ANALYZING SPATIAL COGNITION VIA BEHAVIORAL INTERPRETABILITY

Anonymous authors
Paper under double-blind review

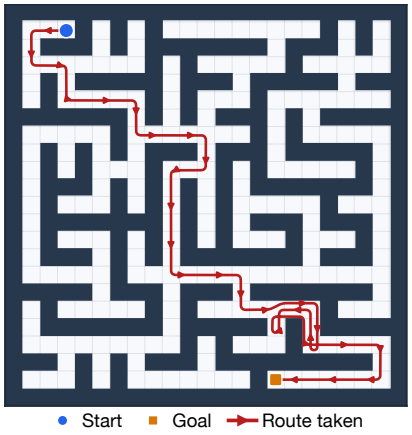
ABSTRACT

What are LLMs learning to solve spatial cognition tasks? Interpretability techniques usually inspect a model's verbalized reasoning or internal activations, but chain-of-thought traces can be unfaithful and activations are inaccessible for closed models. Behavioral psychology faces the same two obstacles, since self-reports are unreliable and the brain is hard to observe, and instead studies the mind by modeling people's choices and mistakes. We propose to study LLMs the same way by synthesizing symbolic programs that predict their choices and mistakes. On four interactive spatial cognition tasks, our evolutionary search finds programs that match each model's action distribution to within 0.07 nats while being two orders of magnitude simpler than a memorization baseline. The programs reveal that the strongest models perform shortest-path planning in maze navigation, raster scanning and spatial memory-based pruning to find treasures in the Cambridge spatial working memory task, and lookahead planning over visual waypoints for egocentric navigation. On the other hand, weaker models perform greedy goal search in mazes and get stuck in dead ends, exhibit poor retention in spatial working memory and reopen boxes known to be empty, and oscillate in place for egocentric navigation.

1 INTRODUCTION

From an early age, children and animals develop strong spatial cognition skills that underpin abilities such as mental rotation, perspective taking, spatial memory, and cognitive mapping (Marshall &

(a) GPT-5.4 navigating a maze



(b) How GPT-5.4 solves mazes

Input: Position $s = (r_s, c_s)$, goal g, waypoints $\mathcal{W}$, graph G
Output: Action distribution $\pi(w)$ over $\mathcal{W} \cup \{\text{stop}\}$

if $s = g$ **then** ▷ Goal reached
  return $\pi(\text{stop}) = 1.0$
end if
for all $w \in \mathcal{W}$ with target $s_w = (r_w, c_w)$ **do**
  if $s_w = g$ **then** ▷ Goal in sight
    $z_w \leftarrow \beta_{\text{goal}}$
  else
    $\text{on_sp}_w \leftarrow \mathbb{I}(d_G(s, g) = d_G(s, s_w) + d_G(s_w, g))$
    $\text{cds}_w \leftarrow \text{IsDeadEnd}(s_w, \text{depth} \le 27)$ ▷ Cul-de-sac pruning
    $\text{axis}_w \leftarrow \mathbb{I}(r_w = r_s \vee c_w = c_s)$ ▷ Corridor momentum
    $\Delta d_{\text{man}} \leftarrow \|s - g\|_1 - \|s_w - g\|_1$ ▷ Manhattan
    $z_w \leftarrow \beta_{\text{sp}}\,\text{on_sp}_w - \beta_{\text{cds}}\,\text{cds}_w + \beta_{\text{axis}}\,\text{axis}_w + \beta_{\text{man}}\,\Delta d_{\text{man}}$
  end if
end for
return $\pi(w) = \dfrac{\exp(z_w) + \varepsilon}{\sum_{w'}(\exp(z_{w'}) + \varepsilon)}$ ▷ Slip noise ε

Figure 1: **What strategies do LLMs learn to solve spatial cognition tasks?** **(a)** GPT-5.4 solves a maze, making (and recovering from) a wrong turn along the way. What are the heuristics it uses to solve mazes? **(b)** We use evolutionary search to synthesize programs that imitate model behaviors, including their errors, providing a compact explanation for their decisions. Our program predicts GPT-5.4's actions with 92% accuracy, identifying that it relies on **shortest paths**, **cul-de-sac pruning** (pruning dead ends up to 27 steps ahead), **corridor momentum** (favoring straight-line continuation), and **Manhattan progress** (greedily move to goal).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Fink, 2001; Waller & Nadel, 2013; Mallot, 2024). However, only in the last year did frontier AI models begin to demonstrate competency – sometimes even super-human performance – on spatial cognition tasks from Ramakrishnan et al. (2025). Figure 1a shows GPT 5.4 solving a maze, making (and recovering from) a wrong turn on its way to the goal. We are interested in explaining: What algorithms do VLMs and LLMs learn to solve spatial cognition tasks?

Interpretability techniques have focused on internal representations or verbalized reasoning traces, but both approaches offer isolated puzzle pieces that rarely add up to a coherent picture of how an agent plans and acts. Mechanistic interpretability focuses on identifying circuits and individual neurons (Elhage et al., 2021; Wang et al., 2023), but aggregating them into a holistic understanding of an agent's sequential policy is difficult, and closed frontier models withhold their weights entirely. On the other hand, chain-of-thought reasoning presents persuasive rationalizations that may not be faithful accounts of the underlying decision process (Turpin et al., 2023; Lanham et al., 2023).

Behavioral psychology faced this exact dilemma when studying biological minds (Tolman, 1948; Binz & Schulz, 2023). Because researchers could not read neural wiring nor trust a person's verbalizations (Nisbett & Wilson, 1977; Jonas & Kording, 2017; Bennett et al., 2009), they characterized cognitive strategies with computational models that predict both successful actions and characteristic mistakes (Newell & Simon, 1972; Castro et al., 2025). This approach uncovered foundational insights about natural intelligence, including how the mind trades off speed against accuracy (Ratcliff, 1978) and how the brain balances deliberate planning against habits (Daw et al., 2005).

Drawing on this method, we propose behavioral program synthesis as an approach to studying frontier models. We reconstruct a symbolic program that imitates an agent's actions, including its errors, thereby obtaining a compact description for the strategies and behaviors that models have learned. We use agentic evolutionary search to optimize Python programs (Lehman et al., 2023; Romera-Paredes et al., 2024) for LLMs from the GPT, Qwen, InternVL, and Gemma families, allowing us to compare the strategies they learn. Figure 1b shows the program found for GPT 5.4.

We apply our framework to four interactive tasks (mazes, spatial working memory, shortcut discovery, and route retracing) for both visual and textual input modalities. Our synthesized programs come within 0.07 nats of each model's action distribution on every task, while being orders of magnitude more compact than the subject model, and providing an interpretable account of model decisions and mistakes. The evolved programs reveal several key behaviors of frontier models:

- **Planning mechanisms:** The behavior of top LLMs for maze navigation with text stimuli is best explained by global shortest-path planning, and is partly influenced by local heuristics such as dead-end avoidance. Intermediate models with text, and top ones with vision, fail to simulate deep branches, frequently entering dead ends due to greedy goal attraction.
- **Working memory mechanisms:** The best models exhibit remarkable performance on the Cambridge Spatial Working Memory task. They develop memory-guided pruning techniques to avoid boxes that previously contained treasures or were found to be empty, and systematically scan boxes in raster-order to find treasures efficiently. The intermediate models, on the other hand, lack good memory-guidance and repeatedly open boxes that are known to be empty. They also suffer from visual biases such as opening boxes labeled 1 or repeating the previous box.
- **Egocentric navigation mechanisms:** The best models can remember routes demonstrated in videos, aim at waypoints several steps ahead along the route, and steer around obstacles to reach them. Weaker models are dominated by habitual action repetition, oscillating in place instead of making progress, and rarely reach the goal. All models struggle to discover shortcuts.

2 Related work

Spatial cognition. A foundational dimension of biological intelligence, spatial cognition enables animals and humans to perceive spatial layouts, build internal cognitive maps, and execute goal-directed actions (Tolman, 1948; O'Keefe & Nadel, 1978; Marshall & Fink, 2001; Waller & Nadel, 2013; Mallot, 2024). Cognitive science broadly categorizes spatial faculties into small-scale abilities—such as mental rotation, viewpoint transformation, and perspective taking (Shepard & Metzler, 1971; Hegarty & Waller, 2004)—and large-scale abilities, such as path integration, environmental mapping, and navigation (Hegarty et al., 2006). In artificial intelligence, early investigations probed whether text-only language models could form cognitive maps or execute multi-step plans in symbolic environments, finding severe limitations in graph traversal and physical plan-

ning (Valmeekam et al., 2023; Momennejad et al., 2023; Yamada et al., 2024). More recently, multimodal benchmarks have examined spatial reasoning in vision-language models across diverse settings, including 2D spatial relational reasoning and core perception (Liu et al., 2023; Fu et al., 2024), 3D embodied question answering (Majumdar et al., 2024), passive video-based spatial recall (Yang et al., 2025), perspective and rotational invariance (Zhang et al., 2026), and multi-step visual simulation of geometric transformations (Li et al., 2026). In this work, we evaluate models on the SPACE benchmark (Ramakrishnan et al., 2025) because it provides a unified diagnostic testbed spanning this broad spectrum of classical small-scale and large-scale spatial cognitive paradigms across egocentric visual, map-based, and textual presentations.

Program synthesis. Cognitive science has a long tradition of inducing symbolic programs to model and characterize human and animal problem-solving behavior. Beginning in the 1950s, Newell & Simon (1972) were among the first to manually induce programs directly from behavioral traces to explain human reasoning under resource constraints. More recently, Goodman et al. (2008) and Lake et al. (2015) used probabilistic program induction to model human concept learning and generalization under uncertainty, demonstrating that human reasoning can be explained by inferring programs over compositional grammars. Castro et al. (2025) used evolutionary search to induce programs from behavioral traces of animal behavior. Recent work in "machine psychology" treats language models as experimental subjects, evaluating their decision-making, causal reasoning, and behavioral biases using diagnostic paradigms adapted from human cognitive psychology (Binz & Schulz, 2023; Hagendorff et al., 2023). Instead of summary statistics, we treat the behavioral trajectories of vision-language models as execution traces, and synthesize symbolic programs to characterize the models' cognitive strategies. Our work is related to Wang et al. (2026), who use program synthesis to compare how LLMs and people play rock-paper-scissors, while we focus on spatial intelligence tasks requiring multiple turns, perceptual grounding, and visual navigation.

Evolutionary search. We use evolutionary search to synthesize Python programs that predict LLM decisions, leveraging an LLM to implement mutation, crossover, and simplification operators rather than bit-string flips (Holland, 1975) or random syntax-tree manipulation (Koza, 1992). Lehman et al. (2023) pioneered the use of LLMs as the mutation and variation operators for code, and Meyerson et al. (2024) formalized crossover and recombination of parent programs with LLMs. LLM-guided evolution has discovered new algorithms for bin packing, the cap set problem, and matrix multiplication (Romera-Paredes et al., 2024; Novikov et al., 2025), reward functions for robot control (Ma et al., 2024), and symbolic programs from visual data (Chiquier et al., 2024; Mall et al., 2025).

Interpretability. Despite the allure to interpret LLM reasoning traces (Wei et al., 2022; Kojima et al., 2022), a growing body of work demonstrates that chain-of-thought explanations are unfaithful, acting as post-hoc rationalizations that mask the actual features driving decisions (Turpin et al., 2023; Lanham et al., 2023; Alon et al., 2026; Parekh & Goyal, 2026; Du et al., 2026). Models can carry out multi-step reasoning in their latent activations without verbalizing it (Pfau et al., 2024; Baherwani et al., 2026), and Panfilov et al. (2026) show that the publicly released chain-of-thought from frontier models often contradicts the private, encrypted reasoning. Hidden representations can reveal reasoning that a chain-of-thought monitor misses (Jeong et al., 2026; Mirtaheri & Belkin, 2026; Skean et al., 2025), but probing them, as in mechanistic interpretability (Elhage et al., 2021; Meng et al., 2022), requires white-box access, which is unavailable for closed models, and the findings are hard to assemble into a holistic account (Méloux et al., 2025b;a).

3 Interpretability via Program Synthesis

Our goal is to characterize the behaviors and algorithms of LLMs and VLMs by synthesizing an interpretable, symbolic program that imitates the model's decisions. Rather than creating a program that optimally solves a task, we want the program to have the same mistakes, biases, and uncertainty as the LLM so that it provides a behavioral explanation for the model.

We aim to produce a single unified program that can characterize the behaviors of all models on a given task. This is crucial to systematically capture the behavioral commonalities and differences across LLMs on the task. We achieve this by first inducing programs independently for each model through evolutionary search, and then refactoring the individual programs to unify them into a common program with model-specific parameters.

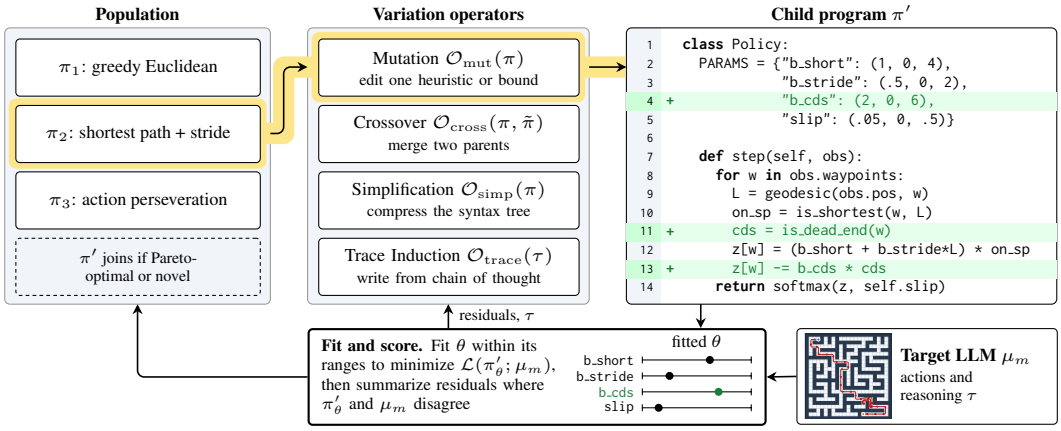


Figure 2: **Agent-driven evolutionary search.** A parent program π, drawn from a population of candidates (left), is edited by one of four LLM variation operators (middle) to produce a child program π' (right). For example, *mutation* expands the frontier with dead-end suppression (the highlighted path) . The other operators will be randomly executed each generation. Each child is scored against the target LLM μ_m, and survivors join the population (bottom loop). Over a full run this fits programs that are predictive of model behavior.

3.1 Agent-driven Evolutionary Search

Consider an LLM $\mu : x \in \mathcal{X} \to a \in \mathcal{A}$ as a black-box policy mapping a task's continuous state space $\mathcal{X}$ to discrete actions $\mathcal{A}$ available in each state. We perform evolutionary search (See Figure 2) to synthesize a concise Python program π that predicts the actions of μ on each state it visits.

Representation. We represent programs as a symbolic structure (a Python class) and continuous parameter values (arguments to the class). The symbolic structure π represents a concrete hypothesis for how a model solves the task, consisting of control flow, state representations, and spatial heuristics. Each program has a parameter vector $\theta \in \Theta_\pi$, which are bounded numerical constants that quantify resource limits (such as search horizons, memory capacities, or decay rates) and behavioral biases (such as backtrack penalties, directional preferences, or action noise). The program $\pi_\theta(a \mid x)$ takes in a state x and predicts a distribution over actions $a \in \mathcal{A}$.

Optimization Objective. To fit program π_θ to the behaviors of LLM μ, we must jointly optimize them using the following negative log-likelihood objective:

$$\pi^\star, \theta^\star = \underset{\pi \in \Pi,\ \theta \in \Theta_\pi}{\arg\min} \; -\mathbb{E}_{(x,a)\sim\mu}\big[\log \pi_\theta(a \mid x)\big], \tag{1}$$

which is measured over the (state x, action a) pairs demonstrated by μ on the task. Minimizing this objective rewards programs that reproduce the LLM's actions, including its errors. A compact program with low held-out loss serves as an inspectable, falsifiable hypothesis about the model's behavior. We treat this as a bilevel optimization problem where each step alternates between optimizing the program structure π using agent-driven evolutionary search, and optimizing the parameters θ for π using coordinate descent. The structure π is optimized over all behavioral samples from the LLM, and the parameters θ are optimized via 4-fold cross-validation on the same samples.

Seed Programs. To initialize the evolution process, we define a list of seed programs for each task that define initial hypotheses for how models may approach that task. Seeds span simple non-spatial biases (e.g., uniform random choice, action perseveration by repeating the previous move, or digit-label frequency preferences), naive spatial heuristics (e.g., greedy Euclidean line-of-sight toward the goal or adjacent-cluster probing), and structured algorithmic planners (e.g., topological shortest-path graph traversal or exact episodic memory exclusion). See appendix Section D.3 for details.

Variation Operators. At each evolution step (generation), the optimization process receives parent programs $\{\pi_p\}_{p=1}^{P}$ from the previous generation.[1] An LLM-agent proposes candidate program structures in parallel using four variation operators (Figure 2): *mutation* $\mathcal{O}_{\text{mut}}(\pi)$ edits the control flow or heuristics of a parent to fix its mispredictions, *crossover* $\mathcal{O}_{\text{cross}}(\pi, \tilde{\pi})$ merges complementary mechanisms from two parents, *simplification* $\mathcal{O}_{\text{simp}}(\pi)$ prunes redundant logic while preserving

[1]For the first generation of evolution, the parent programs are the seed programs.

