```

(committed after 3 steps; the rest of the buffer is still filled in by the ordinary schedule, not frozen mid-word)

Figure 8: Representative PIQA trajectory. The same literal-decoded-text comparison as Figure 6. Prophet's one-shot fill can degrade surrounding text even when the extracted answer is correct.

GSM8K: Collecting cards. Elaine starts with 20 cards, collects 60 in the first month, 40 in the second, and twice the combined first- and second-month amount in the third. The required total is **320**.

Prophet (189/256 steps) **Answer present**

```
So,, now has a total of 320 Pokemon cards.
```

WINO (44/256 steps) **Answer: 290**

```
Second month:
60 ÷ 20 = 30 cards
Total:  20 + 60 + 30 + 180 = 290
So, Elaine now has a total of **290** Pokemon cards.
```

SlowFast (35/256 steps) **Answer: 220**

```
Third month:  2 × 100 = 200
Total:  20 + 60 + 40 + 100 = 220
So, Elaine now has a total of 220 Pokemon cards.
```

C^4 (34/256 steps) **Answer present**

```
Second month:  60 − 20 = 40 cards
Third month:  2 × (60 + 40) = 2 × 100 = 200 cards
Total:  20 + 60 + 40 + 200 = 320
So, Elaine now has a total of 320 Pokemon cards.
```

Figure 9: Arithmetic consistency in a selected GSM8K example. Excerpts show intermediate calculations and final totals, with equations rendered for readability. WINO changes the second-month calculation; SlowFast substitutes a different value in its final sum. Prophet and C^4 both state the correct total. Red highlights the inconsistent calculations. All methods use LLaDA-8B-Instruct.

Carrying intermediate results forward. The card-counting example links each month's contribution to a final sum. Consistency can break when a quantity is first computed or when it is reused: WINO changes the second-month amount, whereas SlowFast obtains the third-month amount correctly but substitutes a different value in the total. The C^4 output preserves this chain of calculations. Prophet also reaches the correct total despite malformed final wording, separating the arithmetic result from the fluency of its verbal expression.

HumanEval: Returning *x* or *y*. Implement `x_or_y(n, x, y)`: return *x* when *n* is prime and *y* otherwise. For example, `x_or_y(7, 34, 12)` returns 34 and `x_or_y(3, 33, 5212)` returns 33.

Prophet (127/384 steps) **Syntax error**

```
while i * i <= n:
    if n % i == 0 or n %i + + == 0 ==
return return
```

WINO (45/384 steps) **Fails tests**

```
if n <= 1:
    return False
if n <= 3:
    return True
```

SlowFast (39/384 steps) **Fails tests**

```
def x_or_y(n, x, y):
    return x if n == 7 else y
```

C^4 (25/384 steps) **Passes tests**

```
if n <= 1:
    return y
if n <= 3:
    return x
```

Figure 10: Return behavior in a selected HumanEval example. Excerpts isolate the relevant branches of each generated program. Prophet's output contains invalid syntax; WINO returns Boolean values; SlowFast handles only the visible prime example. C^4 returns the requested values. Test outcomes refer to the complete programs.

Connecting conditions to return values. In the code example, determining primality is an intermediate operation: the function must use that condition to choose between x and y. Returning a Boolean loses this connection, while checking only $n = 7$ replaces the general rule with a single example. The completed C^4 program connects a general primality check to the requested return values and passes the recorded tests. Together, these examples illustrate how correctness depends on carrying intermediate computations through to their intended use.

E Early Exit with WINO

We apply CVEE to WINO to examine stopping behavior on a different sampling trajectory (Table 2). WINO retains its local update rule, while CVEE evaluates sequence termination using predictions and probabilities from the same forward passes. This configuration pairs WINO commitments with CVEE stopping, so the local sampling trajectory is determined entirely by WINO.

Early stabilization leaves room for termination. On MMLU and HellaSwag, CVEE removes most of WINO's remaining steps with nearly unchanged accuracy (Table 2). Local sampling has accelerated token updates, but stabilization still precedes completion of the scheduled generation. CVEE uses this interval to skip updates after the prediction has settled.

