GRADIENT DESCENT ON TOKENS IS SPARSE: UNDERSTANDING THE COMPUTATIONAL STRUCTURE OF DIFFERENTIABLE TOKEN OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Inference-time computation has become an important factor in language model quality, and differentiable token optimization (DTO) is one way to allocate this computation. DTO treats a model's discrete output tokens as continuous parameters and refines them using gradient descent. Test-time reasoning, adversarial attacks and controllable generation all share this template, and every member's default gives each position the same fixed budget at every step, presuming that the computation is homogeneous. This assumption does not hold, because DTO is a discrete process evaluated through a continuous surrogate, and the two decouple within the first few dozen updates. Once a position's margin exceeds what a single step can overturn, the surrogate's gradients can no longer alter the discrete output. The resulting computation is not merely redundant, since updates to uncommitted positions can continue to alter the output without improving quality. Accuracy falls monotonically from 47.1% at greedy decoding to 34.2% at $T = 60$, with losses outnumbering gains 32 to 1. Because the decoupling arises from the interface between continuous optimization and discrete token assignment rather than from any particular objective, it holds across the family and is measurable. Tracking 340 discrete trajectories over two model families on MATH-500, we find at least 89% of token flips within the first 50 of 59 measurements, the most active fifth of positions absorbing a third of the churn, and a churn plateau that never converts into quality. Three propositions characterize what can be recovered from the decoupling and yield three mechanisms, gradient reuse across stable updates, position-level activity prediction, and objective-driven stopping with a loss bound. Acting on the structure jointly removes 15.8% of model calls at $T = 60$ without measurable loss of accuracy. The budget of DTO is therefore a measurable property of the decoupling rather than a constant to be tuned.

1 INTRODUCTION

The quality of a large language model (LLM) depends not only on how it was trained but also on how much computation it is given at inference time (Wei et al., 2022; Snell et al., 2024). One prominent family of test-time methods performs *differentiable token optimization* (DTO). It treats the model's discrete output tokens as continuous parameters (logits, soft token embeddings, or hidden states) and runs gradient descent on them to maximize an objective. The family spans test-time reasoning that refines an output toward higher-reward regions (Wang et al., 2026), adversarial attacks that optimize suffix tokens (Zou et al., 2023), and controllable generation via energy-based decoding (Qin et al., 2022; Dathathri et al., 2020). These methods differ in objective and parameterization, yet they share one computational template. Each defaults to initializing from a decoded sequence and running gradient descent for a fixed number of steps over every token position. These methods often use substantial optimization budgets, with ∇-Reasoner running $T = 100$ steps over every generated position. The template tacitly assumes that the computational demand of DTO is *homogeneous*, so that every step and every position matter roughly equally (Figure 1).

The assumption does not hold, and the reason is structural. DTO is a discrete process evaluated through a continuous surrogate. What determines the output is the hard argmax assignment, whereas the optimizer moves only the logits feeding it, and the two decouple within the first few dozen up-

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.

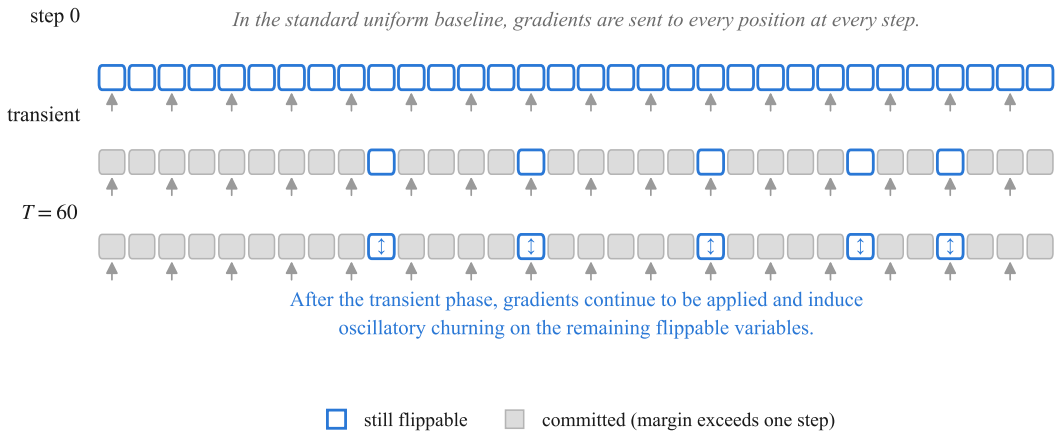


Figure 1: The template every method in the family shares, and where it breaks. Rows show one token sequence at three moments. Grey marks positions whose margin now exceeds what one gradient step can overturn, so their assignment can no longer change. The arrows mark where a gradient is applied to that variable at that step, and are identical across rows. The uniform implementation updates every position whether or not its assignment can still change.

dates. Once a position's margin exceeds what a single step can overturn, the surrogate keeps producing gradients that can no longer reach the discrete output, and the computation spent there is inert. Worse, gradients keep arriving at already-fixed variables while the minority still free to move drifts the output away from the objective. Accuracy therefore does not saturate but falls monotonically with the budget. The default schedule thus does not reflect the heterogeneous dynamics of the discrete optimization process, and the appropriate budget is a property of the decoupling, measurable rather than tunable.

We measure it by recording the discrete trajectory itself. The continuous loss looks like ordinary optimization, and only the hard assignment reveals that most of the computation changes nothing. Across two model families on MATH-500 (Qwen2.5-7B-Instruct on 240 problems, Llama-3.1-8B-Instruct on 100), against a shared reward model (RM), the computation is sparse along three distinct dimensions (Section 4). In time, token changes concentrate in the first few dozen updates, after which the process settles onto a low-amplitude *churn plateau* that never reaches zero. In space, most positions change in only a small fraction of updates, and the churn that remains concentrates in a minority of them. In quality, accuracy peaks at the smallest budget tested and declines monotonically thereafter, with greedy decoding outperforming all evaluated optimization budgets.

The mechanisms already appear in this family's implementations (Wang et al., 2026). The contribution is the measured decoupling they rest on and the theory it implies. Three propositions state the decoupling in closed form, predict where flips remain possible, and bound the cost of stopping. The first two are falsifiable against the measurements, each dimension yields one mechanism, and the three together reduce the number of model calls without measurable loss of accuracy (Section 4.3).

2 Related Work

The use of gradients to optimize discrete text dates to token-substitution attacks (Ebrahimi et al., 2018; Guo et al., 2021), which established that gradients can identify which tokens to change. The decisive step was to let those signals pass through discrete sampling. Gumbel-softmax reparameterization (Jang et al., 2017; Maddison et al., 2017) approximates categorical sampling by a continuous relaxation, and the straight-through estimator (STE) (Bengio et al., 2013) retains the hard assignment in the forward pass while carrying soft gradients backward. Gradient signals have since driven guided discrete replacement (Shin et al., 2020), controllable generation (Dathathri et al., 2020; Qin et al., 2022), prompt tuning (Lester et al., 2021), and, more recently, test-time reasoning (Wang et al., 2026; Hao et al., 2024; Zelikman et al., 2024). These applications differ in objective and in parameterization, but not in computational structure. Objective and parameterization determine what those gradients mean and which variables they reach, while the step budget and the position set determine

how much computation is spent. The last two are fixed in advance, and Section 4 measures what they cost.

Deployed variants (Wang et al., 2026) stabilize this scheme with per-token gating, acceptance criteria for individual updates, gradient caching, and a stronger coupling to the prior ($\beta = 1$ against our 10^{-3}). The first three sit on the axes measured in the sections below. Selectivity acts on the flippable positions, acceptance on updates that do not convert, and caching on the temporal axis. The findings concern the uniform default they stabilize, and the measured structure indicates when they matter.

A second line of work allocates inference-time computation, but does so in the zeroth-order (sampling) regime. Best-of-N and tree search (Stiennon et al., 2020; Nakano et al., 2021; Yao et al., 2023; Hao et al., 2023) widen the candidate pool, while Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) have the model rewrite its own output. Snell et al. (2024) characterize the optimal split between sampling and verification (Lightman et al., 2024). These methods allocate additional samples rather than computation within a single optimization trajectory. A recurring observation in this literature is that more computation does not always improve performance, a phenomenon known as *overthinking* (Zhou et al., 2026; Srivastava et al., 2026). We observe the same behavior in the quality dimension, but what is scaled differs. It is not samples or chain length but the number of updates within a single first-order optimization. The two regimes also fail differently, since a sample is a fresh draw whereas an update moves a continuous parameter and can therefore change the answer the model would otherwise have given. Past the transient, it changes that answer against the objective (Section 4.2.3).

That not all parameters require equal computation is an older idea. In every form it has taken, the decision belongs to a trained router or a halting unit, a gate or an exit that decides how far an input travels (Graves, 2016; Huang et al., 2018; Wang et al., 2018; Teerapittayanon et al., 2016; Xin et al., 2020; Raposo et al., 2024), a per-parameter step size learned during training (Duchi et al., 2011; Kingma & Ba, 2015), or structural sparsity built into the model (Han et al., 2015; Shazeer et al., 2017; Beltagy et al., 2020). The structure we study is none of these, and it is neither assumed nor learned from data. It emerges from the interaction between gradient pressure and the hard assignment and lies outside the model rather than inside it. Mask-GCG (Mu et al., 2025), Probe Sampling (Zhao et al., 2024) and Gradient-Guided Reward Optimization (GGRO) (Lin & Zhang, 2026) approach the same phenomenon indirectly, resting on the observation that most positions never contribute, but none characterizes *when* flips occur or *where* they land. Appendix G relates these results to adaptive computation and classical early stopping.

