

Length Value Model: Scalable Value Pretraining for Token-Level Length Modeling

Zhen Zhang^{1,*} Changyi Yang^{2,4} Zijie Xia⁴ Zhen Yang⁵ Chengzhi Liu¹
 Zhaotiao Weng¹ Yepeng Liu¹ Haobo Chen¹ Jin Pan^{3,4} Chenyang Zhao⁴
 Yuheng Bu¹ Alkesh Patel⁵ Zhe Gan⁵ Xin Eric Wang^{1,*}

¹University of California, Santa Barbara ²Carnegie Mellon University

³University of Wisconsin–Madison ⁴LMSYS Org ⁵Apple Inc.

Abstract

Tokens are the fundamental units of computation in modern autoregressive models, and generation length directly influences both inference cost and reasoning performance. Despite its importance, existing approaches model length primarily at the coarse sequence level. We introduce the Length Value Model (LenVM), a token-level framework that estimates the remaining generation length at every decoding step. By formulating length modeling as a value estimation problem and assigning a constant negative reward to each generated token, LenVM predicts a bounded, discounted return that is a monotone proxy for the remaining generation horizon. This value formulation provides annotation-free, dense, unbiased, and scalable supervision. Experiments on LLMs and VLMs show that LenVM supports exact control, continuous performance–efficiency steering, length prediction, and interpretation. On LIFEbench-token, it raises the exact-length score of Qwen2.5-7B-Instruct from 30.9 to 64.8 in one pass; combined with eight-round LCG, it further raises the score from 71.3 to 83.6 while reducing average absolute deviation from 14% to 6%. On GSM8K near 200 tokens, LenVM retains about 63% Pass@1 versus 6% for a hard token budget, and it outperforms budget-aware prompting and EOS calibration at matched lengths on MATH500. Prompt-boundary prediction improves consistently with scale, reaching MREs of 9.8%, 14.9%, and 17.1% on math, code, and instruction following at 32B; against entropy-guided predictors, LenVM reduces prompt MRE from 32.56 to 26.90 and progressive MRE from 26.71 to 15.10. Its token-level values also identify pivots between longer and shorter trajectories. Results demonstrate that LenVM supports a broad range of applications, including length control, prediction, and interpretation of generation dynamics. They suggest that generation length can be effectively modeled as a token-level value signal, highlighting the potential of LenVM as a general framework for length modeling and as a length-specific value signal that supports future RL training.

Code: [UCSB-AI/Length-Value-Model](https://github.com/UCSB-AI/Length-Value-Model)

1 Introduction

Token generation is the core computational process in modern AI systems. Large language models (LLMs), vision-language models (VLMs), and specialized models for code, mathematics, and agentic tasks all produce outputs sequentially. The number of generated tokens therefore forms a fundamental unit of inference-time computation. It influences both *performance*, since additional tokens can enable more reasoning in challenging settings (Snell et al., 2024), and *cost*, since each token incurs compute, KV-cache memory, latency, and energy (Kwon et al., 2023; Pope et al., 2022). Despite its importance, existing approaches to length modeling and control remain coarse-grained. Sequence-level penalties during

*Correspondence: zhen.zhang@ucsb.edu, ericxwang@ucsb.edu

training (Team et al., 2025), prompt-based instructions at inference (Xiao et al., 2026a), and pre-decode predictors (Zhang et al., 2025a) all operate at the level of whole sequences or fixed pre-generation decisions, rather than the token-level dynamics of generation itself. What is missing is a token-level model of remaining generation length.

To address this gap, we propose the **Length Value Model (LenVM)**, a token-level formulation of generation length modeling. At each decoding step, LenVM predicts a scalar value for the current decoding state. Conceptually, the quantity of interest is the remaining generation length, which depends on how much future generation is still to be realized. We model this quantity through a value formulation: by assigning a constant negative reward to each generated token and discounting future steps by γ , we define a discounted return that serves as a bounded, monotone proxy for the remaining generation horizon. LenVM is trained to predict this return, yielding a principled reduction from token-level length modeling to value estimation and placing the problem in a standard value-function framework. In this view, LenVM evaluates a decoding state by its remaining distance to completion rather than by task utility. This construction also maps highly variable raw lengths into a bounded target space, improving conditioning when generation lengths range from a few tokens to over 32k. The resulting value signal can be used at inference time for length estimation and control, and at training time as a length-aware value baseline.

A central advantage of LenVM is that it naturally supports **scalable value pretraining**. The resulting supervision signal has four favorable properties. First, it is **annotation-free**: training targets are constructed automatically from sampled completions, without any additional human labeling or reward modeling. Second, it is **dense**: every token position contributes a target, rather than only one target per completed response. Third, it is **unbiased** for the policy-conditioned value target: under a fixed rollout policy, the realized return is an unbiased Monte Carlo sample of the target value and introduces no additional annotation or reward-model noise. Fourth, it is **scalable**: supervision grows naturally with both prompt count and the number of completions sampled per prompt, while remaining cheap to construct and well matched to increasing model size. Together, these properties make LenVM well suited to large-scale value pretraining, enabling high-quality token-level supervision without the annotation bottlenecks of conventional reward and value modeling.

We evaluate whether LenVM learns a meaningful token-level signal of remaining generation length. At inference time, LenVM exposes a smooth **performance-efficiency trade-off**: exponentially tilting the token distribution with predicted values traces a Pareto frontier between response quality and generation length. Comparisons with budget-aware prompting, EOS calibration, and a length-penalized RLVR model further test this trade-off at matched generation lengths (§ 3.3). The same signal also supports explicit **length control**, where LenVM-guided decoding improves adherence to Equal To, At Most, and At Least constraints and complements iterative black-box control on LIFE Bench-token (Zhang et al., 2025b) (§ 3.2). In addition, LenVM provides useful **length prediction** from both the prompt boundary and progressive prefix states, across math, code, and instruction-following domains and against specialized entropy-guided predictors (§ 3.4). Its token-level values further offer an interpretable view of generation dynamics, revealing where trajectories shift between shorter and longer reasoning regimes (§ 4). Beyond inference, LenVM admits a natural interpretation as a **length-specific value baseline** in PPO-style RL (Schulman et al., 2017), while its value-learning objective improves consistently with model and data scale (§ 3.5). Finally, packed scoring and entropy skipping substantially reduce **inference overhead** (§ 3.6). Together, these results establish LenVM as a principled and scalable framework for token-level length modeling, while leaving empirical RL fine-tuning to future work.

2 Length Value Model

We propose the **Length Value Model (LenVM)**, a token-level value function over decoding states for modeling generation length. As shown in Figure 1, LenVM treats autoregressive generation as an episodic process and models remaining generation length through a discounted return induced by a constant per-token cost. Rather than predicting raw remaining length directly, it estimates a bounded and monotone transform of the remaining

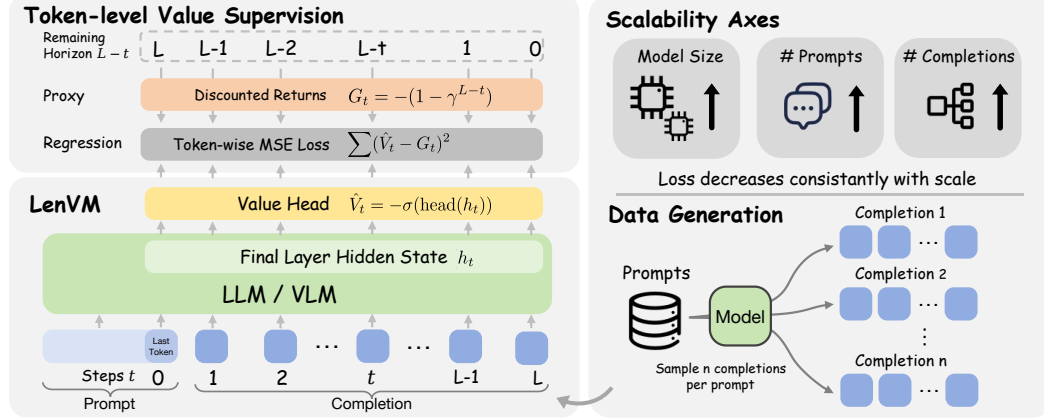


Figure 1: **Length Value Model (LenVM) architecture and Pipeline.** At each decoding step, LenVM attaches a scalar value head to the final-layer hidden state of an LLM or VLM to predict the value $V_t \in (-1, 0)$. LenVM enables annotation-free and scalable value pretraining across model size, prompt count, and number of completions.

generation horizon. Starting from the last prompt token, LenVM outputs a scalar value at each decoding step. We first define the return construction (§2.1), then summarize its key properties, and finally describe the prediction head (§2.2) and training objective (§2.3).

2.1 Modeling Length as a Discounted Return

Let s_t denote the decoding state after the prompt and the first t generated tokens, with s_0 corresponding to the last prompt token. For a sampled completion of generated length L , decoding steps are indexed by $t \in \{0, \dots, L\}$, where $t = L$ is the EOS step. Our goal is to estimate, from each prefix state s_t , how much generation horizon remains under the rollout policy that produced the sampled trajectory.

We assign a constant negative reward at each non-terminal decoding step:

$$r_t = -(1 - \gamma), \quad t = 0, \dots, L-1, \quad (1)$$

where $\gamma \in (0, 1)$ is the discount factor, and define $r_L = 0$. The factor $(1 - \gamma)$ is introduced purely for normalization, so the resulting return lies in a fixed range compatible with a sigmoid-bounded output. The discounted return from step t is

$$G_t \triangleq \sum_{i=0}^{L-t} \gamma^i r_{t+i} = -(1 - \gamma^{L-t}). \quad (2)$$

Thus, for each non-terminal step, $G_t \in (-1, 0)$ and is a strictly monotone function of the remaining generation length $L - t$: states closer to termination have values nearer to 0, while states with longer continuations have values closer to -1 . Compared with raw length, this bounded transform preserves ordering while improving target scale and conditioning. It also satisfies the Bellman recursion

$$G_t = r_t + \gamma G_{t+1}, \quad (3)$$

placing token-level length modeling in a standard value-learning framework.

Key properties of the target. The return construction has three useful properties. First, it is **bounded**: $G_t \in (-1, 0)$ for all non-terminal steps, avoiding the large dynamic range of raw remaining length. Second, it is **monotone**: the ordering over remaining horizons is preserved exactly, so shorter continuations always map to values closer to 0. Third, it is **Bellman-consistent**: the target obeys a one-step recursion, which places token-level length modeling in the same formal framework as standard discounted value estimation. The discount factor γ controls how aggressively long horizons are compressed. Larger γ preserves more resolution over long continuations, while smaller γ concentrates resolution

near termination. Equivalently, γ determines the logarithmic scale on which remaining length is represented. For a fixed model-induced decoding policy π , the corresponding length value function is

$$V^\pi(s_t) = \mathbb{E}_\pi[G_t \mid s_t]. \quad (4)$$

In practice, we sample trajectories from a fixed rollout policy and regress to their realized returns as Monte Carlo supervision. When applied to a realized return, the transform can be inverted exactly to recover the realized remaining token count:

$$l = \frac{\log(1 + G_t)}{\log \gamma}. \quad (5)$$

When applied to a predicted conditional mean value, however, this inverse yields a proxy-derived length estimate rather than, in general, the conditional expectation of raw remaining length. We discuss this distinction in Appendix I.

This formulation is (1) **annotation-free**, since targets are computed directly from observed completion lengths; (2) **dense**, since every non-terminal step contributes a regression target; (3) **unbiased** for the policy-conditioned value target, since under a fixed rollout policy the realized return is an unbiased Monte Carlo sample of $V^\pi(s_t)$ and is computed deterministically from the sampled completion, introducing no additional annotation or reward-model noise; and (4) **scalable**, since multiple completions can be sampled per prompt to automatically construct many supervised trajectories.

