# Episodic Steering Memory: Retrieval-Augmented Test-Time Control for Reasoning LLMs

Anonymous Authors

**Abstract**

Test-time methods for improving LLM reasoning often trade accuracy for length: sampling and search can increase compute, while longer chain-of-thought can drift into unproductive "overthinking". We propose **Episodic Steering Memory (ESM)**, a retrieval-augmented controller that performs sparse, interpretable activation steering at *control points* during generation. ESM views steering vectors as a library of latent "tools" and selects tools *conditionally* based on the current hidden state: it retrieves similar past control-point states from an episodic memory of counterfactual interventions, forms a tool distribution, and probes a small set of candidates before committing the next segment. Offline, ESM mines candidate tools from contrastive rollouts, selects a compact and diverse library, and populates memory with delayed outcome advantages of tool applications. Online, ESM improves the accuracy–length frontier by intervening only when beneficial. We provide a complete experimental protocol, ablations, and implementation details (including EasySteer integration) to enable reproduction.

## 1 Introduction

Large language models (LLMs) exhibit strong few-shot and chain-of-thought (CoT) reasoning, but their test-time behavior is brittle: a model may underthink (terminate early), overthink (produce long but wrong derivations), or drift into irrelevant branches. Test-time methods—self-consistency, best-of-$N$, tree search, verifiers—can improve accuracy but often require significant extra compute. Meanwhile, *activation steering* offers a lightweight alternative: add a direction vector to internal activations to bias behavior without weight updates.

This paper asks: *can we steer an LLM's reasoning trajectory at test time, in a way that is conditional on the current internal state, computationally practical, and robust to overthinking?* A central challenge is that a single global steering vector is rarely optimal: the same prompt can require different interventions at different stages (e.g., "be concise" vs. "re-check arithmetic"). We therefore propose **Episodic Steering Memory (ESM)**:

- ESM mines many candidate steering directions from contrastive rollouts and selects a compact diverse *tool library.*
- ESM builds an *episodic memory* mapping embedded control-point states to the *delayed advantage* of applying a tool.
- At test time, ESM retrieves similar past states, forms a tool distribution, and probes a small number of candidates before committing the next segment.

ESM is designed to improve the accuracy–length frontier while using only a few sparse interventions and modest probing overhead.

## 2 Preliminaries: Activation Steering at Control Points

### 2.1 Base model and frozen decoding

Let $p_\theta$ be a pretrained autoregressive language model with parameters $\theta$. We assume *frozen* test-time decoding: parameters are fixed, and we only intervene through activation modifications.

### 2.2 Steering operator

Let $h_t^{(\ell)} \in \mathbb{R}^d$ denote the residual-stream activation at token position $t$ and layer $\ell$. A steering tool $b = (v_b, \ell_b, \alpha_b)$ applies

$$\mathcal{S}_{\ell_b,t}(h_t^{(\ell_b)}; v_b, \alpha_b) := h_t^{(\ell_b)} + \alpha_b v_b. \tag{1}$$

We apply steering at a *single* token position per control point to keep interventions sparse and interpretable.

### 2.3 Control points and segments

We partition generation into segments separated by control points:

$$y = [y^{(1)}, y^{(2)}, \ldots, y^{(M)}], \tag{2}$$

where $y^{(m)}$ are tokens generated between control points $m-1$ and $m$. In this paper we use a **deterministic** control-point rule to eliminate ambiguity:

- **Token-interval control points (default):** choose a segment length $L_{\text{seg}}$ and set $t_m = m \cdot L_{\text{seg}}$ (clipped by termination). This aligns control points across rollouts and works for any task.
- **Delimiter control points (optional):** define a delimiter token sequence (e.g., "`\nStep`" or "`<SEG>`") and set control points at its generated token positions. This is useful for step-formatted reasoning.

All experiments in Section 7 use token-interval control points unless otherwise stated.

## 3 Problem Setup

We study test-time control for discrete-answer reasoning tasks. Each instance $x$ is a prompt and has a ground-truth answer $a(x)$. A generation $y$ receives a terminal reward $R(x, y) \in \{0, 1\}$ indicating answer correctness. To reflect the common accuracy–length tradeoff, we define a length-regularized reward

$$R_\eta(x, y) := R(x, y) - \eta \cdot \frac{|y|}{T_{\max}}, \tag{3}$$

where $|y|$ is the number of generated tokens and $T_{\max}$ is a max-new-tokens budget.

