# TEMPO: TEMPORAL AND POSITIONAL PRECISION FOR DIFFUSION LANGUAGE MODEL AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Diffusion large language model (dLLM) agents recompute a fixed-length generation region at each denoising step. Reserved positions beyond the emitted call account for most of this computation. Removing them reduces accuracy because they provide context for uncommitted predictions, requiring a reduction in computation per position. Quantisation reduces this cost, but existing quantisers use a fixed configuration across steps and positions, irrespective of the decoding state defined by committed tokens and current predictions. We identify three phenomena that enable precision to adapt to this state. *Filler-first ordering* places end-of-text and padding commitments before content commitments, allowing early steps to run at low precision. *Margin persistence* correlates logit gaps across steps, indicating the risk of quantisation-induced prediction changes before each pass. *Slack tolerance* permits low-precision updates of the reserved region with limited changes in accuracy. These phenomena motivate TEMPO (Temporal and Positional Orchestration of Precision), a training-free hierarchical precision schedule. A step-wise gate selects low-precision steps using cached gaps. On other steps, position-wise routing retains full precision only for the prompt and estimated content region. Across four agent benchmarks and three backbones, TEMPO achieves **84%** mean call fidelity, compared with **76%** for full quantisation and **66%** for Learn2PD. Adding TEMPO to parallel decoding with boundary fill yields a further $\mathbf{1.20}$ to $\mathbf{1.45}\times$ speedup on Spider and API-Bank. The full composition achieves up to $\mathbf{98.8}\times$ at 128 denoising steps. Our code is openly accessible to support future research.

## 1 INTRODUCTION

Diffusion large language models (dLLMs) (Austin et al., 2021; Lou et al., 2023; Sahoo et al., 2024; Shi et al., 2024; Nie et al., 2025) are increasingly studied as backbones for agentic workflows (Li et al., 2026) that involve reasoning, tool selection and the generation of structured function calls, API requests and SQL queries (Yao et al., 2022; Schick et al., 2023; Patil et al., 2024). LLaDA-8B and Dream-7B achieve accuracy comparable to similarly sized autoregressive models on these tasks (Nie et al., 2026; Ye et al., 2025; Gong et al., 2025). A dLLM refines a masked generation region over a fixed number of *denoising steps*. At each step, a *sampler* irreversibly *commits* predictions selected by confidence. Committed tokens and current predictions constitute the *decoding state*. Through bidirectional attention, the representation at every generation position can change as this state evolves.

Consequently, each denoising step recomputes the fixed-length generation region (Gwak et al., 2026). This length is specified before decoding. Agent calls reach $6.1\times$ the median length (§3), requiring sufficient capacity for long outputs to avoid truncating structured calls. We distinguish the *content region* occupied by the emitted call from the *reserved region* beyond it. Their interface is the *semantic boundary*. Even at the shortest generation length that avoids truncation, the reserved region occupies **57–84%** of generation positions for a median-length call (§3) and therefore accounts for most of the computation across denoising steps.

Existing acceleration methods reduce the number of steps, the number of positions or the computation per position. Step reduction and feature reuse decrease the number of evaluations (Wu et al., 2026; Liu et al., 2025b), while position removal reduces the sequence length (Bao et al., 2026; Zuo



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.

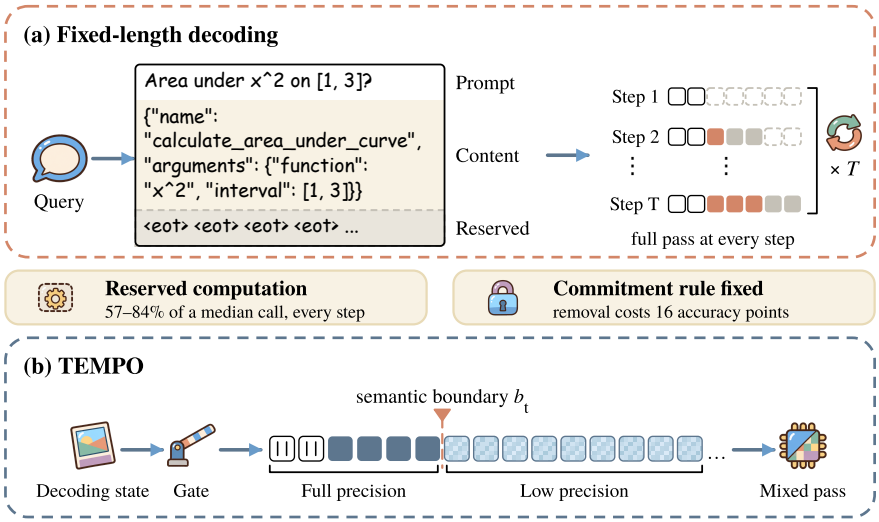


Figure 1. (a) A dLLM repeatedly evaluates a fixed-length generation region. The emitted call occupies the content region, while the reserved region accounts for much of the computation and can affect decoding. (b) TEMPO selects precision from the decoding state and updates the reserved region at low precision.

et al., 2026). However, the reserved region supports decoding by providing context through bidirectional attention for predictions at uncommitted positions. On LLaDA-BFCL, truncation to the oracle output length lowers accuracy by **16 percentage points**, and freezing the region's hidden states lowers it by **11 points** (§3). The reserved region should therefore remain computationally active, requiring a reduction in computation per position. Quantisation reduces this cost, which is dominated by linear layers (Appendix A), while retaining all positions (Dettmers et al., 2022; Frantar et al., 2022; Lin et al., 2024; 2026; Xu & Yang, 2025; Zhang et al., 2026b; Yan et al., 2026). These quantisers apply a single configuration across steps and positions, irrespective of the decoding state. Adaptive methods for autoregressive LLMs (Elbayad et al., 2019; Schuster et al., 2022; Kwon et al., 2026; Tao et al., 2025) select precision or depth once per token, whereas a dLLM's decoding state evolves at each step until the corresponding position is committed. This distinction motivates our central question.

**Can precision adapt to the decoding state without removing any position?**

We characterise precision demand across denoising steps and sequence positions through three phenomena (§3). Under *filler-first ordering*, end-of-text and padding tokens (*filler*) are committed before call tokens (*content*) on all three backbones (Figure 2(b)). This ordering makes early steps predominantly devoted to filler and permits their execution at low precision. Under *margin persistence*, top-1/top-2 logit gaps are correlated across consecutive steps ($r$=0.871), allowing gaps from the preceding pass to indicate whether low precision would alter a content prediction before the current pass is executed. Under *slack tolerance*, low-precision updates of the reserved region change accuracy by at most three percentage points across the evaluated backbones. Positions beyond the semantic boundary can therefore remain active through low-precision updates.

These phenomena motivate **TEMPO** (Temporal and Positional Orchestration of Precision), which provides a training-free hierarchical precision schedule (Figure 1(b)). A step-wise gate uses cached gaps from the preceding pass to identify steps that can execute entirely at low precision. When the gap of a candidate content commitment falls below the threshold, position-wise routing maintains full precision for the prompt and the estimated content region and updates the reserved region at low precision. One offline calibration determines one threshold per backbone. **TEMPO** reduces computation per position while retaining all positions and the original commitment rule. It therefore combines with methods that reduce the step count, including parallel decoding and boundary fill. The latter commits the end-of-text tail once the semantic boundary stabilises (Wu et al., 2026; Bao et al., 2026).

Across LLaDA-8B, Dream-7B and CoDA-1.7B (Chen et al., 2025) on four agent benchmarks (Patil et al., 2025; Li et al., 2023; Huang et al., 2024), **TEMPO** reproduces **84%** of full-precision calls on average, compared with **76%** for full quantisation and **66%** for Learn2PD, while evaluating linear layers at low precision for **53–96%** of positions. On Spider and API-Bank, adding TEMPO to parallel decoding with boundary fill yields a further $\mathbf{1.20}$ to $\mathbf{1.45}\times$ speedup (§5.2). At $T{=}128$, the full composition achieves speedups of $\mathbf{52}\times$ on LLaDA-8B and $\mathbf{98.8}\times$ on Dream-7B. Our contributions are summarised below.

❶ ***Characterisation of Precision Demand.*** Across three backbones, we identify *filler-first ordering*, *margin persistence* and *slack tolerance* as empirical bases for allocating precision across steps and positions.

❷ ***Dynamic Precision across Steps and Positions.*** **TEMPO** combines a step-wise gate based on cached logit gaps with position-wise routing based on the estimated semantic boundary.

❸ ***Cross-Backbone Validation and Composition.*** With one threshold per backbone, **TEMPO** improves mean call fidelity over full quantisation and Learn2PD across four benchmarks and provides additional acceleration when combined with parallel decoding and boundary fill.