2.2 LenVM Head

LenVM is built on top of LLMs or VLMs by attaching a scalar value head to the final-layer hidden state at each decoding step:

$$z_t = \text{head}(h_t). \quad (6)$$

In this work, we instantiate $\text{head}(\cdot)$ as a two-layer MLP with SiLU activation,

$$z_t = W_2 \text{SiLU}(W_1 h_t + b_1) + b_2, \quad (7)$$

followed by

$$V_\theta(s_t) = -\sigma(z_t), \quad (8)$$

which ensures $V_\theta(s_t) \in (-1, 0)$.

2.3 Training Objective

For a minibatch of N prompt-completion trajectories, where the n -th trajectory has generated length $L^{(n)}$, we optimize the token-averaged mean squared error

$$\mathcal{L}_{\text{len}} = \frac{\sum_{n=1}^N \sum_{t=0}^{L^{(n)}-1} (V_\theta(s_t^{(n)}) - G_t^{(n)})^2}{\sum_{n=1}^N L^{(n)}}, \quad (9)$$

where

$$G_t^{(n)} = -(1 - \gamma^{L^{(n)}-t}). \quad (10)$$

Here the trajectories are sampled from a fixed generator checkpoint and decoding policy, and $G_t^{(n)}$ is computed exactly from each realized completion. This objective therefore corresponds to Monte Carlo regression with dense token-level supervision over the rollout state distribution induced by that policy.

More generally, one may consider weighted variants of the same objective, where each token-level squared error is multiplied by a weight. In our setting, we avoid weights that depend on future rollout outcomes relative to the current decoding step, since such weights change the effective regression objective. Trajectory-level averaging (token-mean then sequence-mean) is one such case, as it induces a per-token weight inversely proportional to the realized completion length. We discuss this issue in Appendix H.

Table 1: Datasets used to train general LenVMs.

Domain	Dataset	Scale
Code	OpenCodeReasoning-2 (Ahmad et al., 2025) (Python)	1.42M
Instruction Following	WildChat (Zhao et al., 2024)	529k
Math	DeepMath-103K (He et al., 2025)	103k

Relation to GAE. LenVM also admits GAE (Schulman et al., 2015). With $\delta_t = r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)$, one can construct bootstrapped targets from temporal-difference residuals. This provides an optional bootstrapped variant of the same value-learning problem. In our experiments, however, $\lambda = 1$ performs best. This is consistent with the task structure: supervision is dense and directly recoverable from completed sequences, so full-return regression is already available without additional bootstrapping assumptions.

3 Experiments: Validating LenVM as a Token-Level Length Signal

In this section, we evaluate at inference time how well LenVM captures the token-level signal of remaining generation length. Our experiments ask three questions: (i) Can LenVM support inference-time control without modifying the base generator? (ii) Can it predict generation length from the prompt boundary? (iii) Does its value-learning objective scale with model and data? We answer these questions through four evaluations: (1) **length-controlled generation** (§3.2), (2) **performance–efficiency trade-off** (§3.3), (3) **generation length prediction** (§3.4), and (4) **scalability of LenVM** (§3.5).

Applying LenVM adds forward passes; Section 3.6 reports the resulting latency and an entropy-skipping optimization.

3.1 Experimental Setup

General LenVM. We train LenVMs on a multi-domain mixture of math, code, and instruction-following data rather than training separate domain-specific models. Table 1 summarizes the training mixture. For each prompt, we sample multiple completions and convert them into dense per-token regression targets using the discounted return in §2.

Models. For Qwen2.5 experiments (both LLMs and VLMs), we initialize LenVM from the **Qwen2.5-Instruct** family (Yang et al., 2024). For Qwen3 experiments, we initialize LenVM from **Qwen3-Base** models (Yang et al., 2025).

Further experimental details appear in Appendix B.

3.2 Application 1: Length-Controlled Generation

We first test inference-time control on **LIFEBench-token**, our token-based adaptation of LIFEBench (Zhang et al., 2025b). It contains 360 bilingual instances, three constraints (*Equal To*, *At Most*, and *At Least*), and targets from 32 to 1024 tokens. We retain the tasks, target grid, and metrics but replace word or character counts with model-specific token counts, so scores are not directly comparable with the original leaderboard. We report **Length Score (LS $\uparrow$)** for all settings and **Length Deviation (LD $\downarrow$)** for *Equal To*; Appendix C gives full definitions.

Exact-control baselines. We compare with Prompt Best-of- N (sample and select), CAPEL (prompted countdown) (Xie & yi Lee, 2025), Dynamic Feedback (token-count feedback between segments) (Xiao et al., 2026a), and LCG (iterative revision with length and quality acceptance) (Gu et al., 2025). LCG + LenVM uses guided decoding within each proposal. One call is one generation or refinement pass; Appendix B gives the implementations.

Table 2: **Length control on LIFEbench-token**. LS measures adherence in $[0, 100]$ (higher is better; 100 is perfect). Equal LD is mean absolute relative deviation (lower is better). N/A marks methods evaluated only for exact control.

Model or method	Calls	Equal LD ↓	Equal LS ↑	At Most LS ↑	At Least LS ↑
<i>Closed-source benchmark models</i>					
GPT-4o	1	74%	35.5	77.9	98.5
GPT-5.4	1	135%	37.4	65.4	98.9
GPT-5.4-thinking	1	131%	47.8	72.7	98.9
Claude-Sonnet-4-6	1	105%	34.1	62.9	100.0
Claude-Sonnet-4-6-thinking	1	124%	51.3	69.3	100.0
Claude-Opus-4-6	1	66%	35.5	51.5	100.0
Claude-Opus-4-6-thinking	1	87%	<u>53.2</u>	67.4	100.0
Gemini-3-Flash-Preview	1	123%	40.3	57.3	99.6
Gemini-3.1-Pro-Preview	1	91%	49.3	70.7	100.0
<i>Open benchmark models, LD</i>					
Qwen2.5-3B-Instruct	1	83%	25.6	92.1	94.6
+ LenVM (1.5B)	1	56%	62.6	93.0	93.1
Qwen2.5-7B-Instruct	1	71%	30.9	98.5	89.1
+ LenVM (1.5B)	1	44%	64.8	96.1	99.5
Qwen3-30B-A3B-Instruct	1	90%	36.8	87.0	99.3
+ LenVM (1.7B)	1	57%	67.2	99.4	99.8
<i>Exact-control methods on Qwen2.5-7B-Instruct</i>					
CAPEL	1	141%	28.8	N/A	N/A
Prompt Best-of-8	8	29%	58.1	N/A	N/A
LenVM	1	44%	64.8	N/A	N/A
Dynamic Feedback	8	16%	66.6	N/A	N/A
LCG	8	14%	71.3	N/A	N/A
LCG + LenVM	8	6%	83.6	N/A	N/A

Hard-constraint decoding with LenVM. At each step, we first restrict the vocabulary to a candidate set $\mathcal{V}_t$ using standard truncation strategies, then score each candidate token with LenVM on the next state $\hat{v}(x) = \hat{v}_\phi(s_t \oplus x)$. We then select tokens using:

- **Equal To:** First convert the remaining target length $L - t$ into value space v_t^* and select $\arg \min_{x \in \mathcal{V}_t} |\hat{v}(x) - v_t^*|$, favoring candidates whose predicted value is closest to the target v_t^* .
- **At Least:** $\arg \min_{x \in \mathcal{V}_t} \hat{v}(x)$, favoring more negative values and hence longer continuations.
- **At Most:** $\arg \max_{x \in \mathcal{V}_t} \hat{v}(x)$, favoring values closer to 0 and hence earlier termination.

Results. Table 2 shows that LenVM substantially improves constraint following, especially in the *Equal To* setting. For example, on Qwen2.5-7B-Instruct, LenVM improves Length Score from 30.9 to 64.8 and reduces deviation from 71% to 44%. The closed-source models obtain Equal-To deviations between 66% and 135% and scores below 54 in this token-based setting. Against prompt Best-of- N , CAPEL (Xie & yi Lee, 2025), Dynamic Feedback (Xiao et al., 2026a), and LCG (Gu et al., 2025), the key result is complementarity: adding LenVM to eight-call LCG raises the score from 71.3 to 83.6 and reduces average absolute deviation from 14.0% to 5.8%. Appendix D reports all two-, four-, and eight-call results.

3.3 Performance–Efficiency Trade-off

We next test whether LenVM supports *continuous* control of the performance–efficiency trade-off without modifying the base generator. Instead of enforcing a hard target length, we reweight the next-token distribution so that tokens predicted to lead to shorter continuations become more likely.

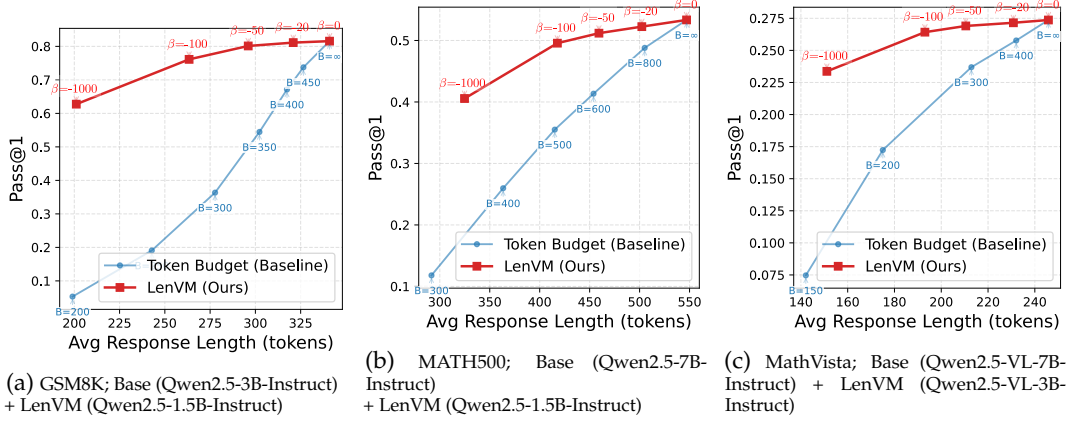


Figure 2: **Performance (Pass@1) versus average generation length.** LenVM-guided exponential tilting (red) consistently outperforms a hard token-budget baseline (blue). The x-axis reports the average *truncated* response length. Since the base generator is unchanged, the result shows that the model already has the ability to solve problems with shorter generations, and LenVM helps uncover those trajectories by reweighting token choices. Models in parentheses represent the initialization checkpoints.

Value-guided exponential tilting. Let p be the base model distribution over candidate set $\mathcal{V}_t$. We define

$$\min_{p'} \mathbb{E}_{p'}[\hat{v}(x)] - \frac{1}{\beta} D_{\text{KL}}(p' \| p), \quad (11)$$

whose solution is

$$p'(x) = \frac{p(x) \exp(\beta \hat{v}(x))}{\sum_{x' \in \mathcal{V}_t} p(x') \exp(\beta \hat{v}(x'))}. \quad (12)$$

The derivation is provided in Appendix J. Here $\beta < 0$ controls the steering strength. When $\beta = 0$, decoding reduces to the original model. As β becomes more negative, generation is increasingly biased toward shorter trajectories.

We evaluate on GSM8K (Cobbe et al., 2021), MATH500 (Lightman et al., 2023), and MathVista (Lu et al., 2023). As a baseline, we use a hard token budget that truncates generation once the token count exceeds threshold B and marks the example as incorrect. For fair comparison, the x-axis reports the average *truncated* generation length for both methods.