We aim to improve the *frontier* of accuracy vs. length by steering at control points. At each control point $m$, the controller observes an embedding $u_m$ of the current hidden state and chooses a tool $b_m \in \mathcal{V} \cup \{\varnothing\}$, where $\varnothing$ is the null (no steering) action. The controller then generates the next segment under the chosen tool and repeats.

## 4 Episodic Steering Memory

ESM is constructed in three offline stages: (i) mine a large pool of candidate steering tools from contrastive episodes, (ii) select a compact and diverse library, and (iii) populate an episodic memory that links *embedded control-point states* to *delayed* outcomes of counterfactual tool interventions.

### 4.1 Stage I: Mining candidate steering tools

**Goal.** Mine a large candidate pool $\widetilde{\mathcal{V}} = \{(v_c, q_c, \ell_c, \alpha_c)\}_{c=1}^{C}$ where each candidate is a direction $v_c$ at layer $\ell_c$ with an initial scale $\alpha_c$, and $q_c$ measures how strongly the direction correlates with success.

**Contrastive activation differences.** For each training instance $x \in \mathcal{D}$, generate $K$ rollouts $\{y^{(k)}\}_{k=1}^{K}$ and terminal scores $R^{(k)}$. Select positives $\mathcal{P}$ (top-$K_+$ rollouts) and negatives $\mathcal{N}$ (bottom-$K_-$ rollouts). For each control point $m$ and each layer $\ell$ in a candidate layer set $\mathcal{L}$, compute

$$\Delta h^{(\ell)}(x,m) \;:=\; \frac{1}{|\mathcal{P}|}\sum_{k\in\mathcal{P}} h^{(\ell)}_{t_m}(x, y^{(k)}) \;-\; \frac{1}{|\mathcal{N}|}\sum_{k\in\mathcal{N}} h^{(\ell)}_{t_m}(x, y^{(k)}). \tag{4}$$

We normalize $v = \Delta h/\|\Delta h\|$ and treat it as a candidate direction. We associate each candidate with an initial scale $\alpha$ via a small calibration grid (Appendix A).

**Quality score.** We score a candidate direction by how much it improves expected reward when applied at the corresponding control point:

$$q(v,\ell,m) := \mathbb{E}_{x\sim\mathcal{D}}\big[R_{\eta_0}(x, y_{\text{steer}}) - R_{\eta_0}(x, y_{\text{base}})\big], \tag{5}$$

estimated on a calibration split $\mathcal{D}_{\text{cal}}$ under a small fixed length penalty $\eta_0$.

### 4.2 Stage II: Selecting a compact and diverse tool library

**Goal.** Select a subset $S \subseteq \widetilde{\mathcal{V}}$ of size $B$ that is both high-quality and diverse. We define a similarity kernel between directions (e.g., cosine similarity between vectors possibly at matched layers), and maximize the quality+diversity surrogate:

$$\max_{S:\,|S|=B} \sum_{i\in S} \log(1+q_i) \;+\; \lambda \log\det\big(K_S + \epsilon I\big), \tag{6}$$

where $K_S$ is the submatrix indexed by $S$. The log-determinant term encourages a spread of directions, hedging against multi-modal failure modes. We optimize Eq. (6) via a greedy algorithm (Appendix B).

### 4.3 Stage III: Populating episodic memory with counterfactual rankings

**Goal.** Build an episodic memory $\mathcal{M}$ consisting of tuples $(u_j, b_j, A_j)$ mapping a state embedding $u_j$ to a tool $b_j \in \mathcal{V}$ and its *terminal advantage* when applied at that state.

**Counterfactual advantage.** For each training instance, we generate a baseline rollout and collect states at control points. At each collected state $j$ (corresponding to control point $m$), we evaluate a small set of candidate tools $b$ by generating the *rest* of the trajectory under that tool (keeping decoding settings fixed). We define the advantage

$$A_j(b) := R_\eta(x, y_j^{\mathrm{cf}}(b)) - R_\eta(x, y^{\mathrm{base}}), \tag{7}$$

where $y_j^{\mathrm{cf}}(b)$ is the counterfactual completion obtained by applying $b$ at state $j$ and then continuing decoding. We store only the top few tools per state to keep memory compact.

