Output-Space Search: Targeting LLM Generations in a Frozen Encoder-Defined Output Space

Tobias Materzok
Independent Researcher
t.materzok@theo.chemie.tu-darmstadt.de

Abstract

We introduce *Output-Space Search* (OS-Search), which turns LLM generation into endpoint search. An outer loop selects a target z^* in a frozen encoder-defined 3D output space Z, and a retrieval-grounded policy trained with sequence-level RL generates outputs whose coordinates land near z^* under standard autoregressive decoding. This enables parallel sweeps and black-box optimization in Z without path-dependent token/program search. On stories, sweeping Z_{text} yields $3.1\times$ higher LLM-scored diversity than prompt-chaining. On code, Bayesian optimization over Z_{code} improves an objective withheld from the controller under matched inference budgets while preserving validity.

1 Introduction

Large language models (LLMs) are now strong generators of text and code, but many workflows require more than "one good sample." Users often want to explore multiple plausible completions in parallel (diverse drafts, branches, alternative solutions), or search for outputs that maximize a downstream score available only through an external evaluator (e.g., heuristics, or learned judges). In practice this is done by repeated sampling in token space or adaptive loops that feed earlier generations back into the context (e.g., prompt chaining or rejection/reranking). These can introduce sequential dependence and/or growing context length, and they still provide no stable low-dimensional interface that an outer loop can optimize directly. We ask whether an autoregressive LLM can expose a simple state-like target that selects where an output should land. Concretely, an outer loop chooses a target point z^* once per sample, the model generates x whose realised coordinates $z(x)$ lie near z^*, and the outer loop can sweep or optimize z^* while keeping decoding standard and autoregressive (Fig. 1). We call this control state-like (in the sense of state vs. path functions, where the objec-

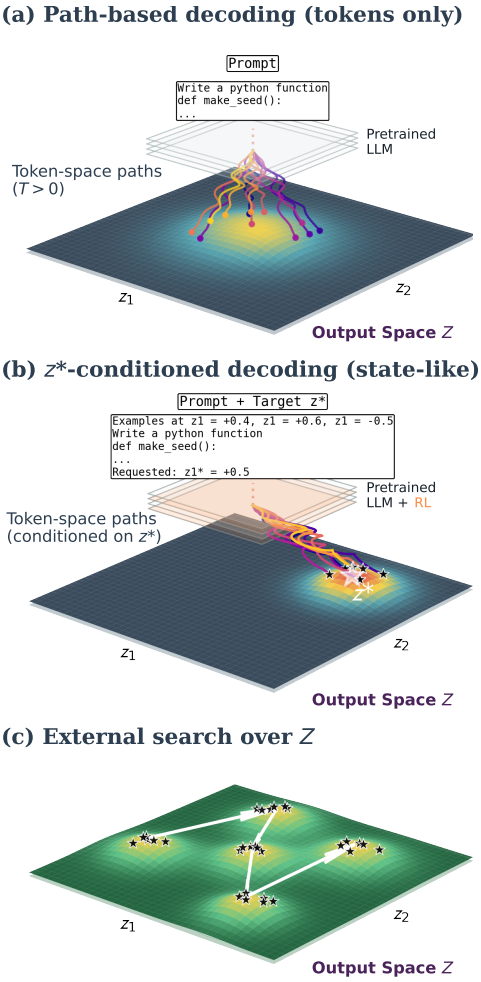

Figure 1: **Path- vs state-like control (schematic 2D slice of Z).** (a) Standard decoding: a fixed prompt yields token-space paths whose endpoints concentrate in one region of the output space Z. (b) z^*-conditioned decoding: the same base prompt plus a requested target z^* (grounded by retrieved exemplars) shifts endpoints toward z^* in Z (up to targeting error), while the decoder remains autoregressive. (c) External search: an optimizer moves z^* across Z to explore regions and maximize a score $f(x)$. Because branches are conditionally independent given the instantiated prompt, multi-branch generation is parallel.

[0]Code (partial release: Z-space construction + code-domain training): https://github.com/TobiasMaterzok/OS-Search. Story-domain training code/data are intentionally not released; see §6.

Proceedings of the First Workshop on Structured Understanding, Retrieval, and Generation in the LLM Era (SURGeLLM 2026), pages 70–92

tive depends on the endpoint rather than the full trajectory) because z^* specifies an intended endpoint region in Z. The outer loop searches over targets z^* rather than token paths or program edits.

We implement this interface with *Output-Space Search* (OS-Search), which keeps representation and actuation modular.

Representation. We define fixed coordinates $z(x)$ for task outputs using a frozen encoder E and a frozen linear projection (Sec. 3, App. A).

Actuation. We train a retrieval-grounded policy conditioned on a requested target z^* so that, via sequence-level RL, its generations land near that request in Z while also self-reporting $\hat{z}$.

Use. The resulting controller exposes a black-box actuator $\mathcal{F}$ that maps (p, z^*) to a sample x, enabling parallel sweeps and black-box optimization over targets in Z without decoding-time guidance.

We evaluate *OS-Search* on both stories (parallel diversity sweeps) and code (outer-loop black-box optimization in Z), and report target-tracking and calibration diagnostics.

Contributions. We make three contributions.

- We propose *OS-Search*: a fixed, external, encoder-defined output space Z together with a z^*-conditioned controller trained with sequence-level RL to (approximately) hit requested coordinates and self-report $\hat{z}_S$.
- For stories, we build an anchored Z_{text} and show that anchoring yields a stable reference axis z_1, which we validate via a monotonic relationship between requested z_1^* and an automated templatedness proxy (EQ-Bench Slop-Score). Sweeping a small grid in Z_{text} yields large embarrassingly-parallel multi-branch diversity gains.
- For code, we build an unanchored Z_{code} and show it supports black-box optimization: Bayesian optimization (BO) over z^* improves a withheld executable objective under matched valid-program budgets.

Tab. 1 contrasts *OS-Search* with direct search methods (e.g., best-of-N, MCTS, evolutionary program search) at the level of the control/search interface.

Trajectory/program search	**OS-Search**
searches token paths / program edits	searches state-like targets z^*
high-dim, discrete search	low-dim, continuous search (d_z=3)
evaluator drives the inner loop	evaluator used only in outer loop, can be withheld
learns objective-specific solver/heuristic	learns an actuator $(p, z^*) \mapsto x$ without access to the downstream objective
sequential dependence across steps	independent samples per z^* (parallel sweeps)
best-of-run artifact	reusable coordinate interface Z

Table 1: Capabilities and typical behaviors of *OS-Search* versus direct trajectory/program search.

2 Related Work

Control for LLMs via prompts, latents, and decoding-time guidance. Our goal is a low-dimensional, user-facing control space that an outer loop can sweep or optimize over task outputs while keeping decoding standard and autoregressive. Many widely used levers are path-based. They modify the token-level sampling process (e.g., temperature, top-p, repetition penalties, logit bias) or optimize preferences over complete trajectories (e.g., RLHF-style objectives) (Christiano et al., 2017; Shao et al., 2024; Guo et al., 2025; Zheng et al., 2025; OpenAI, 2024). Most controllable-generation methods instead inject control variables into the prompt, the model, or the sampling loop. This includes discrete control codes/tokens (Keskar et al., 2019), learned continuous prompts/prefixes/adapters (Lester et al., 2021; Li and Liang, 2021; Houlsby et al., 2019), internal latent-variable control in text VAEs/flows (Gu et al., 2023; Ding et al., 2023; Abeer et al., 2024; Shi and Lee, 2024), and activation-space steering via steering directions or features (Subramani et al., 2022; Liang et al., 2024; Turner et al., 2024; Arditi et al., 2024), including interventions that amplify sparse autoencoder (SAE) features (Huben et al., 2024; Gao et al., 2025). Decoding-time guidance such as PPLM and GeDi recomputes attribute signals on the prefix and modifies logits online (Dathathri et al., 2020; Krause et al., 2021), tightly coupling control to the sampled trajectory.

Treating generation as optimization over a continuous representation space is common in inverse design, e.g., mapping discrete molecules to a continuous latent space and optimizing objectives in that space before decoding back to valid structures (Gómez-Bombarelli et al., 2018).

These mechanisms can be effective, but their control variables are often model-internal, high-

dimensional, or prefix/path-coupled (and may shift across model updates). AxBench finds current steering methods lag strong prompting/fine-tuning baselines for concept control, with SAE-based steering not competitive in their evaluation (Wu et al., 2025). *OS-Search* can be viewed as conditional generation with a continuous condition, but it differs in where the condition lives and how it is used. We define external encoder-defined coordinates $z(x)$ for task outputs, make target requests actionable by pairing each z^* with nearby exemplars in Z, and train a policy to follow these retrieval-grounded requests (with calibration and $\mathrm{Success@}\varepsilon$ diagnostics). This exposes a fixed coordinate interface Z that an outer loop can sweep or optimize once per sample without decoding-time guidance.

Search-based agents and algorithm discovery. Most LLM-based discovery systems perform trajectory/edit search: they iteratively propose token-level actions or program edits, evaluate candidates, and use the evaluator to drive the inner loop (e.g., MCTS, evolutionary program search, or test-time RL) (Silver et al., 2017, 2016; Chen et al., 2024; Xie et al., 2024; Wan et al., 2024; Romera-Paredes et al., 2024; Novikov et al., 2025; Yuksekgonul et al., 2026). *OS-Search* instead aims to make search endpoint-based: we train an objective-agnostic actuator that follows requested coordinates in a frozen output space Z, and use downstream evaluators only in an outer loop that searches over z^* (with $f(x)$ withheld from the controller). A growing line of work pairs strong function approximators with explicit search and automated evaluation to discover policies and algorithms: AlphaZero-style agents combine self-play with Monte Carlo tree search to discover superhuman strategies (Silver et al., 2017, 2016; Chen et al., 2024; Xie et al., 2024; Wan et al., 2024), and related systems have been applied to algorithmic domains such as tensor decomposition and matrix multiplication (Fawzi et al., 2022). More recently, FunSearch and AlphaEvolve use LLM-guided evolutionary search over programs, paired with automated evaluators, to discover new mathematical constructions and optimized heuristics (Romera-Paredes et al., 2024; Novikov et al., 2025). Concurrently, TTT-Discover performs reinforcement learning at test time to adapt an LLM on a single problem instance using continuous evaluator rewards, explicitly prioritizing discovery of one best-of-run solution over average-case performance (Yuksekgonul et al., 2026).

Retrieval grounding for numeric targets in Z. Retrieval-augmented generation (RAG) retrieves documents to inject world knowledge (Lewis et al., 2020). In *OS-Search*, retrieval plays a different role. It grounds otherwise-arbitrary numeric coordinates by retrieving nearby outputs in Z (plus a contrast example), making a requested target z^* interpretable and actionable for the generator (Fig. 2).

Multi-branch story generation. A parallel line of work explicitly constructs branching story trees from a single prompt, either by expanding a latent story graph or by sampling choice points and continuations (Alabdulkarim et al., 2021; Wen et al., 2023; Nottingham et al., 2024; Materzok, 2025). These methods focus on the structure of branching narratives and typically rely on sampling and search over discrete branches. Avoidance Decoding (Park et al., 2025) introduces a decoding-time similarity penalty that discourages new branches from resembling earlier ones, and demonstrates strong gains in multi-branch diversity. Our approach is complementary. Rather than applying guidance at each decoding step, we expose a low-dimensional external output space that can be swept or optimized once per sample via a state-like target z^*.

3 Methods

We define the frozen coordinate map and the sequence-level RL objective for training a z^*-conditioned controller. Alg. 1 summarises the full pipeline, including target sweeps and (when an evaluator is available) outer-loop optimization in Z.