Effect on long-reasoning and code. The same composition is less effective on GSM8K, SVAMP, and HumanEval. With the fixed CVEE configuration, accuracy drops by 23–36 points. We additionally

evaluate 45 combinations of $\tau_{\text{CVEE}} \in \{0.7, 0.8, 0.9\}$, $p_{\min} \in \{3, 5, 8, 12, 16\}$, and $\gamma \in \{2, 4, 8\}$ for each task. Recovering accuracy requires substantially more decoding. The recovered operating points lie close to WINO's original step count. Thus, in these tasks, the remaining trajectory contains little computation that this exit rule can remove while retaining the original task performance.

Effect of WINO commitments on CVEE. The task contrast reflects where useful computation remains after local acceleration. Reasoning trajectories continue changing during much of their shortened decoding horizon, whereas short-answer trajectories can stabilize with several scheduled steps still available for CVEE to remove through global termination.

WINO's local commitments also affect the evidence observed by CVEE. Varying τ_{CVEE} at fixed $(p_{\min}, \gamma)$ does not change accuracy on the three long-output tasks, indicating that confidence is not the limiting condition. This observation is consistent with the temporal condition determining the tested trade-off. Because committed positions retain their tokens, repeated extractions can also reflect local commitments; the threshold sweep alone does not identify the cause of premature exits.