## 5 Online Control: Similarity Retrieval and Tool Probing

At test time, at each control point $m$, ESM computes the current embedding $u_m$ and retrieves the $k$ most similar memory entries using a vector index:

$$\mathcal{N}_k(u_m) := \mathrm{TopK}(u_m; \mathcal{M}). \tag{8}$$

We estimate the advantage of each tool by similarity-weighted averaging of stored advantages:

$$\widehat{A}(u_m, b) := \sum_{(u_j, b', A_j) \in \mathcal{N}_k(u_m)} w(u_m, u_j) \cdot A_j \cdot \mathbb{I}\{b' = b\}, \qquad w(u, u') = \frac{\exp\big(\kappa \cos(u, u')\big)}{\sum_{\tilde{u}} \exp\big(\kappa \cos(u, \tilde{u})\big)}. \tag{9}$$

We convert estimated advantages into a distribution over tools:

$$p(b \mid u_m) \propto \exp\big(\beta\, \widehat{A}(u_m, b)\big), \tag{10}$$

and sample a small candidate set $\{b^{(1)}, \dots, b^{(L)}\}$ plus the null tool $\varnothing$.

**Tool probing.** For each candidate tool, we generate exactly one next segment and compute an acceptance score

$$\mathrm{Score}(\mathrm{segment}; b) := \widehat{A}(u_m, b) + \rho \cdot \log p_\theta(\mathrm{segment} \mid \mathrm{prefix}, b), \tag{11}$$

where the logprob term acts as a confidence proxy (Appendix D). We commit the best-scoring segment and continue to the next control point.

## 6 Implementation Notes and Complexity

**Steering framework.** ESM requires a steering implementation that can (i) read residual activations at a specified layer and control-point token, and (ii) add a vector scaled by $\alpha$ at that location. In Appendix I, we provide a concrete implementation path using EasySteer (a vLLM-based steering framework) (compatible with `direct` vector addition and per-position triggers).

**Compute overhead.** Let $M$ be the number of control points and $L$ the number of probed tools per control point. ESM generates at most $(L+1)$ segments per control point (including null), so the worst-case generation overhead is roughly a factor of $(L+1)$ on the segmented portion of decoding. In practice, we keep $M$ small (e.g., 4–8) and $L$ very small (e.g., 2–4), making the controller practical.

**Algorithm 1** ESM online control (one instance)

```
1: Initialize prefix with the rendered prompt; set $m \leftarrow 1$
2: while not terminated and tokens $< T_{\max}$ and $m \le M$ do
3:     Generate until control point $m$ (or use already-generated prefix)
4:     Compute embedding $u_m$
5:     Retrieve $\mathcal{N}_k(u_m)$ by Eq. (8); compute $\widehat{A}(u_m, \cdot)$ by Eq. (9)
6:     Form $p(b \mid u_m)$ by Eq. (10); sample candidate list $\{b^{(1)}, \ldots, b^{(L)}\}$
7:     Add the null tool $\varnothing$ to the candidate list
8:     for each candidate tool $b$ do
9:         Generate exactly one next segment under $b$
10:        Compute Score(segment; $b$) by Eq. (11)
11:    end for
12:    Choose $b^\star$ with the best score; if $\max_b \widehat{A}(u_m, b) < \tau_{\text{null}}$ set $b^\star \leftarrow \varnothing$
13:    Commit the best segment; set $m \leftarrow m + 1$
14: end while
```

**Practical defaults.** Stable defaults include neighborhood size $k \in [8, 32]$, projection dimension $d_u = 256$, candidates $L \in [2, 4]$, segment length $L_{\text{seg}} \in [48, 96]$, and a small exploration temperature (moderate $\beta$). We recommend applying each tool at a small set of late layers (e.g., the top third of layers) and normalizing vectors.

## 7 Experiments

We design experiments to (i) test whether ESM improves reasoning accuracy under *matched decoding budgets*, (ii) verify that the gains are not merely due to extra compute, and (iii) isolate which components (mined tools, diversity selection, episodic memory, online probing) matter most. Because ESM is a *test-time* method, our comparisons emphasize **fairness under equal budgets** rather than absolute scores.

