GENERATIVE COGNITIVE MAPS FOR STRUCTURE-SENSITIVE PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Existing models of human planning generally explain behavior through limited planning depth or resource-rational constraints on computation. Such models tend to be agnostic to environmental layout, including repeated or compositional structure, despite it being common in real-world environments. Here we examine human planning behavior in regularly structured spatial environments, finding strong sensitivity to structure not captured by current state-of-the-art models of human planning. We instantiate this principle computationally by proposing GenPlan, a computational framework that implements structure sensitive planning through compositional generative map representations and reusable policies. The framework integrates (1) a Generative Map Module that infers latent structural regularities, and (2) a Structure-Based Planner that exploits them for hierarchical planning. We show that GenPlan captures sensitivity to structure empirically observed in human behavior, while reducing planning cost relative to structure-agnostic baselines with only a bounded performance penalty. Together, these results establish structural sensitivity as an important principle of human planning, and demonstrate how it can be incorporated into resource-efficient planning algorithms.

1 INTRODUCTION

People expertly solve real-world planning problems, such as navigating a city without precise knowledge of its street network or building complex objects with multiple subgoals [39] (Fig. 1). Despite the formal intractability of optimal planning in all but trivial settings [27], in natural environments human planning is often highly efficient [4]. What cognitive computations make this efficiency possible?

Existing models of human planning focus on computational approximations, such as limited search depth [33, 58]. However, such models tend to be agnostic to environmental structure. In contrast, we propose and test the hypothesis that human planning is sensitive to structural regularities in the environment, and demonstrate computationally how exploiting such structure can support resource-efficient planning. Our approach is motivated by the theory of *compositionality* in cognitive science, which states that humans learn complex concepts by combining simpler ones [14, 34, 42]. Compositionality has been successful in explaining mental representation in visual [34, 56], auditory [20, 44, 59] and spatial domains [38, 47]. Its cultural universality [42] suggests an evolved adaptation to structured environments [26]. Neural evidence for compositional structures in cognitive maps includes mirror-invariant encoding of natural scenes [11] and reuse of neural reference frames across similar environments [37]. Behavioral evidence includes hierarchical spatial representations [2, 3, 18, 30, 41, 52, 57, 60, 63], and the ability to predict regular layouts in novel environments [47].

Previous work has modeled compositional representations via *program induction* — the process of inferring from examples a symbolic program that can generalize to novel instances of a concept class [34, 35, 56]. Such frameworks explain how humans can learn an underlying generative rule from just a few examples, in contrast to the large datasets required by purely neural models [35, 55]. Building on this line of research, we introduce GenPlan, an algorithmic framework that implements structure-sensitive planning by representing cognitive maps as generative programs and using them for cost-efficient planning with policy reuse. In doing so, our aim is not to outperform domain-general planning methods across benchmarks, or to establish the exact cognitive algorithm by which people

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

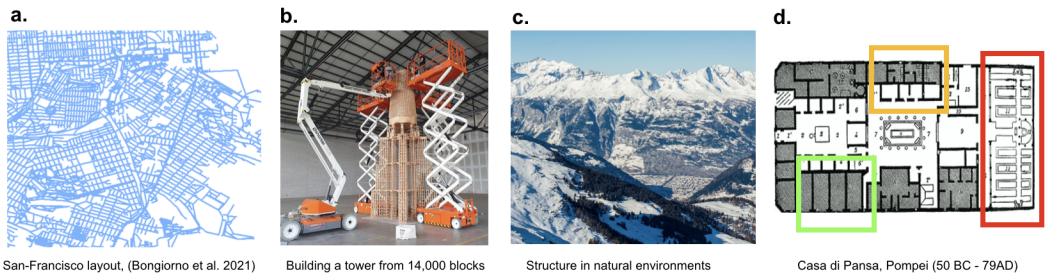


Figure 1: Structure in human environments: city street networks, construction projects, natural landscapes, and building interiors.

plan. Rather, our goal is to examine whether environmental structure shapes human planning and demonstrate how such structure-sensitive planning can be instantiated algorithmically in a practical computational framework. We contribute:

- an empirical result showing that human planning in structured environments is strongly sensitive to structure, in a way that cannot be captured by current alternative models or local-search baselines;
- a generative-program representation of environmental structure, motivated by compositional accounts of human cognition;
- a planning framework (GenPlan) that shows how structure-sensitive planning can be implemented in practice to save computational cost with only a bounded penalty on performance.

2 Methods

2.1 Structured Spatial Domain

We examine people's planning in a controlled experimental paradigm designed to isolate the role of environmental structure in planning. To do this, we adapt the Maze Search Task (MST) [16, 31, 33] previously used to study human planning under partial observability. This domain allows us to parametrically vary the amount of compositional structure, to distinguish predictions of our framework from structure-agnostic baselines.

MST requires finding hidden exits in a series of two-dimensional grid-worlds. The mazes are partially observable; each has only one exit, hidden in a random unobserved location (black cells) (see Fig. 2). Full experiment instructions are given in Appendix H.People can move to any unoccupied grid cell adjacent to the agent (a round avatar) by clicking. The black hidden cells are revealed when they come into the avatar's line of sight. When revealed, the exit becomes visible as a red tile. As soon as the character moves over the exit, the trial ends. In our adaptation of MST, mazes contain varying degrees of repeated structure, with between 2 and 20 recurring units comprising a portion of the layout.

2.2 Computational Models

Planning under partial observability can be modeled by a partially observable Markov decision process (POMDP). A common formulation represents this as a fully observable search through a space of beliefs, where each belief is a probability distribution over possible states [27] - an approach used to model MST in prior work [33]. Formally, a POMDP is a tuple $\langle \Delta(S), A, \tau, r, b_0, \gamma \rangle$, where $\Delta(S)$ is the space of probability distributions over a state space S, A is the set of actions, τ is the belief update function, r is the reward function, b_0 is the initial belief, and γ is the discount factor. The belief state evolves deterministically via τ, reflecting both the agent's actions and observations.