This result illustrates the coupling between local commitment and global termination. Appendix C examines the same interaction within C^4. CCTC instead structures additional commitments through a boundary-anchored core and one-step confirmation alongside CVEE tracking of the answer.

F Computation Profiles and Commitment Thresholds

Where decoding work remains. The block profiles show how stabilization timing shapes computation across the sequence (Figures 11 and 12). Short-answer tasks usually finish within the first one or two blocks, leaving later blocks unused. Reasoning and code continue farther into the sequence, where CCTC reduces the updates within each block. These patterns connect the component contributions in Section 5.3 to the location of the work saved during generation.

How the threshold shapes commitment. The confidence threshold τ_{CCTC} controls which masked positions become eligible for additional commitment. Lower values admit more predictions per update, reducing the opportunity to revise them under later context. The sweeps in Figures 13 and 14 trace the resulting changes in accuracy and computation across the 12 tasks; the main experiments use 0.8 throughout. Sweep speedups use NVIDIA A100 timings normalized to the 0.9 setting.

The threshold has different consequences across task regimes. Short-answer accuracy changes little over much of the tested range, while reasoning and code exhibit sharper declines as commitment becomes more aggressive. Step counts and throughput change more gradually, so a modest computational saving can carry a substantial accuracy cost. The useful operating range therefore depends on how much a task benefits from revising intermediate predictions.

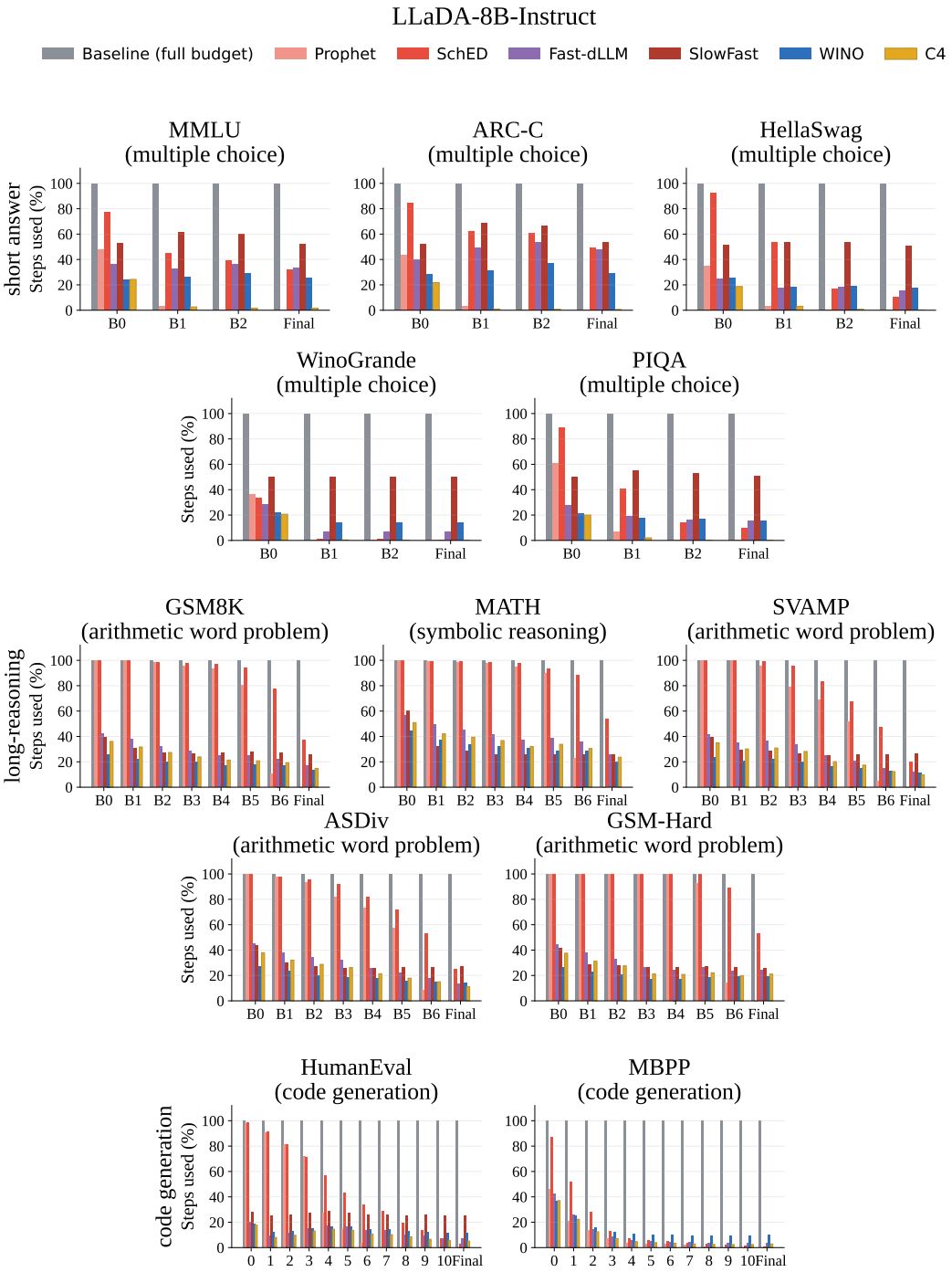

Figure 11: Per-block step usage across all 12 tasks, LLaDA-8B-Instruct. Short-answer tasks terminate early, whereas long-reasoning and code tasks retain more within-block computation.

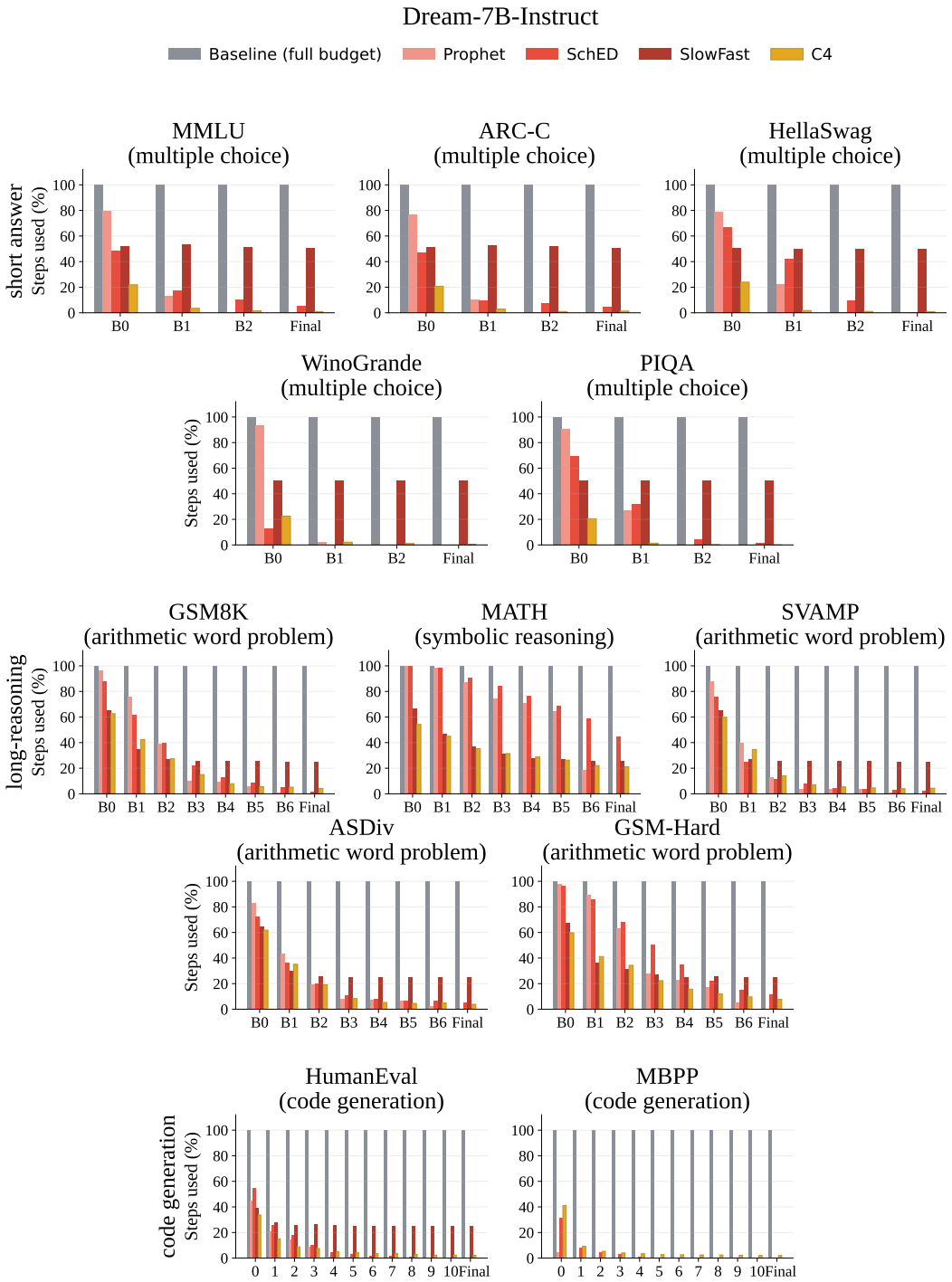

Figure 12: Per-block step usage across all 12 tasks, Dream-7B-Instruct. Short-answer computation is concentrated in early blocks; reasoning and code extend across more of the sequence, with savings from fewer updates within blocks.

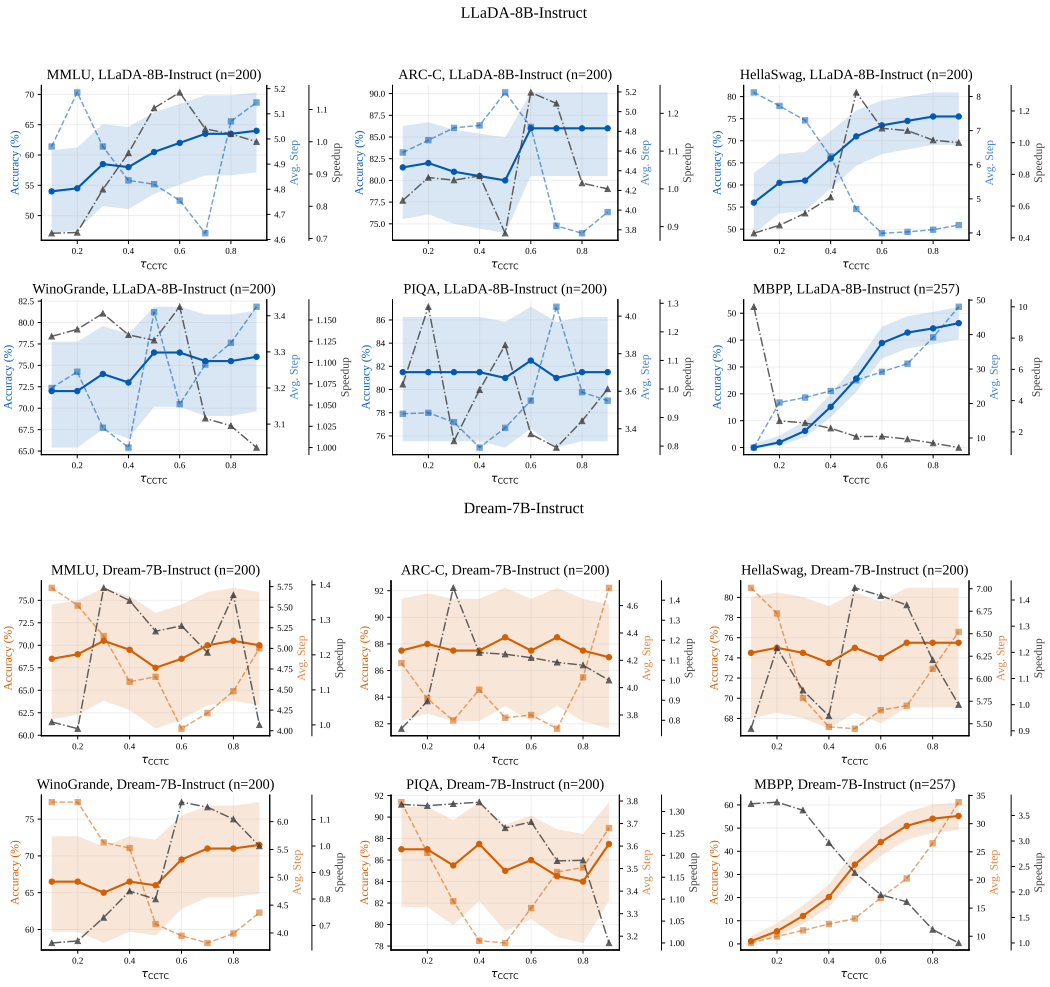

Figure 13: Effect of τ_{CCTC} on short-answer tasks and MBPP. LLaDA-8B-Instruct is shown above and Dream-7B-Instruct below. Solid curves show accuracy with Wilson-score 95% intervals, dashed curves show average Step, and dash-dot curves show Speedup