3 The Decoupling Principle

∇-Reasoner (Wang et al., 2026), GCG (Zou et al., 2023), and the other methods above differ in objective and parameterization but instantiate a single template, which can be formalized once.

3.1 General Formulation

Given a language model π_{LLM}, a context x, and an output sequence $y = (y_1, \ldots, y_L)$ with logits $z = (z_1, \ldots, z_L) \in \mathbb{R}^{L \times V}$, where V is the vocabulary size, let $\mathcal{O}(y|x)$ be a differentiable objective. Gradient-based text optimization solves

$$\tilde{y} = \text{argmax}_y \ \mathcal{O}(y|x) + \beta \cdot \log \pi_{\text{LLM}}(y|x) \tag{1}$$

where $\beta \geq 0$ weights the log-prior term (the *negative log-likelihood* (NLL) fidelity coefficient, $\beta = 10^{-3}$ in our runs, Appendix C).

Because y is discrete, the standard approach parameterizes each position by continuous logits and applies the straight-through softmax estimator (Bengio et al., 2013). The forward pass carries a discrete one-hot while the backward pass flows through the softmax, so that $\nabla_z \mathcal{O}(y(z)|x)$ is computable (Appendix B). Starting from initial logits $z^{(0)}$, typically the LLM's greedy output, gradient descent runs for T steps:

$$z^{(t+1)} = z^{(t)} - \eta \cdot \nabla_z \mathcal{L}(z^{(t)}), \tag{2}$$

where $\mathcal{L}(z) = -\mathcal{O}(y(z)|x) - \beta \cdot \log \pi_{\text{LLM}}(y(z)|x)$ (written $\mathcal{L}$ to avoid clashing with the sequence length L) and η is the learning rate.

Appendix B specifies the two instances, ∇-Reasoner, which optimizes logits (or last-layer hidden states) against a reward model with $\beta = 1$, and GCG (Greedy Coordinate Gradient), which sets $\beta = 0$ and maximizes the log-probability of a target response following an adversarial suffix. Every measurement reported in this paper comes from the reward-model instance.

3.2 When the Discrete Output Stops Responding

The uniform way to run the instances above is to give every position all T updates of Eq. (2), whether or not it ever changes its hard assignment. We call this fixed-budget, all-position scheme the *uniform reference implementation* and take it as the object of study, since it embodies the homogeneity assumption directly.

We study the process through its discrete events. Let $y_i^{(c)}$ denote the hard (argmax) token at position i after update c, and say that position i *flips* at measurement c when $y_i^{(c+1)} \neq y_i^{(c)}$. The condition below characterizes when a single gradient step can produce a flip, which is the decoupling stated in closed form.

Proposition 1 (Single-Step Flip Condition)**.** *Let $z \in \mathbb{R}^V$ be the logits at a single token position, with softmax probabilities $p = \mathrm{softmax}(z)$, and let $j^* = \mathrm{argmax}_j\, p_j$ be the current predicted token. Under a single gradient step $z' = z - \eta \cdot g$ with $g = \nabla_z \mathcal{L}$, the token flips ($\mathrm{argmax}_j\, \mathrm{softmax}(z')_j \neq j^*$) if and only if there exists $k \neq j^*$ such that*

$$\eta \cdot (g_{j^*} - g_k) > z_{j^*} - z_k. \tag{3}$$

Proposition 1 (proof in Appendix A) isolates two quantities that govern flips, the *logit margin* $z_{j^*} - z_k$, which grows as a position commits to a token, and the *gradient differential* $g_{j^*} - g_k$, which carries the optimization pressure. A flip requires the latter to overcome the former. Since the objective enters only through the gradient differential, the condition holds for every instance of the template.

4 Experiments: Structure, Mechanisms, and Generalization

This section measures where the computation of the uniform reference implementation is spent, whether that structure converts into an accelerator, and how far it generalizes. Experiments 1–3 profile the structure, 4–5 build the mechanisms, and 6–7 test how far it replicates on a second family and task.

4.1 Protocol, Datasets and Evaluation Metrics

All main measurements are on MATH-500 (Lightman et al., 2024) with two instruction-tuned models in 4-bit NormalFloat (NF4), Qwen2.5-7B-Instruct (Team, 2024) and Llama-3.1-8B-Instruct (Grattafiori et al., 2024), scored by a shared reward model (Liu et al., 2025) so that the reward term is identical across families. Each problem begins with a greedy decode, after which DTO optimizes the sequence in logit space for $T = 60$ steps, with generations capped at 128 tokens except in the accuracy sweeps, which run at 512. Hyperparameters are fixed across all runs (Adam at learning rate 1.0), since the object of study is the uniform reference implementation rather than a tuned variant. The problem subsets overlap, with 240 on Qwen2.5-7B-Instruct for the structural measurements, 100 on Llama-3.1-8B-Instruct for the cross-family replication, 40 (indices 80–119) for the joint evaluation, and 15 Llama problems for the no-cache controls. 24-problem Qwen smoke runs at $T = 100$ anchor the temporal profile, 80 problems (indices 0–79) supply the predictor's training labels, and the cross-task check uses 200 problems from GSM8K (Cobbe et al., 2021).

We record the hard (argmax) assignment after every update, giving checkpoints $c = 0, \ldots, T-1$. The flip rate at measurement c is the fraction of positions differing between checkpoints c and $c+1$, averaged over problems:

$$r_c = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{L_n} \sum_{i=1}^{L_n} \mathbb{1}\big[y_{i,n}^{(c+1)} \neq y_{i,n}^{(c)}\big], \tag{4}$$

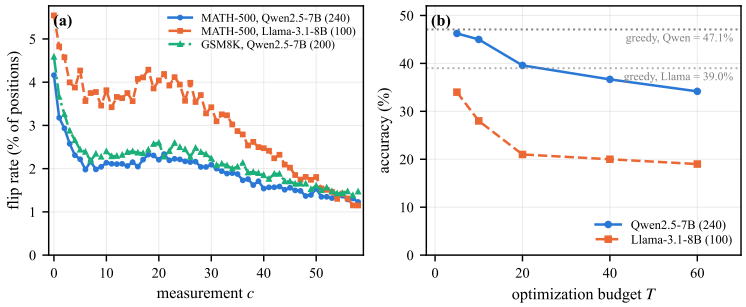


Figure 2: Process and quality profiles of the uniform reference implementation. (a) Flip rate, the fraction of positions changing per update, against the measurement index c, for MATH-500 on Qwen2.5-7B (240 problems) and Llama-3.1-8B (100) and GSM8K on Qwen2.5-7B (200). (b) Accuracy against the optimization budget, with dotted lines for the greedy baselines.

with N problems, L_n positions in problem n, and $\mathbb{1}[\cdot]$ the indicator. The transition from the greedy initialization into checkpoint 0 is not logged, so each curve has $T-1$ measurements. Four quantities are recorded, each tied to the configuration in which it was measured. Hard-decoded accuracy is per-problem correctness, compared pairwise by an exact McNemar test. The flip rate r_c follows Eq. (4). The reward delta RM Δ is the change in the reward model's score on hard-decoded text relative to the greedy output. Model calls are cumulative counts of LM and RM forward and backward passes, with each pass covering one full sequence and no KV state reused across steps. An uncached DTO step therefore costs four calls and the uniform reference costs $4T$ per problem. Because the flip rate and accuracy are computed at the same checkpoints, process and quality align directly. Answer-extraction conventions and the hardware and software are given in Appendix C.

4.2 Measurement: Three Sparse Dimensions

The three subsections below profile the uniform reference implementation along three facets, *when* flips occur, *where* they occur, and what continued optimization converts into.

4.2.1 Temporal profile: flips concentrate in the first few dozen steps (Experiment 1).

All three curves in Figure 2(a) fall steeply, bend near the tenth measurement, and then flatten onto a low but strictly positive level. That flattened tail, rather than the descent, distinguishes DTO from a convergent optimization. Both families settle onto a churn plateau that persists to the end of the run, a little over 1% per step on Qwen and higher on Llama once the cached tail is corrected (Table 1, Appendix D.3). A 24-problem smoke run at $T = 100$ with no caching rules out an artifact of the $T = 60$ budget. Between measurements 70 and 98 the rate holds near 1% (range 0.8–1.4%, mean 1.09%). The structure is visible only because the hard assignment is recorded, rather than the continuous loss or the final output.

Concentration is clearest in cumulative terms, where $S(m) = \sum_{c=0}^{m-1} r_c / \sum_{c=0}^{T-2} r_c$ denotes the share of all flips in the first m measurements. The Qwen curve gives $S(10) = 22.3\%$, $S(30) = 60.1\%$ and $S(50) = 89.4\%$, and the Llama curve $S(50) = 93.0\%$. In absolute terms Qwen flips about 5.3 tokens per update at the start and 1.6 by the final measurement, Llama 6.8 and 1.4.

The process does not freeze. Only 21% (Qwen) and 36% (Llama) of problems have zero flips in their final measurement, so the tail is persistent low-amplitude churn rather than a fixed point. Proposition 1 explains its shape. Committed positions acquire large margins and leave the flippable set, which shrinks rapidly and then stabilizes at the population whose margins remain inside the step-size band.