Following prior work [32], each state $s \in S$ is represented as an $N \times M$ grid whose cells are labeled $\{\text{wall}, \text{empty}, \text{exit}, \text{agent}\}$. The overall state space S consists of all such grids containing exactly one

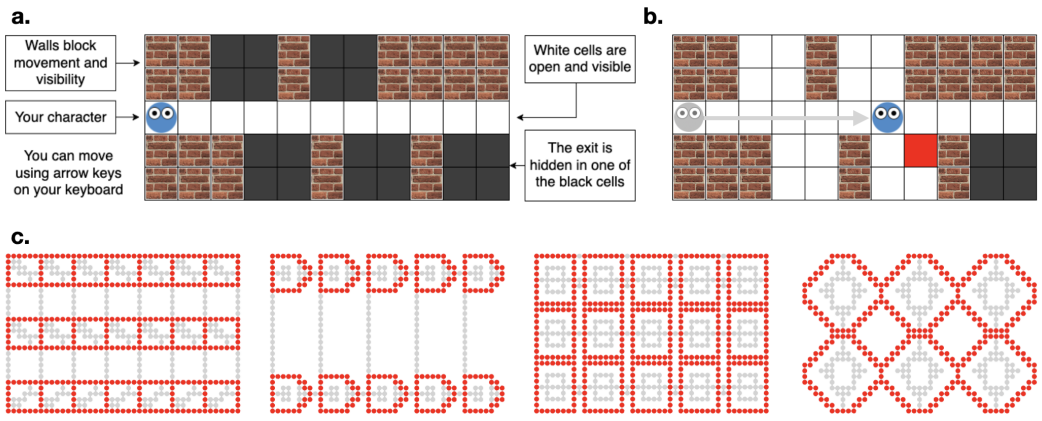


Figure 2: (a) The Maze Search Task. (b) The exit is shown as a red tile when it comes into view. (c) Representative structured environments used to compare GenPlan to a Naive POMCP and DepthPlan. Dots denote floor cells; red dots show structural unit boundaries.

agent and one exit. A belief $b \in \Delta(S)$ is thus a probability distribution over these grids, encoding the agent's uncertainty about the true state. Initially, b_0 assumes that the agent and the walls are known, while the exit is uniformly distributed over all valid, unseen cells. The action space A contains four possible movements (up, down, left, right). Observations $o \in O$ reveal the visible subset of the grid around the agent, with each visible cell labeled $\{\text{wall}, \text{empty}, \text{exit}\}$, and any cell outside the agent's visibility range r labeled as *unseen*. Observations are consistent with the grid structure of the true state $s \in S$.

The belief update function τ is given by

$$b'(s') \;\propto\; Z(o \mid s') \sum_{s \in S} T(s', a, s)\, b(s),$$

where $T(s', a, s)$ is the transition function, and $Z(o \mid s')$ is the observation likelihood. The transition function $T(s', a, s)$ specifies the probability of transitioning to state s' from s after executing action a. Here, actions that would move the agent into a wall result in the agent remaining in its current position, and transitions to an exit state terminate the process. The observation function $Z(o \mid s')$ encodes the likelihood of observing o given s', where observations reflect the visible subset of the grid within range r of the agent's position. Visibility is blocked by walls, such that cells beyond a wall are labeled as *unseen*. Finally, the reward function $r(b, a)$ is the expected reward under the belief b. Since the agent can always see an exit before reaching it, $r(b, a) = 1$ if action a leads the agent to a known exit and 0 otherwise.

Expected Utility Given this, the optimal policy can be found by searching over a tree where each node represents a belief $b \in \Delta(S)$, and edges correspond to action-observation pairs (a, o). Starting from the root node b_0, the tree expands by simulating actions $a \in A$ and updating beliefs using the belief update function τ. For each action a, the agent considers all possible observations $o \in O$, with the likelihood of each observation determined by the observation function $Z(o \mid s')$. At each node, the value of a belief is computed recursively using the Bellman equation:

$$V(b) = \max_{a \in A} \left[r(b, a) + \gamma \sum_{o \in O} P(o \mid b, a) V(\tau(b, a, o)) \right], \tag{1}$$

where $P(o \mid b, a)$ is the probability of observation o after action a under belief b. The optimal policy π^* is derived by selecting the action at each belief node that maximizes the expected value.

Although this is the optimal strategy, human behavior often diverges from its predictions to an extent that depends on cognitive resources allocated to planning [31]. Previous work most readily explains this divergence by a limited planning horizon (a discount factor $\gamma < 1$ in Equation 1) [23, 33]. Here, we test an alternative hypothesis: that human planning is sensitive to environmental structure.

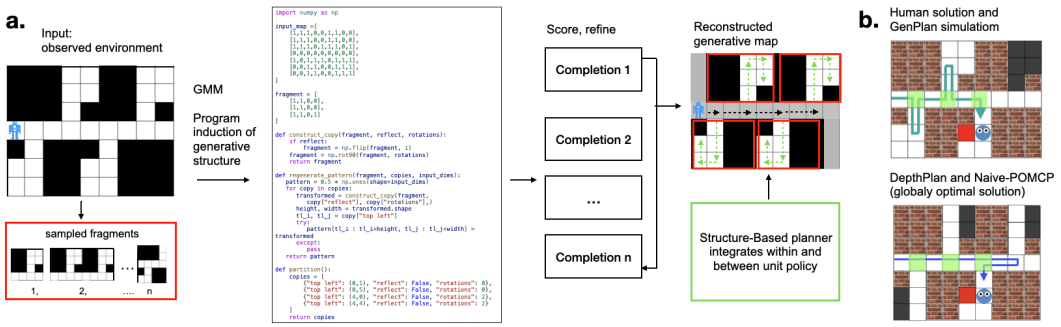


Figure 3: GenPlan architecture and predictions. (a) Based on partial observations of the environment, GMM recovers a small distribution over generative maps, each consisting of a set of candidate units and a program for reconstructing the input from them. Given a GMM's map representation as input, SBP plans reusable options once per structural unit, which are then integrated into a general plan. (b) A human solution agrees with the prediction of GenPlan (top panel), and cannot be captured by DepthPlan and Naive-POMCP (bottom panel), which optimize the global policy. Green highlights show discriminating decisions.

Generative Structure-Based Framework (GenPlan) We formalize structure-sensitive planning in two steps: a Generative Map Module (GMM) builds a compressed programmatic representation of the observed map, and a Structure-Based Planner (SBP) (Fig.3) plans a piecewise policy over repeated structural units in GMM's representation. Our framework combines automatic structure discovery with structure-aware planning and can, in principle, extend to other domains; here, we demonstrate it in the context of MST.

Generative Map Module (GMM) Let I be a partial observation of a ground-truth map that specifies wall locations, but does not reveal the reward location – matching human observations in MST. The GMM implements approximate inference of a posterior distribution $p(\mathcal{M}|I)$ over compressed map representations $\mathcal{M}$ that partition I into structural units $\mathcal{M} = \{U_i\}_{i=1}^{m}$. This is implemented by prompting an LLM to identify repeating units in the input map, and synthesize a Python program that approximately reconstructs the input from them. The prompt (Appendix F) includes Python code describing admissible transformations, as well as a likelihood function for a given $\mathcal{M}$. In our implementation, the input map I is a grid-world, specified by a numerical array, where each grid cell is associated with a number (e.g. wall=1, floor=0). The reconstructions $\mathcal{M}$ do not allow overlapping units, and allow any units that repeat in I at least twice. Our implementation used GPT-4.1, with generalization to other LLM architectures demonstrated in Appendix E.