relative to τ_{CCTC}=0.9. Accuracy remains stable across most of the threshold range on the five short-answer tasks, while MBPP accuracy declines with more aggressive commitment. Measurements use an NVIDIA A100.

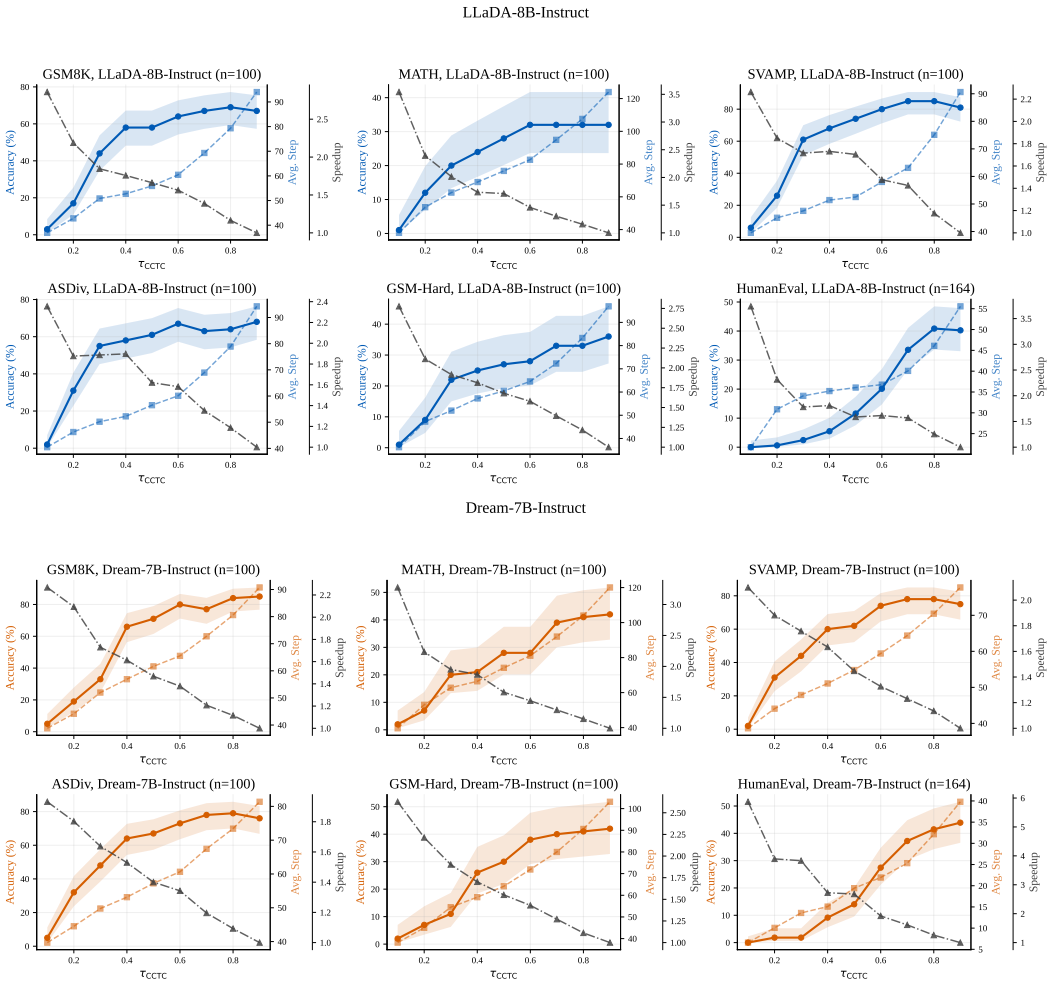

Figure 14: Effect of τ_{CCTC} on long-reasoning tasks and HumanEval. Lower thresholds reduce decoding work but can sharply decrease accuracy, revealing a quality cost to committing more predictions under incomplete context. Top: LLaDA-8B-Instruct. Bottom: Dream-7B-Instruct. Timings are measured on an NVIDIA A100 and normalized to the 0.9 threshold setting.

G Generation Length, Block Length, and Caching

LLaDA-8B-Instruct				C^4					+ DualCache					Fast-dLLM (DualCache+parallel)				
L	B	N	Baseline	Acc	Step	Saving	TPS	Speedup	Acc	Step	Saving	TPS	Speedup	Acc	Step	Saving	TPS	Speedup
HumanEval (code generation), block length swept at fixed generation length 384																		
384	16	24	41.5	39.6	44.5	88.4%	104.5	7.56×	32.3	45.4	88.2%	127.1	9.22×	39.0	64.4	83.2%	102.2	7.41×
384	32	12	40.2	42.1	39.4	89.7%	120.0	8.70×	34.8	44.7	88.4%	154.3	11.18×	37.8	57.2	85.1%	132.2	9.58×
384	64	6	39.6	42.1	36.5	90.5%	128.7	9.30×	36.6	42.2	89.0%	137.9	10.01×	36.6	51.1	86.7%	143.1	10.38×
GSM8K (long-reasoning), generation length swept at fixed block length 32																		