### 7.1 Benchmarks, models, and metrics

**Benchmarks.** We evaluate on a standard suite of discrete-answer reasoning tasks: GSM8K and SVAMP (numeric), MATH (symbolic/numeric), ARC-Challenge (multiple-choice), and StrategyQA (yes/no). All evaluation is automatic via deterministic answer extraction (Appendix H).

**Models.** To demonstrate that ESM is not model-specific, we recommend reporting at least two instruction-tuned LLMs: a **7–8B** model (e.g., Qwen2.5-7B-Instruct or Llama-3.1-8B-Instruct) and a **13–14B** model (e.g., Qwen2.5-14B-Instruct). For each model, we report results with the same ESM hyperparameters whenever feasible, and tune only the steering scale range (Appendix A).

**Metrics.** For each dataset we report: (i) Accuracy, (ii) average generated tokens, and (iii) *budget-sweep AUC* computed from accuracy at multiple token budgets. The AUC summarizes robustness to “thinking longer” and directly measures overthinking mitigation.

## 7.2 Fair compute: token budgets and matched baselines

ESM spends extra tokens during *tool probing* (Online Control section). To avoid attributing gains to extra compute, we evaluate under two complementary protocols.

**(P1) Fixed max-new-tokens.** All methods use the same max-new-tokens $T_{\max}$ per instance (e.g., 256/512/1024). For ESM, probing tokens are counted toward $T_{\max}$: if ESM probes $L+1$ candidates for the next segment, we subtract that cost from the remaining budget.

**(P2) Fixed *total* generated tokens.** We additionally equalize the *total* number of generated tokens per instance across methods (including discarded probes). For sampling-based baselines (self-consistency, best-of-$N$ reranking), we choose $N$ so that expected total tokens match ESM within $\pm 2\%$.

We recommend reporting both (P1) and (P2); (P2) is the primary protocol for claims about efficiency.

## 7.3 Baselines

We compare ESM to strong test-time methods that do not modify model weights:

- **Greedy-CoT**: greedy decoding with a standard chain-of-thought prompt.
- **Best-of-$N$**: sample $N$ complete solutions; select using (i) final-answer consistency or (ii) a verifier LM. Match total tokens under (P2).
- **Self-Consistency**: majority vote over $N$ sampled solutions (numeric / categorical tasks).
- **Prompt-only heuristics**: length penalty and "be concise" / "double-check" prompt variants.
- **Static steering**: apply a single global steering vector (chosen on a dev set) at every control point (no retrieval/memory).
- **ESM ablations**: remove one component at a time (Section 7.6).

Exact hyperparameters and prompts are specified in Appendix E.

## 7.4 Main results

Table 1 is the primary evidence: accuracy under matched budgets and models. We recommend reporting results at $T_{\max} = 512$ (main table) and placing additional budgets in the appendix.

## 7.5 Overthinking mitigation and budget robustness

A core motivation of ESM is to *intervene only when needed* and prevent the model from drifting into unproductive long reasoning. We therefore sweep budgets $T_{\max} \in \{256, 512, 1024\}$ and plot accuracy vs. budget. Overthinking appears when accuracy *decreases* as budget increases.

We summarize robustness using the AUC of the curve and the *overthinking gap* $\Delta = \mathrm{Acc}_{1024} - \mathrm{Acc}_{256}$.

## 7.6 Ablations and analysis

We isolate contributions of (i) mined tools, (ii) diversity selection, (iii) episodic memory, and (iv) probing.

Table 1: Main results: accuracy (%) under a matched max-new-tokens budget ($T_{\max} = 512$) and protocol (P2) fixed total tokens. Fill in the numbers after running the provided evaluation scripts.

| Method | GSM8K | SVAMP | MATH | ARC-C | StrategyQA | Avg |
|---|---|---|---|---|---|---|
| Greedy-CoT | – | – | – | – | – | – |
| Self-Consistency ($N$ matched) | – | – | – | – | – | – |
| Best-of-$N$ (rerank, $N$ matched) | – | – | – | – | – | – |
| Prompt-only heuristics | – | – | – | – | – | – |
| Static steering (global vector) | – | – | – | – | – | – |
| ESM (ours) | – | – | – | – | – | – |

Figure 1: Accuracy vs. token budget. ESM should improve the area under the curve and reduce negative slopes (overthinking).