Notation and setup. Let x denote the task content (the generated `<text>` field) and $z(x) \in Z \subseteq \mathbb{R}^{d_z}$ its frozen coordinates under a fixed encoder+projection (PCA subspace + Varimax rotation (Kaiser, 1958)). Each instance specifies a base prompt p, a requested target z^*, and constrained axes $S \subseteq \{1, \ldots, d_z\}$. We sample a structured completion y containing `<text>` x and a self-report $\hat{z}_S$ (Fig. 2), and write $z := z(x)$. Only axes in S enter the reward, and $|S|$ is the control dimensionality. In all our experiments $d_z = 3$, a deliberately small control dimensionality chosen to keep target tracking feasible for the 1.7B controller and to make target sweeps and BO sample-efficient;

evaluating the precision–expressivity trade-off in higher-dimensional spaces is left for future work.

```
USER:
Guidance examples (retrieved in Z):
[<text> x_1 </text><target> z(x_1) </target>]
(near z*)
[<text> x_2 </text><target> z(x_2) </target>]
(near z*)
[<text> x_3 </text><target> z(x_3) </target>]
(near −z*)
Task + domain constraints
REQUESTED TARGET: z*_S (e.g. z1=+0.5, z2=+0.2,
z3=+0.2)
ASSISTANT:
<think> ... </think>
<title> ... </title>
<text> new program/story x </text>
<target> ẑ_S </target>
```

Figure 2: **Prompt schema for OS-Search.** The prompt includes retrieved exemplars with stored realised coordinates $z(x_j)$ and the numeric request z_S^*. The model outputs a structured completion y containing task content x and a self-report $\hat{z}_S$. We compute realised coordinates by embedding only `<text>` to obtain $z(x)$.

3.1 Representation: constructing the output space for text and code

We construct a low-dimensional output space Z (a coordinate system over task outputs) on top of a frozen encoder. Given task content x (the generated `<text>` block, for code we extract the `make_seed()` function and strip comments before embedding, App. B), the encoder produces an embedding $E(x) \in \mathbb{R}^D$, and a frozen linear map projects it to coordinates

$$z(x) \;=\; U^\top (E(x) - \mu) \in \mathbb{R}^{d_z},$$

where μ is the corpus mean embedding and $U \in \mathbb{R}^{D \times d_z}$ has orthonormal columns. We fit (μ, U) once per domain (PCA followed by an orthogonal Varimax rotation within the PCA subspace, additional anchoring for our story experiments) and then freeze E, U, and μ throughout RL so training changes only the policy.

For stories, we fit Z_{text} on a story corpus and anchor z_1 toward Qwen3-generated "default-style" stories relative to the corpus mean (App. A). For code, we fit Z_{code} on a library of N=188 valid `make_seed()` implementations (used to fit Z_{code}, for the code warm-start stage, App. C, and as the Baseline (Z_{code}) sweep in Sec. 4.4). App. A provides full construction details (including the motivation for a task-specific code space under our `make_seed` constraints), diagnostics, and hyperparameter selection.

Algorithm 1 OS-Search: fixed output coordinates + state-like target search in Z.

Require: Frozen encoder $E(\cdot)$, corpus $\mathcal{D}$, (optional) anchors $\mathcal{A}$, d_z=3.
(A) Build Z and the retrieval library.
Fit μ and U (PCA→Varimax, optionally anchor z_1 with $\mathcal{A}$).
Define $z(x) = U^\top(E(x) - \mu)$.
Build exemplar library $\mathcal{L} = \{(x, z(x))\}$ and a nearest neighbor (NN) index in Z (for code, we optionally grow $\mathcal{L}$ during RL with valid model-generated programs).
Freeze (E, μ, U).
(B) Train a z^*-conditioned controller $\pi_\theta(\cdot \mid p, z^*)$.
for group-based RL updates **do**
  Sample base prompt p, constrained axes S, and target z^* (curriculum).
  Retrieve exemplars from $\mathcal{L}$: two near z^* and one near $-z^*$, instantiate the prompt as in Fig. 2.
  Sample a group of structured completions from $\pi_\theta(\cdot \mid p, z^*)$, each containing `<text>` and `<target>` ($\hat{z}_S$).
  for each completion $y = (x, \hat{z}_S)$ in the group **do**
    $R_{\text{format}} \leftarrow \textsc{Valid}(y) \in \{0, 1\}$. ▷ Parse + domain gate: text coherence, code AST+execute+board
    If $R_{\text{format}} = 0$, set $R = 0$. Else embed only `<text>` to get $z(x)$ and compute
    $R = 3 + 3R_{\text{dist}}(z(x), z^*; S) + 1.5R_{\text{hon}}(\hat{z}, z(x); S)$ (Sec. 3.2).
  end for
  Update θ with group-based RL using group-relative rewards.
end for
(C) Use: sweep or optimize targets in Z (optional evaluator f).
Fix a base prompt p.
Propose targets z^* by grid/random/BO.
for each proposed z^* **do**
  Retrieve exemplars and instantiate the prompt as in Fig. 2.
  Sample K candidates $x_k \sim \pi_\theta(\cdot \mid p, z^*)$ and optionally select x_{best} (e.g., maximize f).
  If using BO, update the surrogate at realised coordinates $(z(x_{\text{best}}), f(x_{\text{best}}))$.
end for

3.2 Actuation: training a z^*-conditioned controller

Goal. Given a base prompt p and a requested target z^* (optionally only on a subset of axes S), we train a controller that samples a structured completion y whose task content x lands near z^* in the frozen output space Z.

Prompt interface and required output. Requests follow the schema in Fig. 2. We retrieve exemplars in Z (two near z^*, one near $-z^*$) and include them together with the numeric request z_S^*. The model outputs a structured completion y containing task content x in `<text>` and a self-report $\hat{z}_S$ in `<target>`. For scoring we embed only the generated `<text>` block to obtain $z(x)$. We also request a brief `<think>` hypothesis (not embedded), which should help the policy verbalize how

it intends to realize the target.

Why exemplars ground numeric targets. Because Z is defined by a frozen encoder plus an arbitrary rotation within a PCA subspace, the meaning of numeric coordinates is not self-evident. Nearest-neighbour exemplars provide local grounding. They show what outputs typically look like around the requested neighborhood in Z. To prevent trivial copying in code, near-duplicate outputs to retrieved exemplars receive zero reward during RL (App. D).

Sequence-level reward for validity, targeting, and honesty. For each sampled structured completion $y \sim \pi_\theta(\cdot \mid p, z^*)$, we parse out x and $\hat{z}_S$ and compute a scalar reward

$$R(y) = R_{\text{format}}(y)\Big(3 + 3R_{\text{dist}}(z(x), z^*; S) + 1.5R_{\text{hon}}(\hat{z}, z(x); S)\Big).$$

Here, $R_{\text{format}}(y) \in \{0, 1\}$ is a hard parse + domain-validity gate. If it fails, we set $R(y) = 0$ and do not score distance or honesty. For code, this gate includes AST sanitization and execution to a valid 16×16 board. R_{dist} rewards targeting: it increases as the realised coordinate $z(x)$ moves closer to the requested target z^* on the constrained axes S. R_{hon} rewards calibration: the model should self-report $\hat{z}_S \approx z_S(x)$. Details are in App. D.

Training. We fine-tune Qwen3-1.7B thinking (Yang et al., 2025) with QLoRA adapters (Hu et al., 2022; Dettmers et al., 2023) using group-relative policy optimization (GRPO) with GSPO-style sequence-level corrections (Shao et al., 2024; Guo et al., 2025; Zheng et al., 2025). Training follows a curriculum over constrained-axis subsets S and target magnitudes (App. C). For code, we interleave two short warm-start (prediction) phases on the fixed N=188 program library (coordinate prediction and axis-level hypotheses) with full target-hitting training (App. B, App. C). Full fine-tuning and training hyperparameters are given in App. D.

Story prompt sets. We train on 14 hand-written prompt templates and evaluate multi-branch diversity on 20 WritingPrompts prompts (Fan et al., 2018). The training and evaluation prompt sets are disjoint.

3.3 Use: target sweeps and outer-loop search in Z

Once trained, the controller defines a black-box map $\mathcal{F} : (p, z^*) \mapsto x$ that supports both target sweeps and (when an evaluator is available) outer-loop optimization in Z. For stories, we sweep targets z^* in Z_{text} for a fixed prompt to obtain conditionally independent branches $x \sim \pi_\theta(\cdot \mid p, z^*)$ that can be generated in parallel without prompt chaining (Sec. 4.3).

In the code experiments, we instantiate the outer loop as BO with a Gaussian-process surrogate and expected improvement (GP–EI).

In the code domain, each completion implements a constrained `make_seed() -> list[str]` and is executed and scored by the withheld CA++ benchmark $f(x) \in [0, 1]$ (App. H). The controller is objective-agnostic because f is never used as an RL reward. Instead, an outer loop proposes z^*, samples K candidates, selects the best valid program x_{best}, and (e.g. under BO) updates at the realised coordinate $(z(x_{\text{best}}), f(x_{\text{best}}))$ rather than at the request z^* (Alg. 1, App. I), since z^* is a request and reachable support in Z_{code} is non-uniform (App. Fig. 11).

4 Results

We first report target-tracking diagnostics in the frozen space Z (Sec. 4.1). We then evaluate *OS-Search* in two settings. For stories, we treat the anchored Z_{text} as a controllability and diversity interface (Sec. 4.2, Sec. 4.3). For code, we treat Z_{code} as a search interface under the withheld CA++ objective (Sec. 4.4).

4.1 Target-tracking accuracy in Z

Figure 3 shows that best-of-5 (Bo5) improves precision. Because the prompt schema is unchanged across checkpoints, the gap between early and late curves largely reflects learning from RL. In the code domain, the exemplar library also grows during RL, so gains reflect both a stronger policy and denser retrieval grounding. Code prompt ablations (App. Tab. 5) highlight the role of retrieval grounding. Removing exemplars or providing mismatched exemplars collapses 3D control (Bo5 $\leq$ 0.006), and dropping exemplars also sharply reduces the fraction of requests that yield any valid sample (ValidReq = 35.3%). By contrast, omitting the explicit `REQUESTED TARGET` line has little effect (Bo1 0.658 vs. 0.664 with the default prompt), which suggests that in code most conditioning is carried

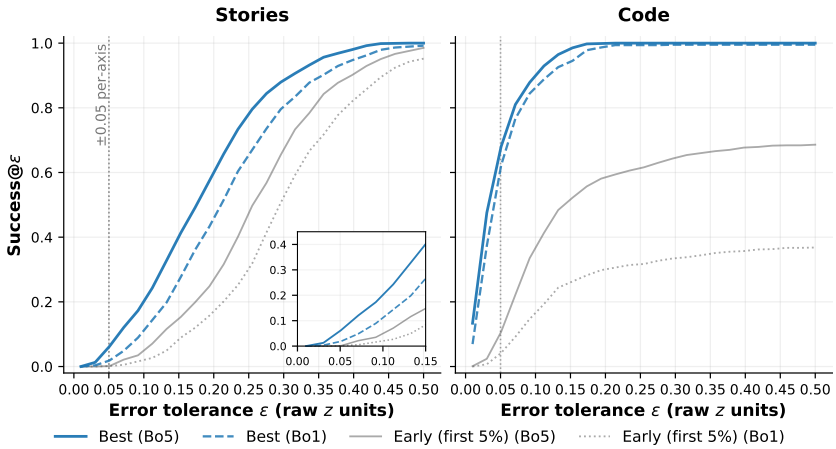

Figure 3: **Success–tolerance curves for 3D control in Z.** Dashed curves show one-shot sampling, and solid curves show best-of-5 at the same request. Malformed/invalid outputs count as failures. Best checkpoint (blue) against first 5% of the RL run (gray).

by exemplar selection rather than direct numeric-coordinate parsing. Calibration plots, with targets spanning $q_{0.01}$ to above $q_{0.99}$, show code tracking within ± 0.05 while stories are looser (Fig. 8, App. E).

4.2 Anchored axis z_1 and EQ-Bench Slop-Score

We use EQ-Bench Slop-Score (Paech, 2023, 2025) as a proxy for templated lexical patterns (colloquially "AI slop"). Because z_1 is explicitly anchored toward Qwen3-generated "default-style" stories relative to the corpus mean, we use Slop-Score primarily as a post-hoc axis-validation diagnostic (not as a general text-quality metric). App. F summarises analyzer details, and provides qualitative excerpts spanning realised z_1.