Figure 2 shows that LenVM-guided decoding consistently achieves a better performance-efficiency frontier than hard truncation. At the same average truncated length, LenVM retains much higher Pass@1. For example, on GSM8K with Qwen2.5-3B-Instruct, at roughly 200 tokens the hard budget baseline achieves about 6% Pass@1, whereas LenVM maintains about 63%. Because the base generator is never modified, this result indicates that the model itself already contains shorter successful reasoning paths. LenVM simply biases decoding toward them. Moreover, varying β changes the frontier smoothly, confirming that LenVM provides a continuous knob for balancing performance and token budget.

Stronger matched-length baselines. We also compare with budget-aware prompting, which requests and enforces a token cap, and EOS calibration, which shifts the EOS logit. At matched average lengths, LenVM improves task accuracy on MATH500 and GSM8K, showing that its shorter outputs preserve answer quality. Full results and protocols appear in Appendices D and B.

Comparison with a length-penalized RLVR model. On AIME2025, LenVM applied to Qwen3-30B-A3B-Instruct-2507 reaches a favorable matched-length point relative to ART, an RLVR model with a length penalty (Wu et al., 2026), and can further steer ART itself. This establishes compatibility rather than uniform superiority across training protocols; Appendix D reports the results.

Table 3: **Generation length prediction.** Panel (a) reports prompt-boundary Mean Relative Error (MRE) across scales and domains; IF is instruction following. Panel (b) compares specialized predictors. MAE is Mean Absolute Error, ρ is Spearman correlation, and W128 is Within-128 accuracy.

(a) Scaling across domains				(b) Specialized baselines					
Model size	MRE ↓			Method	Mode	MRE ↓	MAE ↓	ρ ↑	W128 ↑
	Math	Code	IF						
1.5B	17.0%	29.0%	33.0%	EGTP	Prompt	32.56	228.31	0.344	35.34
3B	13.6%	24.0%	27.2%	LenVM	Prompt	26.90	194.39	0.575	42.65
7B	11.0%	19.5%	23.0%	PLP	Prefix	26.71	218.67	0.746	30.74
14B	10.4%	17.0%	19.8%	LenVM	Prefix	15.10	155.56	0.783	60.70
32B	9.8%	14.9%	17.1%						

Policy dependence. LenVM is policy-conditioned. Our steering stays anchored to candidates from the base generator and evaluates nearby decoding shifts, not arbitrary cross-policy calibration; Appendix B discusses this scope.

3.4 Application 3: Generation Length Prediction

Beyond control, we test whether LenVM can predict generation length from the prompt boundary, i.e., from state s_0 before any response token is emitted. This is useful for scheduling, batching, and memory planning at inference time.

To evaluate prediction accuracy, we use held-out prompts from math, code, and instruction-following data. For each prompt, we sample $N = 64$ completions with lengths $L_1, \dots, L_N$. Because LenVM predicts discounted remaining horizon rather than raw token count, we evaluate in the same transformed space:

$$u(L_i) = 1 - \gamma^{L_i}, \quad \mu_u = \frac{1}{N} \sum_{i=1}^N u(L_i), \quad (13)$$

and define the return-consistent ground-truth horizon as

$$L_{\text{GT}} = \frac{\ln(1 - \mu_u)}{\ln \gamma}. \quad (14)$$

We then invert LenVM’s prediction at s_0 into a length estimate $\hat{L}$ and report Mean Relative Error (MRE):

$$\text{MRE}(\hat{L}, L_{\text{GT}}) \triangleq \mathbb{E} \left[\frac{|\hat{L} - L_{\text{GT}}|}{L_{\text{GT}}} \right]. \quad (15)$$

Further discussion of this target appears in Appendix I.

Specialized prediction baselines. We compare with EGTP, which predicts total length from entropy-weighted prompt states, and PLP, which extends this predictor to generated prefixes (Xie et al., 2026). Appendix B gives the matched training and evaluation protocol.

As shown in Table 3(a), LenVM predicts generation horizon from the prompt boundary with low relative error across all domains, and accuracy improves consistently with scale. At 32B, MRE reaches 9.8% on math, 14.9% on code, and 17.1% on instruction following. This shows that LenVM is not only useful for online control, but also captures predictive information about expected generation length before decoding begins.

Comparison with specialized predictors. Table 3(b) reports results using a frozen Qwen2.5-7B-Instruct generator. Prompt metrics cover 8192 held-out DeepMath-103K prompts, while progressive metrics cover approximately 108,000 prefix states. LenVM improves all four metrics in both settings, reducing prompt MRE from 32.56 to 26.90 and progressive MRE from 26.71 to 15.10.

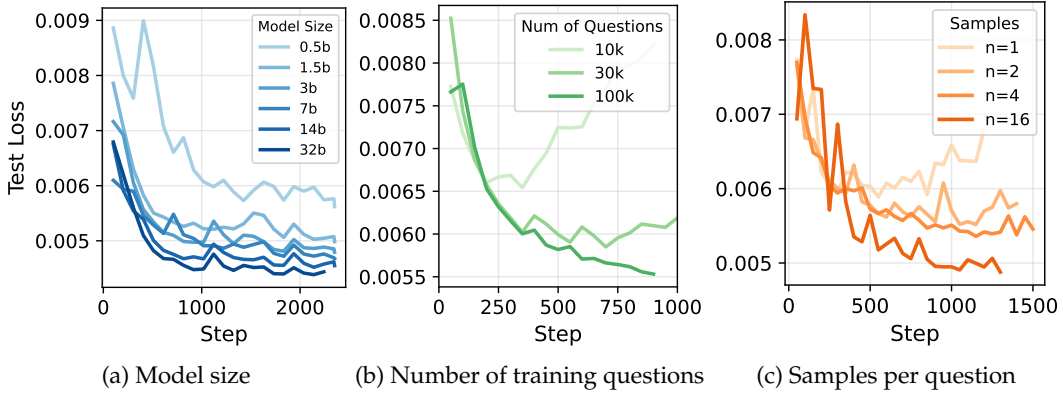


Figure 3: **LenVM scales along three axes.** Validation loss decreases consistently as we increase (a) model size, (b) number of training questions, and (c) samples per question.

3.5 Scalability of LenVM

A key advantage of LenVM is that its supervision is annotation-free, dense, and easy to scale. We therefore study scaling along three axes: **(i) model size**, **(ii) number of training questions**, and **(iii) number of samples per question**.

Figure 3 shows consistent improvements along all three axes. Larger models achieve lower validation loss, matching the gains in prompt-boundary prediction accuracy from Table 3. Increasing the number of training prompts also improves performance, indicating that LenVM benefits from broader coverage of tasks and trajectory types. Finally, increasing the number of sampled completions per prompt further reduces validation loss, showing that supervision can be scaled not only through more prompts but also through more sampled trajectories from the same prompts. Overall, these results support LenVM as a scalable value-pretraining objective.

3.6 Inference Cost

We score the generator’s top- K candidates in one packed SGLang call (Zheng et al., 2024), cache LenVM key-value states, and skip value scoring when generator entropy is below 0.5. Table 4 shows that skipping activates LenVM on 19.5% of decoding steps and reduces measured end-to-end latency from $4.55\times$ to $1.90\times$. Appendix D gives the full protocol and candidate-scoring times.

Table 4: **End-to-end decoding latency.** Active steps are decoding steps on which LenVM is evaluated.

Setting	Active steps	Latency
Vanilla decoding	0%	$1.00\times$
LenVM at every step	100%	$4.55\times$
LenVM with entropy skip	19.5%	$1.90\times$

Additional analyses. Appendix E studies target representation, batching, discount factor, numerical precision, and candidate-set size. Section 4 analyzes tokens associated with shifts toward shorter or longer predicted horizons.

Scope and limitations. Guided decoding remains $1.9\times$ slower than vanilla decoding in our setup, with hardware and serving-load sensitivity. LenVM is policy-conditioned, so calibration under unrelated generators remains open. LIFEbench-token uses token

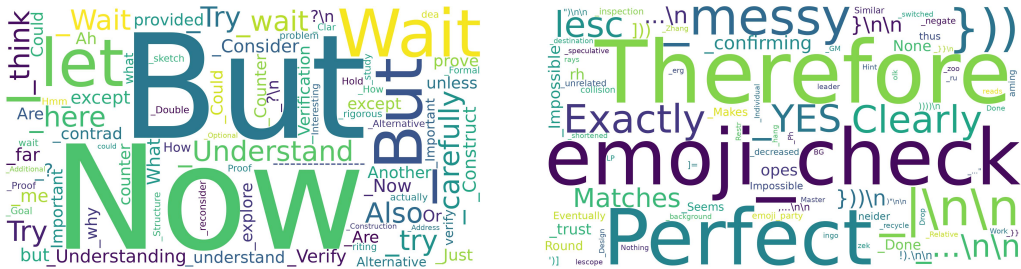
constraints and is not directly comparable with the original leaderboard. Empirical RL fine-tuning with LenVM is future work.

4 Qualitative Case Study: Length Tokens as Markers of Length Shifts

We call tokens associated with changes in LenVM’s predicted remaining horizon *length tokens*. At step t , their temporal-difference residual is

$$s_t \triangleq r_{t-1} + \gamma V_t - V_{t-1}, \quad r_{t-1} = -(1 - \gamma), \quad (16)$$

where $V_t \triangleq V_\theta(s_t)$ and γ is the training discount factor. Positive and negative s_t mark longer and shorter horizons, respectively. Figure 4a contrasts reasoning pivots such as wait, think, and try with closure tokens such as therefore, clearly, and perfect.



(a) Positive length tokens: longer-horizon shifts (b) Negative length tokens: shorter-horizon shifts

Figure 4: **Length tokens as markers of horizon shifts.** Frequent tokens with TD residual s_t above 0.01 or below -0.01 on 8k DeepMath-103K examples. Token size is proportional to frequency.

5 Conclusion

We proposed **LenVM**, a token-level value model for remaining generation length. By assigning a constant per-token reward and predicting the resulting discounted return, LenVM places length modeling in a standard value-learning framework with bounded, dense, and annotation-free supervision. Our experiments show that this simple formulation is both useful and scalable. LenVM supports precise length-controlled generation, exposes a smooth performance–efficiency trade-off without modifying the base generator, predicts expected generation horizon, and improves consistently with model scale and automatically collected supervision. Additional experiments show that LenVM complements iterative controllers, matched-length baselines, and specialized predictors, while entropy skipping reduces overhead. It also steers a length-penalized RLVR model, and the case study identifies token-level markers of horizon shifts. These results suggest that discounted length value is an effective target for token-level length modeling and a promising substrate for scalable value pretraining.

References

- Wasi Uddin Ahmad, Somshubra Majumdar, Aleksander Ficek, Sean Narenthiran, Mehrzad Samadi, Jocelyn Huang, Siddhartha Jain, Vahid Noroozi, and Boris Ginsburg. Opencodereasoning-ii: A simple test time scaling approach via self-critique. *arXiv preprint arXiv:2507.09075*, 2025.
- Adewale Akinfaderin, Shreyas Subramanian, and Akarsha Sehwal. Plan-and-write: Structure-guided length control for llms without model retraining. *ArXiv*, abs/2511.01807, 2025. URL <https://api.semanticscholar.org/CorpusID:282739780>.
- Bradley Butcher, Michael O’Keefe, and James Titchener. Precise length control for large language models. *Nat. Lang. Process. J.*, 11:100143, 2024. URL <https://api.semanticscholar.org/CorpusID:274788732>.

- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- Yifeng Ding, Hantian Ding, Shiqi Wang, Qing Sun, Varun Kumar, and Zijian Wang. Planning-aware code infilling via horizon-length prediction, 2025. URL <https://arxiv.org/abs/2410.03103>.
- Emmanuel Anaya Gonzalez, Sairam Vaidya, Kanghee Park, Ruyi Ji, Taylor Berg-Kirkpatrick, and Loris D’antoni. Constrained sampling for language models should be easy: An mcmc perspective. *ArXiv*, abs/2506.05754, 2025. URL <https://api.semanticscholar.org/CorpusID:279245064>.
- Yuxuan Gu, Wenjie Wang, Xiaocheng Feng, Weihong Zhong, Kun Zhu, Lei Huang, Ting Liu, Bing Qin, and Tat-Seng Chua. Length controlled generation for black-box llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 16878–16895, 2025. doi: 10.18653/v1/2025.acl-long.825. URL <https://aclanthology.org/2025.acl-long.825/>.
- Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, et al. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*, 2025.
- Chenghua Huang, Lu Wang, Fangkai Yang, Pu Zhao, Zhixu Li, Qingwei Lin, Dongmei Zhang, Saravan Rajmohan, and Qi Zhang. Pretrain value, not reward: Decoupled value policy optimization, 2026. URL <https://arxiv.org/abs/2502.16944>.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Haoteng Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023. URL <https://api.semanticscholar.org/CorpusID:261697361>.
- Yanhao Li, Lu Ma, Jiaran Zhang, Lexiang Tang, Wentao Zhang, and Guibo Luo. Leash: Adaptive length penalty and reward shaping for efficient large reasoning model, 2025. URL <https://arxiv.org/abs/2512.21540>.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *The twelfth international conference on learning representations*, 2023.
- Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, Yejin Choi, Jan Kautz, and Pavlo Molchanov. Dler: Doing length penalty right - incentivizing more intelligence per token via reinforcement learning, 2025. URL <https://arxiv.org/abs/2510.15110>.
- Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.
- Ning Miao, Hao Zhou, Lili Mou, Rui Yan, and Lei Li. Cgmh: Constrained sentence generation by metropolis-hastings sampling, 2018. URL <https://arxiv.org/abs/1811.10996>.
- Grzegorz Piotrowski, Mateusz Bystronowski, Mikołaj Hołysz, Jakub Binkowski, Grzegorz Chodak, and Tomasz Jan Kajdanowicz. When will the tokens end? graph-based forecasting for LLMs output length. In Jin Zhao, Mingyang Wang, and Zhu Liu (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pp. 843–848, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-254-1. doi: 10.18653/v1/2025.acl-srw.61. URL <https://aclanthology.org/2025.acl-srw.61/>.

- Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *ArXiv*, abs/2211.05102, 2022. URL <https://api.semanticscholar.org/CorpusID:253420623>.
- John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015. URL <https://api.semanticscholar.org/CorpusID:3075448>.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. URL <https://api.semanticscholar.org/CorpusID:28695052>.
- Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *ArXiv*, abs/2408.03314, 2024. URL <https://api.semanticscholar.org/CorpusID:271719990>.
- Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Feng Tang, Flood Sung, Guangda Wei, Guokun Lai, Haiqing Guo, Han Zhu, Haochen Ding, Hao-Xing Hu, Haoming Yang, Hao Zhang, Haotian Yao, Hao-Dong Zhao, Haoyu Lu, Haoze Li, Hao Yu, Hongcheng Gao, Huabin Zheng, Huan Yuan, Jia Chen, Jia-Xing Guo, Jianling Su, Jianzhou Wang, Jie Zhao, Jin Zhang, Jingyuan Liu, Junjie Yan, Junyan Wu, Li-Na Shi, Li tao Ye, Long Yu, Meng xiao Dong, Neo Y. Zhang, Ning Ma, Qi Pan, Qucheng Gong, Shaowei Liu, Shen Ma, Shu-Yan Wei, Sihan Cao, Si-Da Huang, Tao Jiang, Wei-Wei Gao, Weiming Xiong, Weiran He, Weixiao Huang, Wenhao Wu, Wen He, Xian sen Wei, Xian-Xian Jia, Xingzhe Wu, Xinran Xu, Xinxing Zu, Xinyu Zhou, Xue biao Pan, Y. Charles, Yang Li, Yan-Ling Hu, Yangyang Liu, Yanru Chen, Ye-Jia Wang, Yibo Liu, Yidao Qin, Yifeng Liu, Yingbo Yang, Yiping Bao, Yulun Du, Yuxin Wu, Yuzhi Wang, Zaida Zhou, Zhaoji Wang, Zhaowei Li, Zhengxin Zhu, Zheng Zhang, Zhexu Wang, Zhilin Yang, Zhiqi Huang, Zihao Huang, Ziya Xu, and Zonghan Yang. Kimi k1.5: Scaling reinforcement learning with llms. *ArXiv*, abs/2501.12599, 2025. URL <https://api.semanticscholar.org/CorpusID:275789974>.
- Taiqiang Wu, Zenan Zu, Bo Zhou, and Ngai Wong. The art of efficient reasoning: Data, reward, and optimization. *arXiv preprint arXiv:2602.20945*, 2026.
- Violet Xiang, Chase Blagden, Rafael Rafailov, Nathan Lile, Sang Truong, Chelsea Finn, and Nick Haber. Just enough thinking: Efficient reasoning with adaptive length penalties reinforcement learning, 2025. URL <https://arxiv.org/abs/2506.05256>.
- Meiman Xiao, Ante Wang, Qingguo Hu, Zhongjian Miao, Huangjun Shen, Longyue Wang, Weihua Luo, and Jinsong Su. Can llms track their output length? a dynamic feedback mechanism for precise length regulation. *ArXiv*, abs/2601.01768, 2026a. URL <https://api.semanticscholar.org/CorpusID:284486970>.
- Meiman Xiao, Ante Wang, Qingguo Hu, Zhongjian Miao, Huangjun Shen, Longyue Wang, Weihua Luo, and Jinsong Su. Can llms track their output length? a dynamic feedback mechanism for precise length regulation, 2026b. URL <https://arxiv.org/abs/2601.01768>.
- Huanyi Xie, Yubin Chen, Liangyu Wang, Lijie Hu, and Di Wang. Predicting llm output length via entropy-guided representations. *ArXiv*, abs/2602.11812, 2026. URL <https://api.semanticscholar.org/CorpusID:285540500>.
- Juncheng Xie and Hung yi Lee. Prompt-based one-shot exact length-controlled generation with llms. *ArXiv*, abs/2508.13805, 2025. URL <https://api.semanticscholar.org/CorpusID:280686321>.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu,

- Jianwei Zhang, Jianxin Yang, Jiabin Yang, Jingren Zhou, Jingren Zhou, Junyan Lin, Kai Dang, Keqin Bao, Ke-Pei Yang, Le Yu, Li-Chun Deng, Mei Li, Min Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shi-Qiang Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. 2025. URL <https://api.semanticscholar.org/CorpusID:278602855>.
- Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiabin Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yinyang Wan, Yuqi Liu, Zeyu Cui, Zhenru Zhang, Zihan Qiu, Shanghaoran Quan, and Zekun Wang. Qwen2.5 technical report. *ArXiv*, abs/2412.15115, 2024. URL <https://api.semanticscholar.org/CorpusID:274859421>.
- Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, Xiangpeng Wei, Xiangyu Yu, Gaohong Liu, Juncai Liu, Lingjun Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Ru Zhang, Xin Liu, Mingxuan Wang, Yonghui Wu, and Lin Yan. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks, 2025. URL <https://arxiv.org/abs/2504.05118>.
- Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. Adaptthink: Reasoning models can learn when to think. *ArXiv*, abs/2505.13417, 2025a. URL <https://api.semanticscholar.org/CorpusID:278769267>.
- Wei Zhang, Zhenhong Zhou, Kun Wang, Junfeng Fang, Yuanhe Zhang, Rui Wang, Ge Zhang, Xavier Li, Li Sun, Lingjuan Lyu, et al. Lifebench: Evaluating length instruction following in large language models. *arXiv preprint arXiv:2505.16234*, 2025b.
- Yi-Kai Zhang, Zhiyuan Yao, Hongyan Hao, Yueqing Sun, Qi Gu, Hui Su, Xunliang Cai, De-Chuan Zhan, and Han-Jia Ye. v_0 : A generalist value model for any policy at state zero, 2026. URL <https://arxiv.org/abs/2602.03584>.
- Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. *arXiv preprint arXiv:2405.01470*, 2024.
- Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, 2024.
- Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. Response length perception and sequence scheduling: An llm-empowered llm inference pipeline, 2023. URL <https://arxiv.org/abs/2305.13144>.

A Related Work

A.1 Length-Controlled Generation

Length control has been explored through prompting, fine-tuning, and decoding-time constraints. Prompt-based methods can be attractive for black-box models: for example, Xie and Lee propose a one-shot countdown prompt that improves strict compliance without changing model weights (Xie & yi Lee, 2025). Recent prompt-only methods also incorporate explicit structure and counting to regulate length without retraining; Plan-and-Write uses a plan-first scaffold and word-counting guidance to improve adherence in summarization tasks (Akinfaderin et al., 2025). For black-box settings where fine-tuning is not possible, Gu et al. (2025) propose an iterative sampling framework that combines Metropolis–Hastings

with importance-sampling acceleration to achieve length-constrained generation without modifying model parameters. More broadly, constrained generation has a long history of MCMC-style inference-time methods; CGMH applies Metropolis–Hastings moves to satisfy lexical constraints without parallel training data (Miao et al., 2018), and recent work revisits MCMC constrained sampling with stronger distributional guarantees (Gonzalez et al., 2025). Training-based approaches can yield more precise control by modifying the model’s internal length representation; Butcher et al. (2024) introduce length-difference positional encoding (LDPE) and fine-tuning to encourage coherent termination near the target length, while Xie et al. (2026) propose Hansel, which uses periodically outputted hidden special tokens to track the remaining target length during finetuning. In contrast to approaches that embed length information into the generator, our method trains a *separate* token-level horizon estimator and uses it as a decoding-time control signal, enabling length control without retraining the base LLM.

A.2 Output Length Prediction

Predicting completion length is important for both serving efficiency and stochastic sampling, and can be approached either statically (before decoding) or progressively (during decoding). Recent work shows that hidden representations and entropy signals can provide accurate, low-overhead length prediction; in particular, Xie et al. (2026) propose entropy-guided token pooling for static prediction and progressive length prediction (PLP) for online remaining-length estimation. Beyond entropy-guided methods, Piotrowski et al. (2025) study forecasting remaining output length from frozen layerwise hidden states, exploring aggregation and graph-based regressors. In code generation, horizon-length prediction has also been used to improve fill-in-the-middle planning: Ding et al. (2025) propose a horizon-length prediction objective to better align infills with distant right context. Relatedly, Xiao et al. (2026b) examine whether LLMs can self-track output length and propose a dynamic feedback mechanism that adjusts generation online to meet length targets without retraining. Our LVM is also an online predictor, but is trained as a standalone value model with bounded regression targets and is designed to provide a stable per-token signal across diverse task families and length scales. Variable output lengths complicate batching and can waste compute through padding and fragmentation; length prediction can improve schedulers and load balancing. Zheng et al. (2023) study response length perception and sequence scheduling to improve inference throughput.

A.3 RL Framing and Reward Shaping of Generation Length

Our formulation views length regularization as a dense per-step penalty (i.e., negative reward) with an associated discounted return, making remaining horizon a value-function quantity. This makes LVM naturally compatible with PPO-style training as a length-specific value baseline and advantage component (Section F), while in this paper we focus on inference-time decoding control and analysis. More broadly, recent RLHF work argues for value-centric formulations: DVPO pretrains a global value model on preference data and uses it as a frozen critic for policy optimization (Huang et al., 2026), and V_0 learns a generalist value estimator at the initial prompt for policy-agnostic routing and resource allocation (Zhang et al., 2026). Recent work on efficient reasoning also studies how to adapt length penalties during RL to balance accuracy and conciseness (Xiang et al., 2025; Liu et al., 2025; Li et al., 2025). VAPO further targets stable and efficient value-based RL for long-CoT reasoning by addressing value bias, heterogeneous sequence lengths, and sparse rewards (Yue et al., 2025).