## 2 Related work

**dLLM acceleration.** A dLLM applies bidirectional attention to the combined prompt and generation sequence at every denoising step. The changing hidden states prevent direct reuse of standard autoregressive KV caches, so generation cost depends on both step count and sequence length. This structure is shared by LLaDA (Nie et al., 2026), Dream (Ye et al., 2025) and CoDA (Chen et al., 2025). Existing acceleration methods target the number of steps, the effective sequence length, or feature recomputation. Fast-dLLM (Wu et al., 2026) reduces the step count by committing all masked positions above a confidence threshold and adds a block-wise approximate KV cache. Learn2PD (Bao et al., 2026) shortens the effective sequence by using a lightweight filter to predict whether each position already matches its final output, together with truncation of the end-of-text tail. dLLM-Cache (Liu et al., 2025b) reuses features across steps by caching the prompt over extended intervals and refreshing only response tokens whose value vectors change. These methods save computation through fewer steps, shorter sequences, or fewer feature updates. **TEMPO** instead leaves the denoising steps, sequence positions and commitment rule unchanged and lowers the cost of active computation by varying its numerical precision.

**dLLM quantisation.** Recent work adapts quantisation to the denoising dynamics of dLLMs. Lin et al. (2026) systematically analyse dLLM quantisation and attribute much of its degradation to activation outliers. DLLMQuant (Xu & Yang, 2025) calibrates across denoising time and mask ratio and uses attention interactions to allocate activation precision. Quant-dLLM (Zhang et al., 2026b) quantises weights to two bits, STaR-Quant (Yan et al., 2026) improves consistency across decoding states and steps, and FAIR-Calib (Huang et al., 2026) emphasises positions with fragile predictions during calibration. Despite their different strategies, these methods deploy one low-precision configuration that remains fixed across denoising steps and sequence positions. Unlike these methods, **TEMPO** leaves the underlying quantiser unchanged and uses the online decoding state to schedule an existing low-precision configuration across steps and positions.

**Autoregressive LLM quantisation.** Post-training quantisation converts a pretrained model into a fixed low-precision configuration before decoding. GPTQ (Frantar et al., 2022) and OmniQuant (Shao et al., 2024) reduce weight-quantisation error through layer-wise reconstruction and learnable calibration. SmoothQuant (Xiao et al., 2023) moves activation outliers into the weight domain, while QuaRot (Ashkboos et al., 2024) and SpinQuant (Liu et al., 2025a) use rotations to distribute outlier magnitude more evenly. These methods construct low-precision representations, whereas TEMPO selects when and where to use a deployed configuration. Runtime precision allocation has also been studied for autoregressive LLMs. DP-LLM (Kwon et al., 2026) assigns bitwidths to individual layers at each decoding step using learned precision selectors. Cocktail (Tao et al., 2025) and MixKVQ (Zhang et al., 2026a) adapt KV-cache precision according to the query and context at chunk, channel, or token granularity. These methods schedule precision by layer or KV-cache unit within autoregressive decoding. **TEMPO** instead targets diffusion decoding and provides training-free precision scheduling across denoising steps and sequence positions.

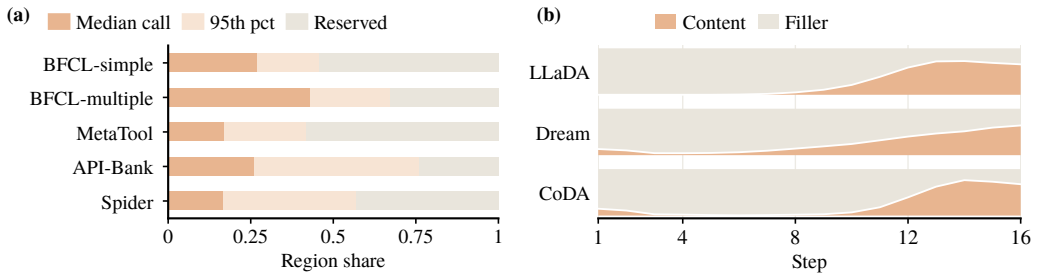


Figure 2. (a) Median and 95th-percentile ground-truth call lengths and the provisioned generation length on five benchmarks. (b) Content share of each step's commits (BFCL, $T$=16, $n$=100).

## 3 Precision demand in dLLM agent decoding

A dLLM specifies one generation length per batch before decoding. Across the five benchmarks in Figure 2(a), the maximum ground-truth call length is 2.3–6.1× the median. Even the shortest length that avoids truncating any call leaves a reserved region occupying 57–84% of the median call's generation region. Every position participates in each forward pass. We characterise this computation through *filler-first ordering*, *margin persistence* and *slack tolerance*. We use *content* for tokens in the emitted call and *filler* for end-of-text and padding tokens.

- **Filler-first ordering.** The samplers of the evaluated backbones rank masked positions by the model's certainty. Filler is highly predictable and tends to be committed before content. Figure 2(b) shows that content commits are concentrated in later steps on LLaDA, Dream and CoDA. Across these backbones and their confidence-based commitment rules, the mean commit step is 5.8–6.4 of 16 for filler and 10.7–12.4 for content. Selecting commitment positions at random reduces the filler-content gap to 0.2 steps (Appendix B). This ordering separates early steps dominated by filler from later steps with more content commitments, motivating different precision allocations across steps.
- **Margin persistence.** A precision decision must precede the current forward pass. The preceding pass provides a useful signal because top-1/top-2 logit gaps are correlated across consecutive steps. Across 28,800 position-step pairs, consecutive gaps correlate at $r$=0.871, and the arg-max remains unchanged in 98.8% of cases (Figure 6). A small gap indicates close competition between the leading tokens and susceptibility to a quantisation-induced arg-max change. Using the threshold calibrated in §4.4, the gap detects 96.9% of these changes at a 28.2% false positive rate (Figure 5(a)). Appendix C compares alternative signals. These observations motivate using cached gaps to estimate whether low precision will change the tokens selected for commitment in the current step.
- **Slack tolerance.** The reserved region affects decoding even though its tokens do not appear in the final call. Before the output layout stabilises, bidirectional attention allows its states to influence predictions at uncommitted positions. On LLaDA-BFCL (Patil et al., 2025), truncating the generation region to the oracle output length reduces accuracy by 16 percentage points, and retaining the region while freezing its hidden states reduces accuracy by 11 points. Updating this region through the deployed low-precision configuration changes accuracy by at most three points across the three backbones (Table 7). Additional interventions are reported in Appendix D. The region therefore tolerates reduced precision even when removal or freezing disrupts decoding, motivating continued updates at lower precision.

## 4 TEMPO

### 4.1 Overview

**TEMPO** converts the observations of §3 into a step-wise gate and position-wise routing without changing the underlying decoding process. Let $L_p$ and $L_g$ denote the prompt and generation lengths, and let $T$ be the number of denoising steps. The token state at step $t$ is $x^{(t)} \in \mathcal{V}^{L_p+L_g}$, where $\mathcal{V}$ is the vocabulary. Uncommitted generation positions contain the mask symbol. Each forward

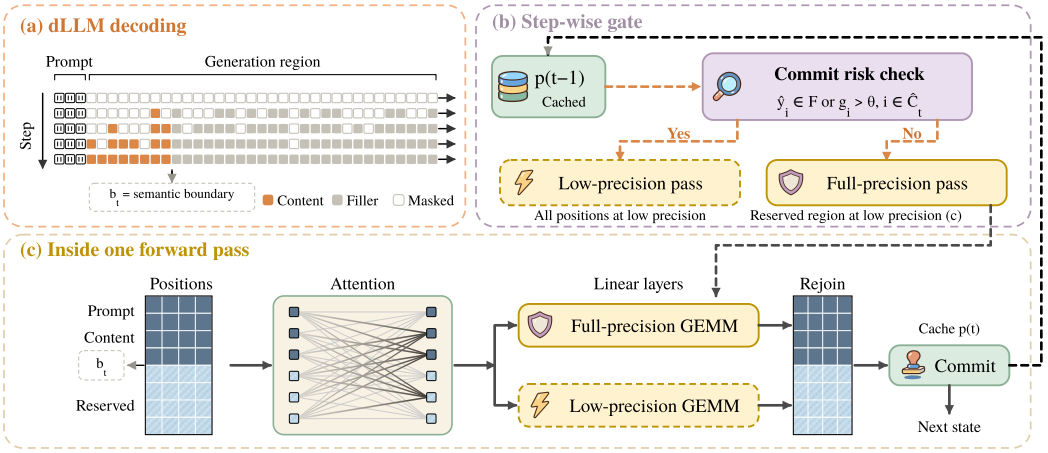


Figure 3. TEMPO allocates precision at two levels. (a) Each step evaluates the full sequence. (b) Cached predictions $\hat{y}_i$ and logit gaps $g_i$ from the preceding pass determine whether the current step runs entirely at low precision. (c) Otherwise, position-wise routing retains full precision through the prompt and the content region and uses low precision for the reserved region beyond the semantic boundary.

pass produces logits for every position. The sampler then selects a commit set $S_t$ and irreversibly assigns the corresponding arg-max tokens. The generation region comprises the *content region*, which contains the valid output, and the *reserved region* beyond the semantic boundary. The prompt precedes both regions. We write $\mathcal{F}$ for the set of filler symbols, comprising end-of-text and padding tokens.

Figure 3 illustrates the step-wise gate (§4.2) and position-wise routing (§4.3).