Proposition 2 (Entropy–Gradient Interaction)**.** *Under a local quadratic approximation of the loss landscape around z, the likelihood that a single gradient step of size η produces a token flip increases with*

$$\mathbb{P}(\textit{flip}) = \Phi\big(H(p) \cdot \|\nabla_z \mathcal{L}\|_2 \cdot \cos(\theta)\big) + O(\eta^2), \tag{5}$$

where Φ is increasing, $H(p) = -\sum_j p_j \log p_j$ is the entropy of the softmax distribution, and $\cos(\theta)$ is the cosine similarity between the gradient direction and the direction toward the runner-up class, with the derivation in Appendix A.

Eq. (5) accounts for the decay-and-plateau shape. As positions commit, their distributions become near-deterministic ($H(p) \to 0$), so the product collapses for all but the small population where both factors remain non-trivial. The account is first-order and ordinal rather than quantitative, resting on a local two-class reduction of the entropy term, with the alignment factor absorbed under an isotropy step in the interpretation. Entropy and gradient norm separate positions that later flip from those that never do with areas under the ROC curve (AUC) 0.81 and 0.80, measured before optimization (Appendix A).

4.2.2 Spatial profile: flips concentrate in a minority of positions (Experiment 2).

Replaying the stored trajectories reveals the spatial counterpart of the temporal pattern. Let f_i denote the number of flips of position i over the run, and order positions by decreasing f_i. The share carried by the most active fraction q of positions is then $C(q) = \sum_{i \le qL} f_i / \sum_{i=1}^{L} f_i$. Over the 59 measurements, 97.7% of Qwen positions and 93.8% of Llama positions change in fewer than 5% of updates on average, and $C(0.2) = 34.0\%$ on Qwen and 33.8% on Llama against 20% for uniformly spread churn. Half of all flips fall on 45 of the 128 positions on Qwen and 46 on Llama. The ratio of the most to the least active position reaches $14.6\times$ and $12.8\times$. A position flips a mean of only 1.1 (Qwen) and 1.7 (Llama) times across the run, so fewer than 3% of (position, update) pairs involve any change.

The two families flip different amounts but allocate the churn nearly identically, their concentration curves differing by less than a point. The allocation is therefore a property of the decoupling rather than of a particular model family.

4.2.3 Quality profile: past the transient the gradients act against the output (Experiment 3).

The temporal and spatial profiles describe *how much* the process moves, while the quality profile asks whether that movement improves the output, and the results indicate that it does not. On the 240 MATH-500 problems, greedy decoding scores 47.1%. After five optimization steps accuracy is 46.3%, statistically indistinguishable from greedy (paired outcomes 0 improved and 2 degraded, exact McNemar $p = 0.50$). It then declines monotonically with the budget, reaching 34.2% at $T = 60$ (exact McNemar $p = 7.9 \times 10^{-9}$). Past the transient, optimization reduces accuracy, which is the inversion the decoupling predicts. In both families the dotted greedy baseline (Figure 2(b)) lies above every optimization budget.

The reward side is no more reliable, and for the same reason. On the hard-decoded text the reward-model score falls by 1.79 points on Qwen and 3.57 on Llama on average, with only 21.7% and 10.0% of problems improving. Gradients routed through a differentiable reward model need not correspond to gains under that same model's hard-decoded evaluation (Appendix E). The answer side is skewed the same way. On the 240 Qwen problems, 13.3% ($32/240$) drift from correct to wrong against a single gain (0.4%), a ratio of 32:1, and Llama shows the same picture at 21:1 (Table 4 and Appendix E). Two further mechanisms keep the asymmetry from closing. Past the transient, *drift* moves the answer without improving any objective. *Persistent churn* keeps a residual set of positions flipping indefinitely, so the process never settles into a state that could be recognized as finished.

The three dimensions are one phenomenon seen three ways. Gradient descent operates on logits, but what determines the trajectory is the hard assignment. Once a position commits (Proposition 1), the continuous motion that remains can no longer change anything discrete, and the churn plateau is the equilibrium of that decoupling. In space the same quantities vary by more than an order of magnitude across positions (Proposition 2), leaving a small active set. In quality the reward's differentiable surface and its hard-decoded evaluation disagree, so progress on the surrogate does not convert into correctness. The uniform reference implementation, computing as if all of this

Table 1: Dynamics and quality profiles across two model families. Flip statistics are over the 59 measurements of a $T = 60$ run. Accuracy is on 240 (Qwen) and 100 (Llama) MATH-500 problems at a 512-token generation cap. Accuracy entries are exact. The Llama flip statistics come from a cached run that understates the tail (Appendix D.3).

	Qwen2.5-7B	Llama-3.1-8B
Temporal profile		
Flip rate, first measurement ($c = 0$)	4.2%	5.6%
Flip rate, final measurement ($c = 58$)	1.2%	1.2%
Share of flips in first 50 measurements	89.4%	93.0%
Problems with zero flips in final measurement	21%	36%
Spatial profile		
Positions changing in $< 5\%$ of updates	97.7%	93.8%
Mean flips per position	1.14	1.74
Quality profile		
Greedy accuracy	47.1%	39.0%
Accuracy at $T = 5$	46.3%	34.0%
Accuracy at $T = 60$	34.2%	19.0%
RM reward delta (hard-decoded text)	-1.79	-3.57
Problems with positive reward delta	21.7%	10.0%

moved together, follows the wrong schedule, and the mechanisms of Section 4.3 recover part of that cost.

4.3 Mechanisms: From Sparse Structure to Acceleration

Measurement alone is diagnostic. It shows that the uniform reference implementation spends most of its compute on updates that are either inert with respect to the discrete output or detrimental to measured accuracy, without saying which parts can be omitted. Each sparse dimension yields one mechanism, gradient reuse across stable updates in time, position-level activity prediction in space, and objective-driven stopping in quality.

4.3.1 Temporal: gradient reuse across stable updates (Experiment 4).

The only product of a DTO step is the update to the token variables. When an update leaves the sequence's hard assignment unchanged, the gradient describes the same discrete problem as the previous full step and can be reused. The objective for the skipped step becomes the inner product $\langle g_{\text{cached}}, \text{soft}(z)\rangle$, whose backward pass reproduces the cached gradient. The optimizer therefore applies the same update as the full pipeline, up to a change in the Jacobian induced by the softmax values, while the skipped step becomes a small matrix operation with no LM or RM passes. Formally, writing $\bar{y}(z) = \text{argmax}_j \, \text{soft}(z)_j$ for the hard assignment, step t is reusable when

$$\bar{y}(z^{(t-1)}) = \bar{y}(z^{(t-2)}), \qquad g_{\text{cached}} = \partial \mathcal{L} / \partial \, \text{soft}(z^{(t-2)}). \tag{6}$$

Reuse under Eq. (6) thus trades storage for computation, unlike gradient checkpointing (Chen et al., 2016). The cache must be invalidated whenever the hard assignment changes, including on skipped steps. The construction is exact under a frozen assignment and empirically token-exact, so reuse leaves the trajectory unchanged and can be enabled by default, with Appendix D specifying the variants and the exactness check. On the 240-problem Qwen configuration with 128-token generations, only 10.6% of updates satisfied the stable-assignment condition and were served from cache. The rate is capped by the churn of Section 4.2.1, since one flipping position invalidates the step. Exact and inexpensive as it is, reuse alone recovers only a modest part of the redundancy.

4.3.2 Spatial: position-level activity prediction.

Section 4.2.2 shows that flips concentrate in a minority of positions. A predictor that identifies which positions can ever change, using only information available *before* optimization, lets the optimizer freeze the rest. Its features follow the flip geometry of Proposition 2, per-position entropy, gradient

norm, their product and the logit margin, plus the relative position, the log sequence length and a last-token flag. The predictor is logistic, $P_i = \sigma(w^\top \varphi_i + b)$, chosen because it must precede optimization, with coefficients readable against those same drivers. Positions with P_i below a threshold ρ are frozen, subject to a cap $f_{\max}$ on the frozen fraction ($\rho = 0.3$, $f_{\max} = 0.6$). Its five-fold cross-validated AUC over 80 problems and 10,218 positions is 0.817. Freezing compounds with reuse, since fewer flippable positions means more steps satisfy the stable-assignment condition, but it removes no LM/RM pass on its own, because a frozen position still attends. Integrated into the joint configuration of Section 4.3.4, it freezes 45.2% of positions on average over the 40 evaluation problems.

4.3.3 QUALITY: OBJECTIVE-DRIVEN STOPPING.

Reuse and position prediction recover process redundancy, whereas stopping recovers the budget that further optimization spends degrading quality (Section 4.2.3). Early stopping in training uses the same principle (Duvenaud et al., 2016), but the criterion differs, held-out loss there, the hard-decoded objective here, at both step and position granularity.

The first rule, the *objective plateau*, stops when the hard-decoded objective, or its training surrogate, has not improved for p consecutive steps.

$$\mathcal{O}^{(t)} \leq \max_{s \leq t-p} \mathcal{O}^{(s)} \implies \text{stop at step } t. \tag{7}$$

Because the residual churn never vanishes (Section 4.2.1), Eq. (7) cannot wait for the token variables to stop moving, only for the objective to stop improving. The second rule, *per-token gradient convergence*, stops updating position i when $\|\nabla_{z_i} \mathcal{L}\|_2 < \varepsilon$, keying on the gradient norm rather than variable displacement so that the rule matches the loss bound stated below.