B Additional Experimental Details

We train LenVM using LlamaFactory. The discount factor γ is set to 0.997 for **Qwen2.5-Instruct** and **Qwen2.5-VL-Instruct**, and to 0.9998 for **Qwen3-Instruct**. We set $\lambda = 1$ for GAE.

For the ablation and scalability experiments, we sample 100k examples from each dataset, except for math, where we use all 95k available examples. For the Length-Controlled Generation and Performance–Efficiency Trade-off experiments, we use the full training sets of all three datasets. We randomly sample 8k examples for validation during training. Unless otherwise specified, data sampling uses temperature 1.0 and top-p 1.0, and we sample up to 16 completions per prompt.

We train LenVM for 2 epochs with learning rate 2×10^{-5} , batch size 1024, and BF16 precision.

In **Length-Controlled Generation**, we use temperature 1.0, top-p 0.999, and top-k 15. All baseline settings follow the original LIFE Bench paper, except that constraints are applied to token count in our LIFE Bench-token experiments. We use the official tokenizer of each model, or the token count returned by the corresponding API response, to measure generation length. Although tokenization schemes differ across models, all methods are evaluated under the same token-based constraint setting.

Exact-control baseline implementations. All exact-control experiments in Table 2 use Qwen2.5-7B-Instruct, the same LIFE Bench-token prompts and targets, and the Qwen2.5 tokenizer for both feedback and evaluation. Unless a method specifies its own decoding rule, generation uses temperature 1.0, top-p 0.999, and top-k 15. For Prompt Best-of- N , we independently sample $N \in \{2, 4, 8\}$ responses from the ordinary target-length prompt and select the response minimizing $|L(y) - L^*|$, where L^* is the target token count. For CAPEL (Xie & yi Lee, 2025), we adapt the published one-pass prompt by requesting token control and appending its countdown markers and explicit counting rules. We remove the countdown markup before computing task metrics and measure the remaining response with the Qwen2.5 tokenizer.

For Dynamic Feedback (Xiao et al., 2026a), we begin with the target-length prompt, pause at detected sentence boundaries, compute the number of generated tokens, and append a feedback string of the form `<used.tokens= $n$ >` before resuming. The prompt explains that this value is supplied by an external counter and should be used to meet the target. Each resume is a new generator call, and the two-, four-, and eight-call settings cap the total number of generation segments accordingly. For LCG (Gu et al., 2025), the first call produces a length-conditioned response. Each subsequent call receives the previous response together with its signed token-count error and is instructed to add or delete the required amount. We retain the method’s importance-guided proposal and Metropolis-Hastings acceptance rule, which favors candidates with better target-length fit while preserving the generator’s response-quality score. The reported call count includes the initial response.

Combining LCG with LenVM. LCG with LenVM keeps the same outer proposal, scoring, and acceptance procedure as LCG. The only change is that every initial-generation or refinement call uses the Equal-To LenVM rule from Section 3.2 when decoding the proposed response. Thus the outer controller still revises complete responses using observed length error, while LenVM guides token selection within each proposal toward the remaining target horizon. We use the same call budgets and $K = 15$ candidate set for the paired LCG and LCG with LenVM comparisons.

In the **Performance–Efficiency Trade-off** experiments, we use temperature 1.0 and top-p 1.0. For LenVM guided decoding, we set min-p to 0.01 to reduce the candidate set size and avoid excessively large LenVM forward batches. We sample 64 completions for each question.

Matched-length baseline implementations. Budget-aware prompting prepends the instruction Solve the problem within B tokens and sets `max.tokens =  $B$` so that the requested and enforced budgets agree. We sweep B to obtain multiple accuracy and average-length points. For EOS calibration, if z_{EOS} is the generator’s EOS logit, we decode with $z'_{\text{EOS}} = z_{\text{EOS}} + b$ and leave all other logits unchanged. Sweeping b changes the stopping tendency without changing model parameters. For each baseline point, we report the measured LenVM point with the smallest absolute difference in average output length. All

methods use Qwen2.5-7B-Instruct, identical benchmark prompts and answer extraction, temperature 1.0, and top-p 1.0.

For the RLVR comparison, we evaluate the released ART checkpoint based on Qwen3-30B-A3B-Instruct-2507 (Wu et al., 2026) on AIME2025 with the same prompts, sampling configuration, and answer checker used for its unmodified base model. Base with LenVM changes only the decoding distribution through β . ART with LenVM applies the same exponential tilting to ART without further parameter updates, so the points isolate decoding-time steering on top of the length-penalized policy.

Policy dependence. LenVM estimates $V^\pi(s_t) = \mathbb{E}_\pi[G_t | s_t]$ for the rollout policy used to construct its supervision, so we do not assume calibration under arbitrary unrelated generators or large policy shifts. In our guided decoding experiments, candidates come from the base generator and exponential tilting remains anchored to its token distribution. This limits, but does not eliminate, off-policy mismatch, and the reported curves test nearby decoding-policy shifts only.

Length-prediction baseline implementations. We implement EGTP and PLP following Xie et al. (2026) with Qwen2.5-7B-Instruct frozen. EGTP computes next-token entropy at every prompt position, applies softmax-normalized entropy weights to the corresponding hidden states, and sends the pooled representation to a lightweight length-prediction head. PLP uses the same prompt representation together with hidden states from the generated prefix and predicts the number of remaining tokens at each sampled prefix. The prediction heads are trained on DeepMath-103K training trajectories without updating the generator. We evaluate EGTP and prompt-boundary LenVM on the same 8192 held-out prompts. For progressive prediction, we sample prefix states from their held-out completions, producing approximately 108,000 matched states for PLP and LenVM. All methods use the same realized completion lengths as targets and the same MRE, MAE, Spearman correlation, and Within-128 calculations.

For the latency study, the generator is **Qwen2.5-7B-Instruct** and LenVM is initialized from **Qwen2.5-1.5B-Instruct**. Candidate next states are evaluated in one packed SGLang call (Zheng et al., 2024), and LenVM key-value states are cached across decoding steps. The entropy-skipping variant invokes LenVM only when the generator entropy is at least 0.5.

C LIFEbench-token Evaluation Details

Benchmark overview. LIFEbench (Zhang et al., 2025b) is a comprehensive benchmark designed to evaluate the ability of large language models to follow explicit length instructions across diverse tasks and length scales. It comprises 10,800 (360 data, 3 length constraints, 10 length target) instances spanning four task categories:

- **Question Answering (QA):** Open-ended questions drawn from Quora, Stack Exchange, WikiHow, and Yahoo Answers (English), and from Zhihu and WikiHow (Chinese).
- **Summarization:** Long documents (10,000–15,000 words) sourced from scientific papers, government reports, book chapters, and Wikipedia (English), and from corporate annual reports, Wikipedia, and XueXiQiangGuo (Chinese).
- **Reasoning:** Eighty problems (40 per language) generated via GPT-4o, covering six reasoning types: deductive, inductive, abductive, analogical, causal, and probabilistic.
- **Creative Generation:** Writing prompts derived from movie scripts, story corpora, and news datasets (English), and from web novels, social commentary, and WeChat public articles (Chinese).

Length constraints. The original benchmark focuses on word and character lengths. Each instance is paired with one of three constraint types (*Equal To*, *At Most*, and *At Least*) and one of ten target lengths: 16, 32, 64, 128, 256, 512, 1,024, 2,048, 4,096, and 8,192 words for English

or characters for Chinese. In LIFEbench-token, we preserve the tasks, target grid, and metrics but apply each constraint to the model-specific token count. Specifically, we prompt the model to generate a response whose total token count is equal to, at most, or at least the target. Tokens directly determine autoregressive decoding steps and key-value cache growth, but this change of unit means that our absolute scores are not directly comparable with the original word-level or character-level leaderboard.

Evaluation metrics. Two metrics are used:

- **Length Deviation (LD):** For example i ,

$$LD_i = \frac{L_{\text{output},i} - L_{\text{constraint},i}}{L_{\text{constraint},i}}, \quad (17)$$

where $L_{\text{output},i}$ is the actual output length and $L_{\text{constraint},i}$ is the target length for example i . Positive values indicate over-generation; negative values indicate under-generation. Following the official aggregation protocol for the *Equal To* setting, we report the mean of $|LD|$ over evaluated examples. The reported LD is therefore nonnegative and equals the mean absolute relative deviation.

- **Length Score (LS):** A bounded score in $[0, 100]$ derived from LD via an exponential penalty function that is adapted to each constraint type:

$$LS_i = \begin{cases} 100 \cdot e^{k_1 LD_i}, & LD_i < 0 \quad (\text{Equal To / At Least}), \\ 100 \cdot e^{-k_2 LD_i}, & LD_i > 0 \quad (\text{Equal To / At Most}), \\ 100, & \text{otherwise,} \end{cases} \quad (18)$$

with $k_1 = 5$ and $k_2 = 2$. The reported score over N examples is

$$\overline{LS} = \frac{1}{N} \sum_{i=1}^N LS_i. \quad (19)$$

A score of 100 indicates perfect adherence; lower scores reflect larger deviations.

Implementation details. Each prompt follows the template [Instruction] + [Requirement], where the instruction specifies the task and input content, and the requirement states the length constraint in natural language (e.g., “Your answer must be equal to 512 tokens.”).

D Additional Experimental Results

D.1 End-to-End Latency

For the latency study, we use Qwen2.5-7B-Instruct as the generator and Qwen2.5-1.5B as LenVM. Candidate tokens are drawn from the generator’s top- K set, so LenVM does not expand the candidate pool. They are scored in one packed SGLang call (Zheng et al., 2024), each candidate attends to the shared prefix but not to other candidates, and LenVM key-value states are cached. We skip LenVM scoring when generator entropy is below 0.5 because low-entropy steps offer little room for length guidance to change the selected token. Table 4 reports the resulting end-to-end latency. This remains a meaningful serving overhead.

D.2 Matched-Length Quality Baselines

Budget-aware prompting requests a solution within B tokens and sets `max_tokens= B`. EOS calibration instead adjusts the EOS probability to move along the accuracy and length curve. All methods in Table 5 use Qwen2.5-7B-Instruct and are compared at their nearest-length operating points. LenVM improves over both baselines on MATH500 and over budget-aware prompting on GSM8K.

Table 5: Task accuracy at matched average output lengths.

Dataset	Method	Avg. tokens	Pass@1 ↑
MATH500	Budget prompt	333	39.66
MATH500	LenVM	317	47.50
MATH500	Budget prompt	412	48.63
MATH500	LenVM	397	50.78
MATH500	EOS calibration	307	28.56
MATH500	LenVM	306	45.59
GSM8K	Budget prompt	198	81.00
GSM8K	LenVM	192	86.28
GSM8K	Budget prompt	211	86.88
GSM8K	LenVM	207	87.88

D.3 Full LIFEbench-token Comparison

Table 6 reports all tested inference budgets for prompt Best-of- N , CAPEL (Xie & yi Lee, 2025), Dynamic Feedback (Xiao et al., 2026a), LCG (Gu et al., 2025), and LCG with LenVM. A round is one generation or refinement call. The methods therefore do not have equal inference cost. Across two, four, and eight rounds, inserting LenVM into LCG consistently lowers average absolute deviation and raises the length score.