### 4.2 Step-wise gate

The step-wise gate uses cached logits to estimate whether quantisation may change the tokens selected for commitment in the current step. *Margin persistence* (§3) motivates using logits from the preceding pass without an additional model evaluation. Let $z_i^{(t-1)} \in \mathbb{R}^{|\mathcal{V}|}$ denote these logits at position $i$, and define

$$\hat{y}_i = \arg\max_v z_{i,v}^{(t-1)}, \qquad g_i = z_{i,\hat{y}_i}^{(t-1)} - \max_{v \neq \hat{y}_i} z_{i,v}^{(t-1)} \tag{1}$$

as the predicted token and top-1/top-2 logit gap. The set $U_t$ contains the positions masked after step $t-1$. The cached ranking defines a candidate set $\widehat{C}_t \subseteq U_t$ containing the top-$2k$ positions for a fixed-count sampler that commits $k$ positions per step, or the supra-threshold positions for a confidence-threshold sampler. The gate assigns the whole step to low precision when every candidate predicts a filler token or a token with gap above $\theta$, as expressed by

$$\text{safe}(t) \;=\; \bigwedge_{i \in \widehat{C}_t} \Big[\, \hat{y}_i \in \mathcal{F} \;\vee\; g_i > \theta \,\Big], \tag{2}$$

where $\theta$ is calibrated as described in §4.4. Otherwise, position-wise routing retains full precision for the prompt and estimated content region (§4.3).

### 4.3 Position-wise routing

Position-wise routing is used when the step-wise gate does not assign the entire step to low precision. It estimates the semantic boundary from the current decoding state and applies low precision only to the reserved region beyond that boundary. Before forward pass $t > 0$, the most recent token available at position $i$ is $x_i^{(t)}$ at a committed position and the cached prediction $\hat{y}_i$ at a masked

position. Indices cover the prompt and generation region. For each generation position $i > L_p$, the indicator $c_i^{(t)}$ identifies tokens outside the filler set $\mathcal{F}$,

$$c_i^{(t)} \;=\; \mathbb{1}\big[\,\tilde{x}_i^{(t)} \notin \mathcal{F}\,\big], \qquad \tilde{x}_i^{(t)} = \begin{cases} x_i^{(t)} & i \text{ committed}, \\ \hat{y}_i & i \text{ masked}. \end{cases} \tag{3}$$

The candidate boundary is placed $\Delta$ positions after the highest-indexed position classified as content,

$$b_t \;=\; \max\{\, i : c_i^{(t)} = 1 \,\} \;+\; \Delta, \tag{4}$$

where $\Delta$ provides a buffer for short-term changes in the content layout. The boundary is re-estimated at every step as parallel commitment modifies the sequence. Irreversible actions such as boundary fill (§5.2) begin only after $b_t$ remains unchanged for two consecutive steps. For a linear layer with weights $W$ and input dimension $d$, the boundary partitions the rows of $H \in \mathbb{R}^{(L_p+L_g)\times d}$. Each group is evaluated by a separate general matrix multiplication (GEMM),

$$Y_{i,:} \;=\; \begin{cases} H_{i,:}\, W^\top & i \le b_t, \\ Q_a(H_{i,:})\, Q_w(W)^\top & i > b_t, \end{cases} \tag{5}$$

Here, $Q_w$ is the group-wise weight quantiser and $Q_a$ is the activation quantiser of the deployed setting. For weight-only quantisation, $Q_a$ is the identity. The outputs of both groups rejoin before attention and subsequent layer propagation, keeping the reserved region active and refreshed at every step.

### 4.4 Calibration

**TEMPO** calibrates the threshold $\theta$ and the deployed low-precision setting once per backbone using unlabelled decoding trajectories. Each denoising step is evaluated at full and low precision, producing a set $\mathcal{D}$ of position-step pairs. Each pair records the full-precision gap $g$ and an indicator $\phi = \mathbb{1}[\text{quantisation changes the arg-max}]$. We choose the smallest gap threshold for which the empirical prediction-change rate above the threshold is at most 0.5%,

$$\theta^\star \;=\; \min\Big\{\, \theta :\; \Pr_{\mathcal{D}}\big[\phi = 1 \,\big|\, g > \theta\big] \le 0.5\% \Big\}. \tag{6}$$

The probability is estimated over $\mathcal{D}$, and $\theta^\star$ is used as the deployed gate threshold. The deployed setting is the lowest-precision weight-only configuration for which Eq. 6 has a solution and a nonzero share of pairs remains above $\theta^\star$. The selected configurations are W3A16 for LLaDA, W4A16 for Dream and W8A16 for CoDA. W$b$A$c$ specifies $b$-bit weights and $c$-bit activations. The resulting threshold is held fixed across all tasks for that backbone (Appendix E). Calibration uses full-precision gaps, whereas the deployed gate can read gaps from a low-precision pass. To assess this difference, a counterfactual gate obtains its candidates, predictions and gaps from an additional full-precision pass. Its fidelity differs from the deployed gate by at most 0.5 percentage points in the BFCL comparison (Appendix F).

The selected precision applies to the attention projections and MLP, using group-wise quantised weights and quantised input activations where specified. Attention scores, normalisation and embeddings remain at full precision.

## 5 Experiments

### 5.1 Setup

We evaluate LLaDA-8B-Instruct (Nie et al., 2026), Dream-v0-7B (Ye et al., 2025) and CoDA-v0-1.7B (Chen et al., 2025) on the simple and multiple splits of BFCL v4 (Patil et al., 2025), API-Bank (Li et al., 2023) and MetaTool (Huang et al., 2024), and additionally measure runtime on Spider (Yu et al., 2018). Unless otherwise stated, experiments use full diffusion with greedy decoding, $T$=16 and $n$=100. Our primary metric is *call fidelity*, the share of valid full-precision calls reproduced exactly on the same items under each benchmark's matcher. Fidelity directly measures quantisation-induced deviations from the full-precision run. Task accuracy can conceal such deviations when correct-to-wrong and wrong-to-correct changes offset each other. Appendix G describes the benchmarks, matchers and evaluation protocol, while Appendices H and E provide the decoding, calibration and compute settings.

Table 1. Main results across all model and benchmark combinations ($T$=16, $L_g$=128, $n$=100). The table reports call fidelity, linear-layer computation saved and task accuracy. L2PD denotes Learn2PD. For computation saved, TEMPO counts positions evaluated at low precision, whereas Learn2PD (Bao et al., 2026) counts positions removed by truncation. TEMPO and full quantisation use the deployed precision setting, while Learn2PD evaluates the retained region at full precision. Bold marks the highest fidelity and computation saved in each row.

| | Call fidelity (%) | | | Comp. saved (%) | | Task accuracy (%) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Fully q. | L2PD | TEMPO | L2PD | TEMPO | Full prec. | Fully q. | L2PD | TEMPO |
| **LLaDA-8B** (Nie et al., 2026), W3A16 | | | | | | | | | |
| BFCL | 66 | 69 | **87** | 21 | **76** | 89 | 81 | 76 | 86 |
| BFCL-mult. | 66 | 69 | **91** | 5 | **76** | 81 | 82 | 70 | 79 |
| API-Bank | 49 | 64 | **67** | 20 | **53** | 55 | 51 | 48 | 51 |
| MetaTool | 93 | 91 | **95** | 21 | **92** | 91 | 86 | 89 | 91 |
| **Dream-7B** (Ye et al., 2025), W4A16 | | | | | | | | | |
| BFCL | 78 | 67 | **88** | 0 | **89** | 74 | 79 | 74 | 74 |
| BFCL-mult. | 74 | 65 | **83** | 0 | **86** | 74 | 77 | 75 | 73 |
| API-Bank | 62 | 55 | **66** | 5 | **68** | 11 | 3 | 16 | 13 |
| MetaTool | 89 | 93 | **98** | 0 | **96** | 86 | 83 | 85 | 86 |
| **CoDA-1.7B** (Chen et al., 2025), W8A16 | | | | | | | | | |
| BFCL | 83 | 49 | **87** | 50 | **70** | 56 | 58 | 33 | 60 |
| BFCL-mult. | **85** | 48 | 81 | 47 | **67** | 52 | 51 | 28 | 51 |
| API-Bank | **64** | 36 | 63 | 48 | **60** | 20 | 15 | 11 | 17 |
| MetaTool | 97 | 83 | **98** | 86 | **89** | 74 | 76 | 65 | 74 |
| *Mean* | *76* | *66* | *84* | *25* | *77* | *64* | *62* | *56* | *63* |

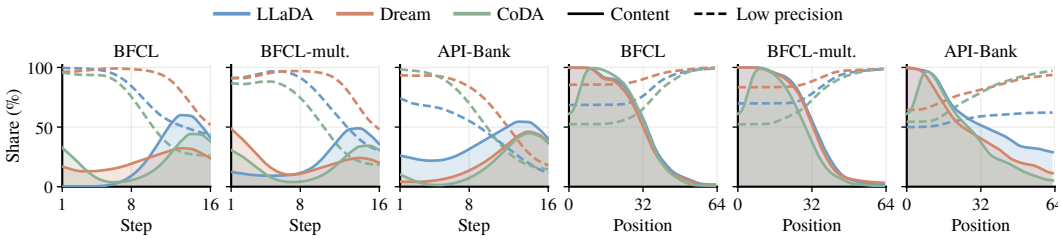


Figure 4. Content-token commits (solid) and low-precision computation (dashed) across denoising steps (left) and sequence positions (right), with $T$=16, $L_g$=128 and $n$=100. The left panels report content-token commits and low-precision computation per step. The right panels report final content-token locations and low-precision computation per position.