Proposition 3 (Early Stopping Safety)**.** *Let $z^* = \operatorname{argmin}_z \mathcal{L}(z)$ be a local minimum. If per-token early stopping is applied when $\|\nabla_{z_i} \mathcal{L}\|_2 < \varepsilon$, the stopped solution $\tilde{z}$ satisfies $\mathcal{L}(\tilde{z}) \leq \mathcal{L}(z^*) + \frac{1}{2\mu} L \varepsilon^2$, where μ is the local strong convexity constant and L the sequence length.*

Proposition 3 (proof in Appendix A) is quadratic in the gradient threshold and linear in sequence length. Loosening the criterion degrades the objective only quadratically in the local regime the bound assumes, while longer sequences demand tighter thresholds. The bound is worst-case, its local assumptions and constants left unverified, so it motivates the form of the rule rather than certifying it. Cutting the budget from $T = 60$ to $T = 5$ leaves accuracy statistically indistinguishable from greedy decoding, so halting early avoids the damage of long schedules. This oracle cutoff, available only with the evaluation outcome, would remove 92% of the calls, but the measured criterion triggers rarely (Section 4.3.4). Stopping therefore recovers accuracy up to the greedy baseline rather than beyond it. The bound covers the per-token rule run to completion. No measured run enables it, and where stopping is enabled, it applies the plateau criterion conservatively (Appendix C).

4.3.4 THE SPARSITY-AWARE LOOP (EXPERIMENT 5).

Algorithm 1 (Appendix B) places the three mechanisms in a single loop, measured jointly rather than extrapolated from their individual effects. On the same 40-problem subset, indices 80–119, which is disjoint from the predictor's training problems and therefore free of label leakage, all three are enabled simultaneously and compared against the uniform reference implementation budget by budget. The saving, defined as $1 - C_{\text{joint}}/C_{\text{uni}}$ in model calls, is 15.8% at $T = 60$, and accuracy is unharmed. The joint and uniform configurations agree pointwise for $T \leq 40$, and at $T = 60$ they differ by a single problem, $3/40$ against $2/40$. Within this subset's 128-token generation cap, only the relative comparison is meaningful. The saving grows monotonically with the budget, from 2.0% at $T = 5$ to 15.8% at $T = 60$. The growth follows from Eq. (6): reuse requires a step in which no position flips, and the per-step flip rate decays toward the churn plateau (Table 1). Longer schedules therefore hold a larger share of reusable steps. The saving is largest where the uniform implementation also degrades most. That saving is a count of passes, and its conversion into wall-clock time depends on the implementation (Appendix F).

Position prediction then adds 2.7% and stopping 0.8% on top of reuse, so the joint 15.8% slightly exceeds the sum of the parts. Reuse remains capped by the one or two positions that still change at

Table 2: Per-mechanism ablation at $T = 60$ on the same 40-problem MATH-500 subset. Each mechanism row includes gradient reuse, which the reference row lacks. Calls are per problem.

Configuration	Calls	Saving
Uniform reference	240.0	—
Gradient reuse alone	213.1	11.2%
Reuse + position prediction	206.6	13.9%
Reuse + stopping	211.3	12.0%
All three (joint)	202.1	15.8%

each update, and early stopping triggers on only 2.5% of problems when measured without position prediction. Both limits trace to the residual churn rather than to the mechanisms.

4.4 Comparison, Ablation and Generalization (Experiments 6 and 7)

The relevant reference is the family's shared default, which runs a fixed budget over every position. Comparing against it isolates the allocation under a fixed model, objective and budget. At $T = 60$ every configuration is within one problem of the reference's, and Table 2 reports the cost of each, on top of reuse and jointly, with accuracy detail in Appendix F.

Two implementation settings vary across the run set, each bounded rather than left free. The generation cap confines absolute accuracy comparisons, 128 tokens for the structural runs against 512 for the accuracy sweeps. The more permissive cache-invalidation variant leaves accuracy invariant on a matched control, with no disagreement across 90 paired checkpoint results, while the flip statistics of the affected runs understate the plateau (Appendix D.3).

Experiments 6 and 7 change the model family and the task, holding the objective fixed. Experiment 6 repeats the temporal, spatial and quality profiles on Llama-3.1-8B-Instruct (100 problems, Table 1). Experiment 7 runs an identical pipeline on GSM8K (200 problems), reproducing the decay-and-plateau flip curve (89.6% of flips in the first 50 measurements against 89.4% on MATH-500) and the spatial concentration, with 96.9% of positions changing in fewer than 5% of updates on average and a most-to-least-active ratio of $23.1\times$ against $14.6\times$. Its accuracy sweep covers 140 of these problems and declines from 84.3% at $T = 5$ to 68.6% at $T = 60$ (exact McNemar $p = 2.7 \times 10^{-5}$), against a greedy baseline of 82.1%, which the smallest budgets meet (exact McNemar $p = 0.45$ and 1.00). Directions replicate across families and tasks, while magnitudes are approximate, since the reward model transfers less reliably between them (AUC 0.81 against 0.94, Appendix C).

5 Conclusion

The three sparse dimensions are directly measured, across two model families under one default configuration, and the three propositions follow from the family's shared template. The prescription is simple because the cause is structural: spend the budget on the transient, watch what updates convert into, and do not assume that more steps help. In our evaluated settings, past a few dozen updates the computation stops being redundant and starts to degrade the answer, and longer schedules degrade it more reliably. Extending the prescription to other objectives, hyperparameters or larger models requires measurement rather than extrapolation. The cross-task check reproduces the process structure and the accuracy decline on GSM8K. The main curves are single-run point estimates at fixed seed 42, with the two bounding approximations documented in the Reproducibility statement.

The broader lesson is methodological. The computational structure of an optimization-based test-time method is an empirical object, visible only when the discrete trajectory is recorded, and it can differ sharply from what the continuous loss suggests. Accuracy falling under longer budgets is direct evidence that the demand is overstated by the default schedule rather than limited by the model. One instrumented run suffices to locate where further updating stops converting into quality. Gradient descent on tokens is, in the regimes of interest, a short and highly localized computation. The default treats it as a long and uniform one, suggesting that iterative discrete-token optimization should allocate computation to the positions that remain responsive to updates.

AI USE STATEMENT

In this work, we did not use generative AI tools for any of the tasks whose disclosure is required, including translation, the implementation of the methods or the experimental code, the development of the theoretical results and the design of the experiments, all of which were carried out by the authors. Additionally, we used generative AI tools to draft initial versions of some sections, to edit the text for readability, logical flow and academic register, and to identify relevant literature. We have reviewed all AI-assisted work. AI-drafted and AI-edited passages were checked against our own measurements and results and rewritten where they did not match them, and the reference list was verified against the original sources by the authors. All core research ideas, technical approaches, mathematical derivations, experimental design, analysis of results and final conclusions were developed and verified independently 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.

ETHICS STATEMENT

This paper studies differentiable token optimization (DTO), the computational framework shared by test-time reasoning methods such as ∇-Reasoner and adversarial attack methods such as GCG. What we evaluate is the framework's computational structure. We release no attack tooling against any specific system, and we do not increase attack capability. All reported measurements come from mathematical reasoning tasks (MATH-500 and GSM8K). The one adversarial instance (GCG) is cited and formalized as an existing method only.

Such an analysis has dual-use potential. A structural understanding of *where gradients in fact change a discrete output* can in principle serve both to reduce inference cost and to construct adversarial examples more efficiently. We judge the benefits here (more honest accounting of what these methods cost, and a reproducible measurement protocol) to outweigh that risk, and this work proposes no new attack objective or transfer technique.

The reward model is validated within each model family (Appendix C), and we recommend same-family validity checks before using reward-model signals for quality judgments on real systems. This research involves no human subjects and uses only publicly available models and datasets.

REPRODUCIBILITY STATEMENT

The DTO optimization loop, the trajectory-replay utilities, and all experiment scripts will be released upon publication. The complete model, hardware and hyperparameter configuration is given in Appendix C, and the measurement provenance in Appendix D. Every quantitative claim in this paper is derived directly from the raw artifacts under `results/` (a per-run results record plus the step-by-step trajectory). The length convention of the `flip_rates` array differs between runs, and Appendix D explains how to tell which applies. Randomness is initialized with a fixed seed of 42, and the initial sequence is produced by greedy decoding, a deterministic procedure. The starting state and its reward score are therefore reproducible. 4-bit quantization and non-deterministic GPU reduction order, however, mean that bit-exact reproduction is not guaranteed, and two approximations bear on the reported numbers. First, gradient reuse was enabled in the large runs, and the accuracy and Llama runs predominantly use the more permissive cache-invalidation variant of Appendix D. Its effect is bounded by the matched no-cache control, on which cached and uncached runs agree exactly on accuracy at all six checkpoints over 15 problems, so the accuracy numbers of Table 1 hold as reported, while the Llama flip statistics are approximate. Second, the reward model is validated within each family, so cross-family comparisons are directional. Appendix D discloses the caching variants and Appendix C the reward-model validity figures.

REFERENCES

I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

K. Cobbe, V. Kosaraju, M. Bavarian, and et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

S. Dathathri, A. Madotto, J. Lan, and et al. Plug and play language models: A simple approach to controlled text generation. In *ICLR*, 2020.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2011.