**Component ablations.** We report the following variants on a representative dataset (GSM8K and MATH) under $T_{\max} = 512$:

- **No memory**: use the same tool distribution at every control point (global prior), but keep probing.
- **No probing**: select the tool with the largest retrieved $\widehat{A}(u_m, b)$ and commit without counterfactual probing.
- **No DPP diversity**: build the library by selecting top-$B$ tools by quality $q$ (Stage I).
- **Random library**: select $B$ tools uniformly at random from the mined pool.
- **Single control point**: allow only one intervention at $m = 1$ (tests whether ESM is more than "one-shot steering").

**Sensitivity to library size and probing width.** We vary $B \in \{16, 32, 64, 128\}$ and probing width $L \in \{0, 1, 3, 5\}$ (with $L$=0 meaning no probing). We recommend plotting accuracy and token cost as a function of $B$ and $L$ (place full grids in the appendix).

**Interpretability.** We report (i) the frequency of chosen tools across control points and datasets, and (ii) a 2D PCA plot of tool vectors. We include 3–5 qualitative case studies showing how different retrieved tools correspond to behaviors such as "be concise", "re-check arithmetic", or "consider alternative hypothesis" (full examples in the appendix).

Table 2: Ablation summary template (report on GSM8K, $T_{\max} = 512$, protocol (P2)).

| Variant | Acc (%) | Tokens / inst. |
|---|---|---|
| ESM (full) | – | – |
| No memory | – | – |
| No probing ($L$=0) | – | – |
| No DPP diversity | – | – |
| Random library | – | – |
| Single control point | – | – |

### 7.7 Implementation and reproducibility

All experiments use the same prompting templates (Appendix E) and default hyperparameters (Appendix J). We implement test-time steering via EasySteer (Appendix I) and release scripts that: (i) generate offline rollouts, (ii) mine and export tool vectors, (iii) build the episodic memory, and (iv) run evaluation under (P1)/(P2) with exact token accounting.

## 8 Limitations and Discussion

ESM is a test-time control method and inherits limitations of steering and retrieval. First, steering directions mined from a finite dataset may not generalize to all distributions. Second, probing increases compute; although $L$ is small, worst-case overhead can still be non-trivial. Third, activation steering can cause unintended side effects (style drift, refusals, or hallucinations) if scale is too large. We mitigate these issues with sparse control points, calibration, a null tool, and matched-budget evaluation.

## 9 Related Work

We relate ESM to (i) test-time compute methods (self-consistency, verifiers, search), (ii) activation steering and representation engineering, and (iii) retrieval-augmented control and episodic memory. We leave a full literature discussion to the final submission.

## 10 Conclusion

We introduced Episodic Steering Memory, a retrieval-augmented controller that performs conditional activation steering at control points. ESM mines a diverse tool library offline, stores delayed counterfactual advantages in an episodic memory, and probes a small candidate set online. We provide a complete experimental protocol and implementation details to enable reproduction and future work on controllable reasoning.

## A Scale selection details

**Recommended calibration protocol.** Fix one control-point index $m$ and a candidate layer $\ell$ for each candidate direction. On $\mathcal{D}_{\text{cal}}$, run baseline completions and completions with the candidate applied *only* at control point $m$. Evaluate $\alpha \in \{0.5, 1, 2, 4\}$ and choose the best $\alpha^\star$ under $R_{\eta_0}$ with $\eta_0 = 0.001$. To stabilize, also evaluate the negative direction $-v$; keep the best of $\pm v$.

## B Greedy optimization for Eq. (6)

We optimize the quality+diversity objective with a greedy forward selection: start from $S = \emptyset$ and iteratively add the element that maximizes the marginal gain of Eq. (6). We maintain the determinant increment efficiently via Cholesky updates on the kernel submatrix.

## C Tool prototypes for fast candidate selection

In large libraries, we optionally cluster tools and maintain a small set of prototypes to reduce retrieval cost. We use k-means on normalized tool vectors and store cluster centroids as prototypes. At test time, we first score prototypes, then sample tools from top clusters.

## D Confidence proxy

We use the (length-normalized) log-probability of the probed segment under the steered model as a confidence proxy. This helps avoid committing low-likelihood segments that may arise from overly strong steering.