### 5.2 Results

**We organise the findings into four observations** that separately examine the full schedule, its temporal and positional components, and the systems-level composition.

**Obs.❶ TEMPO improves the trade-off between fidelity and computation.** Table 1 reports fidelity, saved computation and task accuracy. Learn2PD removes the reserved region while retaining full precision, yet changes more calls on three of four Dream benchmarks and all four CoDA benchmarks than applying the deployed low-precision configuration to all weights. Quantisation can alter content predictions at sensitive steps and positions (§3). **TEMPO** identifies these computations from the decoding state and achieves the highest fidelity on ten of the twelve combinations while keeping 53–96% of positions at low precision. **TEMPO** remains within four accuracy points of full precision on all twelve combinations, with McNemar $p \geq .125$. Full quantisation loses eight points on LLaDA-BFCL ($p$=.021) and Dream-API-Bank ($p$=.039). Table 13 reports the discordant pairs and McNemar tests (McNemar, 1947).

**Obs.❷ The step-wise gate protects content-heavy denoising steps.** Low precision is concentrated in the early denoising steps, whereas most content commits occur in the final third (Figure 4). On 399 BFCL calls, **TEMPO** reaches 90.1%, 90.8% and 84.8% fidelity on LLaDA, Dream and CoDA, respectively. Random allocation at similar rates reaches 71.0%, 79.3% and 83.6%, while reversing

Table 2. Ablations on BFCL reporting computation saved and call fidelity. For the step-wise gate ablations ($n$=399), Random assigns low precision at TEMPO's rate and Inverted reverses the gate decisions. For the reserved-region interventions ($n$=100), Removed truncates the region, Buffered retains eight additional positions, Frozen reuses their hidden states and TEMPO updates them at low precision.

| | LLaDA (Nie et al., 2026) | | Dream (Ye et al., 2025) | | CoDA (Chen et al., 2025) | |
|---|---|---|---|---|---|---|
| | Saved (%) | Fid. (%) | Saved (%) | Fid. (%) | Saved (%) | Fid. (%) |
| *Step-wise gate* | | | | | | |
| Random | **69.6** | 71.0 | **86.2** | 79.3 | **61.0** | 83.6 |
| Inverted | 36.6 | 68.7 | 21.0 | 79.0 | 48.9 | 83.0 |
| **TEMPO** | 66.2 | **90.1** | 84.6 | **90.8** | 57.5 | **84.8** |
| *Reserved region* | | | | | | |
| Removed | **28.0** | 57.0 | **28.1** | 18.0 | **28.6** | 45.0 |
| Buffered | 25.6 | 52.0 | 25.7 | 52.0 | 26.1 | 44.0 |
| Frozen | 23.9 | 82.0 | 24.5 | 68.0 | 24.3 | 14.0 |
| **TEMPO** | 23.7 | **89.0** | 25.8 | **88.0** | 27.3 | **87.0** |

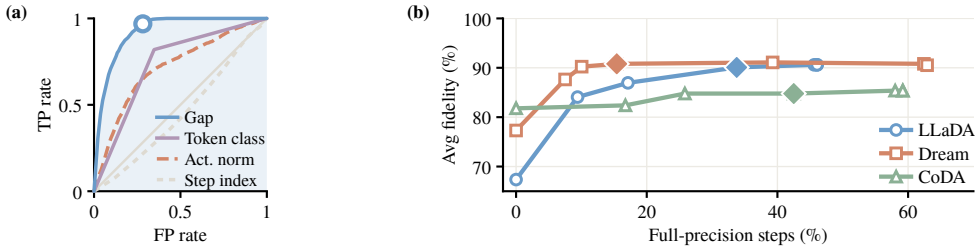


Figure 5. Gate calibration and sensitivity. (a) True-positive (TP) versus false-positive (FP) rates for detecting quantisation-induced prediction changes using logit gaps and three alternative signals (Appendix C). Hollow markers indicate the calibrated thresholds. (b) Fidelity and precision allocation at 0.25, 0.5, 1, 2 and 4 times the calibrated $\theta$ on BFCL ($n$=399, $T$=16). Filled markers indicate the calibrated value.

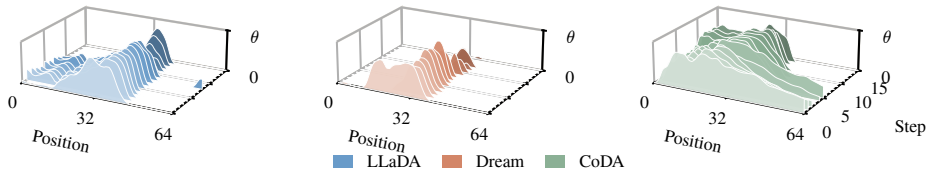


Figure 6. Temporal consistency of cached logit gaps along one greedy BFCL decoding trajectory for each backbone with $T$=16 over the first 64 positions. Curves show the normalised shortfall $\mu_i = \max(0, \theta - g_i)/\theta$ at each step, using the logit gap $g_i$ from Eq. 1 and the gate threshold $\theta$. Darker curves correspond to later steps.

the gate reaches 68.7%, 79.0% and 83.0% (Table 2, step-wise gate block). Doubling $\theta$ improves fidelity by 0.3–0.6 points but assigns 12–24 percentage points more steps to full precision. Halving it reduces fidelity by 3.1 points on LLaDA and 0.6 points on Dream, with no change on CoDA (Figure 5(b)).

**Obs.❸ Low-precision reserved-region updates preserve more calls than removal.** At comparable computation savings, position-wise routing reaches 87–89% fidelity, compared with 14–82% when the reserved region is removed, buffered or frozen (Table 2, reserved-region block). Task accuracy does not capture this difference. Buffered removal improves accuracy by 10 points on Dream and 14 points on CoDA while reproducing only 52% and 44% of their full-precision calls, respectively (Table 7). At the token level, **TEMPO** assigns less low-precision computation to the JSON scaffold and more to the end-of-text tail (Figure 7).

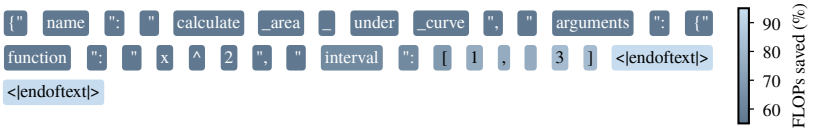


Figure 7. Per-token proportion of linear-layer computation at low precision in one LLaDA-BFCL call ($T$=16).

Table 3. Latency in seconds at $L_g$=1024 on an RTX A6000 ($n$=8). Columns report full precision (FP), parallel decoding (Par.), Par. with boundary fill, and Par. with boundary fill and TEMPO. Red factors compare Par. with FP and both subsequent configurations with Par.

| | | $T$=16 | | | | $T$=64 | | | | $T$=128 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | FP | Par. | +Fill | +TEMPO | FP | Par. | +Fill | +TEMPO | FP | Par. | +Fill | +TEMPO |
| LLaDA-8B | Spider | 3.12 | 2.54 | 2.56 | 1.83 | 11.97 | 3.25 | 2.90 | 2.09 | 23.73 | 3.18 | 2.85 | 2.20 |
| | | | 1.23× | 0.99× | 1.39× | | 3.68× | 1.12× | 1.56× | | 7.46× | 1.12× | 1.45× |
| | API-Bank | 3.29 | 2.10 | 2.00 | 1.54 | 12.44 | 2.77 | 2.56 | 1.78 | 24.65 | 2.77 | 2.55 | 1.78 |
| | | | 1.56× | 1.05× | 1.36× | | 4.48× | 1.08× | 1.56× | | 8.91× | 1.08× | 1.55× |
| Dream-7B | Spider | 3.28 | 2.98 | 2.94 | 2.41 | 13.27 | 7.86 | 5.88 | 4.35 | 25.50 | 8.51 | 5.99 | 4.44 |
| | | | 1.10× | 1.01× | 1.24× | | 1.69× | 1.34× | 1.81× | | 2.99× | 1.42× | 1.92× |
| | API-Bank | 4.84 | 2.98 | 3.04 | 2.54 | 19.43 | 3.67 | 3.68 | 2.54 | 38.11 | 3.80 | 3.80 | 2.62 |
| | | | 1.62× | 0.98× | 1.17× | | 5.29× | 1.00× | 1.44× | | 10.04× | 1.00× | 1.45× |

**Obs.❹ Precision scheduling yields system-level speedups and combines with step reduction.** At sequence length 1234 on an RTX 5090, routing accelerates linear layers by 2.87× and the complete denoising step by 2.19× relative to full precision (Appendix A). Boundary fill commits the end-of-text tail at the semantic boundary (§4.3), terminating decoding early (Bao et al., 2026). On Spider and API-Bank (RTX A6000), parallel decoding yields 1.10 to 10.04× speedups over full precision, increasing to 1.36 to 14.55× with boundary fill and TEMPO (Table 3). Step-reduction gains increase with the original step count. At $T$=128, the full composition achieves 40.6× to 98.8× speedups on BFCL and MetaTool (Figure 8). On Spider and API-Bank, boundary fill alone yields 0.98 to 1.42× relative to parallel decoding. TEMPO adds a further 1.20 to 1.45×, giving total speedups of 1.17 to 1.92× over parallel decoding (Table 3). The full composition matches or exceeds Learn2PD's speedups with higher fidelity (Figure 8 and Table 1).