For final-10% 1D-control rollouts ($S = \{1\}$, $N \approx 7{,}000$), Slop-Score correlates with the requested target z_1^* with Pearson $r \approx 0.57$ ($p < 10^{-3}$). Fig. 4 shows an approximately linear trend: increasing z_1^* shifts the Slop-Score distribution upward by roughly 20–30 points over our target range, with substantial within-setting variability. To test whether this association is specific to the anchored axis, we also correlate Slop-Score with the realised coordinates (z_1, z_2, z_3) on the same samples (App. Fig. 9). Only z_1 shows a substantial association ($r \approx 0.56$), while correlations with z_2 and z_3 are small. We do not explore negative z_1 targets. In our STORIES corpus this half-space is dominated by low-structure diary-style prose, and pushing the small Qwen3-1.7B backbone into $z_1 \leq 0$ reduces coherence.

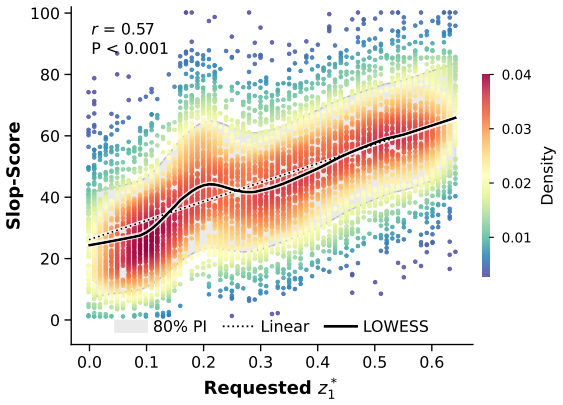

Figure 4: **Relationship between requested z_1^* and Slop-Score.** Slop-Score as a function of the requested target on the first axis, z_1^*, for completions from late-training rollouts (final 10% of GRPO updates) generated under 1D control ($S = \{1\}$) ($N \approx 7{,}000$). Points are coloured by local density. The solid line shows a LOWESS smooth, the dotted line the global linear regression fit ($r \approx 0.57$), and the shaded band an approximate 80% prediction interval around the LOWESS trend.

4.3 Story diversity under path-based baselines versus state-like target sweeps

We compare state-like target sweeping to a strong path-based multi-branch baseline on Writing-Prompts (Fan et al., 2018). As context, Avoidance Decoding applies a decoding-time similarity penalty and reports strong gains under its own harness (Park et al., 2025) (see Sec. 2). Because our backbone, prompt subset, and LLM judge differ, we restrict quantitative comparisons to our fixed harness.

Baselines. We compare to a negative-example prompt-chaining baseline used by Park et al. (2025)

Method	Self-BLEU↓	ROUGE-L↓	METEOR↓	Sent-Sim↓	LLMScore↑	Degen↓
Chained greedy ($T = 0$)	78.90	83.26	85.41	92.36	13.00	**0.06**
Chained temp sweep (best: $T = 0.9$, $p = 0.95$, $k = 20$)	58.45	66.20	70.31	87.89	17.35	**0.06**
Chained top-p sweep (best: $T = 0.6$, $p = 0.95$, $k = 20$)	44.74	54.37	59.83	84.25	18.45	0.07
Chained top-k sweep (best: $T = 0.6$, $p = 0.95$, $k = 40$)	63.96	71.05	74.82	89.38	13.95	**0.06**
OS-Search (Z-grid sweep, $2 \times 3 \times 3$)	**8.01**	**23.27**	**28.40**	**48.17**	**57.15**	0.09

Table 2: **Path-based decoding vs sweeping targets in Z.** Multi-branch diversity metrics computed per prompt and then averaged across prompts. Path-based baselines generate $B = 15$ branches per prompt via chaining with explicit negative examples. *OS-Search* sweeps a fixed target grid in Z_{text}. For comparability we report $B = 15$ branches per prompt (see App. G). Lower scores on Self-BLEU (Papineni et al., 2002; Zhu et al., 2018), ROUGE-L (Lin, 2004), METEOR (Banerjee and Lavie, 2005), and Sent-Sim indicate higher diversity, while higher LLMScore and lower Degen are desirable.

on the base Qwen3-1.7B thinking model with decoding-parameter sweeps to obtain $B = 15$ branches per prompt. Details are in App. G.

OS-Search. We generate branches with the *OS-Search* controller $\pi_\theta(\cdot \mid p, z^*)$ using a fixed sampler and varying only the requested target z^* in Z_{text} (details in App. G). Story targeting is coarse at tight tolerances (Fig. 3), but our grid sweep uses well-separated target settings. This is sufficient to push different branches into distinct regions of the frozen space and drives the diversity gains in Tab. 2.

Tab. 2 shows that sweeping targets in Z_{text} yields substantially higher diversity while keeping degeneration low: the strongest sampling baseline achieves LLMScore $= 18.45$ with Degen $= 0.07$, while *OS-Search* achieves LLMScore $= 57.15$ with Degen $= 0.09$. These gains are consistent across lexical and embedding overlap metrics, indicating substantially greater lexical and embedding-level diversity. Using the ratio ρ_{div} defined in App. G, we obtain

$$\rho_{\text{div}}(\text{OS-Search}) = \frac{57.15}{18.45} \approx 3.10.$$

4.4 Code-space search in Z_{code} under a withheld CA++ objective

We evaluate the code-domain *OS-Search* interface described in Sec. 3.3 on the withheld CA++ benchmark $f(x) \in [0, 1]$ (App. H).

Budget accounting and baselines. We measure evaluation budget by the number of scored valid programs, denoted N_{ok}. This counts candidates that parse, execute, return a valid board, and receive a CA++ score. Our only budget-matched baseline is path-based sampling from the base model (no z^*, Fig. 6, green). For reference (i.e., not budget-matched to the online search runs below), we also report the fixed N=188 program-library sweep

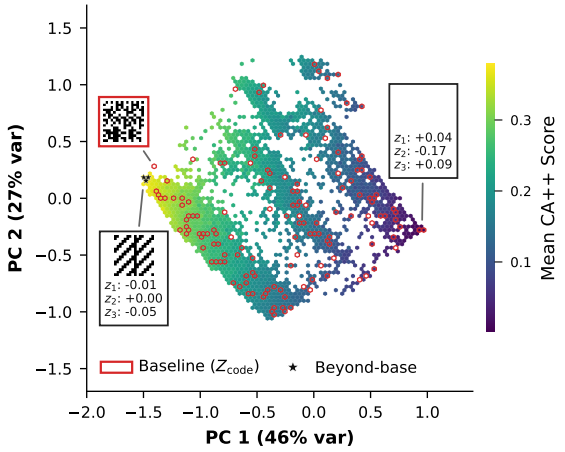

Figure 5: **Programs in CA++ behavioral space (code domain).** Each point is a valid `make_seed()` program scored by CA++. Axes are PCs of the CA++ trajectory-statistic feature vectors of the plotted programs (so proximity indicates similar CA dynamics), not Z_{code}. Hexbins show random-z^* controller samples coloured by mean score per bin. Red outlines mark bins reached by the N=188 program-library baseline, and stars mark bins whose mean exceeds the best baseline score (0.371). Insets show representative seeds generated by corresponding `make_seed()` implementations. The printed (z_1, z_2, z_3) values are the requested targets z^* for those programs.

used to fit Z_{code} (best 0.371), and a larger precomputed pool of random-z^* controller rollouts (valid N_{ok}=13,752, best 0.381), which we use to contextualize coverage and attainable scores.

Coverage and beyond-baseline regions. Random-z^* sampling reaches a broader region of CA behavioral space and uncovers compact high-score pockets beyond the library frontier (Fig. 5).

Outer-loop optimization over z^*: surrogate-update ablation. We run warm-started BO (GP–EI) over requested targets z^* (App. I). Consistent with the realised-coordinate interface described in Sec. 3.3, updating the surrogate at the realised coordinates $z(x_{\text{best}})$ of the best valid program per query outperforms updating at the request z^* in

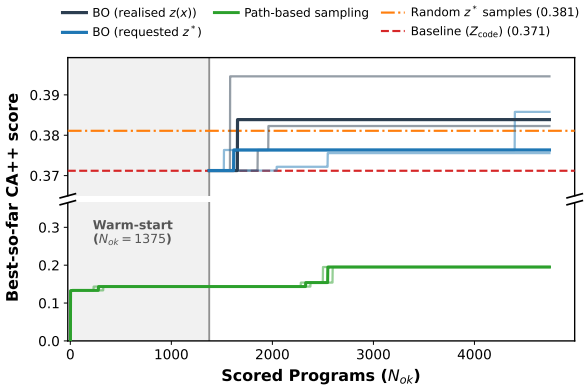

Figure 6: **Best-so-far CA++ score vs scored-program budget.** N_{ok} counts scored valid programs (invalid/malformed outputs are discarded). The grey segment indicates the warm-start phase for BO ($N_{\text{ok}} = 1375$). Thick lines show the median over three seeds, and thin lines show the min/max range (split y-axis for readability).

our surrogate-update ablation (App. Tab. 7). Under a matched N_{ok} budget, BO improves the best-so-far CA++ score up to 0.395 (Fig. 6), compared to 0.371 for the library and well above path-based sampling. For reference, Fig. 12 visualizes the best scoring seed (0.395). Additional Life-torus diagnostics are in App. I.

5 Discussion

OS-Search makes z^* a low-dimensional control knob: after training a controller to approximately hit requested coordinates in a frozen space Z, we can explore or optimize outputs by searching over targets in Z while keeping inference as standard autoregressive sampling.

Empirically, the interface supports two regimes. For stories, a small grid sweep in the anchored Z_{text} produces diverse, non-degenerate branches without prompt chaining (Tab. 2), and the anchored axis provides a stable reference axis (Fig. 4). For code, treating Z_{code} as a design space enables black-box optimization: random target sampling expands coverage and GP–EI over z^* improves a withheld executable objective (Fig. 6). These results suggest that outer-loop search should model actuator noise and update on realised $z(x)$ rather than on requests z^* (App. Tab. 7).

More broadly, *OS-Search* is a modular "outer-loop + evaluator" interface that is complementary to decoding-time control: it shifts work to training so inference can remain standard sampling, and it is most useful when a meaningful space and validity/evaluation signals exist. We expect larger backbones and richer exemplar sets may improve prompt-only use of targets. Our core contribution is the fixed encoder-defined output space Z as a per-sample search/control interface, with GRPO/GSPO training serving as one practical way to make z^* reliably actionable. Next steps include better spaces, stronger actuators, and outer loops beyond Bayesian optimization.

Because group-based RL objectives such as GRPO/GSPO depend on within-prompt diversity for informative relative rewards, *OS-Search* may also be useful during RL by replacing "turn up temperature" with structured target sweeps that deliberately span distinct solution or reasoning modes (e.g., sampling around an initial chain-of-thought-trajectory's $z(x)$ or running a small per-prompt BO on extremely hard prompts).

6 Limitations

OS-Search is an interface and feasibility study, and it comes with important limitations.

Code release scope. Code to construct the frozen output spaces and reproduce the code-domain experiments (including CA++ and the BO loop) is available at `https://github.com/TobiasMaterzok/OS-Search`. We intentionally do not release the story-domain training pipeline, datasets, or checkpoints due to the text-domain safety and dual-use risks discussed below. The released repository is not a mirror of the internal research codebase; components were removed and interfaces simplified for release, so story-domain scripts referenced in the paper are not included. Thus, the story-domain experiments are not fully reproducible from the public release; this is an intentional safety trade-off rather than an accidental omission.

Extreme tail targeting in text can elicit unsafe content. In text domains, the learned control axes in a frozen encoder-defined space may align with broad affective or genre-related factors in the underlying corpus. We observed that pushing requests into extreme tail regions (e.g., large-magnitude z^* on non-anchored axes such as z_2/z_3) can occasionally produce highly negative or graphic content. This can be amplified by retrieval grounding, since nearest-neighbour exemplars in those regions may themselves exhibit the same style and provide in-context steering toward it. Our training objective rewards target tracking and calibration but does not include an explicit content-safety term, so such

behaviours are not disincentivized by default.