We score each candidate $\mathcal{M}$ using a trade-off between reconstruction accuracy and model complexity:

$$l(\mathcal{M}; I) \propto -\frac{1}{d_x d_y} \sum_{x,y} \left(I(x,y) - O(x,y)\right)^2 - \lambda \,\mathrm{MDL}(\mathcal{M}), \tag{2}$$

where O is the reconstruction induced by $\mathcal{M}$, $\mathrm{MDL}(\mathcal{M})$ measures the description length of the program that generates the map, and d_x, d_y are input dimensions. The Minimum Description Length (MDL) term [43] penalizes both (i) the number and size of structural units and (ii) the transformations required to place them (translations, rotations, reflections). This favors reconstructions that explain the map using a small number of reusable components. This formulation is motivated by a well-documented MDL bias in human mental representations [6, 7, 45], favoring concise, reusable structure. We treat this score as proportional to a posterior $p(\mathcal{M} \mid I)$ over map hypotheses, where reconstruction error defines a data fit term and MDL acts as a simplicity prior. The resulting map representations are then used by the planning module to construct a policy. In practice, planning can be performed with either a single high-scoring map or a distribution over candidate maps; in our setting, both approaches yield indistinguishable predictions.

In general, both the input I and reconstruction O can be real-valued arrays. In our implementation, I is binary (wall vs. free space), and $O(x, y) \in [0, 1]$ represents the reconstructed occupancy. To estimate model complexity, rather than use the raw length of the synthesized Python program, we compute a compressed encoding of the inferred structural units and the transformations used to reconstruct the map as a more stable measure of description length. Figures 3 and 2 show examples of high-scoring decompositions used in our experiments.

Structure-Based Planner (SBP) Our approach integrates planning within and between structural units. We first solve planning *within a unit* by searching through the belief space using an approximate online Partially Observable Monte Carlo Planner (POMCP) [48]. A plan *between units* consists of leaving the current unit, searching through unstructured space, and transitioning between the unstructured space and the next unit. These transitions can be repeated several times. The unstructured space may be searched partially each time it is entered, but the structural units are searched fully each time they are entered via policy reuse.

We solve leaving the unit by adapting Monte Carlo Tree Search (MCTS), and introducing an optimistic heuristic valuation for open cells around its boundary (i.e. cells through which we can exit the unit). Upon completing a plan for the current unit, this heuristic introduces a bias to leave the unit in the direction that minimizes the expected global cost of reaching the exit. In the general case, this can be any heuristic that does not overestimate the true cost. Here, we compute the values of boundary cells as inversely proportional to the average of Manhattan distances to the remaining external unobserved cells, hence assigning a higher value to cells that are on average closer to the remaining unseen parts of the map.

We then solve the unstructured search by solving a POMCP on the global map, and implement the option to switch to within-unit planning upon reaching the new unit. Here, we evaluate the option by estimating the average per-step cost to plan within the unit. Therefore, any map regions not captured by any repeating unit are handled by falling back to Naive-POMCP with structured units as options. Once the exit becomes revealed, the SBP terminates this search procedure and transitions to solving a fully observable Markov Decision Process (MDP) using value iteration, which routes the agent to the exit by a shortest direct path.

The pseudo-code for the GMM and SBP algorithms is given in Appendix G. In Appendix B, we analyze the modular planning constraint underlying SBP and show that, under the assumptions of our task, the optimal modular policy incurs at most a constant-factor increase in expected step cost relative to the globally optimal policy.

2.3 Computational Hypotheses

Our behavioral experiment aims to test the hypothesis that human planning is sensitive to repeated environmental structure. To do this, we compare human behavior to an instantiation of structure-sensitive planning along with structure-agnostic alternatives. We evaluate the following computational hypotheses (planning algorithms):

1. **Structure-Naive Planner (Naive-POMCP)**: The model doesn't use generative maps, and plans by optimizing a global policy for the environment.
2. **Structure-Naive Planner With Cognitive Constraints (DepthPlan)**: The model doesn't use generative maps, and plans by optimizing a global policy that discounts future states to model limited planning depth. This model was previously used to describe how people plan in MST [33].
3. **A Generative Planner (Gen-POMCP)**: The model induces a distribution $p(\mathcal{M} \mid I)$ over generative maps and uses the inferred structure to plan a reusable policy (see Fig. 3a and b).
4. **LLM-Direct Planner (LLM-Plan)**: The model receives the map and task description and plans directly using an LLM, without inducing an explicit generative structure or POMCP planning machinery. We use this as a baseline to characterize what structured planning requires beyond general-purpose language model reasoning.

We evaluated LLM-Plan across models ranging from GPT-4.1 to GPT-5.2 at varying reasoning budgets. Smaller or less capable models failed to produce valid paths, either attempting illegal moves

into walls (GPT-4.1) or terminating search prematurely without finding the exit (GPT-5.1). The strongest condition tested — GPT-5.2 at high reasoning effort on small maps — produced valid paths but substituted a full-coverage objective for the correct expected-cost objective, visiting every cell rather than planning against a uniform prior over exit locations. No tested configuration solved the intended planning problem. We therefore do not compare LLM-Plan to human behavior as a cognitive model, but report it as evidence that belief-state planning under a uniform exit prior is not reducible to general-purpose language model reasoning, even at high reasoning budgets (see Appendix I).

Naive-POMCP uses an approximate POMCP planner designed for large partially-observable state spaces. It was previously used to model human spatial planning, although this work did not examine the effect of environment structure on human plans [47]. As GenPlan employs a POMCP planning backend identical to the planner used in Naive-POMCP, Naive-POMCP is a lesioned variant of GenPlan without structural inference. In unstructured environments, GenPlan reduces to Naive-POMCP, and produces identical behavior.

DepthPlan is consistent with prior work that models human deviations from optimal cost minimization by a limited planning horizon, without modeling environment structure [23, 33]. Since DepthPlan internally computes an optimal policy, it can only be applied to smaller environments (generally 6 or fewer structural units, see Experiment 1). Only the third hypothesis is consistent with the GenPlan framework: it represents maps as generative programs, and uses the induced structure to plan.

3 Experiments

We first test whether people's plans exhibit the structure-sensitivity predicted by Gen-POMCP, in contrast to DepthPlan – the state-of-the-art model of planning in MST [33] (Experiment 1). We used a set of 20 structured environments at a scale solvable by DepthPlan, to enable comparison. Based on the sample sizes used in prior work [47] and a preliminary pilot confirming this sample, we recruited 30 participants from Prolific. We also ran a simulation experiment with 10 larger environments that cannot be solved by DepthPlan (Experiment 2) to compare the computational costs of Gen-POMCP (ours) and Naive-POMCP (prior work [47]), demonstrating that Gen-POMCP requires significantly fewer computational resources.

3.1 Experiment 1: Behavioral Validation

Procedure The experiment was conducted in a web browser, using the web-interface of MST [33]. Before beginning the experiment, participants gave informed consent and completed a series of practice trials, followed by an instruction quiz. Following this, they completed a variant of MST with structured mazes, with exit locations randomly chosen. In one maze the exit was randomly placed in plain sight, and it was excluded from the analysis. Participants were told that the exit could be in any of the hidden tiles, and instructed to find it in each environment. As our goal was to observe ecologically valid planning, we did not offer performance-based incentives. Upon completing the experiment, we administered a post-experiment questionnaire collecting demographic information.