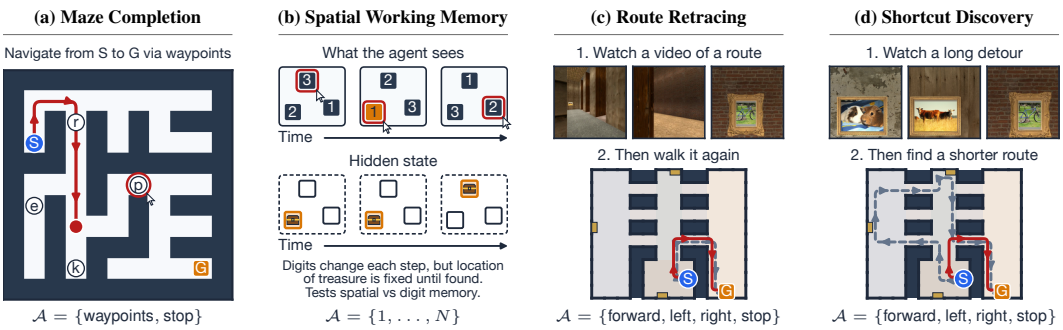


Figure 3: **Interactive spatial cognition tasks (Ramakrishnan et al., 2025).** Agent's actions in red.

predictive accuracy, and *trace induction* $\mathcal{O}_{\text{trace}}(\tau)$ writes new programs from the model's reasoning traces τ, quoted in the residual report of the current best program, without a parent program to edit. Each operator prompt includes a *task contract* defining the executable interface and allowable observation attributes, and a *residual report* of states where the parent's predictions diverge from the model along with the model's verbalized reasoning. All operators except simplification also receive the fit objective and a *briefing* of measurements of the model's behavior (Appendix D.4). Each fitted child joins the population if it is Pareto non-dominated on held-out loss and next-action prediction accuracy, or if its predictions differ from every member's on more than 5% of decisions.

Parameter Optimization. After obtaining a proposed candidate π from the variation operators, we optimize its parameters $\theta \in \Theta_\pi$ to minimize the negative log-likelihood loss in Equation 1. Because resource bounds (such as lookahead depth and memory buffers) are integer-valued and induce piecewise-constant, non-differentiable loss surfaces, we optimize θ using multi-start grid coordinate descent over the program's declared parameter ranges (Wright, 2015).

Unified program generation. After evolutionary search, for each LLM μ_m, we get a compact program structure π_m with parameters θ_m optimized to mimic it. To compare models behaviorally, we need a unified program structure π whose instances π_{θ_m} reproduce every π_m. Rather than re-optimizing, we use agent-driven code refactoring: the structure shared by $\{\pi_1, \cdots, \pi_M\}$ becomes a common Python class, and their differences become per-model settings. The fitted parameters for each model θ_m remain unchanged. The unified programs per task replicate the outputs of the model-specific programs, while reducing the total code by 17.3%. All inter-model variation is therefore captured by $\theta_1, \ldots, \theta_M$, which we will analyze in experiments to compare LLM behaviors.

3.2 Measuring Program Fidelity with KL Divergence

Because LLMs are stochastic, a program that reproduces μ_m exactly would still incur a loss equal to the model's entropy. This floor differs across models and tasks, making raw losses incomparable. We measure the forward Kullback–Leibler (KL) divergence from the model's policy to the program's, which subtracts the entropy of the model's observed actions so that a perfect program scores zero.

Task and Action Abstraction. Computing this divergence requires the model's action distribution at each state, which we can only estimate from repeated decisions at the same state. In an LLM, decisions in a long-horizon context window impact all future decisions, making every state unique. We therefore project raw prompts into discrete situations $\phi(x) = s \in \mathcal{S}$ and surface choices into task-invariant functional actions $\psi(a, x) = \alpha \in \mathcal{A}_{\text{abs}}$ (e.g., advancing along the goal path, exploring a detour, or checking neighboring boxes), and pool decisions within each situation. To isolate the core demands of each task, we define situations as the Cartesian product of discrete feature dimensions $s = (f_1(x), \ldots, f_k(x)) \in \mathcal{S} = \prod_{i=1}^{k} \mathcal{F}_i$, such as memory load in Cambridge Spatial Working Memory or junctions versus corridors in Maze Completion. Appendix E describes the features for each task and gives full operational definitions (Tables 6–13).

Situational Forward KL Divergence. We group the decisions of model μ_m by situation and let N_s be the number of decisions in situation $s \in \mathcal{S}$, out of N in total. For each situation, $\hat{p}_m(\alpha \mid s)$ is the fraction of these decisions on which the model took abstract action α, and $\bar{\pi}_\theta(\alpha \mid s)$ is the program's probability of α averaged over the same decisions. Weighting situations by their

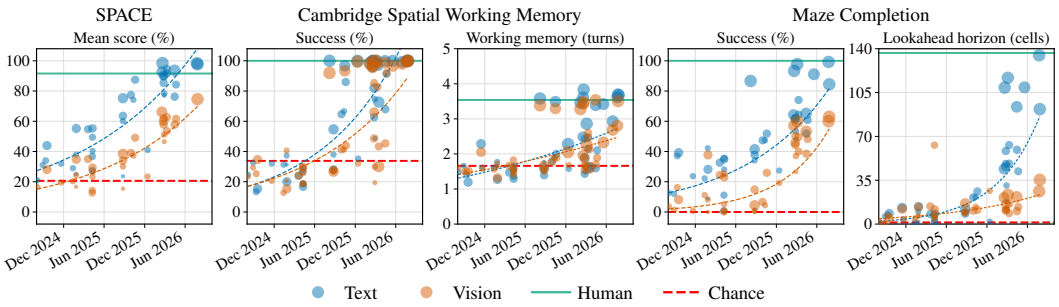


Figure 4: **Spatial cognition is booting up.** The first plot shows that LLM and VLM performances on SPACE are improving rapidly and closing in on humans. Results on text (blue) and vision (orange) are plotted against release date, with marker area encoding parameter count. The next two plots show the success and working memory[2] of models on Cambridge Spatial Working Memory. The last two show the success and lookahead horizon[3] for the Maze Completion task. What computational strategies are models using to solve these tasks?

frequency $p(s) = N_s/N$, the situational forward KL divergence is

$$D_{\mathrm{KL}}(\hat{p}_m \parallel \bar{\pi}_\theta) = \sum_{s\in\mathcal{S},\alpha\in\mathcal{A}_{\mathrm{abs}}} p(s)\,\hat{p}_m(\alpha \mid s)\Big[\underbrace{-\log \bar{\pi}_\theta(\alpha \mid s)}_{\text{program loss}} - \underbrace{\big(-\log \hat{p}_m(\alpha \mid s)\big)}_{\text{empirical entropy}}\Big]. \quad (2)$$

The first term is the loss of Eq. 1 evaluated on abstract situations and actions. The second is the entropy of the model's action frequencies in each situation, computed directly from counts. When $D_{\mathrm{KL}} \to 0$, the program has captured all the reducible uncertainty in the LLM's policy.

4 Experiments

We introduce the task setup in Section 4.1. In Section 4.2, we show that a) we synthesize highly predictive programs that imitate a model's actions and their errors, and b) our programs come within 0.1 nats of performance compared to a memorization ceiling while c) providing an order of magnitude simpler explanation. Finally, we analyze the synthesized programs in Sections 4.3–4.5 to understand the various strategies learned by a spectrum of models.

4.1 Tasks and Setup

We focus on interactive tasks in SPACE (Ramakrishnan et al., 2025), where LLMs and VLMs must build a long-term, dynamic spatial memory and reason through sequential steps to achieve the goal. We evaluate models on the following tasks (also shown in Figure 3):

1. **Maze Completion (MCT)**: The agent navigates a typical maze from start to end. Instead of stepping cell by cell, the model selects among candidate waypoints $w \in \mathcal{W}$, each extending up to 5 steps along a reachable corridor branch, and selects stop once it arrives at g.
2. **Cambridge Spatial Working Memory (CSWM)** (Sahakian et al., 1988): Each round, a treasure is hidden in one of several boxes on screen. The agent must eliminate empty boxes to find it, while remembering past treasure locations, which are never reused.
3. **Route Retracing (RR)**: The agent is given a video walkthrough showing a route to a goal in an interactive 3D maze. It is required to retrace the path from the place where the video started using FORWARD, TURN_LEFT and TURN_RIGHT actions, and selects STOP once it reaches the goal.
4. **Shortcut discovery (SD)**: The agent is given a video walkthrough showing a circuitous route to a goal in an interactive 3D maze. The agent is expected to discover the shortcut route to the goal, navigate to it using navigation actions, and STOP once it reaches.

For MCT and CSWM, the state is encoded either as text or as an image, so we can test whether strategies change between modalities. RR and SD are vision-only.

Figure 4 shows that LLMs have progressed significantly on these tasks, with the latest models nearing human performance. The metrics suggest that models have developed strategies to plan ahead and mechanisms for spatial memory. What algorithms have the LLMs learned to solve these tasks?

[2] **Working memory:** # turns in a round before the model re-opens a box it already opened.

[3] **Lookahead horizon:** largest distance to goal at which a model takes the shortest path $\geq 50\%$ of the time.

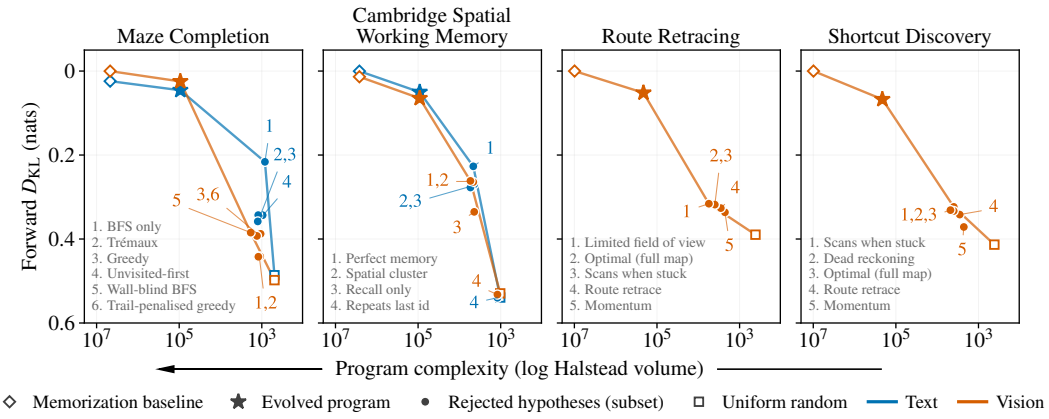


Figure 5: **Behavioral alignment of unified programs vs. program complexity.** We show the behavioral fitness of different programs. Blue and orange show text and vision variants respectively (egocentric tasks are vision-only). Each point represents one program across all models within the task: uniform random control (□), the best evolved program (★), and memorization baseline (◇) that memorizes (observation, model, modality) → action. We also show a subset of other hypotheses (•) that were considered and rejected by the evolutionary search. The horizontal axis is program complexity (Halstead, 1977). See Figures 8 and 9 for full numbers.

We synthesize programs for models across the performance spectrum: GPT 5.4, Anonymous-6rhc[4], Gemma 4 31B, Qwen 3.6 27B, Qwen 3.5 4B, and InternVL 3.5 38B (see appendix Table 1). We evaluate all models three times on each task with high thinking. After ten generations of evolution for each model and task pair, we semi-automatically audit the programs to remove spurious variables that do not contribute to the final result of the program. Finally, we unify the programs within each task into one refactored program and verify that it replicates the originals.

4.2 Fitness of Evolved Programs

Figure 5 plots the fitness of the programs versus their complexity. Since LLMs and VLMs are stochastic, we measure fitness through situational forward D_{KL} (Eq. 2) to subtract away fundamental entropy. We measure program complexity using Halstead volume (Halstead, 1977). We compare our synthesized programs to a uniform random control baseline, competing hypotheses, and a memorization ceiling that maps (observation, model, modality) → action.

Evolutionary search improves fit everywhere: over ten generations the evolved program lowers the situational forward D_{KL} relative to the best seed in all four tasks by $\sim 4{-}16$ times, reaching within 0.07 nats of the model on every task. Importantly, it achieves this fit with programs that are two orders of magnitude lower complexity than the memorization baseline, which is crucial for interpretability. We additionally included the next-action prediction accuracy and per-task breakdowns in appendix Figures 8 and 9. The unified program improves D_{KL} in all 36 model–task–modality cells, by as much as $29\times$, and raises next-action prediction accuracy in 32 of 36.

4.3 Interpreting Planning Strategies

Navigating a maze requires models to look ahead to plan steps, and we can make mazes arbitrarily complex to stress the look ahead depth. Figure 6 (top) shows our synthesized program that imitates the model's decisions when navigating a maze, and it recovers three categories of mechanisms that span the spectrum of model behaviors. **1) Shortest-path planning:** graph-search tests that evaluate whether a candidate waypoint makes wall-aware geodesic progress (w_{geo}), lies on a shortest path to the goal (w_{sp}), or leads into a cul-de-sac that would require backing out (w_{cds}). **2) Goal attraction:** wall-blind coordinate heuristics that favor moving directly toward the goal via Manhattan (w_{man}) and Euclidean (w_{eucl}) progress, alongside bearing alignment (w_{dir}). **3) Kinematics, novelty & readout:** local heuristics including row/column axis continuation (w_{axis}), dead-end avoidance (w_{dead}), and novelty terms that reward unvisited frontier (w_{unv}) and penalize revisits (w_{loop}). These terms are summed into a waypoint utility and passed through a softmax.

[4]Some models are undisclosed due to contractual obligations. Full names will be listed in the final paper.

Unified program evolved for Maze Completion

Task objective: The player navigates a 2D maze to reach a designated goal cell g from start s. At each turn, the player chooses among candidate waypoints $w \in \mathcal{W}$ (each extending up to 5 steps along a reachable corridor branch to target cell t) or selects stop upon goal arrival.

State: Current cell s, goal cell g, maze grid $\mathcal{M}$, visit trail history $v(\cdot)$. **Output:** Action distribution $\pi(w)$ over candidate waypoints $w \in \mathcal{W} \cup \{\text{stop}\}$.

```
// Shortest-path planning
// Goal arrival check: commit stop on goal
if s = g then return {stop : 1.0}
for all waypoints w, target t = W(w) do
    // Wall-aware distance, shortest path, cul-de-sacs
    Δd_geo ← d_geo(s, g) − d_geo(t, g)
    I_sp ← I[d_geo(s, t)+d_geo(t, g)=d_geo(s, g)]
    S_plan ← w_geo Δd_geo + w_sp I_sp
             − w_cds cds(t)
    // Goal attraction
    // Wall-blind Manhattan & Euclidean progress
    Δd_man ← ‖s − g‖_1 − ‖t − g‖_1
    Δd_eucl ← ‖s − g‖_2 − ‖t − g‖_2
    // Directional bearing to goal
    cos_goal ← ((t−s)·(g−s)) / (‖t−s‖_2 ‖g−s‖_2)
    S_goal ← w_man Δd_man + w_eucl Δd_eucl
             + w_dir cos_goal
    // Kinematics, novelty & readout
    // Local heuristics: straight axis & dead ends
    S_local ← w_axis axis(t) − w_dead I_dead(t)
    // Novelty: frontier reward & loop penalty
    S_novel ← w_unv I[v(w)=0] − w_loop v(w)
    // Total waypoint utility and softmax readout
    S(t) ← S_plan + S_goal + S_local + S_novel
end for
return π(w) ∝ exp(S(t)/τ) + ε
```

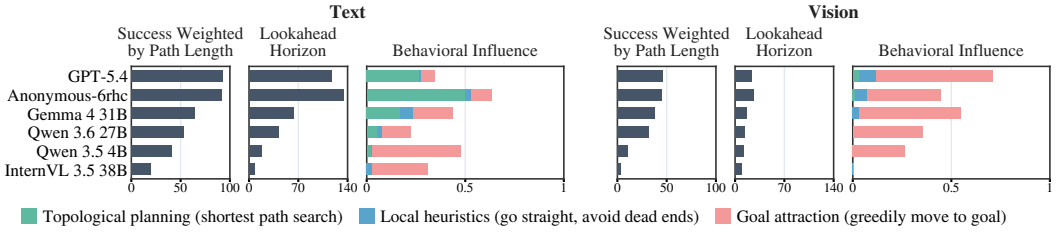


Figure 6: **(Top)** Pseudo-code of the unified program for MCT. The parameters highlighted in color vary across models and modalities. **(Bottom)** Task performance and behavioral influence for text (left) and vision (right). Task metrics are success weighted by path length (SPL), which scales success by navigation efficiency, and lookahead horizon from Figure 4. Each segment of the stacked bars is how much worse the program explains the model's actions without that behavior (rise in negative log-likelihood, nats per decision). A behavior's influence is the largest rise among its variables, and the three goal-attraction weights are removed together.

Figure 6 (bottom) shows which behaviors are important for which models. Shortest-path planning is clearly demonstrated by the best models navigating text mazes. It is the most influential behavior for GPT 5.4 and Anonymous-6rhc. While it has moderate influence for mid-tier models like Gemma 4 31B and Qwen 3.6 27B, these models tend to get distracted by goal attraction heuristics that reduce their effectiveness on this task. For example, GPT 5.4 and Anonymous-6rhc are influenced by goal attraction heuristics in only $2{-}3\%$ of their decisions, while the rest are influenced in $30{-}60\%$ of their decisions. Therefore, shortest-path planning impacts the task performance positively and goal attraction impacts it negatively. Local heuristics like dead-end avoidance, axis continuation and novelty only have limited influence across the models.