## 6 Limitations

**TEMPO** changes only the precision of existing computation and leaves the decoding process unchanged. Parallel decoding reduces the number of steps available for precision scheduling. We therefore measure the joint runtime of the combined methods rather than multiply their isolated speedups. Position-wise routing derives its savings from the reserved region. When the generation length is close to the actual output length, this region is small and the available speedup is correspondingly limited. The step-wise gate and position-wise routing assume irreversible token commitment, as in the samplers of the three evaluated backbones. Samplers that revise committed tokens would require the precision decisions to be updated after those revisions. **TEMPO** allocates precision across steps and positions but does not consider layer-wise allocation.

## 7 Conclusion

**TEMPO** schedules precision hierarchically for dLLM agents without additional training. Its design follows three observations about filler commitment order, margin persistence and slack tolerance. The gate uses cached logit gaps to estimate whether quantisation will change the tokens selected for commitment, while routing retains full precision for the prompt and estimated content region and updates the reserved region at low precision. Across four agent benchmarks and three backbones, **TEMPO** achieves 84% mean call fidelity, compared with 76% for full quantisation and 66% for Learn2PD. On Spider and API-Bank, **TEMPO** adds up to 1.45× speedup to parallel decoding with boundary fill. The full configuration reaches 14.55× over full precision. Precision scheduling reduces the cost of retained computation without removing any sequence positions, complementing methods that reduce the number of denoising steps or the generation sequence length for agents.

### AI USE STATEMENT

In this work, we used generative AI tools for implementing methods (the experiment-orchestration scripts and the code that produces the figures and tables), for feedback on the design of the ablation experiments, and for interpreting results in discussion with the authors. We have not used generative AI tools to generate synthetic data sets, to develop the theoretical model or conceptual framework, to formulate or prove mathematical claims, to propose or refine hypotheses, to clean or reformat data sets, or to support qualitative and thematic analysis. Assistance with translation is not applicable to this work. Additionally, we used generative AI tools to draft and edit parts of the paper, to create and modify the scientific figures (including the method-overview diagram), to create and edit software code, and to identify and summarise relevant literature. We have reviewed all AI-assisted work. Every reported number comes from code run by the authors on the authors' hardware, generated code was checked and tested by the authors, and all generated text was read and revised by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

### REPRODUCIBILITY STATEMENT

Our code is openly accessible to support future research. Appendix G states the evaluation protocol, including the definition of call fidelity and the same-GPU rule under which every comparison is made. Appendix H lists the prompts, the decoding and sampling parameters of every run and the compute used. Appendix E specifies the step-wise gate, position-wise routing, the calibration procedure and the per-backbone settings. Appendix B gives the commit-order study, and Appendix F reports the comparison of cached and full-precision gate signals. The tables and figures are generated by scripts from the raw run files, so that every number in the paper can be traced to a run.

### ETHICS STATEMENT

TEMPO aims to reduce the computational cost of diffusion language model agents. More efficient inference can improve access to these models, but can also lower the cost of harmful or unauthorised tool use. Precision scheduling does not address biases in the underlying models or establish the safety of their outputs. Agreement with full-precision calls does not guarantee that those calls are correct or appropriate. Deployment therefore requires application-specific validation, access controls for tools and human oversight for consequential actions.

## REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

Wenrui Bao, Zhiben Chen, Dan Xu, and Yuzhang Shang. Learning to parallel: Accelerating diffusion large language models via learnable parallel decoding. In *International Conference on Learning Representations*, volume 2026, pp. 83376–83392, 2026.

Haolin Chen, Shiyu Wang, Can Qin, Bo Pang, Zuxin Liu, Jielin Qiu, Jianguo Zhang, Yingbo Zhou, Zeyuan Chen, Ran Xu, et al. Coda: Coding lm via diffusion adaptation. *arXiv preprint arXiv:2510.03270*, 2025.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35: 30318–30332, 2022.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. *arXiv preprint arXiv:1910.10073*, 2019.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, et al. Scaling diffusion language models via adaptation from autoregressive models. In *International Conference on Learning Representations*, volume 2025, pp. 5046–5073, 2025.

Daehoon Gwak, Minhyung Lee, Junwoo Park, and Jaegul Choo. Accelerating masked diffusion large language models: A survey of efficient inference techniques. *arXiv preprint arXiv:2607.12829*, 2026.

Haoyu Huang, Linlin Yang, Sheng Xu, Boyu Liu, Guodong Guo, Zhongqian Fu, Hang Zhou, and Baochang Zhang. Fair-calib: Frontier-aware instability-reweighted calibration for post-training quantization of diffusion large language models. *arXiv preprint arXiv:2606.06547*, 2026.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Gong, et al. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In *International Conference on Learning Representations*, volume 2024, pp. 42978–43007, 2024.

Sangwoo Kwon, Seong Hoon Seo, Jae W Lee, and Yeonhong Park. Dp-llm: Runtime model adaptation with dynamic layer-wise precision assignment. *Advances in Neural Information Processing Systems*, 38:144921–144945, 2026.

Chenqi Li, Minghui Min, Dusit Niyato, and Wei Ni. Diffusion language models for mobile edge agentic ai: Foundations, applications, and challenges. *arXiv preprint arXiv:2609.04778*, 2026.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 3102–3116, 2023.

Haokun Lin, Haobo Xu, Yichen Wu, Ziyu Guo, Renrui Zhang, Zhichao Lu, Ying Wei, Qingfu Zhang, and Zhenan Sun. Quantization meets dllms: A systematic study of post-training quantization for diffusion llms. *Machine Intelligence Research*, pp. 1–17, 2026.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations. In *International Conference on Learning Representations*, volume 2025, pp. 92009–92032, 2025a.

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyan Wei, Shaobo Wang, Yichen Zhu, and Linfeng Zhang. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025b.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. Scaling up masked diffusion models on text. In *International Conference on Learning Representations*, volume 2025, pp. 82974–82997, 2025.

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. *Advances in Neural Information Processing Systems*, 38:50608–50646, 2026.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Gao Peng, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 45472–45496, 2024.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37: 103131–103167, 2024.

Wei Tao, Bin Zhang, Xiaoyang Qu, Jiguang Wan, and Jianzong Wang. Cocktail: Chunk-adaptive mixed-precision quantization for long-context llm inference. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pp. 1–7. IEEE, 2025.

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 *International Conference on Learning Representations*, volume 2026, pp. 57027–57051, 2026.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023.

Chen Xu and Dawei Yang. Dllmquant: Quantizing diffusion-based large language models. *arXiv preprint arXiv:2508.14090*, 2025.

Xin Yan, Aqiang Wang, Zhenglin Wan, Xingrui Yu, and Ivor Tsang. Star-quant: State-time consistent post-training quantization for diffusion large language models. *arXiv preprint arXiv:2606.04945*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

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.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 3911–3921, 2018.

Tao Zhang, Ziqian Zeng, Hao Peng, Huiping Zhuang, and Cen Chen. Mixkvq: Query-aware mixed-precision kv cache quantization for long-context reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7189–7204, 2026a.

Tianao Zhang, Zhiteng Li, Xianglong Yan, Haotong Qin, Yong Guo, and Yulun Zhang. Quant-dllm: Post-training extreme low-bit quantization for diffusion large language models. In *International Conference on Learning Representations*, volume 2026, pp. 79492–79503, 2026b.

Fengrui Zuo, Zhiwei Ke, Yiming Liu, Cheng Tang, Wenqi Lou, Teng Wang, Lei Gong, Chao Wang, and Xuehai Zhou. Window-diffusion: Accelerating diffusion language model inference with windowed token pruning and caching. In *International Symposium on Advanced Parallel Processing Technologies*, pp. 112–125. Springer, 2026.

## A KERNEL AND END-TO-END SPEED

**Kernel setup.** We run the same schedules on INT8/FP8 kernels (`torch._scaled_mm` FP8 and `torch._int_mm` INT8 on the RTX 5090, and a fused W8A16 kernel on the RTX A6000), timed on LLaDA and Dream with BFCL and MetaTool at generation length 1024. CoDA has no kernel path on this hardware. In isolation, the FP8 GEMM on the RTX 5090 runs 2.38× faster than full precision. Inside a complete denoising step at a sequence length of 1234, the routed linear layers run 2.87× faster, yielding a 2.19× speedup for the whole step. The remaining 16% of step time is outside the linear layers.