We recruited 30 (13 female, 17 male, $M_{\text{age}} = 36.7$, $SD_{\text{age}} = 13.5$) English-speaking participants on Prolific, who were paid £9 per hour. None were excluded. On average the experiment took 10 minutes to complete. The experiment was approved by our institution's IRB.

Behavioral metrics We introduce the following *behavioral definitions* to quantify people's alignment with the GenPlan framework.

- A set of *discriminating decisions* $\mathcal{D}(I)$ in a given environment I is the subset of all states in I where Gen-POMCP and DepthPlan predict a different most likely action. Fig. 3b illustrates discriminating decisions in a simple example. Unlike the global solution (bottom panel), Gen-POMCP and most humans search by first searching within each unit locally. The cells highlighted in green are discriminating decisions.
- *Modular fraction* $\sigma(\mathcal{D})$ defines the fraction of decisions in a given set of discriminating decisions $\mathcal{D}$ that are more likely under Gen-POMCP, compared to DepthPlan. Notably, by trading off accuracy against representation and planning costs, Gen-POMCP can capture flexible strategies that integrate global and local search. For simplicity, we evaluate Gen-POMCP

under a strong bias toward structural decomposition rather than global path efficiency. The reported $\sigma(\mathcal{D})$ therefore characterizes human alignment with this structure-biased variant of the model.

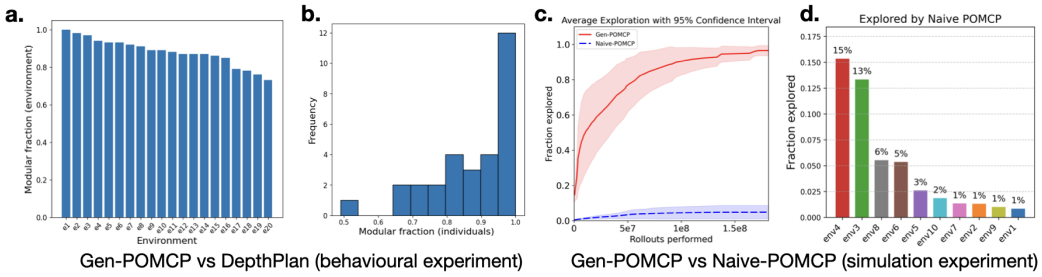


Figure 4: Gen-POMCP predicts human choices better than DepthPlan. (a.) Modular fraction for each environment (b.) The histogram of individual modular fractions per participant, over all discriminating decisions. (c) The fraction of each environment explored by Gen-POMCP across environments (solid line) and Naive-POMCP (dashed lines) as a function of compute budget (number of MCTS rollouts). (d) The fraction of each environment explored by Naive-POMCP, if given the amount of MCTS rollouts at which Gen-POMCP fully explores the given environment. Each bar shows a different environment.

Results We find that Gen-POMCP predicts human behavior significantly better than DepthPlan (Fig. 4) and Naive-POMCP. Across participants, more than 80% of each individual's discriminating decisions were better predicted by GenPlan than by either alternative model (Figure 4(b.)), with 95% CIs excluding chance (0.5) for all participants (Appendix Figure 12). Across all environments, human behavior was likewise better explained by Gen-POMCP over DepthPlan (Figure 4(a.)). Participants rarely shifted toward the global strategy: the mean global-strategy fraction (defined as $1 - \sigma(\mathcal{D})$) was 0.124 (median 0.083), although individual values ranged from 0 to 0.51 (Appendix Fig. 13).

We additionally compare models to human behavior across all decisions, not just the discriminating decisions. Agreement increased with the use of the inferred compositional structure, from 42.5% for Naive-POMCP and 42.2% for DepthPlan, to 52.3% when constraining GenPlan decomposition to just two units, and to 80.7% for the full GenPlan decomposition.

To test whether this advantage could instead be explained by a generic local-search strategy, we additionally compare GenPlan against Steps, a task-specific heuristic previously evaluated alongside DepthPlan [33], which greedily moves toward the nearest unobserved location. Across discriminating decisions between GenPlan and Steps, human choices agreed with GenPlan in 75.7% of cases, with Steps in 19.6%, and with neither model in 4.7%. Fig. 3b illustrates the difference between Gen-POMCP vs. Naive-POMCP and DepthPlan in a simple example. Like Gen-POMCP, the majority of people search the environment by a trajectory shown in the top panel. However, both the optimal policy predicted by the Naive-POMCP, and the depth-limited planning as implemented by DepthPlan, predict the trajectory shown at the bottom – because it allows the agent to quickly reveal a large portion of the global map.

3.2 Experiment 2: Simulation

Next, we compare the computing resources needed by Gen-POMCP and Naive-POMCP to explore 10 large structured environments (e.g., see Fig. 2c). As the objective of MST is to find a hidden exit, each simulation is set up to run until the environment is fully explored (i.e., the exit is not revealed until the end of the simulation). We make two quantitative comparisons. For each environment we compute the fraction of the environment that each model is able to explore given a certain compute budget (Fig. 4c), finding that Gen-POMCP requires a much smaller budget to search the entire environment.

Fig.4d separately shows the fraction of each environment that Naive-POMCP is able to explore, if given the rollout budget at which Gen-POMCP has searched the entire environment. To test robustness

to environments in which compositional structure is only partially repeated, we progressively perturb copies of repeated units while retaining some exact instances. GenPlan's computational advantage decreases smoothly as the fraction of the environment covered by reusable structure decreases, converging toward Naive-POMCP when little repeated structure remains (Appendix D Fig. 10). All environments used in Experiments 1 and 2 are shown in Appendix C.

4 Related Work

Models of Human Planning Human planning has been primarily modeled by approximate planning with a limited planning depth [13, 23, 28, 33, 58]. Recent work further shows that human plan representations are sensitive to both effort minimization (shorter paths) and MDL (shorter policies) [8], which can favor policy reuse similar to our work. Unlike our work, their study does not examine the effects of regular problem-domain structure on human planning.

Cognitive Maps In fully observed domains human cognitive maps omit goal-irrelevant detail [19], which can be viewed as a form of cognitive map compression. Unlike our work, full observability makes goal-relevant structure well defined; our approach operates on partially observed maps, where goal-relevance is not directly established. A recent study found that people can predict unseen parts of structured environments [47], but focuses solely on map prediction rather than planning, and relies on exhaustive enumeration over map representations. In contrast, GMM uses an LLM as a prior over structural decompositions [10, 54] to implement tractable inference from a small number of samples.