## E Prompt templates and control-point settings

This section removes ambiguity by specifying the exact prompting format used in experiments.

**Common system prompt (if applicable).**

```
You are a helpful assistant. Follow the user's instructions carefully.
```

**Chat-format wrapper.** For chat models, we wrap each task prompt into the model's native chat template (e.g., Qwen/Llama). When using vLLM, we recommend applying the tokenizer's chat template to obtain the final string.

**GSM8K / SVAMP (0-shot).**

```
Solve the following problem step by step.
Return the final numeric answer on the last line in the format:
#### <answer>

Problem:
{QUESTION}
```

**MATH (0-shot).**

```
Solve the following problem carefully.
You may show intermediate steps.
Put the final answer in \boxed{<final answer>} on the last line.

Problem:
{QUESTION}
```

**ARC-Challenge (multiple choice).**

```
Choose the correct option (A, B, C, or D).
Briefly explain your reasoning.
Put only the chosen letter on the last line in the format:
#### <A/B/C/D>

Question:
{QUESTION}

Options:
(A) {A}
(B) {B}
(C) {C}
(D) {D}
```

**StrategyQA.**

```
Answer the question with Yes or No.
Put only the final label on the last line in the format:
#### <Yes/No>

Question:
{QUESTION}
```

**Control-point schedule.** We use token-interval control points. Let $L_{\text{seg}}$ be the segment length (in *newly generated tokens*). Control point $m$ occurs right after the model has generated exactly $m \cdot L_{\text{seg}}$ tokens since the start of the assistant response. In code, maintain a running counter of generated token IDs; once the counter hits a multiple of $L_{\text{seg}}$ and the model has not terminated, pause generation, extract the state embedding, and invoke Algorithm 1. We stop at the earliest of: (i) EOS, (ii) reaching $T_{\max}$ total new tokens, or (iii) reaching $M$ control points.

**Delimiter-based control points (optional).** As an alternative, we define control points at occurrences of a delimiter token sequence (e.g., double newline `\n\n`). This setting better aligns with "step-by-step" solutions but requires tokenizer-specific delimiter handling. We report delimiter-based results only in the appendix (Appendix E).

## F Offline rollout generation details

**Stage I rollouts.** To mine contrastive directions, we require diversity across rollouts. Use stochastic decoding with temperature 0.7, top-$p = 0.95$, and a max token budget $T_{\max}^{\text{mine}}$ (e.g., 512). Generate $K$ rollouts per prompt with different random seeds.

**Stage III counterfactual rollouts.** To reduce variance in $A_j(b)$, we recommend greedy decoding ($T = 0$) for both baseline and counterfactual rollouts, keeping the same maximum token budget $T_{\max}^{\text{cf}}$ for fairness. If you prefer stochastic decoding, reuse the same random seed across candidates for each state.

**Prefix selection for memory.** For each prompt, take one baseline rollout and collect state prefixes at each control point $m = 1, \ldots, M$. Optionally subsample control points (e.g., keep only $m \in \{2, 4, 6\}$) to reduce memory size.

## G Memory index and retrieval implementation

We store each memory entry as $(u_j, b_j, A_j)$ where $u_j \in \mathbb{R}^{d_u}$. We recommend using FAISS for approximate nearest neighbors with inner-product search on normalized vectors (cosine similarity). We also recommend storing per-tool aggregated statistics (mean advantage, count) for debugging.

## H Evaluators and answer extraction rules

We use deterministic, fully specified post-processing so that results are reproducible and comparable across methods. All evaluators operate on the *final assistant message* (no access to hidden states).

**Common normalization.** We lowercase for label tasks, strip surrounding whitespace, and remove trailing punctuation. For numeric tasks, we also remove thousands separators (commas) and standardize minus signs.

**GSM8K / SVAMP (numeric exact match).** We extract the answer from the last line matching `####`:

- Regex: `r"*([-+]?[̣,̣]*?̣*̣)"`.
- Normalization: remove commas; cast to Python `Decimal` (or float if needed); compare exact string after canonicalization.
- If no  line exists, fall back to the last number in the response.