Risks of extreme text targeting and dual use. OS-Search exposes a low-dimensional, reusable coordinate interface Z together with an outer-loop search mechanism (sweeps or black-box optimization over z^*). In text domains, this interface can make it easier to systematically search for generations with particular stylistic or rhetorical properties than ad-hoc prompt engineering, because the outer loop can iterate over targets without growing context and can reuse the same actuator across prompts. In the wrong hands, such search could be used to optimize for undesirable behaviors when paired with an automated evaluator. While our story experiments focus on diversity and on validating an anchored axis using Slop-Score as a diagnostic (not as an optimization target), richer spaces, larger backbones, or different anchor choices could enable more extreme forms of style/persona targeting. Practical deployments should therefore incorporate strong content filtering, careful curation/guardrails on the exemplar library used for retrieval grounding, monitoring for adversarial outer-loop objectives, and conservative bounds on target search regions. These risks are not unique to OS-Search, but the state-like target interface can reduce the friction of iterative search in generation space.

Meaning of Z depends on the encoder and corpus. The geometry and semantics of Z depend on the frozen encoder E, the corpus used to fit (μ, U), and embedding/preprocessing choices (e.g., chunking and pooling). Changing any of these can change neighborhoods, axis structure, and which regions are reachable or useful. This dependence is the price of having a stable external interface: portability to new domains requires rebuilding and revalidating the space.

Anchoring is pragmatic, not canonical. In Z_{text} we anchor z_1 with a small set of "default-style" stories to obtain a knob with a clear empirical correlate. Different anchors would produce different axes and may encode different stylistic biases. Beyond the anchored axis, PCA+Varimax can encourage simpler directions but does not guarantee human-interpretable factors.

Control is approximate and reachability is non-uniform. A requested target z^* is a request. Realised coordinates occupy a smaller, non-uniform subset of Z, and some target combinations may be effectively unattainable or attained only with low probability (Sec. 3.3). This is most visible in stories, where strict 3D targeting is coarse at tight tolerances (Fig. 3). Best-of-K sampling can improve targeting, but it increases inference cost. For optimization, the outer loop should model actuator noise and reachable support rather than assuming perfect control. Targets far outside the empirical support of the exemplar library should be treated as unsupported extrapolation: retrieval may no longer provide reliable local grounding, and failures in such regions should not be interpreted as evidence of meaningful controllability.

Dependence on retrieval-grounded prompting. Targets are grounded by retrieving exemplars near z^* (and a contrast example near $-z^*$) under the same frozen space (Fig. 2). Performance therefore depends on library coverage and on the prompt format's interaction with the base model. Ablating exemplar retrieval substantially degrades targeting and (in code) validity (App. Tab. 5). A more exhaustive ablation would further isolate retrieval-only effects and quantify sensitivity to exemplar count, selection strategy, and library quality.

Text evaluation focuses on probes and diversity, not end-to-end optimization. We use Slop-Score mainly as an axis-validation diagnostic (Sec. 4.2) and evaluate branching with overlap metrics and an LLM judge (Tab. 2). These metrics are informative but are not task-level objectives; the LLMScore-based diversity ratio should be interpreted as a within-harness proxy, we do not include human evaluation, and LLM-based judging can introduce model-dependent bias. We do not demonstrate an end-to-end outer-loop text task analogous to the code-domain optimization.

Baselines and domains are not exhaustive. Our story baselines are strong within our fixed harness (prompt-chaining with explicit negative examples plus decoding sweeps), but they do not cover the full space of decoding-time diversity methods or alternative branching pipelines. Broader comparisons would sharpen the empirical positioning.

The code benchmark is synthetic and heavily constrained. CA++ provides a clean executable testbed, but it is not representative of many real software engineering settings. Our programs are single-function and tightly constrained, and the validity gate is correspondingly specific. Extending *OS-Search* to richer objectives raises additional challenges in validity checking, space construction,

and search efficiency. We chose this setting after piloting several code objectives with the 1.7B parameter backbone and a limited GPU-memory budget: generating valid 16×16 boards reliably stayed within the model's capability and kept completion lengths manageable, whereas more realistic tasks often collapsed to very low validity rates and/or produced long `<think>` traces that were infeasible to train with under our constraints.

Compute, scaling, and safety considerations. *OS-Search* shifts effort to training (sequence-level RL plus repeated encoder calls) and outer-loop optimization may require many valid, scored samples when validity rates are low. We also do not compare QLoRA with full-parameter fine-tuning; stronger or larger actuators may improve targeting precision but would change the compute profile. Any method that enables systematic exploration of an LLM's output space can also be misused to search for undesirable behaviours if paired with an inappropriate evaluator. Practical deployments should combine this interface with domain-appropriate safety filters, constraints, and monitoring.

References

A. N. M. Nafiz Abeer, Nathan Urban, M. Ryan Weil, Francis J. Alexander, and Byung-Jun Yoon. 2024. Multi-Objective Latent Space Optimization of Generative Molecular Design Models. *Patterns*, 5(10):101042. ArXiv:2203.00526 [cs].

Amal Alabdulkarim, Winston Li, Lara J. Martin, and Mark O. Riedl. 2021. Goal-Directed Story Generation: Augmenting Generative Language Models with Reinforcement Learning. *arXiv preprint*. ArXiv:2112.08593 [cs].

Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. 2024. Refusal in Language Models Is Mediated by a Single Direction. *arXiv preprint*. ArXiv:2406.11717 [cs].

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024. AlphaMath Almost Zero: Process Supervision without Process. *arXiv preprint*. ArXiv:2405.03553.

Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems 30*, pages 4299–4307.

Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. 2024. Sparse Autoencoders Find Highly Interpretable Features in Language Models. In *International Conference on Learning Representations (ICLR)*.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and Play Language Models: A Simple Approach to Controlled Text Generation. In *International Conference on Learning Representations (ICLR)*.

Daya Guo, Dejian Yang, Haowei Zhang, et al. 2025. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. In *Advances in Neural Information Processing Systems 36*.

Hanxing Ding, Liang Pang, Zihao Wei, Huawei Shen, Xueqi Cheng, and Tat-Seng Chua. 2023. MacLaSa: Multi-Aspect Controllable Text Generation via Efficient Sampling from Compact Latent Space. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 4424–4436, Singapore. Association for Computational Linguistics.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical Neural Story Generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia. Association for Computational Linguistics.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. 2022. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53.

Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus Mcaleer, Ying Wen, Weinan Zhang, and Jun Wang. 2024. AlphaZero-Like Tree-Search can Guide Large Language Model Decoding and Training. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 49890–49920. PMLR.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. 2025. Scaling and evaluating sparse autoencoders. In *International Conference on Learning Representations*.

Yuxuan Gu, Xiaocheng Feng, Sicheng Ma, Lingyuan Zhang, Heng Gong, Weihong Zhong, and Bing Qin. 2023. Controllable Text Generation via Probability

Density Estimation in the Latent Space. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12590–12616, Toronto, Canada. Association for Computational Linguistics.

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. 2018. Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. *ACS Cent. Sci.*, 4(2):268–276.

Michael Günther, Jackmin Ong, Isabelle Mohr, Alaeddine Abdessalem, Tanguy Abel, Mohammad Kalim Akram, Susana Guzman, Georgios Mastrapas, Saba Sturua, Bo Wang, Maximilian Werk, Nan Wang, and Han Xiao. 2023. Jina Embeddings 2: 8192-Token General-Purpose Text Embeddings for Long Documents. *arXiv preprint*. ArXiv:2310.19923 [cs].

Daniel Han, Michael Han, and the Unsloth team. 2023. Unsloth. GitHub repository. Accessed: 2025-07-29.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Patrik O. Hoyer. 2004. Non-negative Matrix Factorization with Sparseness Constraints. *Journal of Machine Learning Research*, 5(Nov):1457–1469.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

Henry F. Kaiser. 1958. The Varimax Criterion for Analytic Rotation in Factor Analysis. *Psychometrika*, 23(3):187–200.

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. CTRL: A Conditional Transformer Language Model for Controllable Generation. *arXiv preprint*. ArXiv:1909.05858 [cs].

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. 2021. GeDi: Generative Discriminator Guided Sequence Generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4929–4952, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, pages 611–626.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Xun Liang, Hanyu Wang, Yezhaohui Wang, Shichao Song, Jiawei Yang, Simin Niu, Jie Hu, Dan Liu, Shunyu Yao, Feiyu Xiong, and Zhiyu Li. 2024. Controllable Text Generation for Large Language Models: A Survey. *arXiv preprint*. ArXiv:2408.12599 [cs].

Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Understanding R1-Zero-Like Training: A Critical Perspective. *arXiv preprint*. ArXiv:2503.20783 [cs].

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

Tobias Materzok. 2025. COS(M+O)S: Curiosity and RL-Enhanced MCTS for Exploring Story Space via Language Models. *arXiv preprint*. ArXiv:2501.17104 [cs].

Kolby Nottingham, Ruo-Ping Dong, Ben Kasper, and Wesley N. Kerr. 2024. Improving Branching Language via Self-Reflection. In *NeurIPS 2024 Workshop on Language Gamification (LanGame)*.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R.

Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint*. ArXiv:2506.13131 [cs].

OpenAI. 2026. GPT-5.1 Model | OpenAI API. Accessed: 2026-01-14.

OpenAI. 2024. OpenAI o1 System Card. *arXiv preprint*. ArXiv:2412.16720 [cs.AI].

Samuel J. Paech. 2023. EQ-Bench: An Emotional Intelligence Benchmark for Large Language Models. *arXiv preprint*. ArXiv:2312.06281 [cs].

Samuel J. Paech. 2025. slop-score. GitHub repository. Accessed: 2025-11-22.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Kyeongman Park, Nakyeong Yang, and Kyomin Jung. 2025. Avoidance Decoding for Diverse Multi-Branch Story Generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7489–7505, Suzhou, China. Association for Computational Linguistics.

William H. Paulsen. 2003. Creating Large Life Forms with Interactive Life. *Complex Systems*, 14(3):275–283.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint*. ArXiv:2402.03300 [cs].

Ye Shi and C. S. George Lee. 2024. Uniform Transformation: Refining Latent Representation in Variational Autoencoders. *arXiv preprint*. ArXiv:2407.02681 [cs] version: 1.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv preprint*. ArXiv:1712.01815.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. MPNet: Masked and Permuted Pre-training for Language Understanding. In *Advances in Neural Information Processing Systems 33*.

Nishant Subramani, Nivedita Suresh, and Matthew Peters. 2022. Extracting Latent Steering Vectors from Pretrained Language Models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 566–581, Dublin, Ireland. Association for Computational Linguistics.

Trieu H. Trinh and Quoc V. Le. 2019. A Simple Method for Commonsense Reasoning. *arXiv preprint*. ArXiv:1806.02847v2 [cs.AI].

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. 2024. Steering Language Models With Activation Engineering. *arXiv preprint*. ArXiv:2308.10248 [cs].

Zhihua Wen, Zhiliang Tian, Wei Wu, Yuxin Yang, Yanqi Shi, Zhen Huang, and Dongsheng Li. 2023. GROVE: A Retrieval-augmented Complex Story Generation Framework with A Forest of Evidence. *arXiv preprint*. ArXiv:2310.05388 [cs].

Stephen Wolfram. 1984. Computation theory of cellular automata. *Communications in Mathematical Physics*, 96(1):15–57.

Zhengxuan Wu, Aryaman Arora, Atticus Geiger, Zheng Wang, Jing Huang, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. 2025. AxBench: Steering LLMs? Even Simple Baselines Outperform Sparse Autoencoders. In *International Conference on Machine Learning (ICML)*.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P. Lillicrap, Kenji Kawaguchi, and Michael Shieh. 2024. Monte Carlo Tree Search Boosts Reasoning via Iterative Preference Learning. *arXiv preprint*. ArXiv:2405.00451.

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, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 Technical Report. *arXiv preprint*. ArXiv:2505.09388 [cs].

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. 2026. Learning to Discover at Test Time. *arXiv preprint*. ArXiv:2601.16175 [cs].

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. 2025. Group Sequence Policy Optimization. *arXiv preprint*. ArXiv:2507.18071 [cs].

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. 2018. Texygen: A Benchmarking Platform for Text Generation Models. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1097–1100.

Appendices

A Output-space construction and evaluation

We detail the construction of the frozen 3D output coordinates used throughout the paper. For each domain we fix a text/code encoder $E(\cdot)$, fit a linear projection, and define

$$z(x) \;=\; U^\top\big(E(x)-\mu\big) \in \mathbb{R}^{d_z}, \qquad d_z = 3,$$

with corpus mean μ and an orthonormal basis $U \in \mathbb{R}^{D\times d_z}$. We freeze (E, μ, U) for all RL experiments.