Hierarchical Reinforcement Learning (HRL) Reinforcement learning methods reduce planning complexity through hierarchical abstractions such as options [53], shared structure across related MDPs [64], and predictive state merging [50]. Approaches include abstraction spaces [46], HTNs [12, 40], and probabilistic propositional planning [36]. Recent work analyzes state abstraction in tree search [1, 21, 22] and structural conditions for efficient planning [61]. Related lines study option discovery [24, 25], generalized planning across task families [9], and symmetry-based representations [49]. These approaches engineer abstractions to improve worst-case or average efficiency. In contrast, GenPlan conditions planning on inferred structure: the GMM recovers repeated fragments from partial observations, and SBP reuses fragment-level policies with a focus on testing structure-sensitive predictions against human planning behavior.

Using LLMs to plan Several related works have used LLMs for offline planning. Similar to our work, Parsel [67] uses LLMs to decompose complex tasks into modular components to solve a larger problem. Unlike our work, it solves problems specified in natural language, rather than inferring latent environmental structure. Kim et al. [29] propose a framework that fine-tunes LLM-based agents to plan in grid environments by constructing an internal representation. Unlike GenPlan, which represents maps as code, their model works with text-based grid-world descriptions and uses unstructured maps. Chain-of-Thought Procedure Cloning enables agents to generalize to new fully observable environment configurations by imitating the intermediate reasoning steps of expert procedures [65]. The ReAct framework [66] guides an LLM through iterative thought-action steps and can be applied to spatial navigation domains. However, ReAct does not provide inference over environmental structure, or mechanisms for policy reuse [17]. Unlike GenPlan, these works are not evaluated in human experiments, and do not target planning in structured spatial environments. Further, they use LLMs as planning agents, whereas GenPlan uses LLMs as a prior over structural representations, separating structure inference from the planning computation.

Successor representations A complementary line of work examines how relational structure is learned from sequential experience. Successor representations encode states through their expected future occupancy and have been linked to spatial inference and planning [5, 51]. CSCGs infer latent graph structure from ambiguous observation sequences [15], while the Tolman–Eichenbaum Machine learns relational representations that generalize across environments [62]. These approaches primarily address how structure is acquired from experience; GenPlan instead examines how compositional structure inferred from spatial observations can be exploited for planning through reusable policies.

5 DISCUSSION

While human planning cognition has been studied extensively in the laboratory, planning in naturalistic domains remains underexplored. Here we establish structure sensitivity as a computational principle of human planning. We show that people's plans are affected by repeated environmental structure in a way not captured by previous structure-agnostic models and local search heuristics. We give a computational instantiation of this principle in GenPlan, a framework that represents maps as generative programs and plans within them using POMCP with policy reuse. GenPlan achieves tractable inference through LLM-driven program generation in a Turing-complete language (rather than exhaustive enumeration) and reduces planning compute through policy reuse, with only a bounded performance penalty.

In the present behavioral task, participants are provided with the maze geometry, and the experiment tests how known environmental structure affects planning. GenPlan itself can operate on partial map observations and exploit reusable structure as soon as sufficient repeated structure has been observed to support a structural hypothesis. Our results both advance fundamental research in cognitive science and demonstrate how to build AI systems that plan by exploiting inductive biases motivated by human planning.

Limitations and future work. In the current GenPlan implementation, structural units are identified using shape-based matching, whereas units that differ in layout are treated as distinct even if they could admit the same policy. We use this as a controlled modeling choice, and do not claim from the present experiment that human structural representations are based on exact geometric equivalence. While the goal of the present experiment is to establish structure sensitivity in human planning, future work could further examine the representations and algorithms through which humans implement it, and its generalization beyond grid domains. Additionally, future work could manipulate incentives and time pressure to assess how flexibly people could trade off structure-sensitive and global planning. Incentivizing shorter paths would test whether participants are able to trade cognitive effort against solution quality. Time pressure could dis-incentivize cognitive effort, and increase reliance on reusable structure, as a motor-cognitive effort trade-off.

Future work can also extend this framework to probabilistic or evolving units. For instance, instantiating maps as probabilistic generative programs using a probabilistic CFG rather than fixed arrays would allow testing how people plan in units that differ in appearance (e.g. square vs. rectangular city blocks) but share equivalent generative constraints. Importantly, our modeling results raise the hypothesis that the inferred generative map structure may support planning policy reuse. Under this hypothesis, people should ignore environmental structures that do not support reusable policies, even when those structures admit compact representations. This hypothesis remains to be tested in future work.

A related direction of future work could examine whether individual differences in structure-based planning reflect variable compression-accuracy tradeoffs in map representations. Depending on available cognitive resources, people may form coarser or finer structural decompositions. Modelling whether people switch between representations in response to changing cognitive demands would provide an empirical test of whether GenPlan's particular representational assumptions can capture variability in human planning in naturalistic environments.

Finally, future work could examine whether goal structure shapes which environmental regularities are encoded. For instance, people tend to perceive San Francisco as having a grid layout despite the underlying complexity of its actual street network. One possible explanation for this perception is that a local grid model suffices for planning pedestrian shortcuts, in line with prior work on goal-dependent cognitive maps [19]. Future work could test this hypothesis empirically, and instantiate it computationally by integrating goal-conditioned representations into GenPlan's MDL criterion. Because our aim is to explain behavior, a comprehensive evaluation of computational complexity and performance across planning benchmarks is outside the scope of this work. We hope the principle of structure-sensitive planning established here, together with GenPlan as one computational instantiation, will motivate further theoretical work on when structural priors are advantageous or suboptimal.

AI USE STATEMENT

LLMs were used as part of the methodology described in this paper, specifically as proposal mechanisms for structural decompositions and reconstruction programs in the Generative Map Module; these uses are described in detail in the Methods and Appendix.

Generative AI tools were also used for routine coding assistance with generating plots, proofreading the completed manuscript, and literature discovery. All AI-assisted content and suggested references were reviewed and verified by the authors, who take responsibility for the final content of the paper.

REFERENCES

[1] Ankit Anand, Ritesh Noothigattu, Parag Singla, et al. Oga-uct: On-the-go abstractions in uct. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 26, pp. 29–37, 2016.

[2] Jeremy N Bailenson, Michael S Shum, and David H Uttal. The initial segment strategy: A heuristic for route selection. *Memory & Cognition*, 28(2):306–318, 2000.

[3] Jan Balaguer, Hugo Spiers, Demis Hassabis, and Christopher Summerfield. Neural mechanisms of hierarchical planning in a virtual subway network. *Neuron*, 90(4):893–903, 2016.

[4] Christian Bongiorno, Yulun Zhou, Marta Kryven, David Theurel, Alessandro Rizzo, Paolo Santi, Joshua Tenenbaum, and Carlo Ratti. Vector-based pedestrian navigation in cities, 2021.

[5] Wilka Carvalho, Momchil S Tomov, William de Cothi, Caswell Barry, and Samuel J Gershman. Predictive representations: Building blocks of intelligence. *Neural Computation*, 36(11): 2225–2298, 2024.