**MATH (symbolic equivalence when possible).** We first try to extract a LaTeX-boxed answer:

- Regex: `r"`
  `boxed{([]`$*$)}"$(take the last match)$.
- `If absent, fall back to the last  line using the GSM8K regex.`

For equivalence, we recommend using a standard symbolic checker (e.g., `sympy` with `latex2sympy` or `math_verify`) and reporting both (i) exact-string accuracy and (ii) symbolic accuracy. If symbolic parsing fails, we fall back to exact string match after whitespace removal.

**ARC-Challenge (multiple choice).** We extract the final option letter from the last  line:

- Regex: `r"*([ABCDE])"`.
- If absent, fall back to the last standalone capital letter in {A,B,C,D,E}.

**StrategyQA (yes/no).** We extract from the last  line:

- Regex: `r"*(yes|no)"` (case-insensitive).
- If absent, fall back to the last occurrence of “yes” or “no”.

**Reportable statistics.** For each benchmark, we report: (i) Accuracy, (ii) average generated tokens, and (iii) token-normalized accuracy (Acc per 1k generated tokens). For budget-sweep experiments, we also report AUC of the accuracy–budget curve.

## I Concrete implementation using EasySteer (vLLM-based steering)

This appendix provides a concrete implementation recipe for ESM using `EasySteer`, a vLLM-based framework that supports steering vectors at inference time. We use the **direct** algorithm (vector addition), matching the steering operator in Section 2.

### I.1 Installation

EasySteer is distributed as a Git repository with a vLLM submodule. A typical installation flow is:

```
conda create -n easysteer python=3.10 -y
conda activate easysteer

git clone --recurse-submodules https://github.com/ZJU-REAL/EasySteer.git
cd EasySteer/vllm-steer
VLLM_USE_PRECOMPILED=1 pip install --editable .

cd ..
pip install --editable .
```

If the precompiled wheel is unavailable for your system, build-from-source instructions are provided in the EasySteer README.

### I.2 Model initialization (inference mode)

EasySteer recommends enabling steering and forcing eager execution:

```
from vllm import LLM, SamplingParams

llm = LLM(
  model="Qwen/Qwen2.5-7B-Instruct",
  enable_steer_vector=True,
  enforce_eager=True,
  enable_chunked_prefill=False,
  tensor_parallel_size=1,
)
```

### I.3 Applying a steering vector during generation

In EasySteer, a steering vector is applied via a `SteerVectorRequest`:

```
from vllm.steer_vectors.request import SteerVectorRequest

req = SteerVectorRequest(
  steer_vector_name="tool_17",
  steer_vector_int_id=17,
  steer_vector_local_path="vectors/tool_17.gguf",
  scale=1.5,
  target_layers=[20,21,22,23,24,25,26,27,28,29,30,31],
  prefill_trigger_tokens=[-1],     # -1: apply to all prompt tokens (optional)
  generate_trigger_tokens=[-1],    # -1: apply to all generated tokens
)

params = SamplingParams(temperature=0.0, max_tokens=128)
out = llm.generate(["<prompt string>"], sampling_params=params,
                   steer_vector_request=req)
text = out[0].outputs[0].text
```

For *position-specific* interventions (e.g., apply only at the last prompt token), EasySteer also supports trigger *positions* via `VectorConfig` (see the README's multi-vector example); we use this mode in an ablation on "prompt-end injection".

## I.4 Segmented generation for ESM

ESM requires pausing at control points, probing candidates for the next segment, and committing one segment. A simple, robust implementation is to run vLLM in a loop, each time generating a short segment of length $L_{\text{seg}}$:

```
prefix = rendered_prompt_string
for m in range(M):
  # build req_m from the selected tool for this step (or null tool)
  params = SamplingParams(temperature=0.0, max_tokens=L_seg)
  seg = llm.generate([prefix], sampling_params=params,
                     steer_vector_request=req_m)[0].outputs[0].text
  prefix = prefix + seg
  if "<eos>" in seg: break
```

In our codebase, we additionally keep exact token accounting so that discarded probe tokens are counted toward the total budget (protocol (P2) in Section 7.2).