D. Duvenaud, D. Maclaurin, and R. Adams. Early stopping as nonparametric variational inference. In *AISTATS*, 2016.

J. Ebrahimi, A. Rao, D. Lowd, and D. Dou. HotFlip: White-box adversarial examples for text classification. In *ACL*, 2018.

A. Grattafiori, A. Dubey, A. Jauhri, and et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

A. Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

C. Guo, A. Sablayrolles, H. Jégou, and D. Kiela. Gradient-based adversarial attacks against text transformers. In *EMNLP*, 2021.

S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. In *NeurIPS*, 2015.

S. Hao, Y. Gu, H. Ma, and et al. Reasoning with language model is planning with world model. In *EMNLP*, 2023.

S. Hao, S. Sukhbaatar, D. Su, and et al. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

G. Huang, D. Chen, T. Li, and et al. Multi-scale dense networks for resource efficient image classification. In *ICLR*, 2018.

E. Jang, S. Gu, and B. Poole. Categorical reparameterization with Gumbel-Softmax. In *ICLR*, 2017.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. In *EMNLP*, 2021.

H. Lightman, V. Kosaraju, Y. Burda, and et al. Let's verify step by step. In *ICLR*, 2024.

H. Lin and R. Zhang. Gradient-guided reward optimization for inference-time alignment. In *UAI*, 2026.

C. Y. Liu, L. Zeng, Y. Xiao, and et al. Skywork-Reward-V2: Scaling preference data curation via human–AI synergy. *arXiv preprint arXiv:2507.01352*, 2025.

A. Madaan, N. Tandon, P. Gupta, and et al. Self-refine: Iterative refinement with self-feedback. In *NeurIPS*, 2023.

C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.

J. Mu, Z. Ying, Z. Fan, and et al. Mask-GCG: Are all tokens in adversarial suffixes necessary for jailbreak attacks? *arXiv preprint arXiv:2509.06350*, 2025.

R. Nakano, J. Hilton, S. Balaji, and et al. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

L. Qin, S. Welleck, D. Khashabi, and Y. Choi. COLD decoding: Energy-based constrained text generation with Langevin dynamics. In *NeurIPS*, 2022.

D. Raposo, S. Ritter, B. Richards, and et al. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

N. Shazeer, A. Mirhoseini, K. Maziarz, and et al. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. In *EMNLP*, 2020.

N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

C. Snell, J. Lee, K. Xu, and A. Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

G. Srivastava, A. S. Hussain, S. Srinivasan, and X. Wang. Do LLMs overthink basic math reasoning? Benchmarking the accuracy–efficiency tradeoff in language models. In *Findings of ACL*, 2026.

N. Stiennon, L. Ouyang, J. Wu, and et al. Learning to summarize with human feedback. In *NeurIPS*, 2020.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

S. Teerapittayanon, B. McDanel, and H. T. Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *ICPR*, 2016.

P. Wang, R. Cai, Z. Wang, H. Mei, Q. Liu, P. Li, and Z. Wang. ∇-Reasoner: LLM reasoning via test-time gradient descent in latent space. In *ICLR*, 2026.

X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez. SkipNet: Learning dynamic routing in convolutional networks. In *ECCV*, 2018.

J. Wei, X. Wang, D. Schuurmans, and et al. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *ACL*, 2020.

S. Yao, D. Yu, J. Zhao, and et al. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023.

E. Zelikman, G. Harik, Y. Shao, and et al. Quiet-STaR: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

Y. Zhao, W. Zheng, T. Cai, and et al. Accelerating greedy coordinate gradient and general prompt optimization via probe sampling. *arXiv preprint arXiv:2403.01251*, 2024.

S. Zhou, R. Ling, J. Chen, and et al. When more thinking hurts: Overthinking in LLM test-time compute scaling. In *Findings of ACL*, 2026.

A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

A Full Theoretical Derivations

A.1 Proof of Proposition 1 (Single-Step Flip Condition)

Proof. The argmax changes exactly when $\mathrm{softmax}(z')_k > \mathrm{softmax}(z')_{j^*}$ for some $k \neq j^*$. Softmax is strictly monotone in its inputs, so this is equivalent to $z'_k > z'_{j^*}$, i.e., $z_k - \eta g_k > z_{j^*} - \eta g_{j^*}$, which rearranges to Eq. (3). □

A.2 Proof of Proposition 2 (Entropy–Gradient Interaction)

We provide the complete derivation of the entropy-gradient interaction as the dominant term in flip probability.

Setup. Consider a single token position with logits $z \in \mathbb{R}^V$. Let $p = \mathrm{softmax}(z)$ be the softmax probabilities. The gradient descent update is $z' = z - \eta g$ where $g = \nabla_z \mathcal{L}$. Under a local quadratic approximation of $\mathcal{L}$ around z:

$$\mathcal{L}(z + \delta) \approx \mathcal{L}(z) + g^T \delta + \frac{1}{2} \delta^T \mathbf{H} \delta \tag{8}$$

where $\mathbf{H} = \nabla_z^2 \mathcal{L}(z)$ is the Hessian. The optimal step under this approximation is $\delta^* = -\mathbf{H}^{-1} g$, and multiple steps of gradient descent approximate this direction.

Flip condition. From Proposition 1, a flip from class j^* to class k occurs when Eq. (3) holds. Define the margin $m_k = z_{j^*} - z_k \geq 0$ (by definition of j^* as argmax). The flip condition becomes:

$$\eta(g_{j^*} - g_k) > m_k \tag{9}$$

Entropy-margin relationship. For a two-class softmax, a reasonable local approximation when probability concentrates on the top two classes:

$$p_{j^*} = \frac{e^{z_{j^*}}}{e^{z_{j^*}} + e^{z_k}}, \quad p_k = 1 - p_{j^*} \tag{10}$$

The margin relates to the log-odds: $z_{j^*} - z_k = \log(p_{j^*}/p_k)$. Eq. (11) gives the two-class entropy:

$$H_2(p_{j^*}) = -p_{j^*} \log p_{j^*} - (1 - p_{j^*}) \log(1 - p_{j^*}) \tag{11}$$

For p_{j^*} near 0.5 (maximal uncertainty), the margin $m_k \approx 0$, making flips easy. For $p_{j^*} \to 1$, $H_2 \to 0$ and $m_k \to \infty$, making flips impossible. More precisely, the margin can be expressed in terms of entropy for the binary case:

$$m_k(p_{j^*}) = \log \frac{p_{j^*}}{1 - p_{j^*}} = f^{-1}(H_2(p_{j^*})) \tag{12}$$

where f^{-1} is a monotonically decreasing function, so higher entropy implies a smaller margin.

Gradient direction. The gradient norm $\|g\|_2$ determines the magnitude of change but not its direction relative to the decision boundary. Let θ_k be the angle between the gradient g and the direction $e_{j^*} - e_k$ (the direction from the current max toward class k).

Eq. (13) projects the gradient onto the competing direction:

$$g_{j^*} - g_k = \sqrt{2}\, \|g\|_2 \cos(\theta_k) \tag{13}$$

Flip probability. Combining the margin condition and gradient projection, a flip occurs when:

$$\sqrt{2}\, \eta \cdot \|g\|_2 \cdot \cos(\theta_k) > m_k(p_{j^*}) \tag{14}$$

Table 3: Per-feature discrimination of later flips, measured before optimization on the predictor's training split (80 problems, 10,218 positions). Each single-feature row is an ROC AUC computed without fitting, and the last row is the fitted logistic predictor of Section 4.3.2. The margin row is reported with its sign reversed, since smaller margins flip more often (Proposition 1).

Feature	AUC
Entropy $H(p)$	0.813
Gradient norm $\|\nabla_z \mathcal{L}\|_2$	0.804
Product $H(p) \cdot \|\nabla_z \mathcal{L}\|_2$	0.809
Logit margin $z_{j^*} - z_k$ (sign reversed)	0.808
Relative position	0.533
Log sequence length	0.500
Last-token flag	0.505
Joint logistic predictor (five-fold CV)	0.817

Treating $\cos(\theta_k)$ as a random variable over possible gradient directions (since the gradient depends on the specific loss surface), Eq. (15) states the resulting flip probability:

$$\mathbb{P}(\text{flip}) \approx \mathbb{P}\big(\sqrt{2}\,\eta\,\|g\|_2 \cos(\theta) > m_k(p_{j^*})\big) \tag{15}$$

For the multi-class case, we aggregate the pairwise conditions over the competing classes and replace the pairwise entropy terms by the full entropy $H(p)$, with the dominant contributions coming from the top few classes. Under the additional assumption that gradient directions are approximately isotropic in high dimensions ($V \sim 10^5$), $\cos(\theta)$ may be treated as a constant factor and absorbed into the proportionality:

$$\mathbb{P}(\text{flip}) = \Phi\big(H(p) \cdot \|g\|_2\big) + O(\eta^2) \tag{16}$$

We stress that this isotropy step is an assumption rather than a consequence of the derivation. The alignment term $\cos(\theta)$ is retained in the general statement of Proposition 2.

The $O(\eta^2)$ term captures higher-order effects from the Hessian that become relevant when the quadratic approximation degrades (large step sizes, highly non-convex regions).