128	32	4	66.0	67.0	44.7	65.1%	77.3	2.51×	63.0	48.6	62.1%	51.7	1.70×	68.0	58.4	54.4%	45.0	1.48×
256	32	8	66.0	64.0	62.5	75.6%	67.5	3.55×	66.0	68.0	73.5%	75.1	3.95×	66.0	80.4	68.6%	63.6	3.35×
512	32	16	72.0	72.0	81.8	84.0%	75.8	5.84×	75.0	88.0	82.8%	108.2	8.34×	73.0	101.9	80.1%	92.4	7.13×
GSM8K (long-reasoning), L and B scaled together at fixed block count 8*, so B/L is held constant*																		
128	16	8	65.0	68.0	47.2	63.1%	73.2	2.39×	66.0	52.4	59.0%	54.5	1.79×	66.0	60.8	52.5%	44.8	1.47×
256	32	8	66.0	64.0	62.5	75.6%	67.5	3.55×	66.0	68.0	73.5%	75.1	3.95×	66.0	80.4	68.6%	63.6	3.35×
512	64	8	70.0	77.0	76.8	85.0%	77.2	5.94×	72.0	80.3	84.3%	115.4	8.90×	74.0	97.0	81.1%	99.8	7.69×
MMLU (short-answer), generation length swept at fixed block length 16																		
16	16	1	56.5	58.0	4.0	75.3%	128.3	3.83×	55.5	4.5	72.1%	82.9	2.80×	55.0	6.2	61.4%	59.3	1.96×
32	16	2	61.5	62.5	4.0	87.5%	247.2	7.37×	59.0	4.6	85.6%	147.9	4.94×	59.5	7.5	76.7%	93.3	3.12×
64	16	4	64.0	64.0	4.8	92.5%	490.4	16.13×	60.0	5.1	92.1%	355.5	11.69×	59.5	20.2	68.4%	77.8	2.56×

Table 7: Generation length and block length. Longer generations offer more computation to skip; block length sets the scope of parallel commitment. DualCache adds per-pass acceleration. L: generation length; B: block length; $N{=}L/B$: block count; Baseline: full-decoding accuracy.