## I.5 Extracting hidden states for mining

EasySteer provides a hidden-state extraction utility based on vLLM's embed mode:

```
import easysteer.hidden_states as hs
from vllm import LLM

llm_embed = LLM(
  model="Qwen/Qwen2.5-7B-Instruct",
  task="embed",
  tensor_parallel_size=1,
  enforce_eager=True,
  enable_chunked_prefill=False,
  enable_prefix_caching=False,
)
```

```
prompts = [
  "What are the future trends in artificial intelligence?",
  "Explain the basic principles of quantum computing",
  "How to effectively learn a new language",
]
all_hidden_states, outputs = hs.get_all_hidden_states(llm_embed, prompts)
# all_hidden_states: list[sample][layer][token] -> torch.Tensor
```

### I.6 Mining and exporting vectors (DiffMean example)

For analysis-based vectors, EasySteer exposes DiffMean extraction and GGUF export:

```
from easysteer.steer import extract_diffmean_control_vector

control_vector = extract_diffmean_control_vector(
  all_hidden_states=all_hidden_states,
  positive_indices=[0,1,2,3],
  negative_indices=[4,5,6,7],
  model_type="qwen2.5",
  token_pos=-1,
  normalize=True,
)
control_vector.export_gguf("vectors/tool_17.gguf")
```

In ESM, `positive_indices` correspond to successful rollouts and `negative_indices` to failed rollouts at the same control point.

### I.7 Batching tips

To reduce overhead:

- **Probe in a batch:** at each control point, create $L+1$ prompts (same prefix) and call `llm.generate` once with a list input.
- **Reuse tokenization:** cache token IDs for prompt prefixes when possible (EasySteer supports KV-cache in its v1 adaptation).
- **Keep segments short:** $L_{\text{seg}} \in [32, 128]$ is usually sufficient; longer segments reduce control granularity.

## J Default hyperparameters (recommended starting point)

This section provides a concrete configuration that is (i) strong enough to show gains in preliminary experiments, and (ii) cheap enough to iterate on during development. Unless otherwise stated, all hyperparameters are shared across datasets.

**Control points.**

- Control-point rule: token-interval (Section 2.3).
- Segment length: $L_{\text{seg}} = 64$ new tokens.
- Max control points: $M = 6$ (thus at most 384 controlled tokens; generation may continue until $T_{\max}$).

**Offline mining (Stage I).**

- Rollouts per prompt: $K = 8$ with temperature 0.7, top-$p = 0.95$.
- Pos/neg sets: $K_+ = 2$, $K_- = 2$ using terminal reward $R_{\eta_0}$ with $\eta_0 = 0.001$.
- Candidate layers: top third of layers (e.g., layers 20–31 in a 32-layer model).
- Candidate pool size: keep top $C = 512$ candidates by quality before diversity selection.

**Library selection (Stage II).**

- Library size: $B = 64$ tools (increase to 128 for 13–14B models).
- DPP tradeoff: $\lambda = 0.5$, $\epsilon = 10^{-4}$ (see Appendix B).

**Episodic memory (Stage III).**

- Counterfactual evaluation per state: $L_{\text{eval}} = 8$ candidate tools (plus null).
- Keep the top $S_{\text{keep}} = 2$ tools per state in memory.
- State embedding: average of the last-token residual streams at layers in $\mathcal{L}$, projected to $d_u = 256$.

**Online control.**

- Retrieval neighbors: $k = 16$; similarity softmax temperature $\kappa = 10$.
- Candidate list: sample $L = 3$ tools from $p(b \mid u_m)$ plus the null tool.
- Probe acceptance: $\rho = 0.1$ (logprob confidence weight), $\tau_{\text{null}} = 0$.

**Budgets.**

- Main-table budget: $T_{\max} = 512$ max-new-tokens.
- Budget sweep: $T_{\max} \in \{256, 512, 1024\}$.

**Steering scales.** Use the calibration protocol in Appendix A. A safe search range is $\alpha \in \{0.5, 1, 2, 4\}$ with sign flip $\pm v$.

**Sanity checks.** Before large runs, verify: (i) steering does not change outputs when $\alpha = 0$, (ii) increasing $|\alpha|$ yields monotonic changes in probe scores at a few states, and (iii) ESM improves or matches Greedy-CoT on a small dev subset (e.g., 200 examples) without exceeding the token budget.

## References

[1] H. Xu et al. EasySteer: A Unified Framework for High-Performance and Extensible LLM Steering. arXiv:2509.25175, 2025.