The picture is starkly different when the maze is shown as an image. All models are heavily influenced by goal attraction heuristics. While GPT 5.4 and Anonymous-6rhc show some semblance of topological planning (via w_{geo}) in $5{-}10\%$ of their decisions, other models are barely influenced by it. They instead rely on goal attraction that allows them to still succeed in some cases.

4.4 Interpreting Memory Strategies

We use the Cambridge Spatial Working Memory task to characterize how models build spatial memories. Figure 7 (top) shows the evolved program for CSWM, which consists of three stages. **1) Episodic memory update:** the program builds an episodic memory of where treasures have been found before ($\mathcal{T}$), and what boxes were opened in search for the current treasures ($\mathcal{M}$). This memory is common and shared across all models. **2) Memory-guided pruning:** This is the mechanism to avoid picking boxes that will not contain the treasure. When selecting a box, the program uses the episodic memory to prune boxes that previously contained a treasure (m_{found}). It also prunes boxes that were previously opened in the round and did not contain a treasure, but that pruning is imperfect. When an empty box is opened, the pruning weight is set to m_{miss} and it fades to a_{miss} over time. Finally, the boxes that contained treasures before or were already opened and found to be empty ($\mathcal{D}$) are (optionally) demoted in the scan order for box selection to suppress them further. **3) Box selection:** After pruning, the program scans the boxes in raster scan order starting from a spatial anchor s_0 (typically the previously opened box) and assigns a rank r to each box. Then, it scans over every box and assigns a spatial score $S(s)$ based on the raster sweep rank r (w_{raster}), spatial proximity to the anchor (w_{prox}), and spread away from previously opened boxes (w_{spread}). The

Unified program evolved for Cambridge Spatial Working Memory

Task objective: There are multiple boxes on a screen with one treasure hidden in an unknown box. Each round, the model must open boxes one at a time, keep track of which boxes were empty, and find the treasure. Once the treasure is found, the next round starts and the treasure is moved to a box where it never appeared before. The game ends when the model finds all the treasures (equal to the total number of boxes).

Observation: Box grid $\mathcal{B}$: $a \mapsto s$ from ephemeral box label a (changes each turn) to persistent spatial location s

State & Memory: Turn t, prior treasures $\mathcal{T}$, round misses $\mathcal{M}$, last probe s_{t-1}, treasure count c_{t-1}, last treasure s_T.

Output: Action distribution $\pi(a)$ over box labels $a \in \{1, \ldots, N\}$, $N = |\mathcal{B}|$.

```
// Episodic memory update
// Treasure found: retire box, reset misses
c_t, s_T ← identify_treasure(B)
if c_t > c_{t-1} then
    T ← T ∪ {s_{t-1}};  M ← ⟨⟩
// Empty probe: within-round miss
else if s_{t-1} ∉ T ∪ M then
    M ← M ⊕ s_{t-1}

// Memory-guided pruning
for all boxes s ∈ B do
    // Prune treasures (s ∈ T) and misses (s ∈ M)
    M(s) ← m_found
           m_miss F(s) + (1 − F(s)) a_miss
    // Fade F(s): 1→0 as the miss ages
end for
// Demotion set: ranked last, not excluded
D ⊆ T ∪ M

// Box selection & action distribution
s_0 ← anchor()  // typically previous box s_{t-1}
// Raster sweep: starts at top-left (or s_0), D last
{(a, s, r)} ← raster_rank(B; s_0, D)
for all boxes (a, s, r) ∈ B do
    // Spatial preference: sweep, proximity, spread
    S(s) ← w_raster raster(r; λ_raster)
           + w_prox prox(s, s_0; λ_prox)
           + w_spread spread(s)
    // Non-spatial visual bias: lure, digits, repeat
    B(s) ← b_lure I_{s=s_T} + b_digit L(a)
           + b_rep I_{s=s_{t-1}}
    x_a ← M(s) · [S(s) + B(s)]          ν = lin
          M(s) · exp(S(s) + B(s))       ν = exp
end for
// Readout: lapse mixture with the uniform
return π(a) ← (1 − ε) x_a / Σ_b x_b + ε / N
```

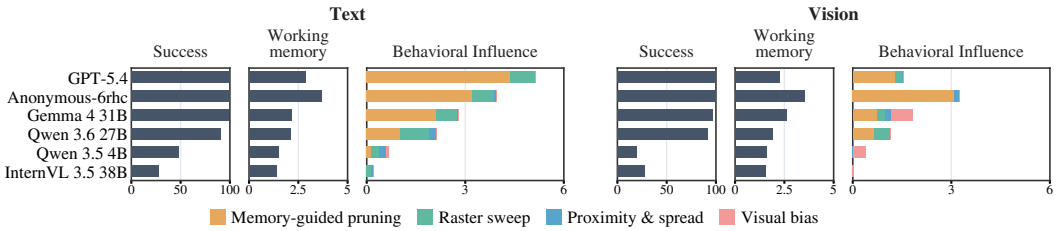


Figure 7: **(Top)** Pseudo-code of the unified program for CSWM. The parameters highlighted in color vary across models and modalities. **(Bottom)** Task performance and behavioral influence for text (left) and vision (right). Task metrics are success and working memory from Figure 4. Each segment of the stacked bars is how much worse the program explains the model's actions without that behavior (rise in negative log-likelihood, nats per decision). A behavior's influence is the largest rise among its variables.

program additionally accounts for selection biases to the recently opened treasure (b_{lure}), to specific numbers assigned to boxes (b_{digit}), as well as repeating the previously opened box (b_{rep}). Finally, the scores are combined through linear or exponential functions and normalized to probabilities.

Figure 7 (bottom) compares influence of the memory and search behaviors across models. The best models show remarkable memory-guided pruning, i.e., deprioritizing past treasures and within-round empty boxes. GPT 5.4, Anonymous-6rhc, Gemma 4 31B, and Qwen 3.6 27B perform incredibly well in this task with both text and vision presentations (over 90% success) and are heavily influenced by memory-guided pruning. Qwen 3.5 4B and InternVL 3.5 38B on the other hand hardly prune empty boxes, which aligns with their poor task success and working memory. While raster-sweep search dominates most models (after pruning), some models search based on spatial proximity to the previous box. In particular, Anonymous-6rhc searches primarily based on spatial proximity when tested with visual inputs (with little influence from raster-sweep). Qwen 3.5 4B and InternVL 3.5 38B are strongly influenced by visual biases: with vision, Qwen 3.5 4B is drawn to the most recent treasure, while InternVL 3.5 38B repeatedly opens the same box.

4.5 Interpreting Egocentric Navigation Strategies

We use the route retracing and shortcut discovery tasks to characterize how models build spatial representations from egocentric videos and navigate to goals based on that representation. The unified programs for the two egocentric navigation tasks, and the behavioral analysis they support, are reported in Appendix B. We find that the successful route retracers exhibit good route memory, aim several steps ahead along the route when planning, and can navigate to the goal while avoiding obstacles. However, weaker models are dominated by repetitive behaviors and are often stuck oscillating in place without making progress. All models struggle on shortcut discovery (the best model achieves only 40% success). The best shortcut model repeatedly moves forward and turns as needed to avoid obstacles.

AI Use Statement

AI systems are central to this work. Coding agents were used for evolutionary program search to arrive at a unified program that explains the behaviors of LLMs and VLMs on given task (Section 3). Multimodal LLMs are the subjects of our evaluation and behavioral analyses (Section 4). Beyond this, we used coding assistants to help write analysis and plotting scripts and to edit the wording of the manuscript. All research ideas, experimental designs, and conclusions are the authors' own. The authors checked all AI-assisted content and take full responsibility for it.

Reproducibility Statement

All tasks are drawn from the publicly available SPACE benchmark (Ramakrishnan et al., 2025), and Section 4.1 describes the task setup. The models we study, their versions, and their inference settings (thinking effort, token budgets, sampling hyper-parameters, and number of evaluation runs) are listed in appendix Table 1 and Appendix D. Some models are undisclosed for contractual reasons and will be named in the final version. Section 3 specifies the program search: the fit objective (Eq. 1), the bilevel optimization, the variation operators, the population admission criteria, and parameter fitting by coordinate descent. The seed programs are in Appendix D.3. The situational forward KL divergence is defined in Eq. 2, and Appendix E gives full operational definitions of every abstract situation and action (Tables 6–13), so the metric can be recomputed from raw model trajectories. We provide pseudo-code for all algorithms in the paper, and the fitted parameters of all algorithms are reported in the appendix.

References

Bar Alon, Itamar Zimerman, and Lior Wolf. Faithful serum: Mitigating the faithfulness gap in textual explanations of LLM decisions via attribution guidance. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6622–6645, San Diego, California, United States, July 2026. Association for Computational Linguistics.

Vatsal Baherwani, Tom Goldstein, and Ashwinee Panda. Not all LLM reasoning is visible in the chain-of-thought, 2026. URL https://arxiv.org/abs/2607.22925.

Craig M Bennett, Michael B Miller, and George L Wolford. Neural correlates of interspecies perspective taking in the post-mortem atlantic salmon: an argument for multiple comparisons correction. *Neuroimage*, 47(Suppl 1):S125, 2009.

Marcel Binz and Eric Schulz. Using cognitive psychology to understand GPT-3. *Proceedings of the National Academy of Sciences*, 120(6):e2218523120, 2023.

Pablo Samuel Castro, Nenad Tomasev, Ankit Anand, Navodita Sharma, Rishika Mohanta, Aparna Dev, Kuba Perlin, Siddhant Jain, Kyle Levin, Noemi Elteto, Will Dabney, Alexander Novikov, Glenn C Turner, Maria K Eckstein, Nathaniel D. Daw, Kevin J Miller, and Kim Stachenfeld. Discovering symbolic cognitive models from human and animal behavior. In *International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=dhRXGWJ027.

Mia Chiquier, Utkarsh Mall, and Carl Vondrick. Evolving interpretable visual classifiers with large language models. In *ECCV*, 2024. URL https://llm-mutate.cs.columbia.edu.

Nathaniel D Daw, Yael Niv, and Peter Dayan. Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature neuroscience*, 8(12):1704–1711, 2005.

Kevin Du, Alexander Miserlis Hoyle, Laura Ruis, and Acyr Locatelli. Legibility is not interpretability: Evaluating judged importance versus actual importance in chain-of-thought reasoning steps. In *Conference on Language Modeling (COLM)*, 2026.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt,

Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A. Smith, Wei-Chiu Ma, and Ranjay Krishna. Blink: Multimodal large language models can see but not perceive. In *ECCV*, pp. 148–166, Cham, 2024.

Noah D. Goodman, Joshua B. Tenenbaum, Jacob Feldman, and Thomas L. Griffiths. A rational analysis of rule-based concept learning. *Cognitive Science*, 32(1):108–154, 2008. doi: 10.1080/03640210701802071.

Thilo Hagendorff, Ishita Dasgupta, Marcel Binz, Stephanie CY Chan, Andrew Lampinen, Jane X Wang, Zeynep Akata, and Eric Schulz. Machine psychology. *arXiv preprint arXiv:2303.13988*, 2023.

Maurice H. Halstead. *Elements of Software Science*. Elsevier, New York, 1977.

Mary Hegarty and David Waller. A dissociation between mental rotation and perspective-taking spatial abilities. *Intelligence*, 32(2):175–191, 2004.

Mary Hegarty, Daniel R. Montello, Anthony E. Richardson, Toru Ishikawa, and Kristin Lovelace. Spatial abilities at different scales: Individual differences in aptitude-test performance and spatial-layout learning. *Intelligence*, 34(2):151–176, 2006.

John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

Seogyeong Jeong, Jaehui Hwang, Dongyoon Han, Geonmo Gu, Alice Oh, and Taekyung Kim. Beneath the surface of chains-of-thought: A mechanistic interpretation of reasoning operations in llms. In *EMNLP*, 2026.

Eric Jonas and Konrad Paul Kording. Could a neuroscientist understand a microprocessor? *PLoS computational biology*, 13(1):e1005268, 2017.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 22199–22213, 2022.

John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, MA, 1992.

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. doi: 10.1126/science.aab3050.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamilė Lukošiūtė, Karina Nguyen, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Robin Larson, Sam McCandlish, Sandipan Kundu, Saurav Kadavath, Shannon Yang, Thomas Henighan, Timothy Maxwell, Timothy Telleen-Lawton, Tristan Hume, Zac Hatfield-Dodds, Jared Kaplan, Jan Brauner, Samuel R. Bowman, and Ethan Perez. Measuring faithfulness in chain-of-thought reasoning, 2023. URL https://arxiv.org/abs/2307.13702.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. Evolution through large models. In *Handbook of evolutionary machine learning*, pp. 331–366. Springer, 2023.

Linjie Li, Mahtab Bigverdi, Jiawei Gu, Zixian Ma, Yinuo Yang, Ziang Li, Yejin Choi, and Ranjay Krishna. Unfolding spatial cognition: Evaluating multimodal models on visual simulations. In *International Conference on Learning Representations (ICLR)*, volume 2026, pp. 93136–93189, 2026.

Fangyu Liu, Guy Emerson, and Nigel Collier. Visual spatial reasoning. *Transactions of the Association for Computational Linguistics (TACL)*, 11:635–651, 2023.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Arjun Majumdar, Anurag Ajay, Xiaohan Zhang, Pranav Putta, Sriram Yenamandra, Mikael Henaff, Sneha Silwal, Paul Mcvay, Oleksandr Maksymets, Sergio Arnaud, Karmesh Yadav, Qiyang Li, Ben Newman, Mohit Sharma, Vincent Berges, Shiqi Zhang, Pulkit Agrawal, Yonatan Bisk, Dhruv Batra, Mrinal Kalakrishnan, Franziska Meier, Chris Paxton, Alexander Sax, and Aravind Rajeswaran. Openeqa: Embodied question answering in the era of foundation models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16488–16498, 2024.

Utkarsh Mall, Cheng Perng Phoo, Mia Chiquier, Bharath Hariharan, Kavita Bala, and Carl Vondrick. Disciple: Learning interpretable programs for scientific visual discovery. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 29258–29267. IEEE, 2025.

Hanspeter A. Mallot. *From Geometry to Behavior: An Introduction to Spatial Cognition*. The MIT Press, Cambridge, MA, 2024.

John C. Marshall and Gereon R. Fink. Spatial cognition: Where we were and where we are. *NeuroImage*, 14(1):S2–S7, 2001.

Maxime Méloux, Giada Dirupo, François Portet, and Maxime Peyrard. The dead salmons of AI interpretability. *arXiv:2512.18792*, 2025a.

Maxime Méloux, Silviu Maniu, François Portet, and Maxime Peyrard. Everything, everywhere, all at once: Is mechanistic interpretability identifiable? In *International Conference on Learning Representations (ICLR)*, 2025b.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 17359–17372, 2022.

Elliot Meyerson, Mark J. Nelson, Herbie Bradley, Adam Gaier, Arash Moradi, Amy K. Hoover, and Joel Lehman. Language model crossover: Variation through few-shot prompting. *ACM Transactions on Evolutionary Learning and Optimization*, 4(4):1–40, 2024. doi: 10.1145/3694791.

Parsa Mirtaheri and Mikhail Belkin. Catching rationalization in the act: Detecting motivated reasoning before and after CoT via activation probing. In *Conference on Language Modeling (COLM)*, 2026.

Ida Momennejad, Hosein Hasanbeig, Felipe Vieira Frujeri, Hiteshi Sharma, Nebojsa Jojic, Hamid Palangi, Robert Ness, and Jonathan Larson. Evaluating cognitive maps and planning in large language models with cogeval. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=VtkGvGcGe3.

Allen Newell and Herbert A. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ, 1972.

Richard E Nisbett and Timothy D Wilson. Telling more than we can know: Verbal reports on mental processes. *Psychological review*, 84(3):231, 1977.

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. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

John O'Keefe and Lynn Nadel. *The Hippocampus as a Cognitive Map*. Oxford University Press, 1978.

Alexander Panfilov, David Schmotz, Ilia Shumailov, Luca Beurer-Kellner, Joachim Schaeffer, Ameya Prabhu, Jonas Geiping, and Maksym Andriushchenko. Stealing reasoning traces from proprietary LLM APIs. *arXiv preprint arXiv:2608.09867*, 2026.

Swapnil Parekh and Naman Goyal. Drop the act: Probe-filtered rl for faithful chain-of-thought reasoning, 2026. URL https://arxiv.org/abs/2605.11467.

Jacob Pfau, William Merrill, and Samuel R. Bowman. Let's think dot by dot: Hidden computation in transformer language models. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=NikbrdtYvG.

Santhosh Kumar Ramakrishnan, Erik Wijmans, Philipp Kraehenbuehl, and Vladlen Koltun. Does spatial cognition emerge in frontier models? In *International Conference on Learning Representations*, volume 2025, pp. 101439–101498, 2025.

Roger Ratcliff. A theory of memory retrieval. *Psychological review*, 85(2):59, 1978.

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. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Barbara J Sahakian, Robin G Morris, John L Evenden, Andrew Heald, Raymond Levy, Michael Philpot, and Trevor W Robbins. A comparative study of visuospatial memory and learning in alzheimer-type dementia and parkinson's disease. *Brain*, 111(3):695–718, 1988.

Roger N. Shepard and Jacqueline Metzler. Mental rotation of three-dimensional objects. *Science*, 171(3972):701–703, 1971.

Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Nikul Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. Layer by layer: Uncovering hidden representations in language models. In *International Conference on Machine Learning*, 2025.

Edward C. Tolman. Cognitive maps in rats and men. *Psychological Review*, 55(4):189–208, 1948.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 74952–74965, 2023.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=YXogl4uQUO.

David Waller and Lynn Nadel (eds.). *Handbook of Spatial Cognition*. American Psychological Association, Washington, DC, 2013.