[6] Nick Chater. The search for simplicity: A fundamental cognitive principle? *The Quarterly Journal of Experimental Psychology Section A*, 52(2):273–302, 1999.

[7] Nick Chater and Paul Vitányi. Simplicity: a unifying principle in cognitive science? *Trends in cognitive sciences*, 7(1):19–22, 2003.

[8] Carlos G Correa, Sophia Sanborn, Mark K Ho, Frederick Callaway, Nathaniel D Daw, and Thomas L Griffiths. Exploring the hierarchical structure of human plans via program generation. *Cognition*, 255:105990, 2025.

[9] Aidan Curtis, Tom Silver, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Kaelbling. Discovering state and action abstractions for generalized task and motion planning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pp. 5377–5384, 2022.

[10] Aidan Curtis, Hao Tang, Thiago Veloso, Kevin Ellis, Joshua Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Llm-guided probabilistic program induction for pomdp model estimation. *arXiv preprint arXiv:2505.02216*, 2025.

[11] Daniel D Dilks, Joshua B Julian, Jonas Kubilius, Elizabeth S Spelke, and Nancy Kanwisher. Mirror-image sensitivity and invariance in object and scene processing pathways. *Journal of Neuroscience*, 31(31):11305–11312, 2011.

[12] Kutluhan Erol. *Hierarchical task network planning: formalization, analysis, and implementation*. University of Maryland, College Park, 1995.

[13] Diogo R Ferreira. The impact of the search depth on chess playing strength. *ICGA journal*, 36 (2):67–80, 2013.

[14] Jerry A Fodor. *The language of thought*, volume 5. Harvard university press, 1975.

[15] D George, RV Rikhye, N Gothoskar, JS Guntupalli, A Dedieu, and M Lázaro-Gredilla. Clone-structured graph representations enable flexible learning and vicarious evaluation of cognitive maps. nat. commun. 12, 2392, 2021.

[16] Maya Geva-Sagiv, Soyeon Jun, Marta Kryven, Josh Tenenbaum, Erie D. Boorman, Randy O'Reilly, Jack J. Lin, Ignacio Saez, and Charan Ranganath. Fronto-hippocampal synchronization in rapid spatial learning in humans. *Current Biology*, 2025. Under review.

[17] Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, et al. Is your llm secretly a world model of the internet? model-based planning for web agents. *arXiv preprint arXiv:2411.06559*, 2024.

[18] Stephen C Hirtle and John Jonides. Evidence of hierarchies in cognitive maps. *Memory & cognition*, 13(3):208–217, 1985.

[19] Mark K Ho, David Abel, Carlos G Correa, Michael L Littman, Jonathan D Cohen, and Thomas L Griffiths. People construct simplified mental representations to plan. *Nature*, 606(7912):129–136, 2022.

[20] Matthias Hofer, Tuan Anh Le, Roger Levy, and Josh Tenenbaum. Learning evolved combinatorial symbols with a neuro-symbolic generative model. *arXiv preprint arXiv:2104.08274*, 2021.

[21] Jesse Hostetler, Alan Fern, and Tom Dietterich. State aggregation in monte carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[22] Marcus Hutter. Extreme state aggregation beyond Markov decision processes. *Theoretical Computer Science*, 650:73–91, October 2016. ISSN 0304-3975. doi: 10.1016/j.tcs.2016.07.032. URL https://www.sciencedirect.com/science/article/pii/S0304397516303772.

[23] Quentin JM Huys, Níall Lally, Paul Faulkner, Neir Eshel, Erich Seifritz, Samuel J Gershman, Peter Dayan, and Jonathan P Roiser. Interplay of approximate planning strategies. *Proceedings of the National Academy of Sciences*, 112(10):3098–3103, 2015.

[24] Alexander Ivanov, Akhil Bagaria, and George Konidaris. Discovering options that minimize average planning time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 17573–17581, 2025.

[25] Yuu Jinnai, David Abel, David Hershkowitz, Michael Littman, and George Konidaris. Finding options that minimize planning time. In *International Conference on Machine Learning*, pp. 3120–3129. PMLR, 2019.

[26] Iain G Johnston, Kamaludin Dingle, Sam F Greenbury, Chico Q Camargo, Jonathan PK Doye, Sebastian E Ahnert, and Ard A Louis. Symmetry and simplicity spontaneously emerge from the algorithmic nature of evolution. *Proceedings of the National Academy of Sciences*, 119(11): e2113883119, 2022.

[27] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[28] Mehdi Keramati, Peter Smittenaar, Raymond J Dolan, and Peter Dayan. Adaptive integration of habits into depth-limited planning defines a habitual-goal–directed spectrum. *Proceedings of the National Academy of Sciences*, 113(45):12868–12873, 2016.

[29] Doyoung Kim, Jongwon Lee, Jinho Park, and Minjoon Seo. How language models extrapolate outside the training data: A case study in textualized gridworld. 2024.

[30] Stephen M Kosslyn, Herbert L Pick Jr, and Griffin R Fariello. Cognitive maps in children and men. *Child development*, pp. 707–716, 1974.

[31] Marta Kryven, Tomer D Ullman, William Cowan, and Joshua B Tenenbaum. Plans or outcomes: How do we attribute intelligence to others? *Cognitive Science*, 45(9):13–41, 2021.

[32] Marta Kryven, Suhyoun Yu, Max Kleiman-Weiner, and Josh Tenenbaum. Planning ahead in spatial search. *The 5th Multidisciplinary Conference on Reinforcement Learning and Decision Making*, 2022.

[33] Marta Kryven, Suhyoun Yu, Max Kleiman-Weiner, Tomer Ullman, and Joshua Tenenbaum. Approximate planning in spatial search. *PLOS Computational Biology*, 20(11):e1012582, 2024.

[34] Brenden M Lake and Steven T Piantadosi. People infer recursive visual concepts from just a few examples. *Computational Brain & Behavior*, 3(1):54–65, 2020.

[35] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[36] Michael L Littman. Probabilistic propositional planning: Representations and complexity. *AAAI/IAAI*, pp. 748–754, 1997.

[37] Steven A Marchette, Lindsay K Vass, Jack Ryan, and Russell A Epstein. Anchoring the neural compass: coding of local spatial reference frames in human medial parietal lobe. *Nature neuroscience*, 17(11):1598, 2014.

[38] William P McCarthy, Robert D Hawkins, Haoliang Wang, Cameron Holdaway, and Judith E Fan. Learning to communicate about shared procedural abstractions. *arXiv preprint arXiv:2107.00077*, 2021.

[39] Ugurcan Mugan, Seiichiro Amemiya, Paul S Regier, and A David Redish. Navigation through the complex world: The neurophysiology of decision-making processes. In *Habits: Their Definition, Neurobiology, and Role in Addiction*, pp. 109–139. Springer, 2024.