Table 6: Full LIFEbench-token comparison across inference budgets. LD is Length Deviation and is reported as the mean absolute relative deviation. Length Score is a bounded adherence score in $[0, 100]$, with 100 indicating perfect adherence. Lower LD and higher Length Score are better.

Method	Protocol	LD ↓	Score ↑
LCG + LenVM	8 rounds	5.8%	83.6
LCG + LenVM	4 rounds	8.6%	80.5
LCG + LenVM	2 rounds	10.7%	73.8
LenVM	1 round	44.0%	64.8
LCG	8 rounds	14.0%	71.3
LCG	4 rounds	15.3%	69.0
LCG	2 rounds	20.3%	60.6
Dynamic Feedback	8 rounds	15.9%	66.6
Dynamic Feedback	4 rounds	25.3%	46.5
Dynamic Feedback	2 rounds	42.9%	30.4
Prompt	Best-of-8	28.7%	58.1
Prompt	Best-of-4	38.4%	47.2
Prompt	Best-of-2	50.2%	38.4
CAPEL	1 round	140.8%	28.8

D.4 RLVR Length-Penalty Comparison

We compare LenVM with ART, which trains Qwen3-30B-A3B-Instruct-2507 using RLVR and a length penalty (Wu et al., 2026). Table 7 separates two questions. The first group compares nearby measured operating points from different protocols. Base + LenVM reaches 54.17 Pass@1 at 5611 tokens, compared with 51.25 at 5599 tokens for ART. The second group applies LenVM directly to ART. Stronger steering shortens the average response from 5599 to 4949 tokens while accuracy decreases from 51.25 to 46.25, exposing a continuous quality and length trade-off.

Table 7: **Comparison with RLVR plus a length penalty on AIME2025.** RLVR denotes reinforcement learning with verifiable rewards, β is the LenVM steering coefficient.

Method	β	Avg. tokens	Pass@1 $\uparrow$
Base	0	6595	60.00
ART	0	5599	51.25
Base + LenVM	-200	5611	54.17
Base + LenVM	-300	5422	52.92
ART	0	5599	51.25
ART + LenVM	-10	5391	49.58
ART + LenVM	-25	5255	46.67
ART + LenVM	-50	4949	46.25

D.5 Candidate-Set Size

Under hard constraints, a larger candidate pool increases the chance that a next state matches the target horizon. Under soft steering, exponential tilting remains anchored to the generator distribution, so low-probability candidates remain suppressed. Table 8 shows that increasing K from 1 to 128 raises the LIFEbench-token score from 29.3 to 72.1 and reduces deviation from 100.2% to 18.5%, although intermediate values are not strictly monotone. On MATH500, most of the change occurs between $K = 1$ and $K = 2$.

Table 8: **Sensitivity to candidate-set size.** K is the candidate-set size (top-k), LB-token denotes LIFEbench-token, and its score is the Length Score in $[0, 100]$, where higher is better. β is the LenVM steering coefficient. MATH500 uses $\beta = -100$.

K	LB-token deviation $\downarrow$	LB-token score $\uparrow$	MATH500 Pass@1 $\uparrow$	MATH500 avg. tokens
1	100.2%	29.3	55.84	636.95
2	30.6%	56.5	52.78	435.56
8	38.6%	56.1	50.72	395.08
15	26.5%	63.5	50.72	391.60
32	21.4%	69.7	50.03	392.26
128	18.5%	72.1	50.88	393.76

D.6 Packed Candidate-Scoring Time

All candidate next states are scored in one packed SGLang call (Zheng et al., 2024), with a tree-structured attention mask that lets each candidate attend to the shared prefix but not to other candidates. Table 9 reports LenVM forward time per active decoding step. Within the tested range, time stays between 32.18 and 34.16 ms as K increases from 2 to 5. This rules out a naive sequence of K independent LenVM calls in this implementation, but it does not imply that runtime is independent of K outside the tested range.

Table 9: **LenVM packed forward time per active decoding step.**

Candidate-set size K	LenVM forward time
2	32.18 ms
3	34.16 ms
5	33.49 ms

E Ablations

We ablate five design choices in LenVM: (i) **length-space representation**, (ii) **batch construction strategy**, (iii) **discount factor γ** , and (iv) **numerical precision** (FP16/BF16/FP32),

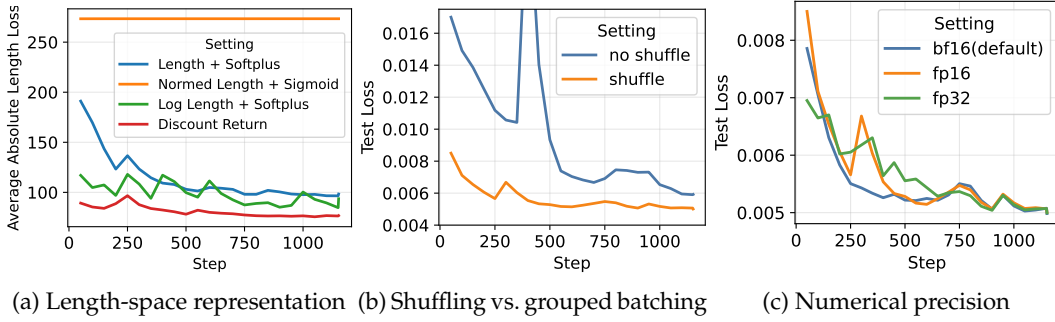


Figure 5: **Ablations on LenVM design choices.** Comparison of (a) target representation, (b) batch construction strategy, and (c) numerical precision.

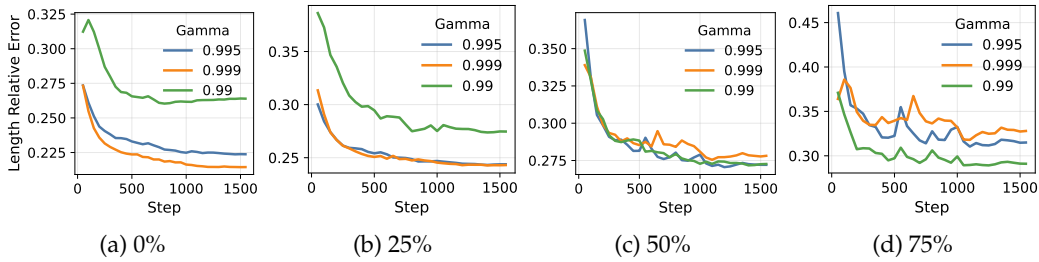


Figure 6: **Ablation on the discount factor γ .** Prediction error at different relative decoding positions. Larger γ performs better earlier in generation, while smaller γ performs better closer to termination.

and (v) **candidate-set size K** . Unless otherwise specified, all experiments keep the model, optimizer, and evaluation protocol fixed.

E.1 Length-Space Representation

We first compare four target parameterizations in Figure 5a: (1) **Length + Softplus**, (2) **Normalized Length + Sigmoid**, (3) **Log Length + Softplus**, and our proposed (4) **Discount Return + Sigmoid**. Among them, **Discount Return + Sigmoid** consistently achieves the lowest average absolute length error throughout training. Raw-length and normalized-length regression perform worse, partly because generation lengths are highly long-tailed. In particular, normalizing by the maximum length compresses the dense short-length region into a narrow interval near zero, where discrimination becomes difficult. Log-length regression is a stronger baseline and substantially improves over raw or normalized length, but still underperforms discount return. Unlike a static scale transform such as log length, discount return is aligned with the stepwise Bellman recursion of autoregressive decoding, yielding a more coherent token-level training signal. This result shows that for token-level length modeling, directly regressing raw length or normalized length is ineffective, while *the discounted-return formulation provides a better-conditioned and more effective target*.

E.2 Batch Construction Strategy

We compare fully shuffled training against grouped batching that keeps multiple completions from the same prompt together, while holding the total number of data per update fixed. As shown in Figure 5b, shuffling consistently improves evaluation loss. This suggests that, unlike the reward model, grouping multiple samples from the same prompt within a batch is unnecessary and can mildly hurt generalization.

E.3 Discount Factor γ

The discount factor γ determines how remaining horizon is distributed in the target space, and therefore where prediction resolution is concentrated along the trajectory. Larger γ compresses long horizons more aggressively, which tends to improve prediction earlier in generation. Smaller γ allocates more resolution near termination, improving prediction later in the trajectory. To evaluate this trade-off, we measure prediction error at relative decoding positions 0%, 25%, 50%, and 75%. Figure 6 shows the expected pattern: larger γ performs better at earlier positions, while smaller γ performs better closer to the end. Thus, γ acts as a simple knob for balancing early-horizon compression against late-horizon discrimination. In practice, we choose an intermediate γ to balance early-horizon compression against late-horizon discrimination. Specifically, we set γ such that the 99th-percentile generation length $L_{0.99}$ satisfies $1 - \gamma^{L_{0.99}} = 0.99$, so that nearly all observed horizons are mapped into the high-resolution region of the target space.

E.4 Numerical Precision

We compare FP16, BF16, and FP32 under the same training and evaluation setup. Figure 5c shows nearly identical loss curves across all three formats, with no substantial difference in convergence or final performance. This indicates that LenVM is numerically stable under common floating-point precisions. Appendix G provides further analysis of finite-precision effects.

E.5 Candidate-Set Size

Candidate-set size affects hard target matching and soft efficiency steering differently. Increasing K from 1 to 128 raises the LIFEbench-token score from 29.3 to 72.1 and reduces deviation from 100% to 19%. On MATH500 with $\beta = -100$, most of the change occurs between $K = 1$ and $K = 2$; beyond $K = 2$, average length and Pass@1 remain stable. Appendix D reports the complete ablation and packed forward times.

F LenVM as a Length-Specific Value Function in RL

This section makes explicit the RL interpretation of LenVM and clarifies two distinct ways in which it can interact with reinforcement learning. First, the return constructed in Section 2 is already a valid discounted reward process, so LenVM can be viewed as a value function for a token-level length objective. Second, because LenVM defines a state-dependent notion of progress toward termination, it can also be used to construct potential-based shaping rewards that provide dense intermediate learning signals without changing the underlying task objective, under the standard fixed-potential assumptions used in reward shaping. Our use of this connection in the main paper is conceptual only: we do not perform RL fine-tuning with LenVM, and all reported results use LenVM at inference time.

F.1 LenVM as a Value Function for a Length Objective

Consider an autoregressive policy π over a generated trajectory of length L . Suppose the RL objective contains both a task component and a length component,

$$J(\pi) = J_{\text{task}}(\pi) + s J_{\text{len}}(\pi), \quad (20)$$

where $s \in \mathbb{R}$ controls the strength and sign of the length term. Following Section 2.1, define the per-step length reward as

$$r_t^{\text{len}} = -(1 - \gamma), \quad t = 0, \dots, L - 1, \quad (21)$$

with terminal reward $r_L^{\text{len}} = 0$. The corresponding discounted return from state s_t is then

$$G_t^{\text{len}} = \sum_{i=0}^{L-t} \gamma^i r_{t+i}^{\text{len}} = -(1 - \gamma^{L-t}). \quad (22)$$

Therefore the length value function under policy π is

$$V_{\pi}^{\text{len}}(s_t) \triangleq \mathbb{E}_{\pi} \left[G_t^{\text{len}} \mid s_t \right]. \quad (23)$$

This is exactly the quantity LenVM is trained to estimate. In this sense, LenVM is not merely correlated with generation length. It is a value function for a well-defined token-level objective in which each additional decoding step incurs a constant discounted cost.