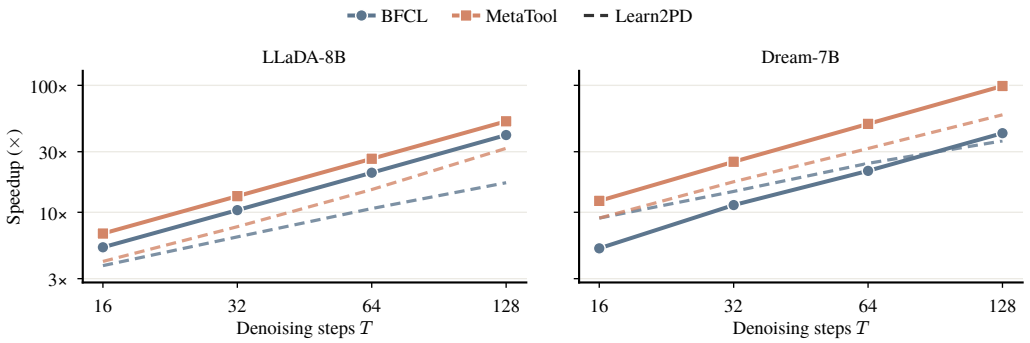


Figure 8. End-to-end speedup relative to full-precision decoding across denoising step counts at $L_g$=1024 on an RTX A6000. Solid curves represent TEMPO with boundary fill and parallel decoding, and dashed curves represent Learn2PD.

Table 4. Task accuracy at $T$=128 and $L_g$=1024 for the runtime comparison in Figure 8. We use $n$=100 for full precision and TEMPO with decoding optimisations, and $n$=50 for Learn2PD.

| | Task | Full precision | Learn2PD | TEMPO+dec. |
|---|---|---|---|---|
| LLaDA-8B | BFCL | 0.89 | 0.88 | 0.89 |
| | MetaTool | 0.92 | 0.90 | 0.96 |
| Dream-7B | BFCL | 0.75 | 0.78 | 0.86 |
| | MetaTool | 0.86 | 0.82 | 0.96 |

Table 5. Wall-clock time and board energy per sample for the full composition on the LLaDA reference INT8 path (RTX A6000, BFCL, $T$=16).

| | $L_g$ | FP wall (s) | Composed wall (s) | Speedup | Energy saved |
|---|---|---|---|---|---|
| LLaDA-8B | 128 | 1.24 | 0.24 | 5.23× | 81% |
| | 512 | 2.57 | 0.52 | 4.98× | 80% |
| | 1024 | 4.19 | 0.84 | 4.98× | 80% |

## B COMMIT ORDER AND THE FILLER-CONTENT SPLIT

Each backbone provides several rules for choosing which masked positions a step commits, all ranking positions by a measure of the model's certainty. Figure 9 compares the filler-content split of §3 under every such rule the three backbones provide. Certainty-ordered rules produce filler-content gaps of 4.2-6.6 steps in all nine backbone-rule combinations. A fourth rule that commits positions at random, which no backbone recommends, is included as a control. It reduces the gap to 0.2 steps, so the separation comes from ranking by certainty rather than from the counting procedure.

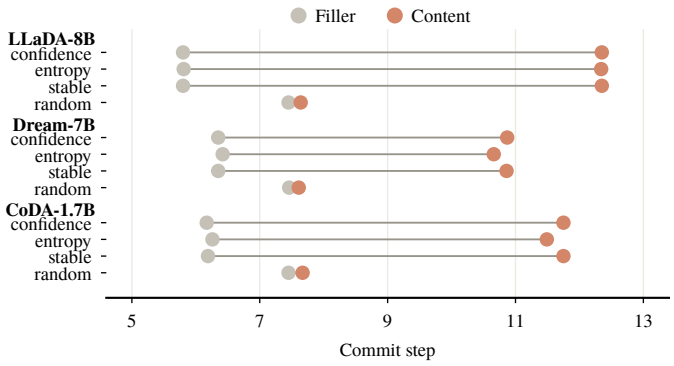


Figure 9. Mean commit step of filler and content under each commit rule (BFCL, $T$=16, $n$=100).

## C SIGNAL COMPARISON FOR THE STEP-WISE GATE

Figure 5(a) compares candidate signals for detecting quantisation-induced prediction changes. The signals that distinguish such changes are already computed by the sampler. Step index produces a curve below the diagonal, ranking quantisation-induced output changes in the opposite direction. Activation norm, used elsewhere in layer-wise mixed precision, follows the diagonal until the false positive rate exceeds 0.3, beyond the operating region a schedule can afford. Token class distinguishes filler from content but provides no ranking within either class, producing a curve with a single bend. A paired forward-pass diagnostic on the fully quantised LLaDA run locates the first output-changing commit at step 12.8 of 16, inside the window where LLaDA writes its content (Figure 2b).

Table 6. Area under the ROC curve for each candidate signal ($n$=21,760, with 95% bootstrap intervals in brackets) and F1 at its best threshold.

| Signal | AUC | F1 |
|---|---|---|
| Top-1/top-2 gap (ours) | 0.923 [.919, .928] | 0.522 |
| Entropy | 0.921 [.917, .926] | 0.506 |
| Top-3 mass | 0.905 [.901, .910] | 0.465 |
| Token class | 0.736 [.726, .745] | 0.291 |
| Activation norm | 0.716 [.702, .729] | 0.288 |
| Step index $t/T$ | 0.454 [.443, .468] | 0.164 |

## D RESERVED-REGION INTERVENTIONS

Table 7 gives the accuracy behind the reserved-region block of Table 2. On LLaDA-BFCL, removal with an exact oracle output length costs 16 points, removal with a small buffer of eight tokens costs 9 points, and removal with the original per-step commit rate preserved on the shortened sequence costs 34-41 points. Varying commit rate and sequence length separately attributes the damage to disrupted position evolution rather than to output-length prediction error. Freezing costs about 11 points, and refreshing the frozen region every fourth step does not recover the loss. The reserved region's hidden states are the sequence's least stable, with mid-depth cosine similarity to the previous step of 0.74 against 0.98 for the prompt.

Table 7. Task accuracy for the reserved-region ablations in the lower block of Table 2 (BFCL, $T$=16, $L_g$=128, $n$=100). Parentheses show the change from full precision.

| | Full precision | Remove | Remove+8 | Freeze | Low precision |
|---|---|---|---|---|---|
| LLaDA-8B | 0.89 | 0.73 (−16) | 0.80 (−9) | 0.78 (−11) | 0.86 (−3) |
| Dream-7B | 0.74 | 0.28 (−46) | 0.84 (+10) | 0.78 (+4) | 0.76 (+2) |
| CoDA-1.7B | 0.57 | 0.54 (−3) | 0.71 (+14) | 0.56 (−1) | 0.59 (+2) |

## E METHOD SPECIFICATION

**Quantisation.** We use group-wise symmetric fake quantisation, from weight-only (W3A16) to progressively narrower activation widths (W4A6, W4A4). The calibration set $\mathcal{D}$ of Eq. 6 contains 21,760 position-step pairs from BFCL decoding trajectories at $T$=16. The resulting thresholds are 4.27 (LLaDA), 3.16 (Dream) and 3.31 (CoDA). Under the narrower W4A6 stress setting, full quantisation loses fidelity on every benchmark and the step-wise gate halves the damage.

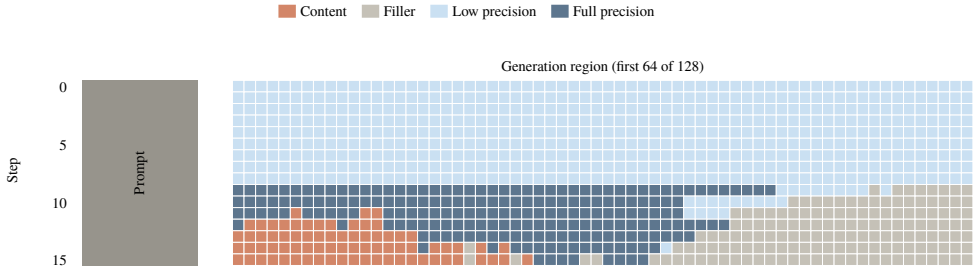


Figure 10. TEMPO on one LLaDA-BFCL trajectory ($T$=16, $L_g$=128), one row per step. The background marks the step-wise decision, and positions in the reserved region beyond the semantic boundary run at low precision within full-precision steps.

The gate checks the top-$2k$ candidates, twice as many positions as the sampler will commit, so a position that rises into the commit set between two steps has usually already been checked. At $t = 0$ no previous distribution exists, so the step runs at low precision and the routing boundary is not consulted.