[40] Dana Nau, Yue Cao, Amnon Lotem, and Hector Munoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*, pp. 968–973, 1999.

[41] Nora Newcombe, Janellen Huttenlocher, Elisabeth Sandberg, Eunhui Lie, and Sarah Johnson. What do misestimations and asymmetries in spatial judgement indicate about spatial representation? *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 25(4):986, 1999.

[42] Benjamin Pitt, Stephen Ferrigno, Jessica F Cantlon, Daniel Casasanto, Edward Gibson, and Steven T Piantadosi. Spatial concepts of number, size, and time in an indigenous culture. *Science Advances*, 7(33):eabg4141, 2021.

[43] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978. ISSN 0005-1098. doi: https://doi.org/10.1016/0005-1098(78)90005-5. URL `https://www.sciencedirect.com/science/article/pii/0005109878900055`.

[44] Martin Rohrmeier. Towards a formalization of musical rhythm. In *ISMIR*, pp. 621–629, 2020.

[45] Joshua S Rule, Steven T Piantadosi, Andrew Cropper, Kevin Ellis, Maxwell Nye, and Joshua B Tenenbaum. Symbolic metaprogram search improves learning efficiency and explains rule learning in humans. *Nature Communications*, 15(1):6847, 2024.

[46] Earl D Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, 5(2): 115–135, 1974.

[47] Sugandha Sharma, Aidan Curtis, Marta Kryven, Josh Tenenbaum, and Ila Fiete. Map induction: Compositional spatial submap learning for efficient exploration in novel environments. *International Conference of Learning Representations*, 2022.

[48] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in Neural Information Processing Systems, 2010*. Neural Information Processing Systems, 2010.

[49] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[50] Satinder Singh, Michael James, and Matthew Rudary. Predictive state representations: A new theory for modeling dynamical systems. *arXiv preprint arXiv:1207.4167*, 2012.

[51] Kimberly L Stachenfeld, Matthew M Botvinick, and Samuel J Gershman. The hippocampus as a predictive map. *Nature neuroscience*, 20(11):1643–1653, 2017.

[52] Albert Stevens and Patty Coupe. Distortions in judged spatial relations. *Cognitive psychology*, 10(4):422–437, 1978.

[53] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

[54] Hao Tang, Darren Key, and Kevin Ellis. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. *arXiv preprint arXiv:2402.12275*, 2024.

[55] Joshua B Tenenbaum, Charles Kemp, Thomas L Griffiths, and Noah D Goodman. How to grow a mind: Statistics, structure, and abstraction. *science*, 331(6022):1279–1285, 2011.

[56] Lucas Tian, Kevin Ellis, Marta Kryven, and Josh Tenenbaum. Learning abstract structure for drawing by efficient motor program induction. *Advances in Neural Information Processing Systems*, 33, 2020.

[57] Momchil S Tomov, Samyukta Yagati, Agni Kumar, Wanqian Yang, and Samuel J Gershman. Discovery of hierarchical representations for efficient planning. *PLoS computational biology*, 16(4):e1007594, 2020.

[58] Bas van Opheusden, Ionatan Kuperwajs, Gianni Galbiati, Zahy Bnaya, Yunqi Li, and Wei Ji Ma. Expertise increases planning depth in human gameplay. *Nature*, pp. 1000–1005, 2023.

[59] Tessa Verhoef, Simon Kirby, and Bart De Boer. Emergence of combinatorial structure and economy through iterated learning with continuous acoustic signals. *Journal of Phonetics*, 43: 57–68, 2014.

[60] Ranxiao Frances Wang and James R Brockmole. Human navigation in nested environments. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 29(3):398, 2003.

[61] Zheng Wen, Doina Precup, Morteza Ibrahimi, Andre Barreto, Benjamin Van Roy, and Satinder Singh. On efficiency in hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 33:6708–6718, 2020.

[62] James CR Whittington, Timothy H Muller, Shirley Mark, Guifen Chen, Caswell Barry, Neil Burgess, and Timothy EJ Behrens. The tolman-eichenbaum machine: unifying space and relational memory through generalization in the hippocampal formation. *Cell*, 183(5):1249–1263, 2020.

[63] Jan M Wiener and Hanspeter A Mallot. 'fine-to-coarse'route planning and navigation in regionalized environments. *Spatial cognition and computation*, 3(4):331–358, 2003.

[64] Aaron Wilson, Alan Fern, and Prasad Tadepalli. Transfer learning in sequential decision problems: A hierarchical bayesian approach. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pp. 217–227. JMLR Workshop and Conference Proceedings, 2012.

[65] Mengjiao Sherry Yang, Dale Schuurmans, Pieter Abbeel, and Ofir Nachum. Chain of thought imitation with procedure cloning. *Advances in Neural Information Processing Systems*, 35: 36366–36381, 2022.

[66] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

[67] Eric Zelikman, Qian Huang, Gabriel Poesia, Noah Goodman, and Nick Haber. Parsel: Algorithmic reasoning with language models by composing decompositions. *Advances in Neural Information Processing Systems*, 36:31466–31523, 2023.

A CODE AVAILABILITY

The Gen-POMCP implementation is available here: `https://anonymous.4open.science/r/GenPlan-FBCD/README.md`

B PERFORMANCE BOUNDS FOR STRUCTURE-BASED PLANNING

In structured environments Gen-POMCP can explore the environment faster than Naive-POMCP (using fewer rollouts and in less time) by taking advantage of limited resources. As Gen-POMCP fully explores each fragment it enters, it can incur longer paths overall. How much does global planning by Naive-POMCP improve on the path length taken by Gen-POMCP (and specifically the Structure-Based Planner)?

We sketch a proof that considers the limit in which each planner fully optimizes its respective objective: Gen-POMCP follows the Bayes-optimal plan in each fragment and between fragments, and Naive-POMCP follows the Bayes-optimal global policy for the maze. We bound the cost difference according to the worst-case cost in steps.

Expected and worst-case The expected number of steps it takes for a policy to explore a maze is the average path length taken by this policy to reach uniformly sampled exit locations.