This interpretation is useful because it makes explicit what happens if LenVM is inserted directly into RL optimization. Using a positive coefficient s in the additive objective above means that the policy is optimized not only for task reward, but also for shorter continuations. Thus, direct optimization of $J_{\text{task}}(\pi) + sJ_{\text{len}}(\pi)$ generally changes the task objective unless generation efficiency is itself part of the intended goal. This distinction is important. LenVM can be used either as a genuine objective component, when one wishes to trade off quality against generation cost, or as a source of auxiliary structure for improving learning without redefining the task. The latter case motivates the shaping perspective developed below.

F.2 Advantage Decomposition and Policy Optimization

With separate task and length rewards, one can also maintain separate value baselines. Let $V^{\text{task}}(s_t)$ denote the value function for the task reward and let $\hat{v}_{\phi}(s_t)$ denote the LenVM prediction for the length reward. Then the temporal-difference residuals can be written as

$$\delta_t^{\text{task}} = r_t^{\text{task}} + \gamma_{\text{task}} V^{\text{task}}(s_{t+1}) - V^{\text{task}}(s_t), \quad (24)$$

and

$$\delta_t^{\text{len}} = r_t^{\text{len}} + \gamma \hat{v}_{\phi}(s_{t+1}) - \hat{v}_{\phi}(s_t). \quad (25)$$

Applying GAE to each component gives

$$A_t^{\text{task}} = \sum_{i=0}^{L-1-t} (\gamma_{\text{task}} \lambda_{\text{task}})^i \delta_{t+i}^{\text{task}}, \quad (26)$$

and

$$A_t^{\text{len}} = \sum_{i=0}^{L-1-t} (\gamma \lambda_{\text{len}})^i \delta_{t+i}^{\text{len}}. \quad (27)$$

In the simplest case, one combines these signals additively:

$$A_t^{\text{total}} = A_t^{\text{task}} + s A_t^{\text{len}}. \quad (28)$$

One may also schedule the scalar coefficient s over training if the intended quality-efficiency trade-off changes over time. Here $s > 0$ discourages longer continuations, while $s < 0$ encourages them.

This decomposition is attractive because the quality and efficiency terms remain explicitly separated. The task critic models the utility of the generated content, while LenVM models the expected future cost of continuing generation. As a result, changing the scalar coefficient changes how these two factors enter the policy update without redefining the underlying value targets themselves.

At the same time, it is important to note that this additive formulation is an objective change, not merely a variance-reduction device. When $s \neq 0$, the induced policy update explicitly prefers trajectories with different length characteristics. Therefore this formulation is most appropriate when one truly wishes to optimize a performance-efficiency trade-off, such as token-budgeted generation, latency-aware decoding, or resource-constrained RL for language models.

E.3 LenVM as a Potential for Policy-Invariant Reward Shaping

The same learned signal also admits a different use that does not directly modify the underlying task objective. Suppose the original RL problem is defined only by task reward r_t^{task} . If one wishes to provide dense intermediate feedback without changing the underlying task objective, a standard construction is potential-based reward shaping:

$$r'_t = r_t^{\text{task}} + \beta \left(\gamma \Phi(s_{t+1}) - \Phi(s_t) \right), \quad (29)$$

where Φ is any scalar potential over states and $\beta \geq 0$ controls the shaping strength.

LenVM provides a natural candidate for such a potential. Because $\hat{v}_\phi(s_t) \in (-1, 0)$ is a bounded monotone transform of expected remaining generation length, it defines a notion of prefix-wise progress toward termination. States closer to EOS have values nearer to 0, whereas states expected to continue longer have values closer to -1 . A simple choice is therefore

$$\Phi(s_t) = \text{sg}(\hat{v}_\phi(s_t)), \quad (30)$$

where $\text{sg}(\cdot)$ denotes stop-gradient, and we set

$$\Phi(s_{\text{EOS}}) = 0. \quad (31)$$

The resulting shaping term is

$$F_t = \gamma \Phi(s_{t+1}) - \Phi(s_t). \quad (32)$$

This construction clarifies the role of LenVM in RL. If $\Phi(s_{t+1}) > \Phi(s_t)$, then the next decoding state is predicted to be closer to termination, so F_t becomes larger. Such a transition receives a more positive shaping signal. Conversely, if $\Phi(s_{t+1}) < \Phi(s_t)$, then the chosen token moves the trajectory toward a longer continuation, and the shaping signal becomes smaller or more negative. Thus LenVM-based shaping does not directly reward short outputs as an endpoint property. Rather, it rewards local transitions that appear to make progress toward termination under the current decoding dynamics.

Under the standard shaping assumptions that the decoding process is treated as a discounted Markov decision process, the same discount factor is used in both the base return and the shaping term, and the potential is fixed during policy optimization, the reason this shaping can alter optimization dynamics without changing the underlying task objective is the telescoping form of the added reward. For any trajectory starting from s_0 ,

$$\sum_{t=0}^T \gamma^t \left(\gamma \Phi(s_{t+1}) - \Phi(s_t) \right) = -\Phi(s_0) + \gamma^{T+1} \Phi(s_{T+1}). \quad (33)$$

When terminal states are assigned zero potential, this reduces to a boundary term depending only on the start state. Therefore all trajectories from the same initial state receive the same offset, and their ordering under expected return is unchanged. In this sense, potential-based shaping modifies the distribution of intermediate learning signals rather than the final task criterion itself.

This distinction is especially relevant for language-model RL. A direct length penalty changes what the policy is asked to optimize. By contrast, LenVM-based shaping can be interpreted as a progress signal that redistributes terminal information over token-level decisions. It may therefore improve credit assignment and sample efficiency even when the intended objective is still purely task success.

In practice, however, LenVM is policy-dependent, since it estimates $V_\pi^{\text{len}}(s_t)$ under the rollout policy. For the shaping interpretation to remain closest to the classical policy-invariance result, it is preferable to use a pretrained and fixed LenVM when constructing Φ . If the potential is updated simultaneously with the policy, then the shaping reward itself becomes non-stationary, and the standard invariance guarantee no longer applies directly. For this reason, if LenVM were to be used for RL shaping in future work, the most stable protocol would be to pretrain it first and then freeze it during policy optimization.

F.4 Why This is Preferable to a Sequence-Level Length Penalty

A common alternative is to add a sequence-level penalty such as $R_{\text{task}} - \alpha L$ at the end of generation. The token-level formulation above has several advantages.

First, it yields dense credit assignment. Each additional token induces a local change in the length value, so the policy can receive an immediate signal about whether continuing the rollout appears worthwhile. In contrast, a terminal penalty must be propagated backward through the full sequence.

Second, it admits a matched value baseline. Since LenVM is trained directly on the return induced by r_t^{len} , the resulting baseline is aligned with the optimization target for the length component. This reduces variance relative to using a terminal length penalty with a generic value function that must jointly absorb task quality and sequence length.

Third, it cleanly separates quality from efficiency. The task value network models the utility of the generated content, while LenVM models the expected future cost of additional decoding steps. This decoupling is especially natural when task reward is sparse or delayed.

Fourth, in the shaping setting, LenVM provides a structured progress potential rather than only a final scalar penalty. This is useful precisely because the reward is changed in a way that preserves the original optimum while making intermediate decisions easier to evaluate during training.

Finally, the trade-off coefficient remains interpretable. In the additive objective formulation, s acts directly on a length-specific advantage term. In the shaping formulation, β controls only the strength of the auxiliary progress signal. These two roles should be distinguished, since one changes the optimized objective while the other changes only the learning dynamics.

F.5 Connection to the Main Paper

The role of this section is to clarify that the LenVM objective is compatible with PPO-style RL because it already defines a proper discounted value-learning problem over decoding states. More importantly, the same learned signal admits two conceptually distinct uses in RL. It can be treated as a length-specific value function and directly incorporated into the policy objective when one wishes to optimize quality-efficiency trade-offs. Alternatively, it can be converted into a potential function for reward shaping under the standard fixed-potential assumptions when one wishes to preserve the original task objective while providing denser token-level learning signals.

This RL perspective also helps explain why LenVM is useful at inference time. Even outside RL, the model provides a prefix-conditioned estimate of how close generation is to termination. That signal can be exploited for length-controlled decoding and performance-efficiency trade-offs at test time. We do not claim empirical RL gains in this paper, and we leave RL fine-tuning with these formulations to future work.

G Finite-Precision Analysis of Relative Length Resolution

LenVM is trained to regress remaining horizons over a wide dynamic range, from 1 to 32k. In this setting, the central numerical question is what relative length resolution the representation can maintain under finite precision. Since longer horizons naturally tolerate larger absolute deviations, the more relevant quantity is the relative perturbation $|\delta l|/l$.

Recall that LenVM predicts $\hat{v} = -\sigma(z)$ and that the length estimate is recovered from

$$1 + \hat{v} = \gamma^{\hat{l}}. \quad (34)$$

Since $1 + \hat{v} = 1 - \sigma(z) = \sigma(-z)$, we have

$$\hat{l} = \frac{\ln(\sigma(-z))}{\ln \gamma}, \quad (35)$$

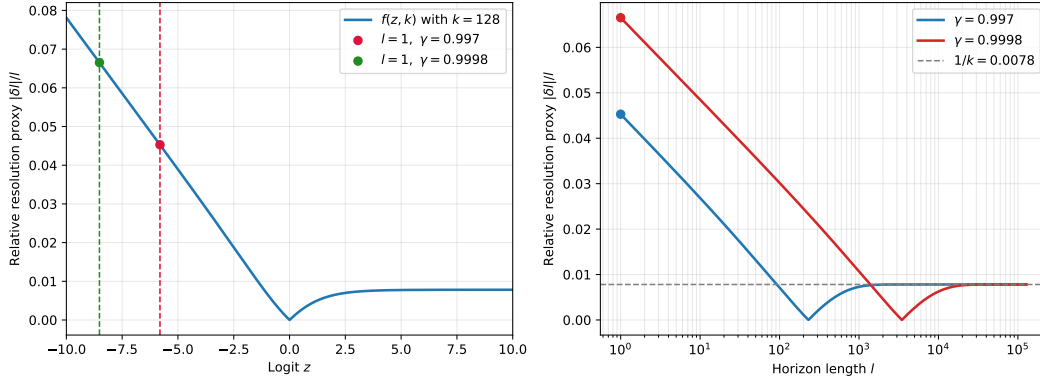


Figure 7: Relative length resolution under finite precision from two complementary views. Left: the proxy $f(z, k)$ as a function of the logit z . Right: the same quantity expressed as a function of the horizon l using $\sigma(-z) = \gamma^l$. Markers indicate the finite-logit locations corresponding to $l = 1$ for representative values of γ .

or equivalently,

$$\sigma(-z) = \gamma^l. \quad (36)$$

Thus, the logit corresponding to horizon l is

$$z(l) = \ln \frac{1 - \gamma^l}{\gamma^l}. \quad (37)$$

A first-order perturbation analysis with respect to z gives

$$\left| \frac{\delta l}{l} \right| \approx m(z) |\delta z|, \quad (38)$$

where

$$m(z) = -\frac{\sigma(z)}{\ln(\sigma(-z))}. \quad (39)$$

To obtain a lightweight finite-precision proxy, we model the local logit perturbation as

$$|\delta z| \approx \frac{|z|}{k}, \quad (40)$$

where k characterizes the effective precision level. In particular, taking $k \approx 2^p$ for a floating-point format with p effective mantissa bits gives $k = 128$ for bfloat16, $k = 1024$ for fp16, and $k = 8388608$ for fp32. Substituting this approximation into the expression above yields

$$\left| \frac{\delta l}{l} \right| \approx f(z, k) = -\frac{\sigma(z) |z|}{k \ln(\sigma(-z))}. \quad (41)$$

We interpret $f(z, k)$ as a proxy for the local relative length resolution under finite precision.