Table 8. Per-backbone configuration from a single calibration. Boundary fill is disabled on CoDA because the boundary locks before the layout is final.

| | LLaDA-8B | Dream-7B | CoDA-1.7B |
|---|---|---|---|
| Deployed precision | W3A16 | W4A16 | W8A16 |
| Weight quantisation | group-wise symmetric, group size 128 | | |
| Gate threshold $\theta$ | 4.27 | 3.16 | 3.31 |
| Gate at step 0 | low precision (no previous distribution) | | |
| Commit-set check | top-$2k$ of previous step's ranking ($k$ = per-step commits) | | |
| Routing boundary | last content position + 8-token buffer, re-estimated each step | | |
| Fill lock (§4.3) | boundary equal on two consecutive steps | | |
| Boundary fill (composition) | on | on | off |
| Fallback | boundary never locks ⇒ fill disabled, precision scheduling continues | | |

## F GATE SIGNAL FROM CACHED VERSUS FULL-PRECISION GAPS

The threshold $\theta$ is calibrated on full-precision gaps (§4.4), whereas the deployed gate uses gaps cached from the preceding pass, which may have run at low precision. The counterfactual comparison uses an additional full-precision pass over the same state to obtain the gate's candidates,

**Algorithm 1** One TEMPO step (state $x^{(t)}$, cached logits from pass $t-1$)

**Require:** state $x^{(t)}$, logits cached by pass $t-1$ and threshold $\theta$
**Ensure:** the pass is executed, its logits cached, and the sampler's commits written
1: **if** $t = 0$ **then** ▷ no cached logits yet
2: run the whole pass at low precision, then cache and commit as in lines 13–14, then **return**
3: **end if**
***Which positions is this step about to commit?***
4: $U_t \leftarrow$ positions masked after step $t-1$
5: $\widehat{C}_t \leftarrow$ the $2k$ positions of $U_t$ the cached confidences rank highest ▷ $k$ = commits per step
***Is every one of them safe to commit at low precision?***
6: *safe* $\leftarrow$ **true** if for every $i \in \widehat{C}_t$ the cached prediction $\hat{y}_i$ is a filler token, or its cached top-1/top-2 gap $g_i$ exceeds $\theta$
***Where is the semantic boundary?***
7: $b \leftarrow$ (last position holding content, using a committed token or the cached prediction where masked) $+ 8$
***Run the pass.***
8: **if** *safe* **then**
9: every position at low precision
10: **else**
11: positions after $b$ at low precision, positions up to $b$ at full precision ▷ one GEMM per group, per linear layer
12: **end if**
13: cache this pass's logits for step $t+1$
14: commit by the sampler's own rule, unchanged

predictions and gaps. The sampler and position-wise routing still use the deployed pass. Both gates are evaluated on BFCL with $n$=399, $T$=16 and the deployed settings, with each pair of runs and its full-precision reference using the same GPU model. Table 9 reports fidelity, the share of full-precision steps and gate-decision agreement evaluated along each gate's own trajectory.

The two gates agree on 90.2–97.2% of step decisions, and their fidelity differs by at most 0.5 percentage points. The cached-gap gate keeps a larger share of steps at full precision on LLaDA (33.8% versus 28.2%) and Dream (15.4% versus 12.8%). The shares are nearly identical on CoDA (42.5% versus 42.4%). These results are consistent with more conservative gating on LLaDA and Dream, while avoiding an additional forward pass. The LLaDA runs use a Quadro RTX 8000 and reach 90.6% fidelity with cached gaps, compared with 90.1% in Table 2. The Dream and CoDA cached-gap results reproduce Table 2.

Table 9. Cached-gap gating versus a counterfactual gate using an additional full-precision pass (BFCL, $n$=399, $T$=16). Agreement is the share of step decisions on which the gates coincide, evaluated along the trajectory of the gate in that row.

| | Gate reads | Fidelity (%) | FP steps (%) | Agreement (%) |
|---|---|---|---|---|
| LLaDA-8B | cached gaps | 90.6 | 33.8 | 90.8 |
| | full-precision gaps | 90.1 | 28.2 | 90.2 |
| Dream-7B | cached gaps | 90.8 | 15.4 | 96.2 |
| | full-precision gaps | 90.5 | 12.8 | 95.8 |
| CoDA-1.7B | cached gaps | 84.8 | 42.5 | 97.1 |
| | full-precision gaps | 85.1 | 42.4 | 97.2 |

## G EVALUATION SETUP

**Backbones.** LLaDA-8B is a masked diffusion model trained from scratch, Dream-7B is adapted from an autoregressive checkpoint, and CoDA-1.7B is a code model adapted in the same way. **Benchmarks.** BFCL v4 uses its simple and multiple splits with an AST-level matcher, API-Bank uses exact call match, and MetaTool is tool selection by exact match over 10 candidates. Spider (Yu

et al., 2018) is used for the length analysis in §3 and the runtime evaluation in Table 3. **Protocol.** Full diffusion means a single block spanning the generation length $L_g$. We use $n = 399$ where stated. Fidelity is computed over the samples for which the full-precision run yields a valid call, and a failed extraction never counts as agreement. Every arm of a comparison is run and scored on the same GPU model. LLaDA and Dream produce identical full-precision outputs across GPU types, whereas CoDA's bf16 outputs agree on 87% of calls between a Turing and an Ampere card and are identical only within the same model, so all CoDA arms use RTX A6000s.

## H PROMPTS, DECODING PARAMETERS AND COMPUTE

**Prompts.** Each benchmark uses one fixed prompt. The model's chat template wraps it as a single user turn. BFCL supplies the function documentation as JSON and asks for a JSON call. API-Bank supplies its own instruction and input fields verbatim. MetaTool lists ten candidate tools and asks for one name.

**BFCL (simple split, while the multiple split lists every function and asks for "the MOST appropriate one of these functions")**

```
You have access to the following function:
{"name": ..., "description": ..., "parameters": {...}}
Question: <user question>
Respond with ONLY a JSON object calling the function, in this exact
format: {"name": "<function_name>", "arguments": {<args>}}
```

**API-Bank**

```
<instruction field of the item>
<input field of the item>
```

**MetaTool**

```
Below is a list of available tools. Pick the ONE tool best suited to
the user request. Reply with only the tool name.
Tools:
- <tool 1>: <description, first 160 characters>
...
- <tool 10>: <description, first 160 characters>
User request: <query>
Tool:
```

**Decoding and schedule parameters.** Table 10 lists every setting that the experiments hold fixed. Decoding is greedy. Each step commits the arg-max token at the positions the commit rule selects, with temperature 0 and no nucleus or top-$k$ truncation, so every run is deterministic given the GPU model (Appendix G).

**Compute.** All accuracy and fidelity runs use fake quantisation (quantise-dequantise in the model's dtype) on NVIDIA RTX A6000 (48 GB) cards across two workstations, six cards in total, with two Quadro RTX 8000 (48 GB) used for LLaDA and Dream runs only. The kernel and wall-clock measurements of §5.2 use an RTX 5090 (24 GB, Blackwell) and RTX A6000 cards. All configurations in Table 3 are evaluated on the same single RTX A6000 using PyTorch 2.13 with CUDA 12.6. These runs extend the fused kernel in Appendix A to Dream's layer shapes, use prefix caching and CUDA graphs for LLaDA, and share fused normalisation and rotary kernels across all configurations. Figure 8 reports earlier measurements using the reference INT8 path for Dream. We use PyTorch 2.13 with CUDA 13.0 and Transformers 4.49, with model weights in fp16 (LLaDA, Dream) or bf16 (CoDA). Table 11 gives the measured cost per run on an A6000. Peak memory is 17.5 GB for LLaDA-8B and 17.4 GB for Dream-7B in plain decoding, rising to about 30 GB under the row-grouped fake-quant path, which evaluates both precision branches of every linear layer. CoDA-1.7B needs 4 to 15 GB. The BFCL ablations of Table 2, Figure 5(b) and Appendix J comprise 81 runs at $n$=399 plus three full-precision repeats for the determinism check, 20.9 GPU-hours in total. The twelve-combination tables at $n$=100 cost about 2.5 GPU-hours, and the $T$=128 accuracy

Table 10. Fixed decoding, schedule and baseline parameters.

| | |
|---|---|
| Denoising steps $T$ | 16 (quality), and 16, 32, 64, 128 (systems, §5.2) |
| Generation length $L_g$ | 128 (quality) and 1024 (systems) |
| Block structure | single block spanning $L_g$ |
| Commit rule | confidence-ranked (low-confidence remasking), $L_g/T$ commits per step |
| Sampling | greedy arg-max, temperature 0, no top-$p$ / top-$k$ |
| Gate candidates $\widehat{C}_t$ | top-$2k$ of the cached ranking, $k = L_g/T$ |
| Gate threshold $\theta$ | 4.27 (LLaDA), 3.16 (Dream), 3.31 (CoDA), step 0 at low precision |
| Routing boundary | last content position $+ \Delta$, $\Delta = 8$, re-estimated each step |
| Fill lock | boundary unchanged on two consecutive steps |
| Weight quantisation | group-wise symmetric, group size 128 |
| Activation quantisation | per-token symmetric, where the setting specifies it |
| Parallel decoding | Fast-dLLM confidence-threshold rule, $\tau = 0.9$ |
| Learn2PD | authors' filter, retrained separately for each backbone and each evaluated step count and generation length |
| Samples per combination | 100 (Tables 12, 13), and 399 for the BFCL ablations |

evaluation of Table 4 about 5 GPU-hours, since a full-precision sample at $L_g$=1024 takes 28 to 30 seconds. Calibration of $\theta$ runs each step of the calibration trajectories once at full precision and once quantised and takes under ten minutes per backbone.

Table 11. Wall-clock cost per run of 399 BFCL calls on one RTX A6000 ($L_g$=128), mean over the runs of Table 2 and Appendix J.

| | LLaDA-8B | Dream-7B | CoDA-1.7B |
|---|---|---|---|
| $T$=16, minutes per run | 16.1 | 10.8 | 4.5 |
| $T$=16, seconds per sample | 2.4 | 1.6 | 0.7 |
| $T$=32, minutes per run | 30.3 | 18.8 | 8.5 |

## I Full results

Figure 11 plots the fidelity and saved-computation columns of Table 1 as aligned dot columns.