Caroline Wang, Daniel Kasenberg, Kim Stachenfeld, and Pablo Samuel Castro. Discovering differences in strategic behavior between humans and LLMs. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Y9RqhWmcyK.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=NpsVSN6o4ul.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 24824–24837, 2022.

Stephen J Wright. Coordinate descent algorithms. *Mathematical programming*, 151(1):3–34, 2015.

Yutaro Yamada, Yihan Bao, Andrew Kyle Lampinen, Jungo Kasai, and Ilker Yildirim. Evaluating spatial understanding of large language models. *Transactions on Machine Learning Research (TMLR)*, 2024.

Jihan Yang, Shusheng Yang, Anjali W Gupta, Rilyn Han, Li Fei-Fei, and Saining Xie. Thinking in space: How multimodal large language models see, remember, and recall spaces. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10632–10643. IEEE, 2025.

Yuyou Zhang, Radu Corcodel, Chiori Hori, Anoop Cherian, and Ding Zhao. SpinBench: Perspective and rotation as a lens on spatial reasoning in VLMs. In *International Conference on Learning Representations (ICLR)*, volume 2026, pp. 70072–70141, 2026.

APPENDIX CONTENTS

A Additional Experiments **19**

B Behavioral Interpretability for Egocentric Navigation **19**

B.1 Mapping egocentric inputs to symbolic states 19

B.2 Route Retracing 19

B.3 Shortcut Discovery 20

C Behavioral influence **22**

D Implementation details **23**

D.1 Model inference 23

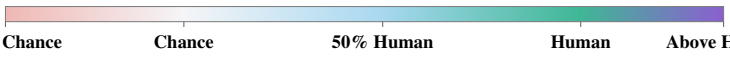
D.2 Fitted parameters 23

D.3 Seed programs for evolution 27

D.4 Evolutionary search implementation 30

E Abstract Situations and Actions **36**

Table 1: **SPACE benchmark results.** Models are grouped by release year and family, and sorted by capacity within each family. Small-scale tasks are grouped by the ability they probe, ordered from strongest to weakest model performance: attention & memory, perception, then visualization. Large-scale tasks evaluate cognitive mapping via metric layout estimation and navigation. Results are the mean over 3 trials. Cells are color-mapped relative to task chance and human performance as defined in the continuous color bar above, with darker fills indicating stronger performance. Cells with hyphen (—) were not evaluated since the models lacked the necessary context length. Cells marked with * are still being evaluated and will be filled in for the final version. The **Avg.** column reports the mean across all 15 tasks (with unavailable tasks treated as 0). Model names marked with † cannot be disclosed due to contractual obligations; the final version will feature real model names.



Release Year	Model	Small-scale spatial cognition										Large-scale spatial cognition					Avg.
		Attention & Memory					Perception		Visualization			Mapping & Navigation					
		CBTT	SAtt	SAdd	CSWM	MCT	WLT	JLO	MRT	PTT	MPFB	Dir. Est.	Dist. Est.	Map Ske.	RR	SD	
	Chance	25.0	25.0	25.0	33.8	0.0	25.0	25.0	25.0	25.0	25.0	25.0	25.0	25.0	0.0	0.0	20.6
	2024 Models																
	MiniCPM-V-2.6	29.8	28.2	14.3	23.3	3.7	17.0	15.8	17.3	10.7	27.5	13.4	23.5	24.2	0.0	0.0	16.6
	Qwen2-VL-7B	39.3	52.7	22.2	29.3	1.5	22.3	26.0	23.8	5.3	31.3	3.2	33.2	10.6	0.0	—	20.1
	Qwen2-VL-72B	52.5	76.8	25.8	34.7	8.1	30.5	22.7	24.4	28.0	30.3	27.1	35.1	24.7	0.0	0.0	28.0
	Pixtral-12B	—	28.3	27.5	25.3	16.3	39.2	27.5	28.2	23.2	32.2	21.2	30.2	—	0.0	0.0	19.9
	InternVL2.5-8B	42.8	44.3	16.8	20.7	0.7	26.5	28.3	24.3	5.8	23.0	8.1	30.7	21.4	0.0	0.0	19.6
	2025 Models																
	MiniCPM-o-2.6	40.3	40.5	22.5	12.7	11.1	24.7	23.5	25.2	12.7	22.8	20.1	26.9	22.5	0.0	0.0	20.4
	Qwen2.5-VL-3B	28.2	27.5	16.5	20.7	0.0	21.8	24.2	24.3	8.2	25.7	7.4	25.5	21.1	0.0	0.0	16.7
	Qwen2.5-VL-7B	47.3	39.9	21.5	40.7	8.1	38.7	24.8	27.2	18.6	20.3	23.4	28.6	26.7	0.0	0.0	24.4
	Qwen2.5-VL-32B	89.3	70.0	62.0	24.0	24.4	31.7	31.5	25.7	30.0	30.8	31.0	37.6	34.2	3.3	0.0	35.0
	Qwen2.5-VL-72B	96.3	74.8	49.3	15.3	37.8	25.5	29.0	27.6	36.1	30.3	23.6	38.4	32.5	4.4	1.1	34.8
	Kimi-VL-A3B	21.7	10.2	21.7	26.0	0.0	36.5	24.7	4.6	17.2	26.3	0.0	0.0	0.0	0.0	0.0	12.6
	Kimi-VL-A3B-Think	14.5	33.2	20.8	30.0	0.7	21.0	21.0	11.0	7.6	15.5	4.9	16.9	16.4	0.0	0.0	14.2
	InternVL3-8B	43.7	51.2	16.2	18.7	5.2	31.3	24.5	25.0	21.5	26.2	25.6	34.4	23.6	0.0	1.1	23.2
	InternVL3-14B	49.3	67.1	42.3	18.7	25.2	34.5	24.7	30.5	25.8	25.0	29.2	27.7	0.0	0.0	0.0	26.7
	InternVL3-38B	48.2	73.8	47.0	29.3	23.0	44.3	26.7	28.1	27.7	27.0	27.8	35.7	0.0	1.1	0.0	29.3
	GPT-5	99.8	75.2	60.0	92.0	*	85.0	35.5	34.5	67.7	29.2	33.4	65.4	66.9	40.0	30.0	54.3
	InternVL3.5-1B	32.7	26.9	5.3	26.7	0.0	24.7	17.7	16.3	2.7	28.7	5.4	22.1	21.9	0.0	0.0	15.4
	InternVL3.5-8B	44.5	71.2	42.8	26.7	0.7	39.8	27.5	28.9	29.6	38.7	25.7	38.0	25.6	5.6	5.6	30.0
	InternVL3.5-38B	42.7	82.8	61.8	28.0	14.8	28.0	32.5	27.5	32.4	29.2	32.8	34.1	41.1	12.2	8.9	33.9
	InternVL3.5-241B-A28B	67.0	82.4	72.2	26.7	4.4	53.0	34.0	28.2	31.8	33.3	34.2	49.4	34.4	10.0	15.6	38.4
	Qwen3-VL-8B-Think	88.0	74.8	55.8	41.3	4.4	28.3	29.8	31.5	40.9	30.3	31.4	60.0	38.6	2.2	2.2	37.3
	Qwen3-VL-30B-A3B-Think	89.8	86.0	61.3	41.3	6.7	53.3	33.8	27.8	49.5	26.0	26.1	40.7	44.2	0.0	0.0	39.1
	Qwen3-VL-32B-Think	99.5	92.3	63.8	44.0	25.9	70.5	42.3	27.4	53.5	38.7	37.2	59.8	62.8	10.0	8.9	49.1
	Anonymous-5wkr †	79.0	87.0	81.5	93.3	*	26.8	37.0	30.0	68.5	44.0	31.3	17.3	10.8	2.2	3.3	40.8
	GPT-5.2	100.0	99.3	100.0	99.3	*	91.5	76.0	30.4	88.5	32.2	42.6	64.6	71.4	33.3	25.6	63.6
	Molmo2-8B	34.0	63.8	24.3	32.0	19.3	30.0	27.5	24.0	24.1	21.2	26.7	0.0	25.6	0.0	0.0	23.5
	2026 Models																
	Qwen3.5-4B	91.2	94.8	81.7	19.3	26.7	66.8	51.3	33.3	57.9	25.8	29.8	58.4	50.8	6.7	3.3	46.5
	Qwen3.5-9B	99.2	98.1	87.8	30.0	37.0	76.2	64.2	34.8	71.6	23.3	32.2	54.3	55.0	3.3	2.2	51.3
	Qwen3.5-27B	99.3	99.7	98.8	85.3	50.4	89.8	75.8	32.7	69.4	33.7	36.0	64.0	69.2	13.3	12.2	62.0
	Qwen3.5-35B-A3B	99.2	99.3	95.2	48.7	44.4	83.8	67.0	32.7	82.6	31.7	32.7	60.6	65.0	7.8	16.7	57.8
	Qwen3.5-122B-A10B	99.0	99.9	94.5	66.0	45.9	91.0	73.5	33.4	79.2	31.0	32.9	63.3	68.3	15.6	8.9	60.2
	Qwen3.5-397B-A17B	99.7	99.8	95.2	95.3	57.0	92.2	77.5	30.9	81.7	35.8	36.9	61.7	75.3	28.9	25.6	66.2
	Anonymous-6xpm †	99.7	100.0	99.3	96.7	*	81.0	82.8	22.7	95.2	29.2	33.1	51.6	33.3	0.0	0.0	55.0
	GPT-5.4 nano	93.2	98.0	94.8	43.3	*	42.3	51.5	27.4	50.9	26.8	28.9	56.5	52.5	6.7	4.4	45.2
	GPT-5.4 mini	89.5	97.8	95.3	30.0	*	88.3	74.7	34.9	93.2	32.7	34.8	54.3	61.4	12.2	6.7	53.7
	GPT-5.4	99.8	100.0	100.0	100.0	60.0	93.0	82.2	31.3	95.7	60.3	42.0	72.4	75.6	55.6	31.1	73.3
	Gemma-4-26B-A4B	81.3	96.2	88.0	82.7	40.0	65.5	68.3	29.7	79.3	24.5	31.8	56.9	46.4	7.8	6.7	53.7
	Gemma-4-31B	100.0	98.5	97.5	97.3	49.6	82.0	75.7	33.4	82.3	41.0	31.4	63.5	64.4	14.4	11.1	62.8
	Kimi-K2.6	99.7	98.3	91.0	98.7	53.3	80.5	67.2	24.6	79.8	6.2	32.7	66.0	55.6	37.8	*	59.4
	Kimi-K3	100.0	100.0	100.0	100.0	63.0	98.0	86.0	26.8	99.2	59.3	48.6	66.4	87.5	45.6	38.9	74.6
	Qwen3.6-27B	99.5	100.0	96.8	92.0	48.1	88.5	69.5	29.3	73.2	35.3	33.4	62.1	62.5	12.2	22.2	61.7
	Qwen3.6-35B-A3B	98.5	99.6	94.2	45.3	38.5	82.7	69.3	32.1	81.3	27.7	36.6	59.9	70.8	14.4	15.6	57.8
	Anonymous-6tdy †	100.0	99.9	95.7	96.7	*	87.8	85.5	23.7	98.5	43.5	29.3	57.8	51.1	0.0	0.0	58.0
	Anonymous-6rhc †	100.0	100.0	100.0	100.0	60.0	99.5	80.5	*	100.0	99.7	51.9	57.8	91.1	53.3	40.0	75.6
	Anonymous-6kwa †	100.0	100.0	96.7	99.3	*	88.3	84.0	26.7	98.0	45.3	43.0	68.3	67.5	4.4	0.0	61.4
	Human	**100.0**	**95.0**	**98.0**	**100.0**	**100.0**	**94.0**	**82.0**	**78.5**	**80.0**	**84.0**	**82.8**	**83.2**	**96.6**	**100.0**	**100.0**	**91.6**

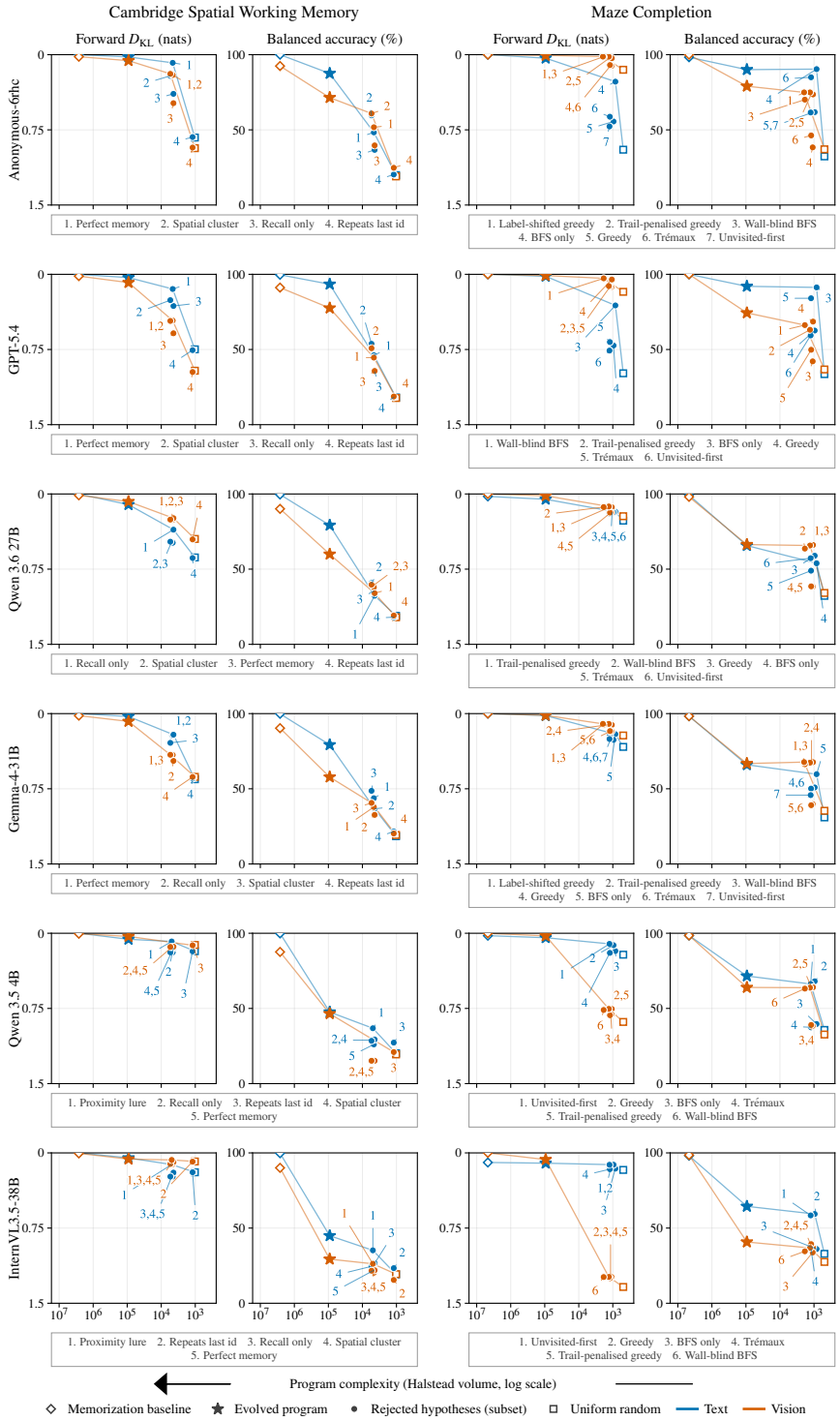


Figure 8: **Behavioral alignment of unified programs on CSWM and Maze Completion.** We show the behavioral fitness of the unified program per task to each model (rows). For each task, the left panel shows situational forward D_{KL} and the right panel shows the stratum-balanced next-action prediction accuracy. Blue and orange show text and vision variants respectively. Uniform random control (□), the best evolved program (★), memorization baseline (◇), and subset of other hypotheses (•) that were considered and rejected by the evolutionary search. The horizontal axis is program complexity (Halstead, 1977). Both axes are oriented so that the top right is best.