For long horizons, corresponding to $z \rightarrow +\infty$, we have

$$\sigma(z) \rightarrow 1, \quad \ln(\sigma(-z)) \approx -z, \quad (42)$$

and therefore

$$\left| \frac{\delta l}{l} \right| \approx \frac{1}{k}. \quad (43)$$

Thus, at large horizons the representation approaches an approximately constant relative resolution floor. In contrast, the variation in relative resolution is concentrated in the short-horizon regime, where the corresponding logits remain finite and finite-precision effects are more pronounced. In practice, this means that the dominant numerical error arises in the small-length region rather than in the long-horizon tail.

Figure 7 summarizes this analysis from two complementary views. The left panel shows the relative resolution proxy $f(z, k)$ directly in logit space, while the right panel rewrites the same quantity as a function of the horizon l using $\sigma(-z) = \gamma^l$. These plots show that the relative error is primarily concentrated in the small- l region, which is consistent with the empirical behavior observed in Figure 6. Together, they illustrate how finite-precision effects in logit space translate into horizon-dependent relative resolution.

H Future-Dependent Weighting Changes the Regression Target

We clarify the population objective underlying LenVM training and show that weights depending on future rollout outcomes shift the regression target away from the state-conditional mean of the return proxy.

Setup. Let τ denote a sampled prompt-completion trajectory with generated length $L(\tau)$. For each non-terminal step $t \in \{0, \dots, L(\tau) - 1\}$, define the per-token squared error

$$\ell_\theta(\tau, t) \triangleq (V_\theta(s_t^\tau) - G_t^\tau)^2, \quad G_t^\tau = -(1 - \gamma^{L(\tau)-t}). \quad (44)$$

H.1 General Weighted Objective

Consider the general weighted population objective

$$\mathcal{J}_w(\theta) = \frac{\mathbb{E}_{\tau, t} \left[\sum_{t=0}^{L(\tau)-1} w_t(\tau) \ell_\theta(\tau, t) \right]}{\mathbb{E}_{\tau, t} \left[\sum_{t=0}^{L(\tau)-1} w_t(\tau) \right]}, \quad w_t(\tau) \geq 0, \quad (45)$$

with empirical estimator

$$\mathcal{L}_w = \frac{\sum_{n=1}^N \sum_{t=0}^{L^{(n)}-1} w_t(\tau^{(n)}) \ell_\theta(\tau^{(n)}, t)}{\sum_{n=1}^N \sum_{t=0}^{L^{(n)}-1} w_t(\tau^{(n)})}. \quad (46)$$

Optimal predictor. Introducing the indicator $\mathbb{I}(s_t^\tau = s)$ and optimizing $V(s)$ pointwise, the first-order condition gives

$$\mathbb{E}_{\tau, t} \left[\sum_{t=0}^{L(\tau)-1} w_t(\tau) \mathbb{I}(s_t^\tau = s) \cdot (V(s) - G_t^\tau) \right] = 0. \quad (47)$$

Solving for $V(s)$ yields the minimizer of $\mathcal{J}_w$:

$$V_w^*(s) = \frac{\mathbb{E}_{\tau, t} [w_t(\tau) G_t^\tau \mid s_t^\tau = s]}{\mathbb{E}_{\tau, t} [w_t(\tau) \mid s_t^\tau = s]}, \quad (48)$$

i.e., a w -weighted conditional mean of G_t^τ at state s . Whether this coincides with the unweighted conditional mean $\mathbb{E}_{\tau, t} [G_t^\tau \mid s_t^\tau = s]$ depends on the structure of $w_t(\tau)$.

H.2 Instantiations

Case 1: Token-uniform weighting ($w_t(\tau) = 1$). Setting $w_t(\tau) = 1$ recovers the token-averaged objective

$$\mathcal{J}_{\text{tok}}(\theta) = \frac{\mathbb{E}_\tau \left[\sum_{t=0}^{L(\tau)-1} \ell_\theta(\tau, t) \right]}{\mathbb{E}_\tau [L(\tau)]}, \quad (49)$$

where every non-terminal decoding step receives equal weight. Since $w_t(\tau) = 1$ is constant, it cancels in Eq. (48), and the optimal predictor is

$$V_{\text{tok}}^*(s) = \mathbb{E}_{\tau,t}[G_t^\tau \mid s_t^\tau = s]. \quad (50)$$

This is the state-conditional mean of the return proxy, which is exactly the quantity LenVM aims to estimate.

Case 2: State-dependent weighting ($w_t(\tau) = c(s_t^\tau)$, $c(s) > 0$). If $w_t(\tau)$ depends only on the current state s_t^τ , then conditioning on $s_t^\tau = s$ makes $w_t(\tau)$ a deterministic constant $c(s)$. By the linearity of conditional expectation, $c(s)$ factors out of both the numerator and denominator in Eq. (48) and cancels:

$$V_w^*(s) = \frac{c(s) \mathbb{E}_{\tau,t}[G_t^\tau \mid s_t^\tau = s]}{c(s)} = \mathbb{E}_{\tau,t}[G_t^\tau \mid s_t^\tau = s]. \quad (51)$$

Such weights only rebalance the relative importance of different states in the global objective; they do not alter the per-state regression target.

Case 3: Future-dependent weighting. If $w_t(\tau)$ depends on outcomes beyond s_t^τ , then conditioned on $s_t^\tau = s$, the weight $w_t(\tau)$ remains a non-trivial random variable that may be correlated with G_t^τ . In general,

$$V_w^*(s) \neq \mathbb{E}_{\tau,t}[G_t^\tau \mid s_t^\tau = s], \quad (52)$$

meaning the regression target itself is shifted. A concrete example is trajectory-level averaging, which sets $w_t(\tau) = 1/L(\tau)$:

$$\mathcal{J}_{\text{traj}}(\theta) = \mathbb{E}_\tau \left[\frac{1}{L(\tau)} \sum_{t=0}^{L(\tau)-1} \ell_\theta(\tau, t) \right]. \quad (53)$$

Because $L(\tau)$ is determined only after the full rollout, $1/L(\tau)$ is correlated with G_t^τ conditioned on $s_t^\tau = s$: trajectories with larger $L(\tau)$ receive smaller weight yet produce more negative G_t^τ . Consequently, $\mathcal{J}_{\text{traj}}$ does not optimize the token-uniform objective in Eq. (49); it optimizes a future-length-reweighted variant with a shifted per-state target.

For LenVM, whose goal is dense token-level Monte Carlo regression over observed decoding states, we therefore adopt the token-averaged objective in Eq. (49).

I Why Inverting the Transformed Horizon Underestimates Expected Remaining Length

In Section 3.4, we evaluate length prediction in the transformed space rather than by directly inverting the predicted horizon into a raw length estimate. This appendix explains why such inversion systematically underestimates the true expected remaining length.

Let L denote the random remaining generation length from a given decoding state, with $L \geq 0$. Define the transformed horizon

$$u(L) = 1 - \gamma^L, \quad (54)$$

where $\gamma \in (0, 1)$.

Suppose the model predicts the conditional expectation in this transformed space exactly:

$$\hat{u} = \mathbb{E}[u(L)]. \quad (55)$$

A natural raw length estimate is then obtained by inversion:

$$\hat{L} = u^{-1}(\hat{u}) = \frac{\ln(1 - \hat{u})}{\ln \gamma}. \quad (56)$$

We now show that this estimate is always less than or equal to the true expected remaining length $\mathbb{E}[L]$.

The first and second derivatives of $u(L)$ are

$$u'(L) = -\gamma^L \ln \gamma, \quad (57)$$

and

$$u''(L) = -\gamma^L (\ln \gamma)^2. \quad (58)$$

Because $\gamma \in (0, 1)$, we have $\ln \gamma < 0$, so $u'(L) > 0$. Thus $u(L)$ is strictly increasing. Also, since $\gamma^L > 0$ and $(\ln \gamma)^2 > 0$, we have $u''(L) < 0$ for all L . Therefore, $u(L)$ is strictly concave.

By Jensen's inequality for concave functions,

$$\mathbb{E}[u(L)] \leq u(\mathbb{E}[L]). \quad (59)$$

Since $u(\cdot)$ is strictly increasing, its inverse $u^{-1}(\cdot)$ is also strictly increasing. Applying u^{-1} to both sides of Equation 59 preserves the inequality:

$$u^{-1}(\mathbb{E}[u(L)]) \leq u^{-1}(u(\mathbb{E}[L])) = \mathbb{E}[L]. \quad (60)$$

Therefore,

$$\hat{L} \leq \mathbb{E}[L]. \quad (61)$$

As a result, the strict convexity of the discounted return mapping, combined with Jensen's inequality, guarantees that the inverted length $\hat{L}$ is bounded above by the true expected length $\mathbb{E}[L]$. The equality $\hat{L} = \mathbb{E}[L]$ holds if and only if the variance of L is zero (i.e., the remaining length is completely deterministic). In all stochastic sequence generation scenarios where the remaining length has variance, the inversion will systematically underestimate the true expected length. In other words, even if the model predicts the conditional mean accurately in the transformed value space, directly mapping that prediction back to the raw length space via u^{-1} yields an underestimate of the true expected remaining length. The underestimation issue is strictly a consequence of converting to the length space; there is no inherent problem when operating directly within the value space.

J Derivation of the Exponential Tilting Solution in the Performance–Efficiency Trade-off Experiment

We derive the closed-form solution to the KL-regularised length-steering objective in Eq. (11). At each decoding step t , let $p(x)$ be the base model's distribution over a finite candidate set $\mathcal{V}_t$ and let $\hat{v}(x)$ be the LenVM's value prediction for token x . The objective has two goals: (1) minimise the expected LenVM value to steer generation toward shorter completions, and (2) penalise large deviations from the base model via KL divergence to preserve generation quality. We seek the distribution p' that solves, for $\beta < 0$:

$$\min_{p'} \mathbb{E}_{p'}[\hat{v}(x)] - \frac{1}{\beta} D_{\text{KL}}(p' \| p) \quad \text{subject to} \quad \sum_x p'(x) = 1, \quad p'(x) \geq 0. \quad (62)$$

Since $\beta < 0$, the term $-\frac{1}{\beta} > 0$, so the KL divergence enters with a positive coefficient, making the objective strictly convex in p' and guaranteeing a unique global minimiser. (If $\beta > 0$ were used instead, the KL term would be subtracted, rendering the objective unbounded below and the problem ill-posed.)

Expanding the KL term and introducing a Lagrange multiplier λ for the normalisation constraint, the Lagrangian is

$$\mathcal{L}(p', \lambda) = \sum_x p'(x) \hat{v}(x) - \frac{1}{\beta} \sum_x p'(x) \ln \frac{p'(x)}{p(x)} - \lambda \left(\sum_x p'(x) - 1 \right). \quad (63)$$

Taking the derivative with respect to $p'(x)$ and setting it to zero:

$$\hat{v}(x) - \frac{1}{\beta} \left(\ln \frac{p'(x)}{p(x)} + 1 \right) - \lambda = 0. \quad (64)$$

Solving for $p'(x)$:

$$p'(x) = p(x) \exp(\beta \hat{v}(x) - 1 - \beta \lambda). \quad (65)$$

The normalisation condition $\sum_x p'(x) = 1$ fixes the constant, giving the final result:

$$p'(x) = \frac{p(x) \exp(\beta \cdot \hat{v}(x))}{\sum_{x' \in \mathcal{V}_t} p(x') \exp(\beta \cdot \hat{v}(x'))}, \quad \beta < 0. \quad (66)$$

This is the Gibbs / softmax form, and it is the unique global minimiser. With $\beta < 0$, tokens with lower predicted values (shorter expected horizons) receive higher probability mass, steering generation toward shorter completions while $|\beta|$ controls how far the distribution drifts from the base model.