The first-order account can be checked factor by factor against the stored predictor data of Section 4.3.2. Each named factor separates positions that later flip from those that never do (Table 3): entropy, gradient norm, their product and the logit margin all reach AUCs between 0.80 and 0.81, and the margin discriminates in the direction Proposition 1 predicts, with smaller margins flipping more often. Position and length features, by contrast, remain near chance. The single-feature AUCs involve no fitting, so the joint predictor's cross-validated 0.817 is the only trained number among them. What the table supports is the ordinal role of the product rather than its exact functional form. The alignment term $\cos(\theta)$ is not separately testable in these data, and the isotropy step above remains an assumption.

A.3 Proof of Proposition 3 (Early Stopping Safety)

Setup. Let $\mathcal{L}(z)$ be μ-strongly convex in a neighborhood $\mathcal{N}$ of a local minimum z^*:

$$\mathcal{L}(z') \geq \mathcal{L}(z) + \nabla\mathcal{L}(z)^T(z' - z) + \frac{\mu}{2}\|z' - z\|_2^2 \quad \forall z, z' \in \mathcal{N} \tag{17}$$

Suboptimality bound. At any point $\tilde{z}$ in $\mathcal{N}$, the strong convexity implies:

$$\mathcal{L}(\tilde{z}) - \mathcal{L}(z^*) \leq \frac{1}{2\mu}\|\nabla\mathcal{L}(\tilde{z})\|_2^2 \tag{18}$$

This follows from minimizing both sides of the strong convexity inequality w.r.t. z' (the right-hand side is minimized when $z' = z - \frac{1}{\mu}\nabla\mathcal{L}(z)$).

Per-token decomposition. The total gradient norm decomposes across token positions:

$$\|\nabla \mathcal{L}(\tilde{z})\|_2^2 = \sum_{i=1}^{L} \|\nabla_{z_i} \mathcal{L}(\tilde{z})\|_2^2 \tag{19}$$

Under per-token early stopping with threshold ε, each token's gradient satisfies $\|\nabla_{z_i} \mathcal{L}(\tilde{z})\|_2 \leq \varepsilon$, giving:

$$\|\nabla \mathcal{L}(\tilde{z})\|_2^2 \leq L\varepsilon^2 \tag{20}$$

Final bound. Substituting into Eq. (18):

$$\mathcal{L}(\tilde{z}) - \mathcal{L}(z^*) \leq \frac{L\varepsilon^2}{2\mu} \tag{21}$$

Thus the suboptimality grows at most quadratically in the gradient threshold and linearly in the sequence length. For illustration, with $\varepsilon = 10^{-3}$ and $L \sim 100$ the bound evaluates to 5×10^{-5} if μ is of order unity. We do not measure μ, and since the bound scales as $1/\mu$, it becomes vacuous as $\mu \to 0$.

Local form of the bound. Where $\mathcal{L}$ is not globally strongly convex, the bound holds locally within any region satisfying the μ-strong convexity condition, the regime the bound presumes. Its role is to certify the suboptimality incurred by stopping near a local minimum. Locating that minimum in practice is left to the plateau criterion of Eq. (7), applied conservatively in the measured runs, where a finite window of unchanged hard assignments certifies that the hard-decoded objective has not improved, which complements the per-token rule rather than duplicating it. The bound presumes the per-token rule applied to every position. If the criterion halts the run while some positions remain above ε, the per-token assumption fails and the stated bound no longer follows.

Diagnostics and scope. The two assumptions behind Proposition 3, local quadratic behavior and strong convexity near z^*, are local properties of the STE-driven loss rather than pointwise facts, and we do not verify them at the states reached in practice. One indirect diagnostic comes from the 240-problem runs. When the hard-decoded response is scored with the reward model at each checkpoint of a fixed-budget trajectory, the score attains its best recorded value within the first 20 update steps for 79.2% of problems and at the first checkpoint for 37.5%. A direct test of the regime the bound describes would require per-token gradient norms at the stopped states, which the runs do not record, so the calibration of ε remains empirical.

B Instances and the Sparsity-Aware Loop

B.1 Instances

∇-Reasoner, logit space. $\mathcal{O}(y|x) = \lambda \cdot r(y|x)$ with r a reward model, λ its scalar weight, and $\beta = 1$, maximizing reward while staying close to the LLM prior (Wang et al., 2026). Our DTO runs use the smaller fidelity coefficient $\beta = 10^{-3}$ of Eq. (1), which weakens the prior term, and the instance is otherwise unchanged. The coefficient is held fixed across every DTO run reported here, so the accuracy curve of Section 4.2.3 is measured at this weak coupling to the prior. How far the decline depends on β, the learning rate or an adaptive step-size rule is not measured, and a sweep over them is left to future work. The reported configuration also differs in budget, using $T = 100$ steps there against $T = 60$ in every run here. The same method can instead optimize the LLM's last-layer hidden states $h \in \mathbb{R}^{L \times d}$, projecting back to logits through the LM head ($z = W_{\text{lm}} h$), a variant that would test whether the conclusions transfer across optimization spaces.

GCG (Greedy Coordinate Gradient). $\mathcal{O}(y|x) = \log \pi_{\text{LLM}}(y_{\text{target}} \mid x \oplus y_{\text{suffix}})$, the log-probability that a target harmful response follows an optimized adversarial suffix, with no KL regularization ($\beta = 0$) (Zou et al., 2023).

Algorithm 1 Sparsity-aware DTO

Require: LM π_{LLM}, objective $\mathcal{O}$, context x, initial logits $z^{(0)}$, budget T, thresholds $\rho, f_{\max}, \varepsilon$, patience p, minimum steps $t_{\min}$

$P_i \leftarrow \sigma(w^\top \varphi_i + b)$ for every position i // logistic predictor, §4.3.2
active $\leftarrow \{i : P_i \geq \rho\}$
if $|\text{active}| < (1 - f_{\max})L$ **then** add the highest-P_i positions until $|\text{active}| = (1 - f_{\max})L$
$\bar{y}^{(0)} \leftarrow \operatorname{argmax} z^{(0)}$; $g_{\text{cache}} \leftarrow$ NONE
for $t = 1$ **to** T **do**
  if $t > 1$ **and** $\bar{y}^{(t-1)} = \bar{y}^{(t-2)}$ **and** $g_{\text{cache}} \neq$ NONE **then**
    $\mathcal{L} \leftarrow \langle g_{\text{cache}}, \operatorname{soft}(z) \rangle$ // cache hit, §4.3.1
  else
    $\mathcal{L} \leftarrow -\mathcal{O}(\text{STE}(z)) - \beta \log \pi_{\text{LLM}}(\text{STE}(z))$
    $g_{\text{cache}} \leftarrow \partial \mathcal{L} / \partial \operatorname{soft}(z)$ // refresh the cache
  end if
  $\mathcal{L}$.backward(); $z_i \leftarrow z_i - \eta \nabla_{z_i} \mathcal{L}$ for all $i \in$ active
  $\bar{y}^{(t)} \leftarrow \operatorname{argmax} z$; **if** $\bar{y}^{(t)} \neq \bar{y}^{(t-1)}$ **then** $g_{\text{cache}} \leftarrow$ NONE
  active $\leftarrow$ active $\setminus \{i : \|\nabla_{z_i} \mathcal{L}\|_2 < \varepsilon\}$ // per-token stopping, §4.3.3
  if $\big(t \geq t_{\min}$ **and** $\bar{y}^{(t)} = \bar{y}^{(t-1)} = \cdots = \bar{y}^{(t-p)}\big)$ **or** active $= \emptyset$ **then**
    break // settled assignment, a conservative instance of Eq. (7)
  end if
end for
return hard decode of z

Straight-through parameterization. Both instances parameterize each position by continuous logits z_i and read out a token through the straight-through softmax estimator (Bengio et al., 2013),

$$y_i = \text{one-hot}(\operatorname{argmax}_j \operatorname{softmax}(z_i/\tau)_j) + \operatorname{softmax}(z_i/\tau) - \text{sg}[\operatorname{softmax}(z_i/\tau)], \qquad (22)$$

with temperature $\tau = 1$ throughout, so that the forward pass carries a discrete one-hot while the backward pass flows through the softmax.

B.2 The Sparsity-Aware Loop

Algorithm 1 places the three mechanisms of Section 4.3 in a single loop.

C Experimental Details, Datasets, and Metrics

C.1 Models, hardware, and software

- **Language models:** Qwen2.5-7B-Instruct (240 MATH-500 problems) and Llama-3.1-8B-Instruct (100 problems), both loaded in 4-bit NF4 (double quantization, bfloat16 compute), as required by the single-GPU setup we used (NVIDIA RTX 3080 Ti, 12GB).
- **Reward model:** Skywork-Reward-V2-Qwen3-4B (4-bit NF4), shared across both LMs.
- **Software:** PyTorch, HuggingFace `transformers` and `datasets`, and a custom DTO optimization loop (STE in logit space).

C.2 DTO configuration

- **DTO:** logit-space STE, Adam, learning rate 1.0, NLL fidelity coefficient $\beta = 10^{-3}$ (the coefficient β of Eq. (1)), $T = 60$ steps, with greedy decoding throughout. Where stopping is enabled, it uses the conservative instance of Eq. (7) shown in Algorithm 1, stopping at the first step $t \geq 10$ at which the hard assignment has been unchanged for $p = 5$ consecutive steps. An unchanged hard assignment leaves the hard-decoded objective unchanged, so the instance satisfies the condition of Eq. (7) whenever it fires. The per-token ε rule of Proposition 3 is not enabled in any measured run.
- **Checkpoints:** the hard assignment is recorded after every update, so checkpoint 0 is the state after the first update, and the greedy initialization is stored separately rather than as a checkpoint.
- **Measurement definitions:** as in Section 4.1.