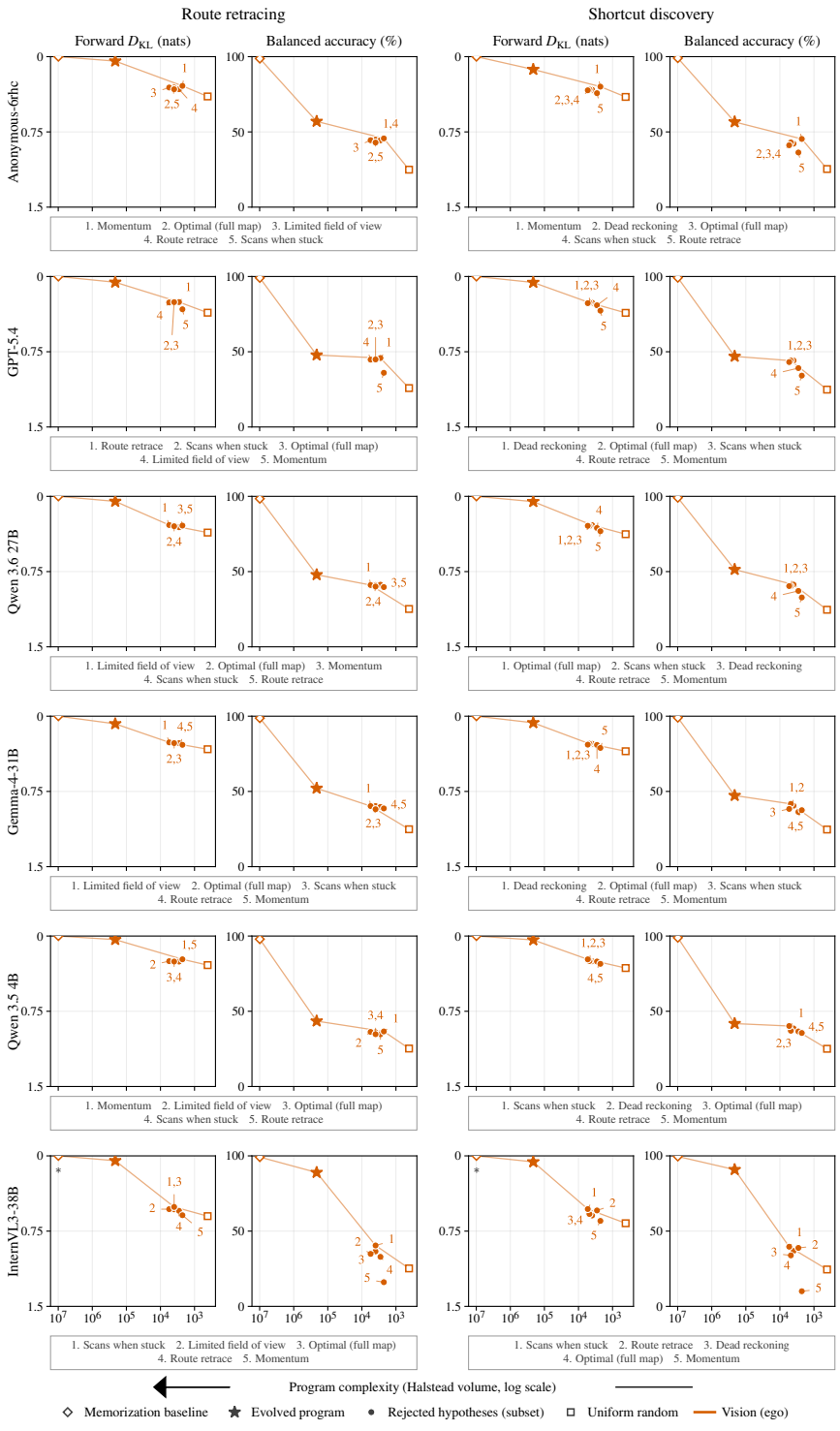


Figure 9: **Per-model behavioral alignment of programs vs. program complexity: Route retracing and Shortcut discovery.** As in Figure 8, for the egocentric tasks, which are vision-only. These tasks were run on InternVL3-38B rather than InternVL3.5-38B. * marks the two cells where the agent loops and the memorization baseline sees too few distinct observations to serve as a meaningful ceiling.

A Additional Experiments

SPACE benchmark results. Table 1 reports the performance of every evaluated model on all 15 SPACE tasks, including the models whose behavior we synthesize programs for.

Fidelity of programs per model. Figures 8 and 9 break down the fidelity measurement of Section 4.2 (Figure 5) by model, for CSWM and Maze Completion and for the egocentric tasks respectively. For each model and task they show the situational forward D_{KL} and the next-action prediction accuracy of the evolved program against the uniform random control, the memorization baseline, and the seed hypotheses that the evolutionary search considered and rejected.

B Behavioral Interpretability for Egocentric Navigation

In Sections 4.3 and 4.4, we performed behavioral interpretation of several models on Cambridge Spatial Working Memory and Maze Completion. A key enabling factor was the ability to precisely remap the textual / visual inputs of LLMs to symbolic variables that the programs can operate on, i.e., a program's inputs match the LLM's inputs. However, such a mapping does not exist for egocentric navigation tasks. The LLM here only receives RGB camera observations combined with brief text prompts to convey the task details. To approximate this mapping from model inputs to program variables, we replace camera images with an accurate 2D reconstruction of the world visible from the agent's camera (or the video walkthrough). While this approximation limits a program's ability to mimic the LLM behaviors, we nevertheless derive useful insights, as we will discuss next.

B.1 Mapping egocentric inputs to symbolic states

First, we have to map the LLM's observations to a program's symbolic inputs. With access to the underlying 3D scene, we can construct the scene geometry, and determine what is visible from each recorded pose in the walkthrough video and the agent's camera poses. We build a 2 cm resolution occupancy map inside the camera's 90° field of view and pool this map onto the 0.25 m lattice that the agent's actions are performed on. Each grid cell of the map defines whether that location in the 3D scene is navigable, free space, or unknown (e.g., due to occlusions). At each step, a program receives a partially observed top-down occupancy map of the scene geometry, the agent's camera pose (x, y, θ), and information about wall paintings visible in the scene such as their positions in the array, their category and whether they are the goal. This reconstruction provides perfect perception near the agent's vicinity. However, we introduce mechanisms in the seed programs to simulate perception failures such as narrow field-of-view, and frame dropouts from the walkthrough. Next, we study the programs evolved for the route retracing and shortcut discovery tasks.

B.2 Route Retracing

The model watches an egocentric video of a route walkthrough from a start location to a goal painting. It is then placed back at the start, and must retrace the route using FORWARD, LEFT, RIGHT and STOP actions. We present this task from the top-down symbolic perspective to programs. Figure 10 (top) shows the evolved program, which consists of five stages. **1) Memory:** the program stores the walkthrough as a route R of camera poses, and optionally builds an initial occupancy map M_0 from the walkthrough, either every visible cell or only those within d_{far}. **2) Perception limits:** to capture potential perception issues in models, the program keeps every s-th waypoint of the route, updates the map using optionally degraded version of the occupancy cells that the model gets at each step. This includes limiting the perception to a field-of-view α_{fov} and depth d_{far}, and retaining cells within depth d_{near}. **3) Planning:** the program localizes the agent at the nearest route waypoint k_t and aims at p_{target}, the waypoint L steps further along, or at the goal once it is visible. When on route, it heads straight to the target. Otherwise, it uses shortest-path search over the map to find a way to the target. **4) Action selection:** every action starts from a prior $w_0 = [w_0^f, w_0^t, w_0^s]$ that sets the baseline score per action. FORWARD is scored by how well the current heading θ points at the target p_{target} ($w_{\mathrm{a}}^{\mathrm{f}}$), or updated by a constant $w_{\mathrm{blk}}^{\mathrm{f}}$ when the cell ahead is blocked. Turn actions are scored by how much it improves the aim to the target ($w_{\mathrm{a}}^{\mathrm{t}}$) and the free space available in the direction after turning ($w_{\mathrm{clr}}^{\mathrm{t}}$). Forward and turn actions are also scored by habits: repeating the previous action (w_{mom}) and

Unified program evolved for Route Retracing

Task objective: The agent watches a walkthrough video of a route to a goal in a continuous 3D environment, then retraces that route using translation and rotation actions, executing STOP on reaching the goal.

State: Walkthrough $V = \langle \mathbf{p}_k, \theta_k, C_k \rangle_{k=1}^{K}$, pose $\mathbf{p}$, heading θ, visible cells C, occupancy map M, previous action a_{t-1}.

Output: Action distribution $\pi_t(a)$ over $a \in \{\text{FWD}, \text{LEFT}, \text{RIGHT}, \text{STOP}\}$.

```
// Memory: walkthrough encoding
R ← {v_1, v_2, ..., v_K}
// Initial map: by distance at most, never by the cone
M_0 ← ⋃_{k∈V} {c ∈ C_k : ‖c − p_k‖ ≤ d_far}   depth
      ⋃_{k∈V} C_k                             all
      ∅                                       none
// Perception limits
// Keep every s-th waypoint of the route
R̂ ← R[:: s]
// Cone α_fov out to d_far, or anything within d_near
Ĉ ← {c : ‖c − p‖ ≤ d_far
      ∧ |∠c − θ| ≤ α_fov/2
      ∨ ‖c − p‖ ≤ d_near}
// Update the map, starting from M_0
M ← M ∪ Ĉ
// Planning: localize and aim
// Nearest route step, searched near the last one
k_t ← localize(R̂, p)
g_t ← ‖p_{k_t} − p‖
// Target: the goal if visible, else L steps ahead
p_target ← p_goal if visible, else p_{k_t+L}
// Detour through the map when off route
b_t ← ∠BFS(M, p, p_target)_1   g_t > d_off
      ∠p_target                otherwise
// Action selection
// Forward: prior and floor if blocked, else aim and momentum
S[FWD] ← w_0^f + w_blk^f   if blocked(M)
         w_0^f + w_a^f align(b_t − θ)   otherwise
               + w_mom^f [a_{t−1} = FWD]
S[LEFT] ← w_0^t + w_a^t Δalign(b_t, θ − 30°)
               + w_clr^t clear(M, θ − 30°)
               + w_mom^t [a_{t−1} = LEFT]
               + w_rev^t [a_{t−1} = RIGHT]
S[RIGHT] ← w_0^t + w_a^t Δalign(b_t, θ + 30°)
               + w_clr^t clear(M, θ + 30°)
               + w_mom^t [a_{t−1} = RIGHT]
               + w_rev^t [a_{t−1} = LEFT]
// Reaching: stop
// Prior, end-of-route drive, goal proximity
S[STOP] ← w_0^s + w_end [k_t ≥ |R̂| − 1]
               + w_stop ρ(d_goal)
// Readout
π_t ← (1 − ε) norm(S) + ε/4          ν = lin
      (1 − ε) softmax(S/τ) + ε/4     ν = exp
```

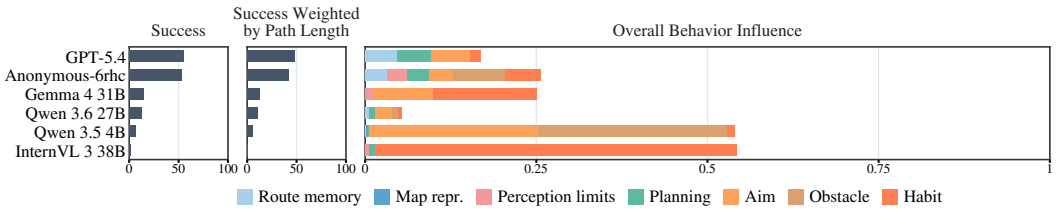


Figure 10: **(Top)** Pseudo-code of the unified program for route retracing. The parameters highlighted in color vary across models. **(Bottom)** Task performance and behavioral influence. Task metrics are success and success weighted by path length (SPL) from Table 1. The stacked bars show route memory, map representation, perception limits, planning and the components of action selection (aim, obstacle and habit). Each segment is how much worse the program explains the model's actions without that behavior (rise in negative log-likelihood, nats per decision). A behavior's influence is the largest rise among its variables (see Appendix C). The action priors are not shown.

reversing the previous turn ($w^{\text{t}}_{\text{rev}}$). **5) Reaching:** stop is scored by a prior (w^{s}_0), a bonus at the end of the route (w_{end}), and proximity to the goal (w_{stop}). Finally, the scores are normalized linearly or through a softmax and mixed with a uniform distribution.

Figure 10 (bottom) compares the influence of these behaviors across models. The most successful models, GPT 5.4 and Anonymous-6rhc, navigate under the combined influence of route memory, planning and low-level navigation behaviors such as aiming and habitual momentum to keep moving forward. Anonymous-6rhc also explicitly models obstacle avoidance and that plays an important role in navigation. All other models perform poorly on this task and the influence values explain why. Route memory and planning do not play a significant role in their decision making. Instead, Gemma 4 31B and InternVL 3 38B are driven by habitual oscillations, and Qwen 3.5 4B by moving forward and reacting to obstacles.

B.3 Shortcut Discovery

The model watches an egocentric video of a deliberately long route walkthrough from a start location to a goal painting. It is then placed back at the start, and must reach the same goal by a shorter path using forward, left, right and stop actions. As in route retracing, we present this task to programs from the top-down symbolic perspective. Figure 11 (top) shows the evolved program, which consists of five stages. **1) Memory:** the program stores the walkthrough as a route R of camera poses, remembers the goal cell g seen in it, and builds an initial occupancy map M_0 from the walkthrough, using either every visible cell, only those within d_{far}, or those that pass the same perception-limit filtering that happens during navigation. **2) Perception limits:** to capture perception limits of models, the program keeps every s-th waypoint of the route, updates the map with the degraded version of occupancy cells that the model gets at each step. This limits perception to a field-of-view α_{fov} and maximum depth of d_{far}, but includes anything within d_{near}. The goal cell is updated whenever

Unified program evolved for Shortcut Discovery

Task objective: The agent watches a walkthrough video of a route to a goal in a continuous 3D environment, then reaches the same goal by a shorter path than the one demonstrated using translation and rotation actions, executing STOP on reaching the goal.

State: Walkthrough $V = \langle \mathbf{p}_k, \theta_k, C_k, P_k \rangle_{k=1}^{K}$, pose $\mathbf{p}$, heading θ, visible cells C, visible paintings P, occupancy map M, goal cell g, previous action a_{t-1}.

Output: Action distribution $\pi_t(a)$ over $a \in \{\text{FWD}, \text{LEFT}, \text{RIGHT}, \text{STOP}\}$.

```
// Memory: walkthrough encoding
R ← {v_1, v_2, ..., v_K}
g ← goal(V)
// Initial map: all, by distance, or through the gate
M_0 ← ⋃_{k∈V} C_k                                              all
      ⋃_{k∈V} {c ∈ C_k : ‖c − p_k‖ ≤ d_far}                    depth
      ⋃_{k∈V} C̃_k                                              cone
// Perception limits
// Keep every s-th waypoint of the route
R̂ ← R[:: s]
// Cone α_fov out to d_far, or anything within d_near
C̃ ← {c : ‖c − p‖ ≤ d_far
        ∧ |∠c − θ| ≤ α_fov/2
        ∨ ‖c − p‖ ≤ d_near}
// Update the map from M_0, and the goal if sighted
M ← M ∪ C̃,   g ← goal(P) if sighted
// Planning: localize, target and plan
// Nearest route step, over R̂ or near the last one
k_t ← localize(R̂, p)
// Target: the goal region, or the route step L ahead
p_target ← region(g)                                           goal
           p_{k_t+L}                                           route
// Shortest path over the map to the target
Π ← BFS(M, p, p_target)
// Bearing and its source: plan, else route, else goal
b_t, σ_t ← ∠Π_{L},  plan                                       Π ≠ ∅
           ∠p_{k_t+L},  route                                  R̂ ≠ ∅
           ∠g,  goal                                           g in view
// Action selection
// Aim weight for the bearing's source
w_t ← w_{σ_t} ∈ {w_plan, w_route, w_goal}
S[FWD] ← w_0^f + w_blk^f                                       if blocked(M)
         w_0^f + w_t align(b_t − θ)
               + w_free^f                                      otherwise
               + w_mom^f [a_{t−1} = FWD]
S[LEFT] ← w_0^t + w_t Δalign(b_t, θ − 30°)
               + w_clr^t clear(M, θ − 30°)
               + w_mom^t [a_{t−1} = LEFT]
               + w_rev^t [a_{t−1} = RIGHT]
S[RIGHT] ← w_0^t + w_t Δalign(b_t, θ + 30°)
               + w_clr^t clear(M, θ + 30°)
               + w_mom^t [a_{t−1} = RIGHT]
               + w_rev^t [a_{t−1} = LEFT]
// Reaching: stop
S[STOP] ← w_0^s + w_stop ρ(d_goal)
// Readout
π_t ← (1 − ε) norm(S) + ε/4                                    ν = lin
      (1 − ε) softmax(S/τ) + ε/4                               ν = exp
```

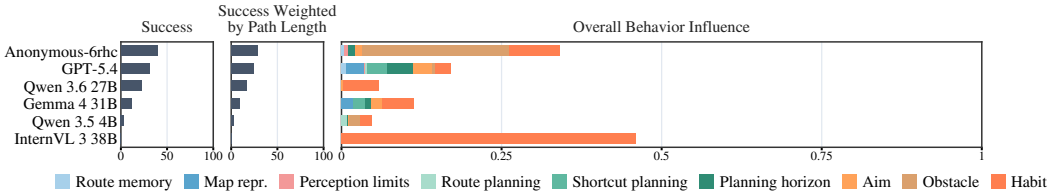


Figure 11: **(Top)** Pseudo-code of the unified program for shortcut discovery. The parameters highlighted in color vary across models. **(Bottom)** Task performance and behavioral influence. Task metrics are success and success weighted by path length (SPL) from Table 1. The stacked bars show route memory, map representation, perception limits, planning and the components of action selection (aim, obstacle and habit). Each segment is how much worse the program explains the model's actions without that behavior (rise in negative log-likelihood, nats per decision). A behavior's influence is the largest rise among its variables (see Appendix C). Planning is split into route planning (steering along the demonstrated route), shortcut planning (a shortest-path search over the map to the goal) and the planning horizon L. The action priors are not shown.

the goal painting is sighted. **3) Planning:** the program localizes the agent on the route, picks either the region around the goal or the waypoint L-steps ahead along the route as the target, and runs a shortest-path search over the map to that target. The steering bearing b_t follows the planned path if one exists, else the route, else the goal when it is in view. **4) Action selection:** every action starts from a prior (w_0^{f}, w_0^{t}). The aim weight w_t depends on where the bearing came from (w_{plan}, w_{route} or w_{goal}). FORWARD receives a blocked term $w_{\text{blk}}^{\text{f}}$ when the cell ahead is blocked, and otherwise is scored by how well the heading points along the target bearing and by a free-forward bonus $w_{\text{free}}^{\text{f}}$. Turn actions are scored by how much they improve alignment with b_t and by the free space available after turning ($w_{\text{clr}}^{\text{t}}$). Forward and turn actions are also scored by habits: repeating the previous action (w_{mom}) and reversing the previous turn ($w_{\text{rev}}^{\text{t}}$). **5) Reaching:** stop is scored by a prior (w_0^{s}) and proximity to the goal (w_{stop}). Finally, the scores are normalized linearly or through a softmax and mixed with a uniform distribution.