Corpora. Z_{text} **(stories).** We combine multiple public English story datasets (short stories and longer narratives), including the STORIES corpus of Trinh and Le (2019). We keep only text fields and filter documents to roughly 800–25,000 characters.

Z_{code} **(code).** We build a task-specific library of $N = 188$ diverse Python implementations of the constrained function `make_seed() -> list[str]` and fit Z_{code} on this set. We use a task-specific corpus because spaces fit on general-purpose code corpora tended to allocate axes to variations that are irrelevant or disallowed under our `make_seed` constraints.

Encoders and pooling. Stories are embedded with a `sentence-transformers` encoder based on all-mpnet-base-v2 (Reimers and Gurevych, 2019; Song et al., 2020). Code is embedded with jina-embeddings-v2-base-code (Günther et al., 2023). For long inputs we use automatic chunking. We mean-pool chunk embeddings and apply ℓ_2 normalization.

PCA and rotation template (both domains). Given a corpus of N texts/programs, we embed to a matrix $X \in \mathbb{R}^{N\times D}$ and set $\mu = \frac{1}{N}\sum_i X_i$. We run PCA on the centered embeddings to an intermediate dimension d_{inter}, yielding orthonormal components $C_{\text{inter}} \in \mathbb{R}^{D\times d_{\text{inter}}}$ and eigenvalues $\lambda_1, \ldots, \lambda_{d_{\text{inter}}}$. Optionally, we whiten inside the PCA space with $s_j = (\lambda_j + \varepsilon)^{-1/2}$ and $S = \mathrm{diag}(s_1, \ldots, s_{d_{\text{inter}}})$. We then choose $d_z = 3$ directions inside this intermediate space (with anchoring for stories, no anchoring for code), map them back to the encoder space, and orthonormalize to obtain U.

Finally, we apply an orthogonal rotation within the retained d_z-dimensional subspace (Varimax, anchored Varimax for stories) to encourage simpler axis loadings. This preserves the projector $UU^\top$ and changes only the reported axis orientation. For code, we apply this template with no anchoring, fitting on the $N = 188$ valid `make_seed()` library.

Anchored story space Z_{text}**.** To make one axis have a stable semantic reference, we anchor the first axis toward a small set of Qwen3-generated "default-style" short stories. Let $\mathcal{A}$ be the anchor set and $a = \frac{1}{|\mathcal{A}|}\sum_{x\in\mathcal{A}} E(x)$ be their mean embedding. Define the anchor direction relative to the corpus mean as $a_{\text{dir}} = a - \mu$. We project this direction into the intermediate PCA space (and apply whitening if enabled) and normalize it to obtain a unit anchor $w \in \mathbb{R}^{d_{\text{inter}}}$. We then choose the remaining $(d_z - 1)$ axes to capture as much residual PCA variance as possible subject to orthogonality to w (i.e., we take the leading eigenvectors of the PCA-variance matrix after projection onto $w^\perp$). After mapping back and orthonormalization, we apply an anchored Varimax rotation: we keep the first axis aligned with the anchor direction and Varimax-rotate only the remaining $d_z - 1$ axes so that z_2 and z_3 are easier to interpret while preserving the

anchored z_1.

Model selection and diagnostics. We select hyperparameters via a small grid search over: $d_{\text{inter}} \in \{8, 10, 12, 14, 16, 18, 20, 24, 28, 32, 36, 40, 48\}$, $d_z \in \{3, 4, 5, 6, 7, 8\}$ (with $d_z \leq d_{\text{inter}}$), whitening on/off, and (for stories) anchoring on/off. For each configuration we compute:

- **Reconstruction:** mean $\|x - \hat{x}\|_2$ where $\hat{x} = \mu + UU^\top(x - \mu)$, and reconstruction R^2 (relative to centered embeddings).
- **Neighborhood preservation:** k-NN recall and trustworthiness in Z at $k \in \{10, 20\}$.
- **Axis diagnostics:** per-axis variance and mean Hoyer sparsity (averaged across axes) (Hoyer, 2004).
- **Stories only:** anchor capture $\|U^\top a_{\text{dir}}\|_2 / \|a_{\text{dir}}\|_2$ and the fraction of captured anchor energy that lies on z_1.

A scalar preference score combines these quantities with fixed weights (reported in the code release) and a mild quadratic penalty on $(d_z - 3)^2$. For anchored story configurations, we discard settings whose anchor capture is below a fixed threshold or whose captured anchor does not load primarily onto z_1. The best configuration per domain is used in all subsequent experiments.

B Prompt format, parsing, and embedding pipeline

Structured envelope and parsing. Completions follow the structured envelope in Fig. 2. A lightweight parser enforces the required fields and checks that `<target>` contains numeric values for the requested axes. Malformed outputs receive zero reward via R_{format}. For all downstream computations we ignore `<think>` and `<title>` and embed only the task-content field `<text>`.

Retrieval grounding for numeric targets. Given a base prompt p and a requested target z^*_S, we retrieve exemplars from the corresponding library indexed in the frozen space using Euclidean nearest neighbours in Z: two exemplars nearest to z^* and one contrast exemplar nearest to $-z^*$. We insert these exemplars together with the numeric request into the prompt (Fig. 2, Sec. 3.2). In exemplar blocks, `<target>` contains stored realised coordinates $z(x_j)$, in model outputs, `<target>` contains the self-report $\hat{z}_S$.

Computing realised coordinates $z(x)$. After generation, realised coordinates are computed by embedding only `<text>` and projecting into the corresponding frozen space. For stories, we embed `<text>` with E_{text} and project to Z_{text}. For code, we deterministically extract the `make_seed()` function definition from `<text>`, strip comments, embed the extracted code with E_{code}, and project to Z_{code}.

Code constraints, validation, and scoring. In the code domain, `<text>` must implement a single function `make_seed() -> list[str]` that returns exactly 16 strings of length 16 over `'.'` and `'#'`. We extract `make_seed()` with a deterministic parser and treat a completion as valid only if it passes AST sanitization and executes in a restricted environment to return a correctly formatted 16×16 board. Invalid candidates are discarded (and receive zero reward). Valid programs are scored by the cellular-automaton evaluator in App. H.

In two brief warm-start stages, we train on library programs by asking the model to state brief hypotheses about which code features influence each axis and to predict the program's z-coordinates, using far-away library examples as context to prevent trivial copying.

Curriculum and reward pointers. Across both domains, training requests vary the constrained-axis subset S and target values z^*_S according to App. C. Reward components (format checks, distance shaping, and self-report scoring) are implemented as detailed in App. D.

C Curriculum and target sampling

One training "request" and phase budget. We define a request as a sampled pair (S, z^*). The set $S \subseteq \{1, 2, 3\}$ specifies which axes are constrained, and z^* gives the requested target in the frozen space. Because the controller is trained with the retrieval-grounded prompt schema, we can request arbitrary z^* values and retrieve the closest exemplars from the current library to instantiate the prompt.

Each target-hitting curriculum phase contains 15,000 distinct prompt–target requests. Concretely, we run 1,500 GRPO optimizer steps with gradient accumulation 10.

Phase schedule (stories vs. code). The curriculum differs slightly by domain:

- **Stories:** three phases that increase control dimensionality: $S = \{1\}$, then $S = \{1, 2\}$, then $S = \{1, 2, 3\}$.

- **Code:** an interleaved procedure: (i) a warm-start phase on the fixed $N = 188$ `make_seed` library to train coordinate prediction and axis-level hypotheses, then (ii) target-hitting training with full 3D control $S = \{1, 2, 3\}$, then (iii) a second brief warm-start phase (same prediction format), then (iv) a final target-hitting phase.

Exemplar library initialization and growth. We initialize the exemplar library by embedding a fixed corpus and keeping documents from the outer quantile bands of each axis, so the initial library covers mostly extreme regions of Z. For code, we additionally grow this library during GRPO by adding valid model-generated `make_seed` programs. During RL we also penalize near-duplicate copying of retrieved exemplars (App. D).

Frozen axis statistics and target ranges. We precompute frozen per-axis statistics from a large embedded reference set (empirical quantiles and IQR). For each axis i we define a robust scale

$$s_i = \max\left(|q_{0.10}|,\ |q_{0.90}|,\ 10^{-4}\right),$$

and freeze (s_1, s_2, s_3) for the entire training run. These scales are used both to sample requested targets and to normalize distance violations in the reward. Targets are drawn from an axis-aligned hypercube whose side lengths are proportional to (s_i) by choosing a sign and sampling a magnitude from a fixed band of multiples of s_i, intentionally reaching into tail regions (often up to and beyond $q_{0.99}$ on skewed axes). For context, Fig. 8 overlays reference-corpus quantiles $(q_{0.01}, \ldots, q_{0.99})$. We request magnitudes up to $1.75\, s_i$ for text and $1.5\, s_i$ for code, so some requests lie beyond $q_{0.99}$ on heavy-skew axes, i.e., we deliberately train on extrapolation into rare tail regions of the corpus.

Prototype targets and within-phase shaping. Early in training we anchor sampling on a small set of prototype centers in Z (a few dozen sign patterns over the axes, scaled by the frozen (s_i)). Later batches draw targets by sampling magnitudes from a bounded band around the origin and choosing a random subset of axes to constrain, with a bias toward non-negative z_1 for stories.

Each request also carries a per-example distance exponent α (Fig. 7) that controls how sharply reward concentrates around the target. Within each curriculum phase, α is linearly annealed from 1.5 to 0.8 over the 1,500 RL updates, then reset to 1.5 at the start of the next phase (from roughly MSE-like to more sharply peaked near the target).

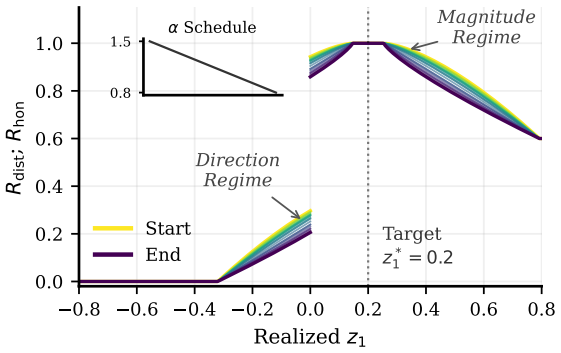

Figure 7: **Evolution of the curriculum reward surface.** Reward profiles for a target value $z_1^* = 0.2$ are shown within a single curriculum phase. In each phase, the distance exponent α is linearly annealed from 1.5 to 0.8 over the 1 500 RL updates (inset), progressively narrowing the reward basin in the magnitude regime to demand higher precision later in that phase, while maintaining a distinct penalty step in the direction regime (sign mismatch). At the start of every new phase, α is reset to 1.5.

D RL configuration and reward implementation

Reward implementation. We compute R_{format}, R_{dist}, and R_{hon} as defined in Sec. 3.2, using the parsing/embedding pipeline in App. B and frozen target scales from App. C. Samples failing the format/validity gate receive reward zero. Both R_{dist} and R_{hon} are down-weighted when the `<think>` trace or `<text>` block are very short, so trivial outputs receive no distance or honesty credit. In addition, we scan `<text>` for explicit mentions of the control interface (e.g., mentions of Z or axis coordinates). If such a leak is detected we skip the embedding step and set $R_{\text{dist}} = R_{\text{hon}} = 0$.

For code, when exemplar retrieval is used, we additionally penalise copying of retrieved guidance examples. Concretely, we detect near-duplicate outputs relative to any retrieved exemplar and apply zero reward to discourage verbatim replication of the guidance library.

Sampling and training. To enable fine-tuning under our computational constraints, we use Unsloth (Han et al., 2023) with 4-bit Quantized Low-Rank Adaptation (QLoRA) (Hu et al., 2022; Dettmers et al., 2023). We set the LoRA rank to $r = 64$ and scaling factor $\alpha = 64$, applying LoRA to all linear layers.