C.3 Answer Extraction and Generation Caps

The MATH-500 partition across experiments, the GSM8K cross-task set and the definitions of the four evaluation metrics are given in Section 4.1, which leaves two details unstated. Answer extraction follows each benchmark's convention. For MATH, the content of `\boxed{}` and the "answer is" pattern with a trailing-number fallback. For GSM8K, the value after `####` with numerical normalization. The extraction logic is unit-tested 10/10 on the task's answer conventions. The two generation caps are not directly comparable. Generation is capped at 128 tokens except in the accuracy sweeps, which run at 512, and the difference affects absolute accuracy.

C.4 Reward-model validity probe

The shared RM was validated by scoring generations of both models against correctness. On Qwen ($n = 40$, greedy) it separates correct from incorrect outputs with AUC 0.937. On Llama the same measurement gives AUC 0.775 on greedy generations alone ($n = 40$, the same condition as the Qwen number) and 0.805 when 40 temperature-0.6 generations are pooled in ($n = 80$). The pooled set shows a clear mean gap ($+5.19$, Cohen's $d = 1.23$), and no score collapse was observed.

A likely source of this gap is a family mismatch in the evaluation setup rather than a failure of the reward signal itself. Our reward model is *Qwen3*-based (Skywork-Reward-V2-Qwen3-4B), yet it is used to score *Llama* outputs, and the separation is markedly weaker on the cross-family side. The main text quotes the pooled figure (0.805), and the same-condition comparison is 0.937 against 0.775. A Llama-family reward model, or a same-condition re-validation, is left to future work. We therefore use the same RM for both families.

D Measurement Provenance and the Gradient-Cache Approximation

D.1 Caching variants and equivalence

Gradient reuse (Section 4.3.1) was used in the large runs to reduce cost. Two variants are relevant. In the *exact-invalidation* variant the cached gradient is discarded whenever the hard assignment changes, including on a skipped step. In the *latching* variant that case is missed, so a skipped step that flips a token can leave a stale gradient in place for the steps that follow, which inflates the reuse rate.

The Qwen flip-curve run (240 problems, the main temporal data) used the exact-invalidation variant. Its cached and uncached trajectories were verified token-exact over all 60 steps on two MATH-500 problems, and the reuse rate measured in this run (10.6%) is the only reuse rate we cite. The GSM8K flip and accuracy runs and the post-resume segment of the Llama accuracy run used the exact-invalidation variant as well. The Qwen accuracy run and the Llama flip-curve run used the latching variant, as did the first 90 problems of the Llama accuracy run, which was later resumed under the exact-invalidation variant. The latching effect is bounded by the 15-problem no-cache control of Section D.3. The 24-problem no-cache smoke runs ($T = 100$, a 128-token flip curve and a 512-token accuracy sweep) show the same qualitative behavior. Flips decay to a plateau, and accuracy stays at or below the greedy baseline at every budget. All reward-model validity figures involve no DTO and are unaffected.

D.2 Flips and the unlogged first transition

The flip-rate curves are computed over measurements $c = 0, \ldots, T - 2$, because the current implementation does not log the transition from the greedy initialization into checkpoint 0. Earlier internal analyses reported a terminal flip rate of zero. That was an artifact of a padded array slot that was never accumulated.

The stored `flip_rates` arrays follow one of two conventions, depending on the code version that produced them. Runs written earlier (all of the MATH-500 flip curves in this paper) store T entries with a trailing padding zero. Runs written later, including the GSM8K cross-task check, store $T - 1$ real measurements with no padding. Readers recomputing statistics can tell them apart by length.

An array whose length equals the run's `steps` carries the padding, so its last element should be dropped. An array of length `steps`-1 should be used in full. Every statistic reported here uses only real measurements, which is what reveals the residual churn plateau (Section 4.2.1). Because the first transition is excluded, our reported flip counts are, if anything, conservative.

D.3 No-cache control results

The control serves two purposes. It certifies that the approximate runs' trend conclusions hold without caching and supplies a no-cache baseline at the 512-token cap. We ran the no-cache control on the same 15 problems (flip at a 128-token cap, accuracy at 512, neither enabling the cache).

The accuracy side is unambiguous. On the first 15 problems of the Llama run (indices 0–14), the cached and uncached trajectories agree on correctness problem by problem at all six checkpoints (greedy and $T = 5, 10, 20, 40, 60$), 90 paired results with no disagreement. The latching variant therefore does not change the optimization result, and the Llama accuracy numbers of Table 1 need no correction.

The flip measurements behave differently. The latching variant reproduces the early shape faithfully (flip rates differ by no more than 0.06 percentage points over the first 10 measurements, diverging gradually thereafter to a maximum difference of 1.29 points in the tail), but it understates the residual plateau by roughly half (1.11% against 2.04%) and raises the first-50 concentration from 90.1% to 94.2% (both computed on the 15 control problems, and the same statistic on the 100-problem main run is 93.0%). The caching approximation has a measurable effect on the *process* and none on the *outcome*.

E Full Statistics for the Degradation Attribution

E.1 Degradation mechanics in full

On this benchmark the default objective converts movement into accuracy only within the transient. Past it, the collected trajectories attribute the loss of conversion to three mechanisms. First, *surrogate mismatch*, where the gradient signal routed through the reward model is not aligned with its hard-decoded evaluation (Section 4.2.3). Second, *drift*, where past the transient updates move the answer without improving any objective. Third, *persistent churn*, where a residual set of positions flips indefinitely, so any mechanism that assumes convergence underestimates the run's stability. We label each of the 240 Qwen problems by its hard-decoded correctness at six budget checkpoints (greedy and $T = 5, 10, 20, 40, 60$) and pair those labels with the problem's churn and reward trajectories.

Loss and gain are asymmetric under both anchorings. Against the greedy answer, 13.3% ($32/240$) of problems drift to wrong (correct at greedy, wrong at $T = 60$), whereas only one problem (0.4%) improves, a ratio of 32:1. Anchoring at $T = 5$ gives the same picture. Of the problems correct at $T = 5$, 27% ($30/111$) are wrong at $T = 60$, while only 0.8% ($1/129$) of those wrong at $T = 5$ recover. Drift also has no single onset. The drifting problems begin in the five budget intervals in counts $2/5/9/10/6$, and nearly four fifths of them ($25/32$) start after $T = 10$, which rules out the explanation that a small set of problems is destroyed at the outset.

A second mechanism is a mismatch between the surrogate and its own hard evaluation. If the reward direction were reliable, drifting problems would show reward increases more often than stable ones, which they do not. The share with RM $\Delta > 0$ is nearly flat across outcomes (19% for drift, 20% for stable wrong, 25% for stable correct), and the reward of the single improving problem *falls*. The rate at which the reward score rises is therefore unrelated to the hard-decoded outcome, so the surrogate's directional signal has ceased to track quality once discretization is applied. The mean reading of Section 4.2.3 (RM $\Delta = -1.79$) is not an artifact of averaging but a systematic per-problem failure.

The third mechanism is persistent churn, which correlates with damage only loosely. That observation is precisely the design constraint on the stopping rule of Section 4.3.3. A stopping rule cannot wait for flips to stop entirely, since they never do, nor can it rely on churn volume alone, since the separation between high- and low-churn groups is limited. It must instead detect whether movement

Table 4: Per-problem outcome classes and mean features on the 240 Qwen problems. Outcomes are defined by hard-decoded correctness at six budget checkpoints. Churn and reward features come from the paired flip run, associated by problem index and therefore problem-level approximations.

Outcome class	n	Share	RM Δ mean	RM $\Delta > 0$	Flips/step (1–5)	Flips/step (50–59)	Text change
Stable correct (all checkpoints correct)	77	32.1%	-1.74	25%	3.57	1.64	0.39
Stable wrong (all checkpoints wrong)	123	51.2%	-1.81	20%	4.05	1.78	0.33
Drift to wrong (greedy right, T=60 wrong)	32	13.3%	-1.87	19%	3.45	1.61	0.36
Optimization gain (greedy wrong, T=60 right)	1	0.4%	-1.05	0%	9.40	2.30	0.95
Non-monotone (mixed, e.g. right→wrong→right)	7	2.9%	-1.71	43%	5.34	2.07	0.37

Table 5: Representative drift-to-wrong failures on Qwen. The first five are ranked by RM Δ. The sixth is an early-drift problem whose reward *decreases*, showing that drift does not require the reward to rise. Trajectory entries give the hard-decoded correctness at greedy, 5, 10, 20, 40 and 60 steps, and problem text is truncated.