Longer budgets create more removable computation. At fixed block length, increasing generation length adds blocks to the reference schedule (Table 7). MMLU uses nearly constant steps under C^4, while GSM8K steps grow more slowly than the full budget. Savings also increase when generation and block length grow together at fixed block count. The benefit therefore extends beyond simply having more blocks to skip: longer within-block budgets also leave work for local acceleration to remove.

Block length changes how computation is organized. At fixed generation length, the HumanEval block-length sweep changes the scope of parallel updates and how often generation advances to a new block. Step savings vary less than in the generation-length sweeps, while accuracy changes across configurations. Block length thus shapes the prediction trajectory and the efficiency of local updates, rather than merely increasing the total budget available to skip.

Caching changes the cost of the remaining passes. Fast-dLLM combines caching with parallel commitment; C^4 additionally skips later blocks through global stopping, with a pronounced advantage on longer MMLU generations. DualCache can further accelerate C^4 on longer GSM8K generations, but its benefit is not uniform across settings. On HumanEval, cached C^4 uses fewer steps than Fast-dLLM, while uncached C^4 achieves higher accuracy than either cached variant. Reducing the number of passes and reducing their cost are therefore complementary directions, but caching also changes the predictions that drive commitment and stopping.

H Response Quality and Post-Exit Refinement

The answer and its surrounding explanation can settle at different stages of decoding. On short-answer tasks, CVEE may reach its stopping condition while parts of the continuation remain masked. Filling these positions in one pass completes the output, but can introduce repetition and disrupt fluency. This section examines that behavior on the five short-answer tasks in Table 1, then explores how additional decoding shapes the completed response.

Quality of the completed response. We retain the full text generated by C^4 and full-budget decoding on the same examples. Table 8 characterizes these outputs using generative perplexity under a frozen GPT-2 (*PPL*), the fraction of outputs that differ from full-budget decoding (*Changed*), and the fraction of adjacent token repetitions (*Repeat*).

The extracted answer matches full-budget decoding on 98–100% of examples, while the completed text changes more frequently. Quality also varies across tasks with similar exit times: ARC-C exhibits more repetition than PIQA. The text still unresolved at exit therefore appears relevant to completion quality alongside the timing of termination.

Allocating computation to refinement. After CVEE first fires at step t, delayed filling runs ordinary decoding for $\lceil \alpha\sqrt{t} \rceil$ additional steps before filling the remaining masks. The parameter α controls the refinement budget, with $\alpha{=}0$ corresponding to the immediate fill used in the main experiments. These extra updates progressively resolve the continuation before its final completion.

	LLaDA-8B-Instruct				Dream-7B-Instruct			
Task	Base PPL	C^4 PPL	Changed	Repeat	Base PPL	C^4 PPL	Changed	Repeat
MMLU	27.3	95.3	57%	19.5%	56.5	97.6	22%	7.0%
ARC-C	21.4	73.0	86%	26.9%	61.2	81.9	24%	12.6%
HellaSwag	44.6	65.1	35%	3.6%	48.4	100.5	49%	4.7%
PIQA	37.5	59.9	38%	2.1%	51.3	68.6	26%	1.4%
WinoGrande†	518.7	865.2	17%	—	1216.5	1198.6	2%	—

Table 8: Full-response quality after one-shot completion. PPL is GPT-2 perplexity; Changed measures full-text differences from full-budget decoding; Repeat measures adjacent token repetition. Early completion often raises perplexity and introduces repetition in the continuation. †WinoGrande outputs have a median length of two words, limiting the interpretation of perplexity and repetition.