For RL, we use vLLM for efficient sampling (Kwon et al., 2023) with temperature 0.8,

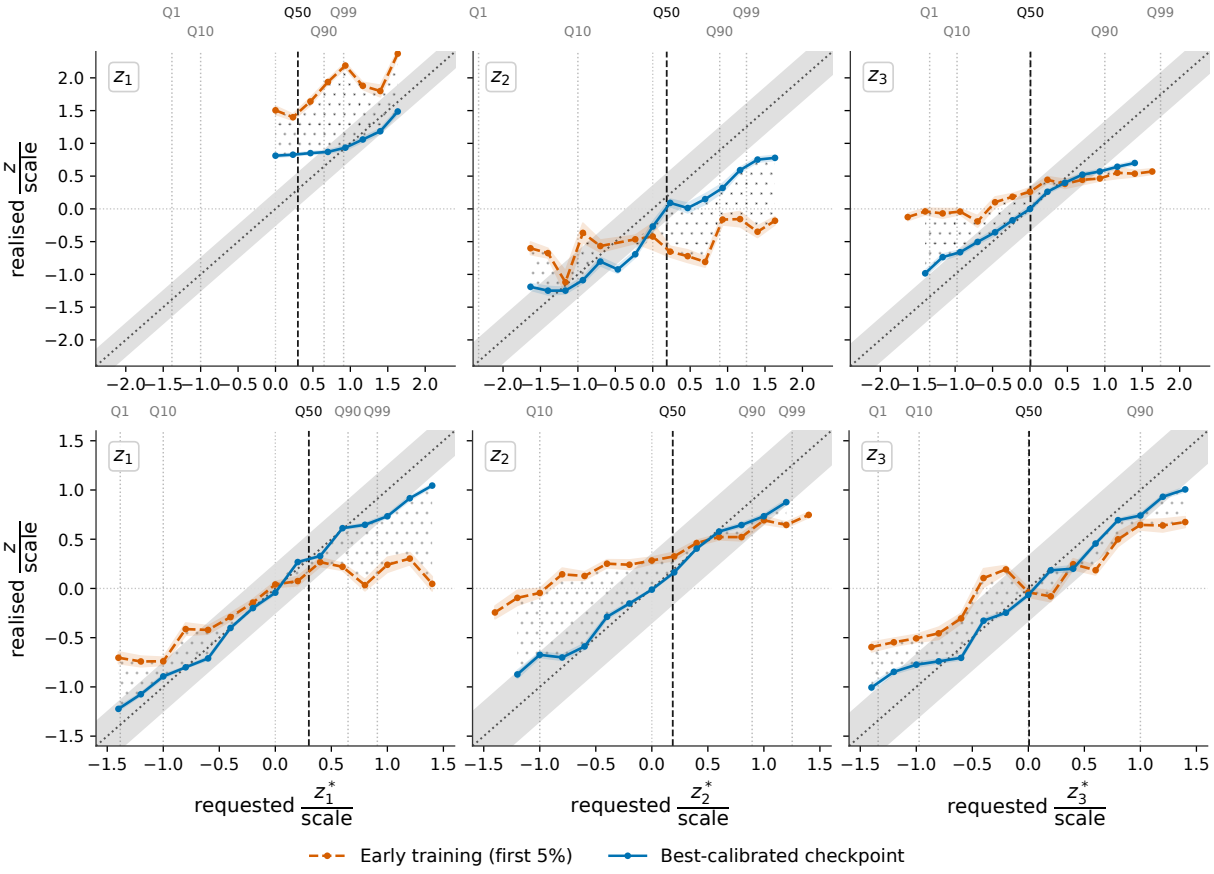

Figure 8: **Calibration of state-like control in Z.** Binned mean realised coordinate z_i versus requested target z_i^* for each constrained axis. The diagonal indicates perfect calibration. Blue: best checkpoint. Orange: early/baseline (first 5% of RL). The grey band marks the reference tolerance $|z_i - z_i^*| \leq 0.05$ (raw z units). Vertical markers show reference-corpus quantiles $(q_{0.01}, \ldots, q_{0.99})$ of z_i Curves are computed from 7,000 sampled evaluation completions per domain (usable records: stories $N = 6{,}946$, code $N = 6{,}860$). **Top:** story policy, fixed exemplar library. **Bottom:** code policy, exemplar library grows during RL.

top-$p = 0.95$, and top-$k = 40$ for text. For code we use temperature 1.0, top-$p = 0.95$, and top-$k = 50$. We generate up to a fixed maximum number of tokens per completion (1600 token output limit) and stop at the model's end-of-sequence token.

We train QLoRA adapters with per-device batch size 1, 7 generations per prompt, 10 gradient accumulations, learning rate in the range $2 \cdot 10^{-5}$, AdamW (Kingma and Ba, 2015; Loshchilov and Hutter, 2019) with weight decay 0.01, and a constant-with-warmup schedule 0.1. We use group-relative policy optimization (GRPO) (Shao et al., 2024; Guo et al., 2025) with GSPO-style sequence-level importance sampling and sequence-level clipping (Zheng et al., 2025), using a clipping window $(\varepsilon, \varepsilon_{\text{high}}) = (0.01, 0.03)$. Reward scaling is disabled following DR-GRPO (Liu et al., 2025), so the geometry of the composite reward is preserved.

E Targeting accuracy and calibration

We report targeting accuracy for the *OS-Search* controllers (text and code) and calibration diagnostics for the frozen output spaces Z_{text} and Z_{code}.

Metrics. For a sample $x \sim \pi_\theta(\cdot \mid p, z^*)$ with constrained-axis set $S \subseteq \{1, 2, 3\}$, define the per-axis absolute error $e_i(x, z^*) = |z_i(x) - z_i^*|$ for $i \in S$ and the joint (worst-axis) error $e(x, z^*; S) = \max_{i \in S} e_i(x, z^*)$. We use $\varepsilon = 0.05$ as a tight reference tolerance ("± 0.05 per-axis" under the ℓ_∞ joint error). All errors are in raw z units.

For per-axis results we report $\text{Success@}\varepsilon = \mathbb{P}[e_i \leq \varepsilon]$ estimated over constrained axis instances. For joint results we report $\text{Success@}\varepsilon = \mathbb{P}[e \leq \varepsilon]$ estimated over requests (with the relevant constrained set S).

Evaluation protocol. We evaluate each domain on 7,000 sampled requests. A completion is counted as usable if it passes the output-envelope

Setting	med e	p90 e	Success@0.05
Stories (per-axis, $e = \lvert z_i - z_i^* \rvert$)			
z_1	0.071	0.165	0.364
z_2	0.080	0.193	0.326
z_3	0.053	0.131	0.471
Code (per-axis, $e = \lvert z_i - z_i^* \rvert$)			
z_1	0.0203	0.072	0.804
z_2	0.0193	0.080	0.787
z_3	0.0211	0.083	0.780

Table 3: **Per-axis targeting error at $\varepsilon = 0.05$ (best checkpoints).** Per-axis statistics pool constrained axis instances. Errors are in raw z units.

Setting	med e	p90 e	Success@0.05
Stories (joint, $S = \{1, 2, 3\}$)			
3D	0.127	0.214	0.066
Code (joint, $S = \{1, 2, 3\}$)			
3D	0.032	0.102	0.667

Table 4: **Joint targeting error at $\varepsilon = 0.05$ (best checkpoints).** Joint error is in raw z units, reported for full 3D control ($S = \{1, 2, 3\}$).

parser and yields a valid `<text>` block that can be embedded (and for code, the extracted `make_seed()` implementation must pass AST sanitization and execute to return a valid 16×16 board). Calibration curves and error summaries below are computed on usable completions (stories: $N = 6{,}946$, code: $N = 6{,}860$).

Calibration curves. Fig. 8 reports binned mean realised coordinates $z_i(x)$ versus requested targets z_i^* for each constrained axis, for an early checkpoint and the best checkpoint. Because the prompt interface is unchanged across checkpoints (numeric targets plus exemplar retrieval), the improvement reflects sequence-level RL training. In the code domain it also reflects the growing exemplar library used for retrieval during RL.

Absolute targeting error at $\varepsilon = 0.05$. The error summaries in Tabs. 3 and 4 are computed on the same 7,000 sampled evaluation requests used for the calibration curves in Fig. 8. Tab. 3 reports per-axis error statistics and per-axis Success@0.05 over constrained axis instances. Tab. 4 reports joint error under full 3D control ($S = \{1, 2, 3\}$), matching the joint-error definition used in Fig. 3.

Exemplar grounding ablation (code). Because $z(x)$ is a frozen, arbitrarily rotated coordinate map, the numeric request z^* is not inherently meaningful to the model without grounding. We therefore ablate the prompt interface at inference time while holding the generator fixed: all variants use the same best-checkpointed code controller and the final grown exemplar library from the corresponding RL run, and we evaluate on $N_{\text{req}} = 1000$ random targets drawn from the same $1.5\times$ $(q_{0.10}, q_{0.90})$-scale box used by the code-domain outer-loop search (Sec. I). We report request-level 3D Success@0.05 for one-shot (Bo1) and best-of-5 (Bo5), where Bo5 selects the sample with the smallest joint error and invalid outputs count as failures.

Tab. 5 shows three clear patterns. First, retrieved exemplars are essential: removing exemplars or providing mismatched exemplars collapses targeting (Bo5 ≤ 0.006) and, without exemplars, validity also degrades sharply (only 35.3% of requests yield any valid sample under Bo5). Second, the explicit numeric target line is largely redundant once exemplars are dense and correct: hiding the `REQUESTED TARGET` line yields essentially unchanged performance (Bo1 0.658 vs. 0.664, Bo5 0.717 vs. 0.716). Third, the exemplar block structure matters: anonymizing the near/contrast labels and shuffling their order reduces success to 0.343 (Bo5), suggesting the policy has learned to use the local neighborhood and the prompt semantics (near vs. opposite), not just generic in-context examples. Overall, in the code setting the control signal is carried primarily through exemplar selection in Z_{code} rather than direct numeric-coordinate parsing, consistent with the retrieval-grounded design of *OS-Search*.

F Anchored story axis diagnostics

EQ-Bench Slop-Score implementation details. We use the EQ-Bench Slop-Score (Paech, 2023, 2025) as an automatic proxy for stereotyped, templated "AI slop" writing. The analyzer compares a model completion against a fixed list of overrepresented slop words and trigrams, and also detects overused contrast constructions of the form "not X, but Y" using a two-stage pattern-matching pipeline (regex followed by a lightweight POS-tag-based check) (Paech, 2025). The final Slop-Score is a weighted composite of these components (60% slop words, 25% not-X-but-Y patterns, 15% slop trigrams), with higher scores indicating more stereotyped language.

Qualitative samples along the z_1^* axis. We include a compact qualitative sanity check for the anchored story axis. Fig. 10 shows three independent

Prompt variant	ValidReq(%)	Success@0.05 (Bo1)	Success@0.05 (Bo5)
Default (target + exemplars)	98.6	0.664	0.716
Target hidden†	98.6	0.658	0.717
Anonymized exemplars	97.3	0.315	0.343
Mismatched exemplars	96.0	0.003	0.006
No exemplars	35.3	0.000	0.001

Table 5: **Code-domain prompt ablations at $\varepsilon = 0.05$ (request-level 3D control).** All variants use the same best-checkpointed code controller and the final grown exemplar library. We ablate only the prompt interface at inference time. Targets are $N_{\text{req}} = 1000$ random draws from the $1.5\times$ per-axis scale search box (based on the frozen s_i, App. C). ValidReq is the fraction of requests that yield at least one valid sample under Bo5. Invalid outputs count as failures for Success@0.05. †Exemplars are still retrieved using the held-out request z^*, but the explicit `REQUESTED TARGET` line is omitted from the prompt.

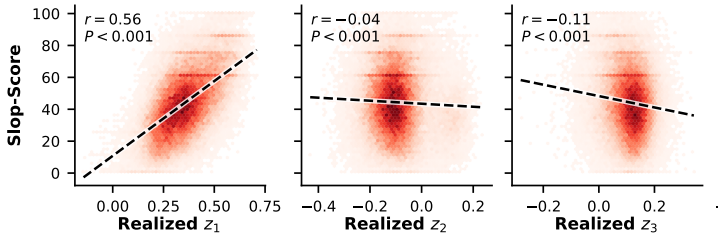

Figure 9: **Slop-Score vs realised coordinates in Z.** Hexbin plots of Slop-Score against realised coordinates z_1, z_2, z_3 for all story completions. Each panel shows a global linear regression fit (black dashed line) and the corresponding Pearson correlation r with two-sided p-value. Only the first axis exhibits a strong positive association with Slop-Score ($r \approx 0.56$, left), while correlations on the remaining axes are small ($r \approx -0.04$ and -0.11 for z_2 and z_3, respectively). For these samples we constrain only the anchored axis z_1 (i.e., $S = \{1\}$), z_2 and z_3 are unconstrained and shown as diagnostics of the frozen space.

completions to the same GRPO training prompt, selected to span low/mid/high realised z_1 (with z_2, z_3 roughly matched) under the same harness as Sec. 4.2. For space we show only opening and ending excerpts.