#	Problem (idx)	Trajectory	Drift onset	RM Δ	Flips/ step	Late 10/step	Attribution
1	Expand and simplify: $x(x(1+x)\ldots$ (66)	111100	$T20\to T40$	$+3.84$	3.12	2.10	surrogate mismatch, persistent churn
2	Two-thirds of the students at Baker Middle School… (182)	111000	$T10\to T20$	$+1.22$	2.31	1.50	surrogate mismatch, persistent churn
3	Find the midpoint of the segment between $(-5,5)$ and… (218)	100000	greedy$\to T5$	$+0.66$	0.86	0.60	surrogate mismatch, persistent churn, early drift
4	The quadratic $x^2+2.6x+3.6$ in the form… (159)	110000	$T5\to T10$	$+0.34$	2.22	1.50	surrogate mismatch, persistent churn, early drift
5	Simplify $\sec x/\sin x - \sin x/\cos x$ (74)	111000	$T10\to T20$	$+0.28$	0.93	0.50	surrogate mismatch, persistent churn
6	Find the product of $6_8 \cdot 7_8$ in base 8 (30)	110000	$T5\to T10$	-0.50	1.66	0.60	persistent churn, early drift

is still converting into the objective. The measured instance does so with a finite window of $p = 5$ consecutive flip-free steps.

On Qwen the drift label correlates only weakly with every explicit feature we extract (total and early churn, reward delta and volatility, and text rewriting) with $|r| \leq 0.12$ throughout. The text level agrees, since drift rewrites as much text as the stable classes, so degradation is not predictable from those features. The cross-family comparison below shows that the claim does not survive as a cross-model statement, but the 100 Llama problems show the same asymmetry (21 drift to wrong against 1 gain, against 32:1 on Qwen), which is the decoupling picture of Section 4.2.3. What makes a problem wrong is not a class of bad problems or a bad stretch of dynamics, but the general drift of discrete–continuous decoupling over long schedules.

The outcome classes and representative failures behind the attribution above are given in Table 4 and Table 5. The per-feature statistics that the main text summarizes are given in full in the following subsections.

E.2 Tail churn and damage

Churn and damage move in the same direction but are only loosely coupled. Splitting the tail churn (mean flips per step over steps 50–59) at its median (1.6 positions per step), the share of $T = 5$-correct problems that fall to wrong by $T = 60$ is 33% in the high-churn group and 23% in the low-churn group, so even the low-churn group still loses about a quarter of its problems. Residual churn continues to change the discrete state (share with differing state at $T = 5$ versus $T = 60$ is 14% high against 12% low), but those changes run almost entirely one way, with recovery from wrong to right at 2% in the high-churn group and 0% in the low-churn group.

E.3 Feature correlations

On Qwen, degradation is not predictable from explicit features. The drift label correlates weakly with every feature extracted. Total churn $r = -0.12$ ($p = 0.065$), early churn (first five steps) -0.11, late churn -0.05, RM Δ -0.01, RM volatility 0.04, text rewriting 0.00 ($|r| \leq 0.12$). The text level agrees. Drift rewrites as much text as the stable classes (0.36 against 0.39 and 0.33), so rewriting volume does not separate outcomes.

E.4 Cross-family differences

On Llama, only 1.5% (1/66) of problems wrong at $T = 5$ recover by $T = 60$, and drifting problems have a mean RM Δ of -4.92 with a 5% rate of reward increase (against -1.87 and 19% on Qwen).

Two cross-family differences carry information of their own rather than needing only a magnitude caveat. First, late churn loses all separation on Llama. At the late-churn median (1.35), the share of $T = 5$-correct problems falling to wrong is 47% in *both* the high- and low-churn groups (against 33% and 23% on Qwen), so predicting damage from churn volume does not survive as a cross-model claim. Second, the correlation between RM Δ and drift on Llama is weak but nominally significant ($r = -0.21$, $p = 0.035$), unlike the $|r| \leq 0.12$ seen for every Qwen feature. The claim that degradation is unpredictable from explicit features therefore holds on Qwen only, and we do not state it as universal. Both families agree in direction, and the magnitude differences (Appendix C) make the comparison directional only.

F Comparison and Ablation Details

Two further points are not stated in Section 4.4. First, the positive interaction reported in Section 4.3.4 is consistent with freezing raising the reuse rate, with cache hits per problem rising from 6.7 to 8.3 in the prediction configuration. Second, accuracy is not disturbed by any of the configurations, which agree at every checkpoint through $T = 40$, each at 3/40, and at $T = 60$ none is worse than the uniform reference, with position prediction and the joint configuration staying at 3/40 and reuse and stopping matching the reference's 2/40. Each call in the metric of Section 4.1 is one pass, and an uncached DTO step runs four of them, one LM forward, one RM forward, and the backward pass through each. Because every executed step has this composition, the reported saving is invariant to the relative cost of the four passes, up to the small inner-product backward each cached step still runs. The protocol generates no candidate pool and carries no KV state across steps. We do not report end-to-end latency, since the realized speedup depends on how the skipped backward passes map onto the batching and kernels of a given implementation, and measuring it is left to future work.

Gradient caching is the temporal axis of Section 4.3.1, reusing a gradient while the hard assignment is unchanged. Per-token gating is the position axis of Section 4.3.2. Position prediction selects which positions receive updates and caps how many can be frozen at once. Acceptance criteria for individual updates are the stopping axis of Section 4.3.3, on which our runs stop once a window of unchanged assignments certifies that updates have stopped converting. The remaining element, the strength of the coupling to the prior, is held fixed across our runs (Appendix B), so the uniform reference serves as the ablation base for the others, and Table 2 quantifies what they contribute within the tested regime.

The savings of Section 4.3.4 are bounded by the churn the mechanisms leave in place. Reuse can serve an update only when the preceding update left the entire hard assignment unchanged. A single

flipped position breaks the condition for the whole update, so the flip-free share of updates is an upper bound on the reuse rate. Over the 240-problem run that share is 11.1%, rising to 19.0% over the last ten updates as the flip rate decays toward the churn plateau. The implemented cache is consulted exactly on the updates the condition admits, so the realized rate is set by this bound rather than by the implementation. Freezing raises the bound itself, from 11.7% to 14.3% on the 40-problem subset, by removing positions that would otherwise flip. Stopping is limited instead by how rarely it fires, on only 2.5% of problems before position prediction. Larger savings therefore require suppressing the churn, through finer reuse that survives a flip or through schedules that settle the late phase (Appendix G), rather than skipping more of the updates the churn produces.

These configurations also frame the comparison with prior positive DTO reports. Our negative numbers are measured for the uniform default at $\beta = 10^{-3}$ and $T = 60$. The stabilized variants of Section 2 target the axes quantified here, deciding which updates can be reused, which positions receive updates and which updates are accepted. Since the uniform schedule spends the majority of its budget past the point at which our data show updates stop converting, the measurements bound the uniform default rather than contradict stabilized results, and whether the stabilized configurations convert the post-transient trajectory into gains is not measured here. The direct test is a stabilized pipeline run at its recommended coupling, acceptance and gating under this protocol, which we leave to future work.

G Relation to Prior Work

Our measurements echo several existing lines and give them a quantification they previously lacked. Like adaptive computation time (Graves, 2016) and per-token depth routing (Raposo et al., 2024), we ask where and how much computation is enough. The difference is that those works decide with a learned router or a halting head, whereas we supply the measured structure such decisions should rest on. Over sixty steps, 94–98% of positions change in fewer than 5% of updates on average, with the most active fifth absorbing at least a third of the churn. That sets an empirical upper bound on the compressible space for any position-level router.

Entropy-triggered interventions take a complementary route. GGRO (Lin & Zhang, 2026) monitors token-level entropy during decoding and injects reward-gradient nudges into high-uncertainty regions only, acting on the uncertainty factor that Proposition 2 identifies as a driver of flipping. Our offline measurements bear on the choice among such rules. An entropy-only gate scores close to the fitted predictor (0.813 against 0.817, Table 3), so the measured sparsity, rather than the particular selective rule, carries most of the signal. The difference between the two gates lies in what they cost to obtain. An entropy gate needs no labels, while the fitted predictor is trained on flip labels recorded over the training problems. We keep the fitted predictor so that the measured saving reflects the stronger of the two. End-to-end comparisons against alternative selective schemes under this protocol, including entropy-triggered and token-wise gating variants, are left to future work.

The residual churn is a target in its own right. Over the last ten updates, 80.1% of the flips on Qwen and 78.3% on GSM8K return a position to a token it previously held. Its volume does not predict the late-stage damage across families, which tracks the drift of the hard assignment instead (Appendix E), and three remedies act on the churn directly. Annealing the step size raises the bar in Eq. (3), since a flip requires $\eta \cdot (g_{j^*} - g_k)$ to clear the logit margin. Locking positions whose hard assignment has been stable for k consecutive updates extends position freezing beyond the pre-optimization features, without labels. Accepting a step only when the hard-decoded objective does not decrease would price each reversal directly. None of the three is tested here, and whether any settles the churn without disturbing the trajectory is left to future work.

Consistent with classical early stopping (Duvenaud et al., 2016), we find the effective budget concentrated in the first few steps. The key difference is that residual churn never reaches zero (Section 4.2.1), so halting criteria premised on convergence underestimate the needed budget, which is why Section 4.3.3 grounds stopping in objective conversion. The quality side is analogous. Accuracy declining past an optimal budget matches what the overthinking literature (Zhou et al., 2026; Srivastava et al., 2026) reports for test-time compute, though what is being scaled here is not reasoning-chain length or sample count but the number of updates inside a single DTO run, and the degradation is explained by drift of the hard assignment rather than by generated redundancy. Compared with work on test-time compute allocation (Snell et al., 2024), which allocates samples or

candidates in the zeroth-order regime, our results show the same allocation problem arising *inside* a single first-order optimization, and that measured sparsity is itself the direct basis for position-level allocation.