Figure 11 (bottom) compares the influence of these behaviors across models. All models perform poorly on this task, with the best model Anonymous-6rhc achieving only 40% success. Anonymous-6rhc is driven primarily by repeatedly moving forward while avoiding obstacles, and has very limited influence from planning and memory. GPT 5.4 on the other hand exhibits signs of shortcut planning with an influential planning horizon term, but its performance is still inhibited by frequent turning in place (only 31% of its actions are forward movements, against Anonymous-6rhc's 43%). As in route retracing, the models performing poorly primarily exhibit signs of repetitive actions, or forward movements with some collision avoidance.

C BEHAVIORAL INFLUENCE

In Figures 6, 7, 10 and 11, we presented pseudo-code of unified programs we evolved to explain the behaviors of models on each task. Additionally, we visualized the behavioral influence of each mechanism to compare different LLMs perform the same task. We now describe the motivation and definition of behavioral influence.

Motivation. The fitted parameters of a unified program do not directly convey how much a mechanism matters. They have different units (weights, distances, angles, step counts), interact through the readout, and correlated terms can trade off against each other. Instead, we ask how much of a model's behavior is explained by a mechanism: we remove the mechanism from the model's fitted program and measure how much worse the program predicts the model's own actions.

Definition. Let π_{θ_m} be the unified program with the fitted parameters of the LLM μ_m, and let $\mathcal{D}_m$ be the decisions (x_t, a_t) recorded from the μ_m on the task. The program is replayed on $\mathcal{D}_m$ with the LLM's own history (teacher forcing), so its states are built from the model's past actions and every variant of the program is scored on exactly the same decisions. The loss is the negative log-likelihood from Equation 1, evaluated on the model's recorded decisions:

$$\mathcal{L}_m(\theta) = \frac{1}{|\mathcal{D}_m|} \sum_{(x_t, a_t) \in \mathcal{D}_m} -\log \pi_\theta(a_t \mid x_t). \tag{3}$$

Removing a variable v sets it to its no-op value, the value at which its term has no effect, giving parameters $\theta_m^{(-v)}$. The behavioral influence $I_{m,k}$ of a mechanism k on model μ_m through its variables $\mathcal{V}_k$ (highlighted in the pseudo-code) is the largest rise in loss caused by removing any one of them[5]:

$$I_{m,k} = \max_{v \in \mathcal{V}_k} \max\Big(0,\ \mathcal{L}_m\big(\theta_m^{(-v)}\big) - \mathcal{L}_m(\theta_m)\Big). \tag{4}$$

$I_{m,k}$ is measured in nats per decision. Intuitively, this means that the program's probability of the model's action shrinks by $e^{-I_{m,k}}$ without the mechanism. The stacked bars show $I_{m,k}$ for each mechanism.

No-op values. Each kind of variable has its own neutral value:

- **Additive weights** are set to 0, which removes their term, e.g., raster sweep $\mathrm{w}_{\mathrm{raster}}$ in CSWM, shortest-path admissibility w_{sp} in MCT, and momentum w_{mom} in route retracing.
- **Suppression factors** are set to 1, so boxes that held a treasure or were found empty are no longer pruned ($\mathrm{m}_{\mathrm{found}}$, $\mathrm{m}_{\mathrm{miss}}$ in CSWM).
- **Perception limits** are made unlimited: an unbounded depth d_{far}, a full 360° field of view α_{fov}, and no near-field override d_{near}.
- **Integer resource bounds** are set to 1: a lookahead of $L = 1$ aims at the next waypoint, and a stride of $s = 1$ keeps every walkthrough frame.
- **Discrete switches** turn a whole mechanism off: the remembered route R is cut to the final walkthrough frame, and no initial map M_0 is built from the walkthrough. In shortcut discovery, route planning takes no bearing from the route and shortcut planning runs no search over the map to the goal; in both cases the program falls back to its own next target.

Some variables are removed together because their terms substitute for one another. The three wall-blind goal-attraction weights in MCT ($\mathrm{w}_{\mathrm{man}}$, $\mathrm{w}_{\mathrm{eucl}}$, $\mathrm{w}_{\mathrm{dir}}$) all favor moving toward the goal. Removing only one of them barely changes the program's predictions, because the other two still push it toward the goal, so each would look inert even when goal attraction explains much of the model's behavior. Removing all three at once measures the mechanism itself. For the same reason, memory-guided pruning in CSWM is removed as a whole by switching off the suppression of known boxes.

[5] For example, for the mechanism 'topological planning' in Maze Completion, the variables are the geodesic progress, shortest-path and cul-de-sac weights $\mathrm{w}_{\mathrm{geo}}$, w_{sp} and $\mathrm{w}_{\mathrm{cds}}$.

Figure 12: **Fitted parameters of the unified MCT program.** Left: the program, with each mechanism's lines banded in the colour its influence is drawn in (Figure 6). Right: the fitted value of every variable for each model, in text (T) and vision (V), grouped by mechanism in the same colours. Faint values are weights fitted to zero, which switch their term off for that model.

Unified program for Maze Completion

```
// Shortest-path planning
// Goal arrival check: commit stop on goal
if s = g then return {stop : 1.0}
for all waypoints w, target t = W(w) do
    // Wall-aware distance, shortest path, cul-de-sacs
    Δd_geo ← d_geo(s, g) − d_geo(t, g)
    I_sp ← I[d_geo(s, t)+d_geo(t, g)=d_geo(s, g)]
    S_plan ← w_geo Δd_geo + w_sp I_sp
             − w_cds cds(t)
    // Goal attraction
    // Wall-blind Manhattan & Euclidean progress
    Δd_man ← ‖s − g‖_1 − ‖t − g‖_1
    Δd_eucl ← ‖s − g‖_2 − ‖t − g‖_2
    // Directional bearing to goal
    cos_goal ← ((t−s)·(g−s)) / (‖t−s‖_2 ‖g−s‖_2)
    S_goal ← w_man Δd_man + w_eucl Δd_eucl
             + w_dir cos_goal
    // Kinematics, novelty & readout
    // Local heuristics: straight axis & dead ends
    S_local ← w_axis axis(t) − w_dead I_dead(t)
    // Novelty: frontier reward & loop penalty
    S_novel ← w_unv I[v(w)=0] − w_loop v(w)
    // Total waypoint utility and softmax readout
    S(t) ← S_plan + S_goal + S_local + S_novel
end for
return π(w) ∝ exp(S(t)/τ) + ε
```

Fitted parameters (T: text, V: vision)

		GPT 5.4		Anonymous-6rhc		Gemma 4 31B		Qwen 3.6 27B		Qwen 3.5 4B		InternVL 3.5 38B	
		T	V	T	V	T	V	T	V	T	V	T	V
Planning													
w_{geo}	geodesic	0	0.11	0	0.08	0	0	3.35	0.62	0	0	0	0
w_{sp}	shortest	2.85	0	3.24	0	1.65	0	0	0	0.47	0	0	0
w_{cds}	cul-de-sac	9.62	0	0	0	0	0	5.47	0	0	0	0	0
Goal attraction													
w_{man}	Manhattan	0.26	0	0	0.30	0.30	0	0	0	0.32	0.08	0.35	0.16
w_{eucl}	Euclidean	0	0.60	0	0.07	0	0.98	0	0.93	0	0	0	0
w_{dir}	bearing	0	0.94	1.86	1.26	0	0	1.95	1.69	0.86	0.58	0	0
Local heuristics													
w_{axis}	axis	1.19	0	0	0	0	0	0	0	0	0	0.95	0
w_{dead}	dead end	0	0	2.62	0	1.75	0	5.11	0	0	0	0	0
w_{unv}	unvisited	0	2.12	0	1.46	0	0	0	0	0	0	2.62	0
w_{loop}	revisit	0	0.98	0	1.19	0.10	1.48	0	0	0	0	0	0
Readout													
τ	temperature	1.00	0.94	1.00	1.05	1.00	1.00	5.00	4.31	1.00	1.00	1.00	1.00
ε	lapse	0.173	0.000	0.001	0.000	0.270	0.031	0.200	0.001	0.000	0.022	0.000	0.050

D Implementation Details

D.1 Model Inference

We query API-based models with high thinking effort and serve open-weight models with vLLM with thinking enabled. All models generate up to 32k tokens and use their provider's recommended sampling hyper-parameters (temperature 1).

D.2 Fitted Parameters

Figures 12, 13, 14 and 15 list the fitted parameters of the unified programs for Maze Completion, Cambridge Spatial Working Memory, route retracing and shortcut discovery. Each figure shows the program beside a table of the fitted values of its variables for every model (and modality), grouped by mechanism in the colours of the code.

Figure 13: **Fitted parameters of the unified CSWM program.** Left: the program of Figure 7. Right: the fitted value of every variable for each model, in text (T) and vision (V), grouped by mechanism in the colours of the code. Faint values are weights fitted to zero or variables left at their neutral value, which switch their term off for that model; a dash marks a variable the model's program does not have.

Unified program for Cambridge Spatial Working Memory

// Episodic memory update
// Treasure found: retire box, reset misses
$c_t, s_T \leftarrow$ identify_treasure($\mathcal{B}$)
if $c_t > c_{t-1}$ **then**
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{s_{t-1}\}$; $\mathcal{M} \leftarrow \langle\,\rangle$
// Empty probe: within-round miss
else if $s_{t-1} \notin \mathcal{T} \cup \mathcal{M}$ **then**
  $\mathcal{M} \leftarrow \mathcal{M} \oplus s_{t-1}$

// Memory-guided pruning
for all boxes $s \in \mathcal{B}$ **do**
  // Prune treasures ($s \in \mathcal{T}$) and misses ($s \in \mathcal{M}$)
  $M(s) \leftarrow \mathbf{m}_{\text{found}}^{\mathbb{I}_{s\in\mathcal{T}}} \mathbf{m}_{\text{miss}}^{F(s)+(1-F(s))\,\mathbf{a}_{\text{miss}}}$
  // Fade $F(s)$: $1 \to 0$ as the miss ages
end for
// Demotion set: ranked last, not excluded
$\mathcal{D} \subseteq \mathcal{T} \cup \mathcal{M}$

// Box selection & action distribution
$s_0 \leftarrow$ anchor() // typically previous box s_{t-1}
// Raster sweep: starts at top-left (or s_0), $\mathcal{D}$ last
$\{(a, s, r)\} \leftarrow$ raster_rank($\mathcal{B}$; s_0, $\mathcal{D}$)
for all boxes $(a, s, r) \in \mathcal{B}$ **do**
  // Spatial preference: sweep, proximity, spread
  $S(s) \leftarrow \mathbf{w}_{\text{raster}}\,\text{raster}(r; \lambda_{\text{raster}}) + \mathbf{w}_{\text{prox}}\,\text{prox}(s, s_0; \lambda_{\text{prox}}) + \mathbf{w}_{\text{spread}}\,\text{spread}(s)$
  // Non-spatial visual bias: lure, digits, repeat
  $B(s) \leftarrow \mathbf{b}_{\text{lure}}\mathbb{I}_{s=s_T} + \mathbf{b}_{\text{digit}}L(a) + \mathbf{b}_{\text{rep}}\mathbb{I}_{s=s_{t-1}}$
  $x_a \leftarrow M(s)\cdot[S(s)+B(s)]$ $\quad \nu = \text{lin}$
  $\quad\;\; M(s)\cdot\exp(S(s)+B(s))$ $\quad \nu = \text{exp}$
end for
// Readout: lapse mixture with the uniform
return $\pi(a) \leftarrow (1-\varepsilon)\,\frac{x_a}{\sum_b x_b} + \frac{\varepsilon}{N}$

Fitted parameters (T: text, V: vision)

		GPT 5.4		Anonymous-6rhc		Gemma 4 31B		Qwen 3.6 27B		Qwen 3.5 4B		InternVL 3.5 38B	
		T	*V*	*T*	*V*	*T*	*V*	*T*	*V*	*T*	*V*	*T*	*V*
Memory-guided pruning													
$\mathbf{m}_{\text{found}}$	treasure	0	0	0	0	0	0.01	0	0	0.01–1.0	1.00	1.00	1.00
$\mathbf{m}_{\text{miss}}$	miss	–	0.06	0	0	0.41	0.04	0.09	0.90	0.90	1.00	1.00	1.00
$\mathbf{a}_{\text{miss}}$	faded miss	–	0	0	–	1.00	1.00	1.00	1.00	–	1.00	1.00	1.00
$\mathcal{D}$	demoted	$\mathcal{T}\cup\mathcal{M}$	$\mathcal{T}\cup\mathcal{M}$	$\mathcal{T}\cup\mathcal{M}$	$\mathcal{T}\cup\mathcal{M}$	$\mathcal{T}\cup\mathcal{M}$	$\mathcal{T}\cup\mathcal{M}$	$\mathcal{T}$	$\mathcal{T}\cup\mathcal{M}$	–	–	–	–
Raster sweep													
$\mathbf{w}_{\text{raster}}$	raster	1.00	1.92	3.66	0.44	1.96	1.00	1.00	3.18	2.70	0	1.67	0
λ_{raster}	decay	4.00	–	3.00	–	–	3.00	2.50	0.77	2.00	0	1.00	0
Proximity & spread													
$\mathbf{w}_{\text{prox}}$	proximity	0	0.18	0.61	3.00	0.48	0.58	1.00	1.77	0.85	0	0	0
λ_{prox}	decay	0	0	3.00	–	–	1.44	1.70	0.37	0.16	0	0	0
$\mathbf{w}_{\text{spread}}$	spread	0	0.81	0	2.18	0	0	0	0	0	0	3.66	0
Visual bias													
$\mathbf{b}_{\text{lure}}$	lure	0	0	0	0	0	0.30	0	0	0	3.20	0	0
$\mathbf{b}_{\text{digit}}$	digit	0	0	0.13	0.34	2.48	0	0	0	1.10	0	0.91	0
$\mathbf{b}_{\text{rep}}$	repeat	0	0	0	0	0	0	0	0	1.40	0	0	0.56
Readout													
ν	form	lin	exp	lin	exp	exp	lin	lin	exp	lin	exp	exp	exp
ε	lapse	.001	.009–.14	.010	.010	.001	.010	≈.07	.258	.142	.001	.06–.84	.06

Figure 14: **Fitted parameters of the unified route retracing program.** Left: the program of Figure 10. Right: the fitted value of every variable for each model grouped by mechanism in the colours of the code. Faint values are weights fitted to zero or variables left at their neutral value, which switch their term off for that model; a dash marks a variable the model's program does not have.

Unified program for Route Retracing

```
// Memory: walkthrough encoding
R ← {v_1, v_2, ..., v_K}
// Initial map: by distance at most, never by the cone
M_0 ← ⋃_{k∈V} {c ∈ C_k : ‖c − p_k‖ ≤ d_far}        depth
      ⋃_{k∈V} C_k                                  all
      ∅                                            none
// Perception limits
// Keep every s-th waypoint of the route
R ← R[:: s]
// Cone α_fov out to d_far, or anything within d_near
C̃ ← {c : ‖c − p‖ ≤ d_far
        ∧ |∠c − θ| ≤ α_fov/2
        ∨ ‖c − p‖ ≤ d_near}
// Update the map, starting from M_0
M ← M ∪ C̃
// Planning: localize and aim
// Nearest route step, searched near the last one
k_t ← localize(R̂, p)
g_t ← ‖p_{k_t} − p‖
// Target: the goal if visible, else L steps ahead
p_target ← p_goal if visible, else p_{k_t+L}
// Detour through the map when off route
b_t ← ∠BFS(M, p, p_target)_1                       g_t > d_off
      ∠p_target                                    otherwise
// Action selection
// Forward: prior and floor if blocked, else aim and momentum
S[FWD] ← w_0^f + w_blk^f                           if blocked(M)
         w_0^f + w_a^f align(b_t − θ)              otherwise
               + w_mom^f [a_{t−1} = FWD]
S[LEFT] ← w_0^t + w_a^t Δalign(b_t, θ − 30°)
               + w_clr^t clear(M, θ − 30°)
               + w_mom^t [a_{t−1} = LEFT]
               + w_rev^t [a_{t−1} = RIGHT]
S[RIGHT] ← w_0^t + w_a^t Δalign(b_t, θ + 30°)
               + w_clr^t clear(M, θ + 30°)
               + w_mom^t [a_{t−1} = RIGHT]
               + w_rev^t [a_{t−1} = LEFT]
// Reaching: stop
// Prior, end-of-route drive, goal proximity
S[STOP] ← w_0^s + w_end [k_t ≥ |R̂| − 1]
               + w_stop ρ(d_goal)
// Readout
π_t ← (1 − ε) norm(S) + ε/4                        ν = lin
      (1 − ε) softmax(S/τ) + ε/4                   ν = exp
```

Fitted parameters

		GPT 5.4	Anonymous-6rhc	Gemma 4 31B	Qwen 3.6 27B	Qwen 3.5 4B	InternVL 3 38B
Perception limits							
$\mathbf{d}_{\text{near}}$	near field	–	–	0.35	–	–	0.5
$\mathbf{d}_{\text{far}}$	depth	1	1.07	1	–	1	1.5
$\boldsymbol{\alpha}_{\text{fov}}$	field of view	30	67.5	30	360	30	30
$\mathbf{s}$	stride	1	3	1	1	1	3
Route memory							
$\mathbf{R}$	route	V	V	V	V	V	V
Map representation							
$\mathbf{M}_0$	initial map	all	∅	∅	∅	depth	all
Planning							
$\mathbf{L}$	horizon	5	3	1	4	1	3
$\mathbf{d}_{\text{off}}$	off route	1.33	0.25	0.5	0.952	0	0
Action selection							
$\mathbf{w}_0^{\text{f}}$	forward	0.208	0	0.612	1	0.05	0
$\mathbf{w}_0^{\text{t}}$	turn	0.208	0	0.2	1	0.05	0
$\mathbf{w}_{\text{a}}^{\text{f}}$	aim forward	0.367	1.41	0.612	6.72	1.78	1
$\mathbf{w}_{\text{a}}^{\text{t}}$	aim turn	0.367	0.877	0.365	0.5	0.1	0.5
$\mathbf{w}_{\text{blk}}^{\text{f}}$	blocked	0.367	−0.5	–	1.34	0.634	−1
$\mathbf{w}_{\text{clr}}^{\text{t}}$	clearance	0	1.61	0	0.2	1.11	0
$\mathbf{w}_{\text{mom}}^{\text{f}}$	repeat forward	0.272	0.727	2.83	1.59	0.612	3.9
$\mathbf{w}_{\text{mom}}^{\text{t}}$	repeat turn	0.14	0.727	0.68–2.24	0.1	0.12/0.37	−0.1
$\mathbf{w}_{\text{rev}}^{\text{t}}$	reverse turn	0	0	0.37–4.50	0	0.24	3.16
Reaching							
$\mathbf{w}_0^{\text{s}}$	stop	0.01	−4	0.01	0.001	0.05	−5
$\mathbf{w}_{\text{end}}$	route end	∞	2.13	0.5	0.025–0.1	0.20	3
$\mathbf{w}_{\text{stop}}$	proximity	0	6.07	0.6	0.5	0.5	2
Readout							
$\boldsymbol{\nu}$	form	lin	exp	lin	lin	lin	exp
$\boldsymbol{\tau}$	temperature	–	1.12	–	–	–	1
$\boldsymbol{\varepsilon}$	lapse	0	0	0.215	0.0601	0	0.0634

Figure 15: **Fitted parameters of the unified shortcut discovery program.** Left: the program of Figure 11. Right: the fitted value of every variable for each model grouped by mechanism in the colours of the code. Faint values are weights fitted to zero or variables left at their neutral value, which switch their term off for that model; a dash marks a variable the model's program does not have.