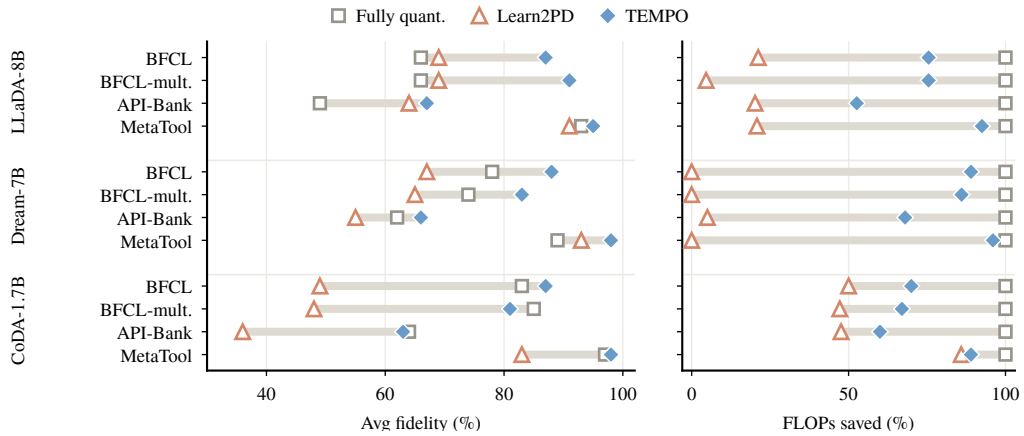


Figure 11. Fidelity (left) and share of linear-layer computation at low precision (right) per combination ($T$=16, $L_g$=128, $n$=100). TEMPO and full quantisation use the deployed setting. Learn2PD (Bao et al., 2026) runs at full precision over a truncated region.

## J The gate against a fixed tail

Table 14 compares the gate with schedules that keep the last $k$ of $T$ steps at full precision, at $T$=16 and $T$=32 (BFCL, $n$=399, deployed settings, all arms with position-wise routing). At equal budget

Table 12. Fidelity per combination in percent ($n$=100, with means quoted in §5.2).

| | Task | Fully quant. (%) | Learn2PD (%) | TEMPO (%) |
|---|---|---|---|---|
| LLaDA-8B | BFCL | 66 | 69 | 87 |
| | BFCL-multiple | 66 | 69 | 91 |
| | API-Bank | 49 | 64 | 67 |
| | MetaTool | 93 | 91 | 95 |
| Dream-7B | BFCL | 78 | 67 | 88 |
| | BFCL-multiple | 74 | 65 | 83 |
| | API-Bank | 62 | 55 | 66 |
| | MetaTool | 89 | 93 | 98 |
| CoDA-1.7B | BFCL | 83 | 49 | 87 |
| | BFCL-multiple | 85 | 48 | 81 |
| | API-Bank | 64 | 36 | 63 |
| | MetaTool | 97 | 83 | 98 |
| | *Mean* | *76* | *66* | *84* |

Table 13. Task accuracy at the deployed setting ($n$=100). Discordant pairs are correct→wrong : wrong→correct with McNemar $p$, fully quantised against full precision.

| | Task | Full precision | Fully quant. | Discordant, $p$ | Learn2PD | TEMPO |
|---|---|---|---|---|---|---|
| LLaDA-8B | BFCL | 0.89 | 0.81 | 9:1, .021 | 0.76 | 0.86 |
| | BFCL-multiple | 0.81 | 0.82 | 7:8, 1.0 | 0.70 | 0.79 |
| | API-Bank | 0.55 | 0.51 | 15:11, .56 | 0.48 | 0.51 |
| | MetaTool | 0.91 | 0.86 | 6:1, .125 | 0.89 | 0.91 |
| Dream-7B | BFCL | 0.74 | 0.79 | 10:15, .42 | 0.74 | 0.74 |
| | BFCL-multiple | 0.74 | 0.77 | 7:10, .63 | 0.75 | 0.73 |
| | API-Bank | 0.11 | 0.03 | 10:2, .039 | 0.16 | 0.13 |
| | MetaTool | 0.86 | 0.83 | 7:4, .55 | 0.85 | 0.86 |
| CoDA-1.7B | BFCL | 0.56 | 0.58 | 1:3, .625 | 0.33 | 0.60 |
| | BFCL-multiple | 0.52 | 0.51 | 3:2, 1.0 | 0.28 | 0.51 |
| | API-Bank | 0.20 | 0.15 | 7:2, .18 | 0.11 | 0.17 |
| | MetaTool | 0.74 | 0.76 | 0:2, .5 | 0.65 | 0.74 |

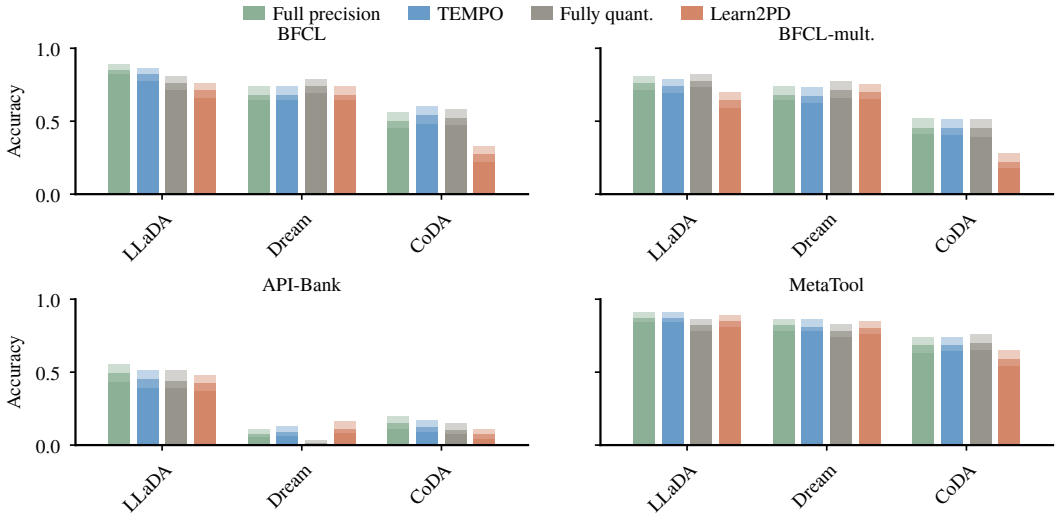


Figure 12. Task accuracy with bootstrap lower tails (5,000 resamples of 100 samples), with the median, 10th and 1st percentile shaded from palest to darkest.

the gate matches a fixed tail on LLaDA and Dream. The tail's $k$ has to be chosen per backbone and step count, while the gate's budget follows from the one calibrated $\theta$. On CoDA no schedule moves

fidelity by more than four points above full quantisation at either step count. The inverted criterion of Table 2 spends 1.2-5.1× the gate's full-precision budget and lands within two points of protecting no step.

Table 14. The gate against a fixed tail of $k$ protected steps (BFCL, $n$=399).

| | LLaDA | | Dream | | CoDA | |
|---|---|---|---|---|---|---|
| | Saved (%) | Fid. (%) | Saved (%) | Fid. (%) | Saved (%) | Fid. (%) |
| $T$=16 | | | | | | |
| Protect last 12 | 25.0 | 91.4 | 25.0 | 91.4 | 25.0 | 85.4 |
| Protect last 8 | 50.0 | 90.6 | 50.0 | 90.8 | 50.0 | 84.5 |
| Protect last 6 | 62.5 | 89.6 | 62.5 | 91.4 | 62.5 | 84.5 |
| Protect last 4 | 75.0 | 86.7 | 75.0 | 89.4 | 75.0 | 84.8 |
| Protect last 2 | 87.5 | 84.1 | 87.5 | 87.6 | 87.5 | 82.1 |
| **TEMPO gate** | 66.2 | **90.1** | 84.6 | **90.8** | 57.5 | **84.8** |
| $T$=32 | | | | | | |
| Protect last 24 | 25.0 | 95.1 | 25.0 | 92.0 | 25.0 | 90.5 |
| Protect last 16 | 50.0 | 95.1 | 50.0 | 92.5 | 50.0 | 90.8 |
| Protect last 12 | 62.5 | 94.0 | 62.5 | 92.5 | 62.5 | 90.5 |
| Protect last 8 | 75.0 | 93.0 | 75.0 | 92.5 | 75.0 | 90.2 |
| Protect last 4 | 87.5 | 87.6 | 87.5 | 90.4 | 87.5 | 89.2 |
| **TEMPO gate** | 83.8 | **89.9** | 92.1 | **91.2** | 76.7 | **90.2** |

## K LEARN2PD ON DREAM

The released Learn2PD implementation targets LLaDA. Its decoding loops access the model only through the logits of a full forward pass and two token IDs (mask, end-of-text). Dream differs in three ways. Its logits are next-token aligned (position $i$ scores token $i$+1), and its mask and end-of-text IDs differ. We therefore wrap the model in an adapter that shifts the logits right by one (the same correction Dream's own sampler applies) and parameterise the two IDs. The baseline's decoding loops run unchanged. The Learn2PD filter is model-specific by construction and is retrained on Dream trajectories with the authors' own collection procedure (oracle unmasking on GSM8K-train), at the same generation length as the evaluation.