G Story decoding sweeps for path-based baselines

We describe the decoding sweeps used to construct the path-based multi-branch baselines in Sec. 4.3. These baselines follow the negative-example prompt-chaining recipe of Avoidance Decoding (Park et al., 2025) but are evaluated under our fixed harness.

Model and prompts. All experiments in this appendix use the Qwen3-1.7B thinking model with its default chat template. We use a fixed set of 20 prompts from WritingPrompts (Fan et al., 2018) and treat each prompt as an independent task. This matches the dataset used by Park et al. (2025) but not necessarily their exact prompt subset.

Chaining with explicit negative examples. Following the baseline configuration in Avoidance Decoding, we use iterative multi-branch generation with explicit negative examples in the prompt. For a given prompt, branch b is generated from a user message of the form

> You must always only respond with the story, no boilerplate.
> Please write a story from the following prompt.
> Do NOT generate responses that resemble the following examples:
> [1] story_1
> [2] story_2
> ...
> PROMPT: <*original prompt*>

where `story_1`, ..., `story_b-1` are the previously generated branches for that prompt, stripped of any `<think>` content. We include all earlier branches as negatives. This procedure reproduces the "feed all previously generated outputs back into the input" sampling baselines in Park et al. (2025).

Family	Swept parameter	Values	Fixed parameters
Temperature	T	$\{0.5, 0.7, 0.9\}$	top-$p = 0.95$, top-$k = 20$
Top-p	p	$\{0.90, 0.95, 0.98\}$	$T = 0.6$, top-$k = 20$
Top-k	k	$\{10, 20, 40\}$	$T = 0.6$, top-$p = 0.95$
Greedy	–	–	$T = 0$, no stochastic sampling

Table 6: **Decoding grids for story baselines.** Each configuration defines one run identified by its (T, p, k) hyperparameters and an associated sweep group (`temp`, `top-p`, `top-k`, or `greedy`).

Decoding grids. We sweep three families of sampling parameters plus a greedy baseline, as summarised in Tab. 6.

GRPO training prompt template: *Write a unique story with around 500 words!*

Low z_1
$(z_1, z_2, z_3) = (0.21, -0.08, 0.05)$
(Excerpt)
The morning sun spilled golden light through the kitchen window as I stirred the pot simmering in the saucepan. The recipe for my grandmother's famous plum sauce had been passed down through generations, each step a testament to patience and care. I had always assumed I could replicate her delicate art, but today the ingredients felt heavier, weighted by years of expectation. First, the apples: a blend of tart and sweet, peeled and sliced with meticulous care.

[. . . middle omitted . . .]

By afternoon, the apples had softened, their edges no longer rigid. I let them take their place in the simmering pot, refusing to force them into submission. In the end, the sauce was not what I expected, but it felt right—familiar, and profoundly different.

Slop-Score: 34

Mid z_1
$(z_1, z_2, z_3) = (0.36, -0.13, 0.02)$
(Excerpt)
The day following the storm, the city felt different. Streets that had once teemed with hurried footsteps now carried the stillness of a clock stopped. The clock above the town square had been the only constant in the chaos—its brass hands trembling slightly as if remembering something.

[. . . middle omitted . . .]

I laid the gear in a wooden box, where it would be found by someone else—maybe a child, or an old man who had no love for clocks. But in that box, the clock's final tick was not lost; it had only been waiting for the day to come when it would be forgotten.

Slop-Score: 54

High z_1
$(z_1, z_2, z_3) = (0.52, -0.14, 0.08)$
(Excerpt)
The storm had passed, leaving the town of Elmsworth in a quiet hush. Rain drizzled the windows, turning them into shivering lattices, while the street, once a thoroughfare for hurried lives, stood abandoned. The manor was a relic of a time when stone walls had guarded secrets and doors led to worlds beyond sight.

[. . . middle omitted . . .]

I had followed the echoes, not to escape, but to understand that some mysteries were meant to be shared. The Labyrinth of Echoes had shown me that the past and present were not separate, but intertwined. And as I turned back to the world that had been, I carried with me the knowledge that truth, once discovered, could bridge the gaps between time, memory, and the infinite.

Slop-Score: 82

Figure 10: **Qualitative excerpts along the anchored story axis z_1 (GRPO training prompt template).** Three independent completions to the same prompt template used during GRPO training, shown at increasing realised z_1 (with z_2, z_3 roughly matched). We report frozen encoder-defined coordinates $z(x)$ in Z_{text} and show only short excerpts (opening + ending) for space. The shift toward more "default-style"/templated narration at higher z_1 is consistent with the Slop-Score trends in Fig. 4.

Automatic diversity metrics. For each prompt and run we collect the B branches and compute the six metrics that appear in the main text and tables of Park et al. (2025):

- **Self-BLEU:** symmetric pairwise BLEU averaged over all unordered pairs of branches.
- **ROUGE-L** and **METEOR:** averaged over all unordered branch pairs.
- **Sent-Sim:** mean cosine similarity between SBERT embeddings (all-MiniLM-L6-v2) of branches (Reimers and Gurevych, 2019).
- **LLMScore:** an LLM-based diversity score in $[0, 1]$ assigned per prompt and run. Like Park et al. (2025), we also multiply by 100.
- **Degen:** an LLM-based degeneration score in $[0, 1]$ assigned to each branch and averaged over branches.

LLM-based judges. LLMScore and Degen are computed by a GPT-5.1 (OpenAI, 2026) judge run in low-reasoning mode via the OpenAI Batch API, using the prompts from the degeneration and diversity sections in App. E–F of Park et al. (2025). Note that Park et al. (2025) evaluate with a different judge model ("GPT-o4-mini" in their paper), so absolute LLMScore/Degen values are not directly comparable across papers even when prompts are reused.

Run-level aggregation, ranking, and proxy ratios. For each run we aggregate prompt-level metrics via simple means, yielding one row per configuration. We then compute a composite diversity rank by ranking runs on six metrics (Self-BLEU, ROUGE-L, METEOR, Sent-Sim, LLMScore, Degen) and averaging the rank positions. For each sweep group (temperature, top-p, top-k, greedy) we pick the configuration with the best composite rank and use these runs as path-based baselines in the main text (e.g. in Sec. 4.3 and Tab. 2), where they are compared to the Z-grid sweep of the RL-trained *OS-Search* controller.

As an optional context-only reference (not a benchmark), we also compute for each method M

the relative LLM-based diversity gain

$$\rho_{\text{div}}(M) \;=\; \frac{\text{LLMScore}_M}{\max_{B\in\mathcal{B}_{\text{sampling}}} \text{LLMScore}_B},$$

where $\mathcal{B}_{\text{sampling}}$ is the set of pure sampling baselines (temperature, top-p, top-k) on the same backbone and dataset with Degen ≤ 0.1. We apply this definition within our Qwen3-1.7B sweeps for within-harness comparisons (e.g., Sec. 4.3).

Branch-count normalisation for the OS-Search Z-sweep. In the small-model setting we sweep the three fixed axes of Z_{text} that the policy is trained to track and use a fixed 3-level grid per axis based on the frozen per-axis scale s_i (defined from $q_{0.10}/q_{0.90}$ in App. C). For z_2 and z_3 we use targets $\{-s_i, 0, +s_i\}$. For the anchored story axis z_1 we avoid low z_1 targets and instead use $\{\frac{1}{2}s_1, +s_1\}$. We take the Cartesian product of these levels (a $2\times3\times3$ grid). For each WritingPrompts prompt we pair it with all $2\times3^2 = 18$ grid targets and sample one completion per target, yielding $B = 18$ branches. To match the $B = 15$ branch count used in Park et al. (2025) and in our path-based baselines, the summary metrics reported for *OS-Search* in Tab. 2 are computed on a fixed subset of 15 branches per prompt: we enumerate the 18 grid targets in a fixed nested-loop order (outer: z_1^*, middle: z_2^*, inner: z_3^*), assign branch indices 0–17 in that order, and keep indices 0–14 (dropping the remaining three). This truncation is deterministic and independent of the generated text.

H CA++ benchmark and composite score

Each candidate program deterministically returns a 16×16 seed board as 16 strings of length 16 over the alphabet $\{.,\#\}$, which we map to a binary array $s\in\{0,1\}^{16\times16}$ with $\# \mapsto 1$.

Grids, rules, and horizons. We evaluate each seed on two toroidal grids $N\in\{16,24\}$ with periodic boundary conditions. For $N = 16$ we use the seed directly. For $N = 24$ we embed the 16×16 seed into the center of a 24×24 grid (zero padding elsewhere). We simulate three 2D totalistic cellular-automaton rules with Moore neighborhoods (8 neighbors) and synchronous updates. We use LIFE (B3/S23), HIGHLIFE (B36/S23), and SEEDS (B2/S0). For a grid of size N, we run a fixed horizon $T_N = \max(64, 4N)$ update steps (thus $T_{16} = 64$ and $T_{24} = 96$), producing states $x_0, x_1, \ldots, x_{T_N}\in\{0,1\}^{N\times N}$.

Per-run subscores. For each pair (N, rule), we compute five subscores from the spacetime trajectory:

1. **Activity**:

$$\text{act} = \frac{1}{T_N}\sum_{t=0}^{T_N-1}\frac{1}{N^2}\,\|x_{t+1}-x_t\|_0,$$

the mean fraction of cells that flip per step.

2. **Diversity (components)**: let $c(x_t)$ be the number of 4-neighbor connected components of live cells in x_t (with wraparound adjacency on the torus). Define

$$\bar{c} = \frac{1}{T_N}\sum_{t=1}^{T_N} c(x_t),$$
$$\text{div} = \min\left(1, \frac{4\bar{c}}{N^2}\right).$$

3. **Patch entropy**: for every time $t\in\{0,\ldots,T_N\}$ and cell location (i,j), we extract the 3×3 Moore neighborhood pattern around (i,j) with wraparound, yielding a 9-bit code in $\{0,\ldots,511\}$. Let p_k be the empirical frequency of code k over all neighborhoods across the full spacetime. The normalized entropy is

$$\text{pent} = \frac{1}{9}\Big(-\sum_{k:\,p_k>0} p_k\log_2 p_k\Big),$$

where $9 = \log_2(512)$.

4. **Compression contrast**: we serialize the full spacetime $(x_t)_{t=0}^{T_N}$ into a row-major ASCII string over $\{\texttt{0},\texttt{1}\}$ of length $L = (T_N+1)N^2$. We compress this byte string using DEFLATE at maximum compression level (9), let $r = \frac{|\text{compressed}|}{|\text{raw}|}$ be the byte-length ratio, and define

$$\text{ccont} = 2\,\min(r,\,1-r)$$

with $|\text{compressed}|$ never being larger than $|\text{raw}|$.

5. **Balance**: let $\ell_t = \frac{1}{N^2}\sum_{i,j} x_t[i,j]$ be the live-cell fraction and $\bar{\ell} = \frac{1}{T_N}\sum_{t=1}^{T_N}\ell_t$ its mean (excluding x_0). We define the balanced-density score

$$\text{bal} = 4\,\bar{\ell}\,(1-\bar{\ell}).$$

CA++ composite score. For each run (N, rule) we compute the weighted sum

$$\text{score}(N, \text{rule}) = 0.30\,\text{act} + 0.20\,\text{div} + 0.25\,\text{pent} + 0.20\,\text{ccont} + 0.05\,\text{bal}.$$

The overall CA++ score is the mean over all $|\{16, 24\}| \times |\{\text{LIFE}, \text{HIGHLIFE}, \text{SEEDS}\}| = 6$ runs:

$$f(x) = \text{clip}_{[0,1]}(\bar{f}(x)),$$
$$\bar{f}(x) = \frac{1}{|\mathcal{N}|\,|\mathcal{R}|} \sum_{N \in \mathcal{N}} \sum_{r \in \mathcal{R}} \text{score}(N, r),$$

where $\mathcal{N} = \{16, 24\}$ and $\mathcal{R} = \{\text{LIFE}, \text{HIGHLIFE}, \text{SEEDS}\}$, and $\text{clip}_{[0,1]}(y) = \min(1, \max(0, y))$. We treat $f(x)$ as the executable black-box objective used in Sec. 3.3 and Sec. 4.4.