Unified program for Shortcut Discovery

```
// Memory: walkthrough encoding
R ← {v_1, v_2, ..., v_K}
g ← goal(V)
// Initial map: all, by distance, or through the gate
M_0 ← ⋃_{k∈V} C_k                                              all
      ⋃_{k∈V} {c ∈ C_k : ‖c − p_k‖ ≤ d_far}                    depth
      ⋃_{k∈V} C̃_k                                              cone
// Perception limits
// Keep every s-th waypoint of the route
R̂ ← R[:: s]
// Cone α_fov out to d_far, or anything within d_near
C̃ ← {c : ‖c − p‖ ≤ d_far
        ∧ |∠c − θ| ≤ α_fov/2
        ∨ ‖c − p‖ ≤ d_near}
// Update the map from M_0, and the goal if sighted
M ← M ∪ C̃,   g ← goal(P) if sighted
// Planning: localize, target and plan
// Nearest route step, over R̂ or near the last one
k_t ← localize(R̂, p)
// Target: the goal region, or the route step L ahead
p_target ← region(g)                                           goal
           p_{k_t+L}                                           route
// Shortest path over the map to the target
Π ← BFS(M, p, p_target)
// Bearing and its source: plan, else route, else goal
b_t, σ_t ← ∠Π_{1:}, plan                                       Π ≠ ∅
           ∠p_{k_t+L}, route                                   R̂ ≠ ∅
           ∠g, goal                                            g in view
// Action selection
// Aim weight for the bearing's source
w_t ← w_{σ_t} ∈ {w_plan, w_route, w_goal}
S[FWD] ← w^f_0 + w^f_blk                                       if blocked(M)
         w^f_0 + w_t align(b_t − θ)
               + w^f_free                                      otherwise
               + w^f_mom [a_{t−1} = FWD]
S[LEFT] ← w^t_0 + w_t Δalign(b_t, θ − 30°)
          + w^t_clr clear(M, θ − 30°)
          + w^t_mom [a_{t−1} = LEFT]
          + w^t_rev [a_{t−1} = RIGHT]
S[RIGHT] ← w^t_0 + w_t Δalign(b_t, θ + 30°)
           + w^t_clr clear(M, θ + 30°)
           + w^t_mom [a_{t−1} = RIGHT]
           + w^t_rev [a_{t−1} = LEFT]
// Reaching: stop
S[STOP] ← w^s_0 + w_stop ρ(d_goal)
// Readout
π_t ← (1 − ε) norm(S) + ε/4                                    ν = lin
      (1 − ε) softmax(S/τ) + ε/4                               ν = exp
```

Fitted parameters

		GPT 5.4	Anonymous-6rhc	Gemma 4 31B	Qwen 3.6 27B	Qwen 3.5 4B	InternVL 3 38B
Perception limits							
$\mathbf{d}_{\text{near}}$	near field	–	–	–	–	0.5	–
$\mathbf{d}_{\text{far}}$	depth	∞	∞	1	∞	1	4.01
$\boldsymbol{\alpha}_{\text{fov}}$	field of view	360	360	30	360	30	30
$\mathbf{s}$	stride	1	3	1	1	1	1
Route memory							
$\mathbf{R}$	route	V	V	$\emptyset$	$\emptyset$	V	V
Map representation							
$\mathbf{M_0}$	initial map	all	all	all	all	cone	depth
Planning							
$\mathbf{L}$	horizon	4	4	5	1	1	1
Action selection							
$\mathbf{w}^{\text{f}}_{\mathbf{0}}$	forward	1	0	1	0.693	0	1
$\mathbf{w}^{\text{t}}_{\mathbf{0}}$	turn	1	0	1	0.645	0	1
$\mathbf{w}_{\text{goal}}$	goal in view	0.5	0	0	0.5	0	0
$\mathbf{w}_{\text{plan}}$	planned path	1.81	1.05	0.5	0.5	0.5	0.5
$\mathbf{w}_{\text{route}}$	route	0.5	1.05	0	0	0.5	0.2
$\mathbf{w}^{\text{f}}_{\text{blk}}$	blocked	−0.204	−0.375	–	−0.261	–	−1
$\mathbf{w}^{\text{f}}_{\text{free}}$	free ahead	0	2.46	0	0	0.388	0
$\mathbf{w}^{\text{t}}_{\text{clr}}$	clearance	0.5	1.22	0	0.2	0	1.88
$\mathbf{w}^{\text{f}}_{\text{mom}}$	repeat forward	0.1	1.12	2.8	1.48	0.388	3
$\mathbf{w}^{\text{t}}_{\text{mom}}$	repeat turn	1.03	1.12	0	0.294	0	0
$\mathbf{w}^{\text{t}}_{\text{rev}}$	reverse turn	0	0	0	1.03	0	6
Reaching							
$\mathbf{w}^{\text{s}}_{\mathbf{0}}$	stop	0.05	0	0.05	0.001	−2	0.05
$\mathbf{w}_{\text{stop}}$	proximity	2	0	1	0.5	0.5	2
Readout							
$\boldsymbol{\nu}$	form	lin	exp	lin	lin	exp	lin
$\boldsymbol{\tau}$	temperature	1	1.25	1	1	0.969	1
$\boldsymbol{\varepsilon}$	lapse	0.001	0.04	0	0.001	0	0

Table 2: **Seed programs for Cambridge Spatial Working Memory.** The same eight seeds are used for text and vision.

Type	Seed	What it does
Label controls	Uniform	Picks any box label at random.
	Round-robin	Cycles through the labels in a fixed order, ignoring the board.
	Repeats last id	Repeats the label it said last turn, although labels are reshuffled every turn.
	Label frequency	Favors the labels it has said most often so far.
Spatial memory	Perfect memory	Plays correctly: never reopens a box that held a treasure, or one already found empty in the current search.
	Recall only	Avoids boxes that held a treasure, but forgets which boxes it already checked in the current search.
	Spatial cluster	Plays correctly, but searches locally: boxes next to the last one opened, then nearby boxes, then the rest.
	Proximity lure	Is drawn to boxes next to the last one opened, even those it has just found empty.

D.3 Seed programs for evolution

Evolution for each task starts from a small set of hand-written seed programs, listed in Tables 2–4. Each seed states one hypothesis about how a model might approach the task. The seeds span three kinds of hypothesis: *controls* that ignore the spatial structure of the task, a *competence reference* that solves the task correctly, and *bounded variants* that limit one capacity of the reference, such as memory, perception, or orientation. Seeds are fitted in the same way as evolved programs (Section 3) and remain in the population throughout the search.

Table 3: **Seed programs for Maze Completion.** Each step, the model chooses one of the offered waypoints. The vision seeds pair each perception filter with each planner ($5 \times 6 = 30$ seeds), plus the two controls shared with text. A vision seed with the faithful filter takes its planner's name; other filters prefix it (e.g., wall blindness with BFS is *Wall-blind BFS*).

Type	Seed	What it does
Controls	Uniform	Picks any offered waypoint at random.
	Alphabetical bias	Prefers letters early in the alphabet.
Text planners	Greedy	Heads for the waypoint closest to the goal as the crow flies, ignoring walls.
	BFS only	Follows the shortest route through the maze to the goal.
	Trémaux	Explores new branches first, backtracks through branches visited once, and avoids exhausted ones.
	Unvisited-first	Heads straight for the goal, but avoids waypoints it has already visited.
Vision perception	Faithful	Sees the maze as drawn.
	Wall-blind	Misses the interior walls, so the maze looks open.
	Local horizon	Sees only the waypoints within a limited radius; the goal is always visible.
	Letter misattribution	Reads the letters correctly, but attaches them to the wrong positions.
	Trail amnesia	Under-reads the shaded trail, so it partly forgets where it has been.
Vision planners	BFS only	Follows the shortest route to the goal through the maze as perceived.
	Wall-blind BFS	Plans as if the interior walls were absent, so it heads for the waypoint closest to the goal in city-block distance.
	Greedy	Heads for the waypoint closest to the goal in a straight line.
	Goal bearing	Heads for the waypoint that points most directly toward the goal.
	Trail-penalised greedy	Heads toward the goal, but avoids retracing its own trail.
	Trémaux	Prefers unvisited branches, with a weak pull toward the goal.
	Heading momentum	Keeps going in the direction it was already moving.

Table 4: **Seed programs for the egocentric tasks.** Columns R and S mark the seeds used for Route Retracing and Shortcut Discovery. Actions are move forward, turn left, turn right, and stop.

Type	Seed	R	S	What it does
References	Uniform	✓	✓	Picks any of the four actions at random.
	Optimal (full map)	✓	✓	Builds a map from the video and its own views, then follows the shortest path to the goal.
Route retracing	Route retrace	✓	✓	Finds where it is on the demonstrated route, walks toward a point a little further along, and stops at the end.
	Sparse-keyframe retrace	✓		Retraces the route from a few remembered keyframes, so it loses detail at turns.
	Dead-reckoning retrace	✓		Retraces the route while its sense of heading drifts with every turn, until it sees the goal painting.
	Scans when stuck (retrace)	✓		Retraces the route, but turns in place to look around when it gets stuck.
Bounded planners	Limited field of view	✓	✓	Plans on the map, but sees only nearby cells in a narrow cone ahead, missing side corridors.
	Decaying-memory planner		✓	Plans on a map that quickly forgets what it has seen, so it re-explores known areas.
	Dead reckoning		✓	Plans on the map while its sense of heading drifts with every turn, until it sees the goal painting.
	Scans when stuck (plan)		✓	Follows its planned path, but turns in place to look around when it is stuck or lost.
Reactive	Momentum	✓	✓	Has no memory: walks forward while the way is clear, keeps turning the same way when blocked, and stops once it sees the goal.

D.4 EVOLUTIONARY SEARCH IMPLEMENTATION

Each variation operator in Section 3 is one call to the proposing LLM. Its prompt is assembled from parts shared by all operators (the task contract, the fit objective, the briefing and a residual report) followed by an instruction specific to the operator. Table 5 lists which parts each operator receives, and Figures 16–19 quote the prompts verbatim. The LLM returns its programs as a JSON object that gives, for each program, a one-sentence hypothesis naming the computation, the complete Python source of the program, and a rationale stating what in the evidence the program is meant to explain.

Task contract. Each task has one contract, which describes the task the model faced, the interface of the program, the observation attributes and geometry helpers it may use, the form of its output, and the rules it must obey (Figure 16). A program declares its hypothesis as a sentence and its parameters as `(initial, low, high)` ranges, which bound the coordinate descent; the LLM does not tune the parameter values itself. The contracts for route retracing and shortcut discovery additionally ask for a policy in three stages, perception, demonstration memory and action selection, and name candidate limits for each, such as a depth horizon, a field of view and a video stride. The fit objective states how programs are scored: held-out NLL and next-action accuracy, balanced over the two strata of the task (e.g., junctions and corridors in Maze Completion).

Residual report. The report is computed for the program the operator builds on: the parent for mutation and simplification, parent A for crossover, and the current best program for trace induction (Figure 17). It gives the program's held-out scores, the model's failure rate in each decision bin beside the program's misprediction rate in the same bin, and six counterexamples. The counterexamples are the distinct decisions to which the program assigns the lowest probability of the model's action. Each shows the observation and the model's recent actions, the action the model took, the probability the program gave it and the action the program preferred, and the model's reasoning at that decision, cut to its first and last 700 characters.

Briefing. The briefing reports measurements of the model's behavior computed from its recorded decisions: the number of episodes and decisions, and the model's error rate on each kind of decision. For CSWM, route retracing and shortcut discovery it also gives the model's task performance and the NLL of choosing uniformly at random, and summarizes the population: programs in the population with their held-out loss, and the hypotheses of programs that were dominated, so that the LLM does not propose them again. The briefing is recomputed at every generation.

Parent selection. Every generation uses $P = 3$ parents: the current best program, and two programs drawn without replacement from a pool of the 8 best programs, the 8 programs with the highest error-site lift, 8 members of the Pareto front, and all seed programs. The best program has the lowest held-out balanced NLL plus balanced error rate. The error-site lift is the program's mean predicted error, i.e., the probability it assigns to incorrect actions, on the decisions where the model erred, divided by its mean over all decisions; a lift of 1 means the program has no information about where the model fails. A pool member is drawn with probability proportional to $1/(1+f)+\ell/\ell_{\max}$, where f is its fitness and ℓ its lift.

Operator schedule. Mutation is applied to each parent. Crossover is applied to the two most complementary pairs among the 20 best members of the Pareto front, and to the best program with one drawn parent. Two members are complementary when each is better than the other on some objective. Pairs whose predictions differ on fewer than 5% of decisions are skipped, and the remaining pairs are ranked by the smaller of the two members' distances from the best objective values of the pair. Simplification is applied to the best program, and trace induction runs once.

Validation. Before fitting, each proposed program is checked statically. It must parse; define a single `Policy` class with a `step` method; declare a hypothesis of at least six words, at most 16 parameters with valid ranges, and whether it needs the full observation history; and use no imports, no randomness and only whitelisted builtins and the `math` module. Violations are returned to the LLM in the same dialog, and it may repair the program for up to five turns. Programs then run in sandboxed subprocesses with a 45-second limit on replaying the model's decisions, and a program that fails to run at its initial parameters is discarded.

Table 5: **Variation operators.** The programs each operator builds on, the parts of its prompt, and the number of programs it requests per call. Every prompt also carries the task contract.