	LLaDA-8B-Instruct						Dream-7B-Instruct					
	MMLU			ARC-C			MMLU			ARC-C		
α	Acc	Step	Repeat	Acc	Step	Repeat	Acc	Step	Repeat	Acc	Step	Repeat
Baseline	64.0	64.0	0.0%	86.5	64.0	0.0%	71.5	64.0	0.0%	87.5	64.0	0.0%
0	64.0	4.8	19.5%	86.5	3.9	26.9%	70.0	4.5	7.0%	88.5	4.2	12.6%
1	63.5	6.9	18.9%	86.5	5.9	26.7%	70.0	6.4	7.8%	88.5	6.1	5.8%
2	63.5	8.4	15.5%	87.0	7.8	23.0%	70.5	7.2	5.6%	88.5	7.1	4.9%
4	63.5	9.9	12.6%	87.0	9.9	20.5%	70.5	7.6	5.0%	88.5	7.4	4.4%
8	63.5	12.7	7.9%	87.0	14.3	12.0%	70.5	8.3	2.4%	88.5	7.7	2.8%
16	63.5	17.8	1.7%	87.0	22.7	2.4%	70.5	9.4	0.2%	88.5	8.2	1.0%

Table 9: Additional refinement trades decoding steps for lower repetition. Larger α allocates more updates between the CVEE trigger and the final fill. Repetition generally falls as this budget grows, with small changes in task accuracy.

The additional updates mainly improve the continuation: repetition falls while task accuracy changes little. Dream reaches low repetition with fewer added steps than LLaDA, suggesting that refinement cost depends on the backbone as well as the task. These results identify a further use for the computation saved by early exit: part of that budget can support response refinement after the stopping criterion is met. Extending this policy to account for coherence and completeness would connect the observed reduction in repetition to broader response quality.

I Future Directions

The interaction between local commitment and global stopping opens several directions for allocating computation as predictions evolve.

Multimodal diffusion decoding. MMaDA (Yang et al., 2025) extends diffusion modeling to textual reasoning, multimodal understanding, and text-to-image generation, while LaViDa (Li et al., 2025) applies diffusion language modeling to multimodal understanding. MMaDA-Parallel (Tian et al., 2026) further enables bidirectional text–image interaction throughout denoising for thinking-aware editing and generation. These models offer settings in which to study local commitment and global stopping beyond text-only inputs. For visually grounded responses, an extension of CVEE could combine textual agreement with evidence that predictions remain consistent with the image. The coupled trajectories in MMaDA-Parallel motivate a further question: different modalities and regions may stabilize at different times, calling for modality-specific commitment criteria and stopping decisions that account for cross-modal consistency. Evaluating these criteria would require both task-level correctness and perceptual quality, with calibration reassessed for the new modalities.

Adaptive response refinement. Delayed filling reduces repetition by spending part of the saved budget on the continuation (Appendix H). This suggests adapting the refinement budget to the amount and structure of unresolved text. Immediate one-shot filling can leave a repetitive or less fluent response, so extending refinement to coherence, completeness, and instruction following would broaden the quality objectives guiding computation.

Revisiting early commitments. Later context can provide useful evidence for revising earlier token choices. CCTC currently retains committed tokens, while CVEE tracks agreement without independently verifying those tokens. Selective revision could use changes in context to identify positions worth reopening. Jointly deciding which tokens to revisit and when to stop would extend adaptive computation to the revision of committed text.

Completion across structured outputs. Long programs and multi-part responses require coordination among dependent components. Extending CVEE's current single-unit tracking to multiple regions could make stopping sensitive to both local completion and cross-region dependencies. Completed regions could remain fixed while unresolved components receive further updates, with execution or consistency checks informing the allocation of computation.

Transfer with statistical risk control. The current results establish empirical behavior on the evaluated tasks and backbones; formal risk guarantees remain an open direction. Statistical risk control (Xie et al., 2026; Wynn et al., 2026) offers a basis for connecting early termination to explicit error tolerances. Applying it to adaptive commitment would require modeling how local updates change the observations used for stopping and how calibration transfers under distribution shift.