I Code-space evaluation and search

We describe the Bayesian optimization setup and additional diagnostics for the code-space experiments, complementing Sec. 3.3.

Query protocol and budget. Each BO query follows Alg. 1(C): given a proposed z^*, we retrieve exemplars in Z_{code}, sample K candidates from the controller, validate and score each valid program with f (App. H), and return the best-scoring valid program x_{best}. When updating the surrogate we use the realised coordinate $z(x_{\text{best}})$ rather than the request z^*. We measure evaluation budget by N_{ok}, the number of scored valid programs, matching Sec. 4.4. The sampling baseline draws directly from the base model with the same validity gate and CA++ evaluator but without any z^* request or exemplar retrieval. Decoding settings match App. D.

BO details and ablation. We use a standard BO loop with expected improvement (GP–EI) over requested targets z^* within axis-aligned bounds derived from frozen axis statistics (App. C). We ablate whether the surrogate is updated at the realised coordinate $z(x_{\text{best}})$ or at the request z^*.

We additionally ablate whether we allow online growth of the exemplar retrieval library during BO, and the bounds scale ($1.5\times$ vs. $2.5\times$ the per-axis scale used to define the search box). Tab. 7 reports the median and range (min–max) over three random seeds for the best-so-far CA++ score at $N_{\text{ok}} = 2000$ scored valid programs and at the end of the run (Best@end), where N_{ok} counts only scored valid programs.

Update	Growing	Scale	Best@2000	Best@End
$z(x)$	no	2.5	0.376 [0.376,0.384]	0.382 [0.376,0.384]
$z(x)$	yes	1.5	0.379 [0.376,0.384]	0.382 [0.379,0.384]
$z(x)$	yes	2.5	0.384 [0.382,0.395]	0.384 [0.382,0.395]
z^*	no	2.5	0.373 [0.371,0.376]	0.373 [0.371,0.376]
z^*	yes	1.5	0.376 [0.371,0.376]	0.384 [0.376,0.389]
z^*	yes	2.5	0.376 [0.371,0.376]	0.376 [0.376,0.386]

Table 7: **BO ablations over Z_{code} (3 seeds each).** Best-so-far CA++ under a matched valid-program budget ($N_{\text{ok}} = 2000$) and at the end of the run (End) for ablations over update (surrogate update at realised $z(x)$ vs requested z^*), online exemplar-library growth (Grow), and scale of bounds. Entries are median and range (min–max) over seeds, N_{ok} counts only scored valid programs.

Transient–cycle decomposition on a finite Life torus. For the Life-only diagnostic on the 16×16 torus (periodic boundary conditions), the global update rule defines a deterministic map on a finite state space of size 2^{256}, so every trajectory must eventually repeat a previously seen configuration and then evolve periodically thereafter (Wolfram, 1984). We therefore decompose each trajectory into a transient (preperiod) length μ and a cycle length (period) λ: if x_t denotes the full 16×16 configuration at generation t, we define μ as the smallest index such that x_μ lies on a recurrent orbit and λ as the smallest positive integer with $x_{\mu+\lambda} = x_\mu$. We also report the first-repeat time $t_{\text{repeat}} = \mu + \lambda$.

Emergent long transients and torus-spaceship cycles. Although our *OS-Search* code objective does not include any term that rewards long transient length μ or long period λ, the resulting candidate pool contains seeds whose Life dynamics on the 16×16 torus remain non-repeating for hundreds of generations. In our evaluated set, the largest first-repeat time observed was $t_{\text{repeat}} = 702$ (with $\mu = 701$, $\lambda = 1$), i.e. a long transient that ultimately converges to a fixed point. Strikingly, the extreme tail by first-repeat time is dominated by very small attractor periods ($\lambda \in \{1, 2\}$), indicating that the longest-lived non-repeating behaviour in this regime is primarily slow relaxation rather than entry into a long limit cycle. Conversely, when ranking by attractor period λ, we frequently observe $\lambda = 64$ cycles. Visual inspection indicates these are traveling "spaceship"-type objects whose translation symmetry closes only after wrapping around the 16-cell torus. For example, a Life glider

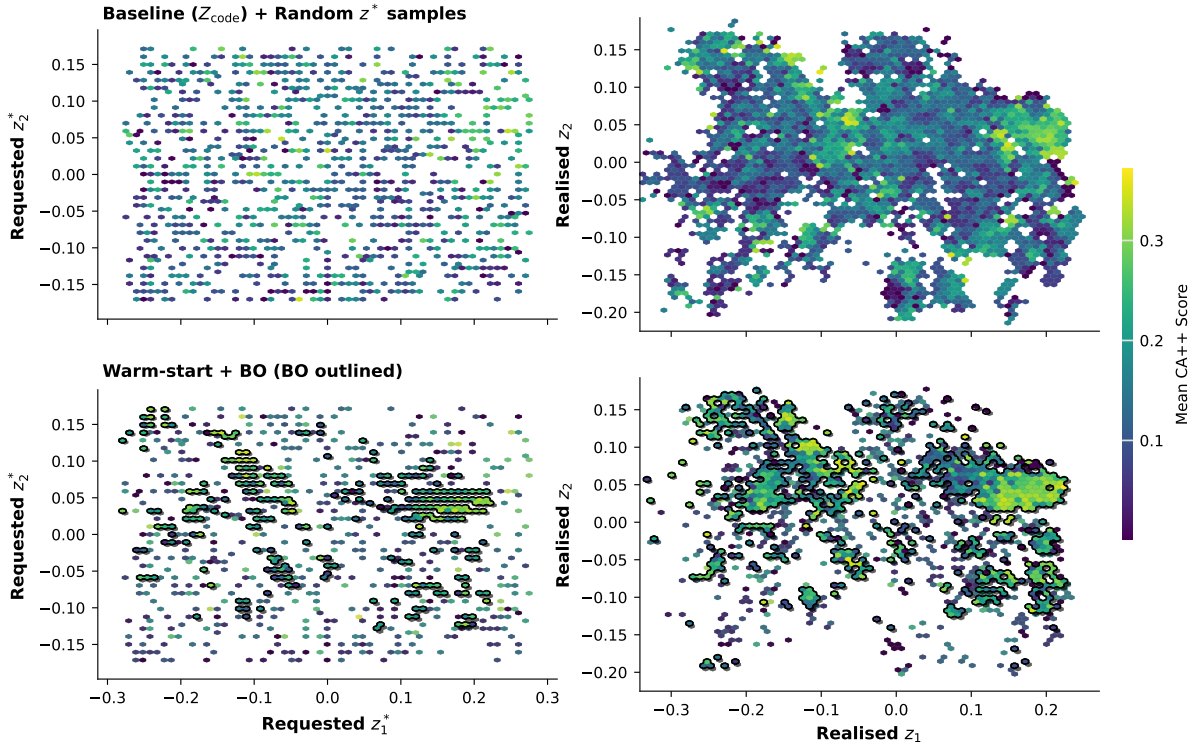

Figure 11: **Score structure in the (z_1, z_2) plane (code domain).** Mean CA++ score as a function of (left column) the requested targets (z_1^*, z_2^*) and (right column) the corresponding realised coordinates $(z_1, z_2) = z(x)$ of generated programs in Z_{code}. **Top:** the combined candidate pool from the Baseline (Z_{code}) sweep and random z^* samples (last valid 13,752 GRPO rollouts). **Bottom:** warm-start subset (1375 points) selected from the random z^* rollout pool plus additional candidates evaluated online during Bayesian optimization (BO), with BO-visited bins outlined in black. Warm-start requested targets (background) span a broad axis-aligned rectangle by construction, while BO focuses on a subset of bins (outlined). The corresponding realised coordinates concentrate on a smaller, non-uniform subset of Z_{code}, reflecting both the frozen encoder-defined geometry and residual policy limitations. Some requested target combinations are therefore effectively unattained (or attained only at very low probability) under our `make_seed()` constraints and 1.7B model. Highest scores concentrate in localized pockets rather than varying smoothly along a single axis.

translates by one diagonal cell every four generations (Paulsen, 2003), so on a 16×16 torus its return time is $4 \times 16 = 64$ generations, consistent with the observed dominant period $\lambda = 64$ in the top-by-period list. In both rankings (top-K by t_{repeat} and top-K by λ), the extreme tails are dominated by seeds produced by the z^*-conditioned RL policy rather than the base sampling pool (e.g. $49/50$ of the top-50 in each list in our run), despite the fact that neither μ nor λ is directly optimized. We emphasize that these values are maxima within our evaluated candidate pool and experimental configuration (Life, 16×16 torus).

For reference, Fig. 12 visualizes the BO-best seed over the CA++ $T_{16} = 64$ horizon under the three evaluation rules generated by Fig. 13.

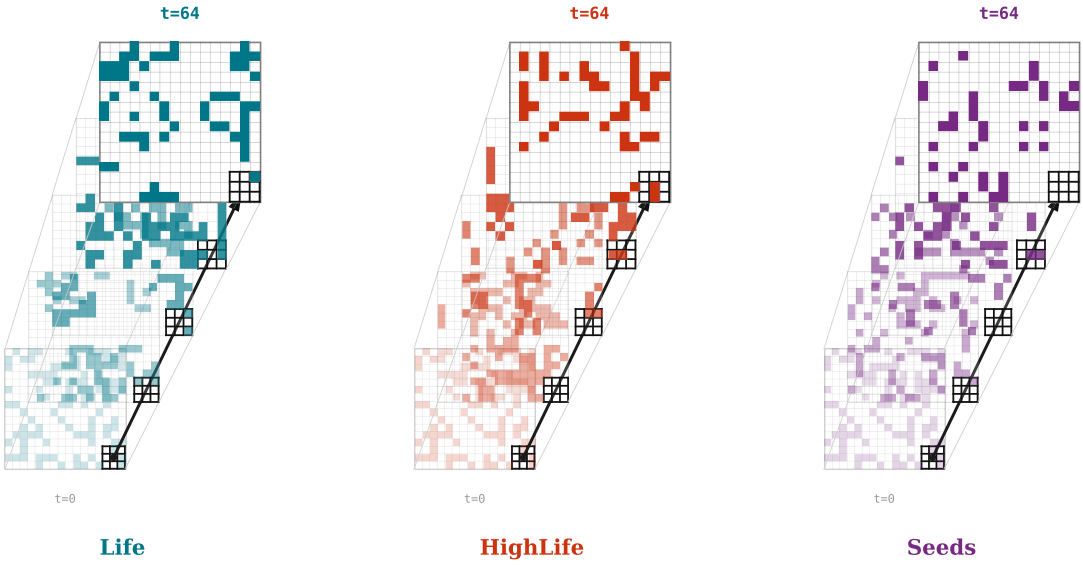

Figure 12: **Qualitative evolution of the BO-best seed under CA++ rules (N=16 horizon).** Spatiotemporal snapshots of the best-performing BO-discovered `make_seed()` program, visualized over the CA++ horizon $T_{16} = 64$ on the 16×16 torus under LIFE, HIGHLIFE, and SEEDS. Each panel shows five snapshots from $t = 0$ to $t = 64$ (earlier layers are lighter).

```python
def make_seed() -> list[str]:
    n = 16
    out = []
    for i in range(n):
        row = []
        for j in range(n):
            x = (i + 1) * 200 + (j + 1) * 400 + i * i * 10 + j * j * 20 + i * j * 30
            v = (1303515245 * x + 12657) % 10007
            if (v % 9) < 4:
                row.append('#')
            else:
                row.append('.')
        out.append(''.join(row))
    return out
```

Figure 13: **BO-best `make_seed()` program** BO-best `make_seed()` program used to generate Fig. 12.