Operator	Applied to	Residual report of	Objective and briefing	Programs per call
Mutation	each of the 3 parents	the parent	✓	3
Crossover	3 pairs of programs	parent A	✓	2
Simplification	the best program	the best program		1
Trace induction	no parent	the best program	✓	3

Figure 16: **Task contract and fit objective for Maze Completion.** Both are quoted verbatim; together they open every operator prompt for the task (simplification omits the objective). At most 16 parameters are allowed, and `slip` is the conventional name of the residual-noise term.

```
You are writing one candidate explanation of how a large language model navigates
a single-frame waypoint maze, as a Python program. The program is a scientific claim
about the model's policy, not an attempt to solve the maze optimally.

## The Task the Model is Doing
The model navigates a perfect maze from a starting cell to a goal 'G'.
Each turn:
1. The model is shown the full maze as a 2D ASCII grid with visit counts:
   - 'W': Wall
   - 'F': Free space, never visited
   - '1', '2', '3': Visited that many times (3 means 3 or more)
   - 'A': Current agent position
   - 'G': Goal position
2. Instead of single steps, the model chooses among reachable waypoints up to 5 steps away.
   - Waypoints are assigned random lowercase letters each turn (e.g. 'e', 'k', 'p').
   - Moving to a waypoint automatically walks the agent along the single path to that destination.
   - When standing on the goal, the model must issue "stop".

## What the Program Is
A single top-level `class Policy` with:

    HYPOTHESIS = "one sentence describing the behavioral mechanism you claim the model runs"
    PARAMS = {"name": (init, lo, hi), ...}   # fitted numerically for you
    NEEDS_HISTORY = False

    def __init__(self, params): ...
    def reset(self): ...                      # called at start of episode
    def step(self, obs, H) -> dict[str, float]: # non-negative scores over candidate waypoints
    def commit(self, action: str): ...        # the action the model actually took

`obs` provides:
* `obs.observation`: Full prompt observation text verbatim.
* `obs.offered_waypoints`: Dict mapping offered letters to coordinates, e.g. {'e': (3, 5), 'k': (7, 1)}.
* `obs.pos`: Agent's current (row, col) coordinates.
* `obs.goal`: Goal (row, col) coordinates.
* `obs.grid`: Static binary grid (1=free, 0=wall).
* `obs.turn`: Integer turn index.
* `obs.standing_on_goal`: True if agent is currently on the goal.

`H` is a fast geometry helper facade:
* `H.bfs_distances(source)` -> dict of (r, c) -> distance
* `H.geodesic(a, b)` -> shortest path length
* `H.manhattan(a, b)` -> Manhattan distance
* `H.degree(cell)` -> number of open neighbors
* `H.visit_count(cell)` -> visit count of cell

The output of `step(obs, H)` must be a dict mapping offered letters (and "stop" if on goal)
to non-negative scores. The harness normalizes them to a probability distribution.

## Hard Rules
* No imports. `math` is already in scope. No randomness of any kind.
* Builtins allowed: range len min max abs sum sorted enumerate zip map filter list
  dict set tuple frozenset float int bool str round all any reversed isinstance.
* Call your residual noise term `slip` by convention.
* At most 16 parameters.
* Keep it under 120 lines. Must run in milliseconds per decision.

Evaluate programs by held-out Negative Log-Likelihood (NLL) and Accuracy balanced
equally across two topological strata:
1. 'junction': Decisions within 1 step of an intersection (degree >= 3).
2. 'corridor': Decisions along linear corridors (degree <= 2).

A lower NLL and higher accuracy on junctions is the primary indicator of a successful
explanation of the model's spatial planning.
```

Figure 17: **Residual report.** The template of the report that an operator prompt carries for the program it builds on. Slots are filled from the program's predictions on the model's recorded decisions.

```
## How ‘⟨program id⟩‘ currently fits

| metric | value |
|---|---|
| held-out NLL per decision | ⟨value⟩ |
| ⟨first stratum⟩ NLL | ⟨value⟩ |
| next-action accuracy | ⟨value⟩ |
| ⟨first stratum⟩ accuracy | ⟨value⟩ |
| catastrophe rate (model's action given <1e-4) | ⟨value⟩ |
| error-site lift (1.0 = you have no idea where it fails) | ⟨value⟩ |
| error-site top-k precision vs chance | ⟨value⟩ vs ⟨value⟩ |

The reference model's failure rate, against the ⟨bin caption⟩ (this is the pattern a mechanism has to reproduce):

| bin | model failed | n | rate |
|---|---|---|---|
⟨one row per bin with at least 10 decisions⟩

Where your program disagrees with the model, binned the same way (a bin where your rate is far from the model's is
where the mechanism is wrong):

| bin | you mispredicted | n | rate |
|---|---|---|---|
⟨one row per bin with at least 10 decisions⟩

### Counterexample 1: ⟨episode⟩ turn ⟨turn⟩
⟨state of the decision, in task terms⟩
the model played ‘⟨action⟩‘; your program gave that ⟨probability⟩ and preferred ‘⟨action⟩‘
a correct move here is ⟨actions on a shortest path⟩

‘‘‘
⟨observation shown to the model, and its recent actions⟩
‘‘‘

What the model said while deciding:
> ⟨first 700 characters of the reasoning⟩
> [...]
> ⟨last 700 characters of the reasoning⟩

⟨counterexamples 2 to 6, in the same form⟩
```

Figure 18: **Mutation and crossover prompts**, verbatim. Italic slots are filled at run time; the crossover prompt carries the residual report of parent A.

Mutation $\mathcal{O}_{\text{mut}}(\pi)$

⟨*task contract*⟩
⟨*fit objective*⟩
⟨*briefing*⟩

```
## The program to improve

Its hypothesis: ⟨parent's hypothesis⟩

'''python
⟨parent's source code⟩
'''
```

⟨*residual report*⟩

```
## Your task

Return 3 *different* edits of this program. Each must change the mechanism, not just a
constant -- the constants are fitted for you. Aim at the counterexamples above: a good
edit is one that would have got those decisions right for a reason you can state, while
leaving the decisions it already gets right alone.

Say in the hypothesis what resource the computation is short of, and make sure the
program's failures follow from running out of it.
```

Crossover $\mathcal{O}_{\text{cross}}(\pi, \tilde{\pi})$

⟨*task contract*⟩
⟨*fit objective*⟩
⟨*briefing*⟩

```
## Parent A: ⟨hypothesis of parent A⟩

'''python
⟨source code of parent A⟩
'''

## Parent B: ⟨hypothesis of parent B⟩

'''python
⟨source code of parent B⟩
'''
```

⟨*residual report*⟩

```
## Your task

Return 2 programs that genuinely *compose* these two mechanisms rather than switching
between them on a fitted threshold. A composition is interesting when the two resources
interact -- one bound applies inside the other -- and boring when it is a weighted
average of two answers.

State in the hypothesis what the combined computation is short of, and what the
composition predicts that neither parent does alone.
```

Figure 19: **Simplification and trace-induction prompts**, verbatim. Trace induction carries the residual report of the current best program, whose counterexamples quote the model's reasoning, and one seed program as an example of style.

Simplification $\mathcal{O}_{\text{simp}}(\pi)$

⟨*task contract*⟩

```
## The program to simplify

It currently uses ⟨number⟩ AST nodes and ⟨number⟩ parameters.

'''python
⟨parent's source code⟩
'''
```

⟨*residual report*⟩

```
## Your task

Return 1 program that behaves as closely as possible to this one while being materially
shorter and easier to read. Remove branches that never fire, collapse duplicated logic,
and drop parameters the mechanism does not need. Do not change what it claims -- the
hypothesis should come out the same, said more plainly.
```

Trace induction $\mathcal{O}_{\text{trace}}(\tau)$

⟨*task contract*⟩
⟨*fit objective*⟩
⟨*briefing*⟩

⟨*residual report*⟩

```
## Your task

Ignore the existing programs. Read the quoted reasoning above as evidence about the
*procedure* the model is running -- what it enumerates, in what order, where it stops,
what it re-checks, what it never mentions -- and write 3 programs implementing the
procedures it suggests.

A trace that lists the recorded decisions in reading order is different evidence from
one that reasons through a single route; a trace that names the target first is
different again. Take the order and the stopping points seriously, because those are
the parts a resource bound would explain.

Here is a seed program for reference on style and structure only, not content:

'''python
⟨style seed program⟩
'''
```

Table 6: **Abstract situations for Maze Completion.** Abstract situations $s = \phi(x) = (f_1, \ldots, f_5) \in \mathcal{S}$ are formed by the Cartesian product of 5 discrete feature dimensions ($2 \times 3 \times 3 \times 2 \times 2 = 72$ bins).

Situation Feature Dimensions Discrete state space $\mathcal{S} = \prod_{i=1}^{5} \mathcal{F}_i$, $|\mathcal{S}| = 72$

Feature Dimension	Values	Operational Definition
Stratum	`junction` `corridor`	`junction`: at or next to a branch point (a cell with three or more open neighbours); `corridor`: any other cell.

Goal relation	`on_goal`
	`goal_in_horizon`
	`goal_unseen`
		`on_goal`: standing on the goal; `goal_in_horizon`: the goal is among the offered waypoints (within 5 steps along the maze); `goal_unseen`: otherwise.

Unvisited branches	$u \in \{0, 1, \geq 2\}$	Number of offered waypoints, including the goal, not visited before in the episode.
Anti-greedy	`True`, `False`	Whether the offered waypoint on the shortest path is farther from the goal in Euclidean distance than the current cell.
Current visits	$v = 1$, $v \geq 2$	Number of times the current cell has been a decision position in the episode, including now: first time versus again.

E Abstract Situations and Actions

We define situations as the Cartesian product of discrete feature dimensions chosen to isolate the core demands of each task:

- *Cambridge Spatial Working Memory*: Features separate search phases, track memory load via remaining viable boxes, and isolate spatial clustering from reading-order biases.
- *Maze Completion*: Features separate branch choices at junctions from corridors, sensory steering from occluded navigation, and expose deceptive local gradients.
- *Shortcut Discovery*: Features capture adherence to familiar routes versus breakaways onto shortcut chords, goal proximity, and heading alignment toward landmarks.
- *Route Retracing*: Features track adherence to the outbound path envelope, distance back to the starting origin, and egocentric alignment with return waypoints.

Tables 6–13 provide the operational definitions for all abstract situations $\phi(x) = s \in \mathcal{S}$ and abstract actions $\psi(a, x) = \alpha \in \mathcal{A}_{\text{abs}}$ across the interactive spatial tasks.

Table 7: **Abstract actions for Maze Completion.** Abstract actions $\psi(a, x) = \alpha \in \mathcal{A}_{\text{abs}}$ map waypoint navigation choices to functional navigational behaviors.

Abstract Actions $\psi(a, x) = \alpha \in \mathcal{A}_{\text{abs}}$

Type	Action	Operational Definition
Progress	`move_towards_goal`	Moving to an unvisited waypoint on the shortest path to the goal.
	`reach_goal`	Choosing the goal waypoint.
Exploration	`explore_detour`	Advancing to a previously unvisited waypoint that deviates from the shortest path to the goal.
	`backtrack_from_deadend`	Moving to a previously visited waypoint on the shortest path to the goal, e.g. returning from a dead end.
	`reexplore_detour`	Revisiting an already explored waypoint that lies off the shortest path to the goal.
Stops	`stop_at_goal`	Issuing `stop` while standing on the goal.
	`stop_premature`	Issuing `stop` anywhere else.
Invalid	`invalid_lost_turn`	Choosing a label that is not an offered waypoint; the turn is lost without moving.

Table 8: **Abstract situations for Cambridge Spatial Working Memory.** Abstract situations $s = \phi(x) = (f_1, \ldots, f_5) \in \mathcal{S}$ are formed by the Cartesian product of 5 discrete feature dimensions ($2 \times 5 \times 2 \times 2 \times 2 = 80$ bins).

Situation Feature Dimensions Discrete state space $\mathcal{S} = \prod_{i=1}^{5} \mathcal{F}_i$, $|\mathcal{S}| = 80$

Feature Dimension	Values	Operational Definition
Search phase	`fresh` `informed`	`fresh`: the first probe of a round, i.e. the game's first turn or the turn after a treasure is found; `informed`: later probes in the same round.
Unvisited load	$u \in \{1, 2, 3, 4, \geq 5\}$	Number of viable boxes, those not yet opened in the current round that have never held a treasure.
First round	`True`, `False`	`True` until the first treasure is found.
Adjacent viable box	`True`, `False`	Whether any viable box is adjacent to the last opened box, enabling cluster search. Always `False` on the game's first turn.
Reading-order top adjacency	`True`, `False`	Whether the first viable box in reading order (left to right, top to bottom) is adjacent to the last opened box. Always `False` on the game's first turn.

Table 9: **Abstract actions for Cambridge Spatial Working Memory.** Abstract actions $\psi(a, x) = \alpha \in \mathcal{A}_{\text{abs}}$ map raw box choices to functional search and memory behaviors.

Abstract Actions $\psi(a, x) = \alpha \in \mathcal{A}_{\text{abs}}$

Type	Action	Operational Definition
Memory errors	`reopen_past_treasure`	Opening a box that held a treasure in an earlier round; a between-round memory error.
	`reopen_in_current_search`	Re-opening a box already opened in the current round; a within-round memory error.
Spatial search	`adjacent`	Opening a viable box adjacent to the last opened box.
	`nearby`	Opening a viable box beyond that but within 3 grid cells.
	`far`	Opening a viable box beyond 3 grid cells.
Order biases	`raster_head_adjacent`	Opening the first viable box in reading order when it is adjacent to the last opened box.
	`raster_head_isolated`	Opening the first viable box in reading order when it is not adjacent, or on the game's first turn.
	`opening_other`	Opening any other box on the game's first turn.

Table 10: **Abstract situations for Route Retracing.** Abstract situations $s = \phi(x) = (f_1, \ldots, f_4) \in \mathcal{S}$ are formed by the Cartesian product of 4 discrete feature dimensions ($2 \times 4 \times 3 \times 2 = 48$ bins).

Situation Feature Dimensions Discrete state space $\mathcal{S} = \prod_{i=1}^{4} \mathcal{F}_i$, $|\mathcal{S}| = 48$

Feature Dimension	Values	Operational Definition
Location type	`on_route` `off_route`	`on_route`: on the demonstrated route or next to it; `off_route`: elsewhere.

Goal proximity	`at_goal`
	`near_goal`
	`mid_distance`
	`far_from_goal`
		Walking distance to the goal in 0.25 m steps: `at_goal` ≤ 4, `near_goal` 5–10, `mid_distance` 11–20, `far_from_goal` > 20 or unreachable.

Facing relation	`facing_route`
	`facing_goal`
	`facing_other`
		Heading within $\pm 30^\circ$ of the direction to the nearest route cell (`facing_route`), otherwise of the direction to the goal (`facing_goal`), otherwise `facing_other`.

Forward passage	`open` `blocked`	`blocked` if the cell one step ahead is not free floor; `open` otherwise.

Table 11: **Abstract actions for Route Retracing.** Abstract actions $\psi(a, x) = \alpha \in \mathcal{A}_{\text{abs}}$ map egocentric motor choices (forward steps, rotations, stops) to functional retracing behaviors.

Abstract Actions $\psi(a, x) = \alpha \in \mathcal{A}_{\text{abs}}$

Type	Action	Operational Definition
Steps on route	`advance_route`	Forward step along the route that reduces the distance to the goal.
	`backtrack_route`	Forward step along the route that increases the distance to the goal.
	`deviate_from_route`	Forward step from the route to off the route.
Steps off path	`reenter_route`	Forward step from off the route back onto it.
	`approach_goal_off_path`	Forward step that stays off the route and reduces the distance to the goal.
	`wander_off_path`	Forward step that stays off the route and does not reduce the distance to the goal.
Collision	`collide`	Attempting a forward step directly into an obstacle or boundary wall.
Turns	`align_route`	Turn that reduces the heading error to the nearest route cell.
	`align_goal`	Otherwise, a turn that reduces the heading error to the goal.
	`turn_disoriented`	Any other turn.
Stops	`stop_at_goal`	Issuing `stop` within 4 steps of the goal.
	`stop_premature`	Issuing `stop` anywhere else.

Table 12: **Abstract situations for Shortcut Discovery.** Abstract situations $s = \phi(x) = (f_1, \ldots, f_4) \in \mathcal{S}$ are formed by the Cartesian product of 4 discrete feature dimensions ($3\times4\times4\times2 = 96$ bins).

Situation Feature Dimensions Discrete state space $\mathcal{S} = \prod_{i=1}^{4} \mathcal{F}_i$, $|\mathcal{S}| = 96$

Feature Dimension	Values	Operational Definition

Location type	`on_shortcut`
	`on_route`
	`off_path`
		`on_shortcut`: on or next to a shortest path from start to goal; `on_route`: otherwise on the walkthrough route; `off_path`: elsewhere.

Goal proximity	`at_goal`
	`near_goal`
	`mid_distance`
	`far_from_goal`
		Walking distance to the goal in 0.25 m steps: `at_goal` ≤ 4, `near_goal` 5–10, `mid_distance` 11–20, `far_from_goal` > 20 or unreachable.

Facing relation	`facing_shortcut`
	`facing_goal`
	`facing_route`
	`facing_other`
		Heading within $\pm30^\circ$ of, in order: the next step on a shortest path to the goal (`facing_shortcut`), the goal (`facing_goal`), the nearest route cell (`facing_route`); otherwise `facing_other`.

Forward passage	`open` `blocked`	`blocked` if the cell one step ahead is not free floor; `open` otherwise.

Table 13: **Abstract actions for Shortcut Discovery.** Abstract actions $\psi(a, x) = \alpha \in \mathcal{A}_{\text{abs}}$ map egocentric motor choices (forward steps, rotations, stops) to functional shortcut navigation behaviors.

Abstract Actions $\psi(a, x) = \alpha \in \mathcal{A}_{\text{abs}}$

Type	Action	Operational Definition
Steps on shortcut	`advance_shortcut`	Forward step within the shortcut that reduces the distance to the goal.
	`backtrack_shortcut`	Forward step within the shortcut that increases the distance to the goal.
	`deviate_from_shortcut`	Forward step from the shortcut to outside it.
Steps on route	`advance_route`	Forward step along the route that reduces the distance to the goal.
	`backtrack_route`	Forward step along the route that increases the distance to the goal.
	`deviate_from_route`	Forward step off the route that does not reduce the distance to the goal.
	`breakaway_to_shortcut`	Forward step from the route into the shortcut.
	`breakaway_to_goal`	Forward step off the route that reduces the distance to the goal.
Steps off path	`reenter_shortcut`	Forward step from off both paths into the shortcut.
	`reenter_route`	Forward step from off both paths onto the route.
	`approach_goal_off_path`	Forward step that stays off both paths and reduces the distance to the goal.
	`wander_off_path`	Forward step that stays off both paths and does not reduce the distance to the goal.
Collision	`collide`	Attempting a forward step directly into an obstacle or boundary wall.
Turns	`align_shortcut`	Turn, while on the shortcut, that reduces the heading error to the next step on a shortest path to the goal.
	`align_goal`	Otherwise, a turn that reduces the heading error to the goal.
	`align_route`	Otherwise, a turn that reduces the heading error to the nearest route cell.
	`turn_disoriented`	Any other turn.
Stops	`stop_at_goal`	Issuing `stop` within 4 steps of the goal.
	`stop_premature`	Issuing `stop` anywhere else.