Lemma 1. *There exists an $n \times n$ fragment that requires $\Theta(n^2)$ steps to explore fully and, under a uniform distribution over possible exit locations, $\Theta(n^2)$ steps to search in expectation.*

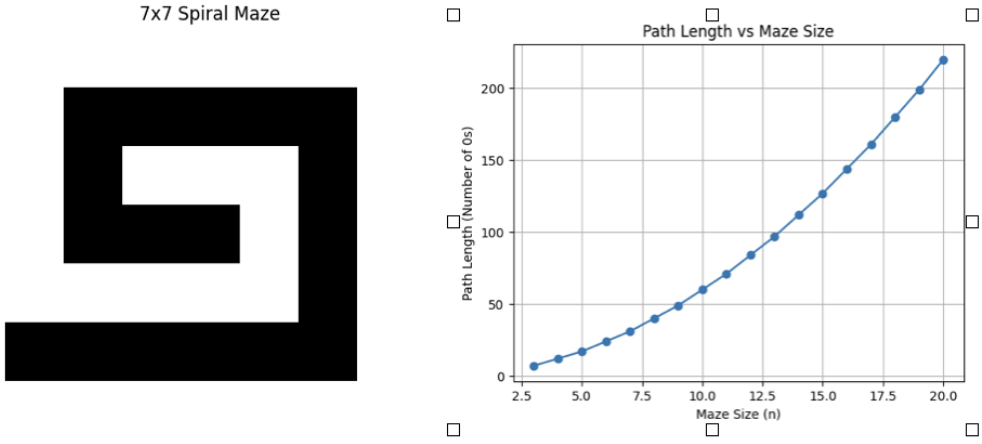


Figure 5: A spiral-maze construction. White cells are traversable and black cells are impassable. The traversable corridor has length $\frac{1}{2}n^2 + O(n)$, and therefore scales as $\Theta(n^2)$.

Proof. Consider the spiral-corridor construction in Figure 5, with the agent starting at one endpoint of the corridor. Let

$$m = \left\lfloor \frac{n}{2} \right\rfloor .$$

The corridor consists of successive horizontal and vertical segments whose lengths decrease by two at every other segment. Its total length is

$$L_n = n + \sum_{i=0}^{m-1} 2(n - 2i - 1) \tag{3}$$

$$= n + 2m(n-1) - 2m(m-1) \tag{4}$$

$$= n + 2m(n-m) \tag{5}$$

$$= \frac{1}{2}n^2 + O(n). \tag{6}$$

Hence,

$$L_n = \Theta(n^2).$$

Because the traversable cells form a single corridor, fully exploring the fragment requires traversing to its far endpoint and therefore takes $L_n - 1 = \Theta(n^2)$ steps.

Now suppose that the exit is distributed uniformly over the traversable cells of the corridor, other than the starting cell. If the cells are indexed by their distance $1, 2, \ldots, L_n - 1$ from the starting position, then the expected number of steps required to reach the exit is

$$\frac{1}{L_n - 1} \sum_{j=1}^{L_n - 1} j = \frac{L_n}{2} = \Theta(n^2).$$

Thus, the fragment requires $\Theta(n^2)$ steps both to explore fully and to search in expectation. □

Theorem 2. *In an $n \times n$ maze, the expected number of steps taken by SBP may exceed the expected number of steps of an optimal policy by $\Omega(n^2)$.*

Proof. Construct two fragments, A and B. Each fragment consists of an open room and a spiral corridor: $A = R_A \cup S_A$ and $B = R_B \cup S_B$. Within each fragment, the room is connected to the spiral by a single doorway. The two fragments are connected by a doorway between R_A and R_B. Choose the sizes of the four regions so that they contain equal numbers of candidate exit locations. Since the exit is uniformly distributed over candidate locations,

$$P(R_A) = P(R_B) = P(S_A) = P(S_B) = \frac{1}{4}.$$

The spiral corridors are chosen according to the construction of Lemma 1, so fully exploring either spiral requires $\Theta(n^2)$ steps. In contrast, the open rooms can be checked in $O(n)$ steps.

Assume the agent starts in the unstructured fragment located at the doorway between fragments A and B, and enters R_A first. Under the SBP algorithm, before SBP may leave A for B, it must fully explore A. In particular, if the exit is in R_B, SBP must still first fully explore S_A and return to the doorway connecting the two fragments. Therefore, conditional on the event that the exit lies in R_B,

$$C_{\text{SBP}} - C_{\text{global}} = \Theta(n^2),$$

because a global planner can instead check R_A and then R_B before exploring either spiral.

In contrast, consider a global policy that first checks R_A, then checks R_B, and, if the exit is in neither room, returns to search S_A and then S_B. On exit locations outside R_B, checking R_B before beginning the spiral search introduces at most an $O(n)$ additional detour relative to SBP.

Hence,

$$\mathbb{E}\left[C_{\text{SBP}} - C_{\text{global}}\right] \geq P(R_B)\Theta(n^2) - O(n).$$

Since $P(R_B) = 1/4$,

$$\mathbb{E}\left[C_{\text{SBP}} - C_{\text{global}}\right] \geq \frac{1}{4}\Theta(n^2) - O(n) = \Omega(n^2).$$

The globally Bayes-optimal policy performs at least as well as this particular global policy. Therefore, the expected cost of SBP may exceed that of the globally optimal policy by $\Omega(n^2)$.

□

Theorem 3. *The number of steps required to fully explore an $n \times n$ maze is $O(n^2)$.*

Proof. Represent the traversable cells of the maze as a connected graph with $v \leq n^2$ vertices. A depth-first traversal of a spanning tree of this graph traverses every tree edge at most twice. Since a spanning tree has $v - 1$ edges, the maze can be fully explored in at most

$$2(v-1) \leq 2n^2 - 2 = O(n^2)$$

steps. □

Theorem 4 (Pointwise constant-factor guarantee)**.** *Let M be a finite connected grid maze with v traversable cells, partitioned into connected fragments $U_1, \ldots, U_m$. Every traversable cell other than the starting cell is a valid potential exit location, and the exit is drawn uniformly from the candidate locations that are unseen after the initial observation. Assume that at least one candidate exit location remains unseen initially.*

Assume there exists a constant q, independent of maze size, such that the initial observation reveals at most q candidate exit locations and each subsequent action reveals at most q previously unseen candidate exit locations.

Let $C_{\mathrm{opt}}(M)$ denote the expected cost of the globally Bayes-optimal policy, and let $C_{\mathrm{mod}}(M)$ denote the expected cost of the optimal policy subject to the modular constraint that each previously unexplored fragment is fully explored upon first entry and is not fully explored again.

Then, for every maze M,

$$C_{\mathrm{mod}}(M) \leq (6q+3) C_{\mathrm{opt}}(M).$$

Consequently, the modular-optimal and globally optimal policies have the same $\Theta(n^2)$ worst-case expected-cost scaling over the fragmented $n \times n$ maze class.

Proof. We first lower-bound the expected cost of any policy. Let N be the number of candidate exit locations that remain unseen after the initial observation.

Consider first a deterministic policy π. Run π along the hypothetical observation history in which every candidate location revealed so far is empty. Until the true exit is revealed, this is exactly the observation history experienced by the policy. This hypothetical execution therefore defines an ordering of the N candidate locations according to the time at which they are first revealed.

Let

$$\tau_1 \leq \tau_2 \leq \cdots \leq \tau_N$$

denote these first-revelation times. Because each action reveals at most q previously unseen candidate locations,

$$\tau_i \geq \left\lceil \frac{i}{q} \right\rceil \geq \frac{i}{q}.$$

The exit is uniformly distributed over the N candidate locations. Therefore, its expected revelation time satisfies

$$\begin{aligned} \mathbb{E}[T_{\mathrm{reveal}}] &= \frac{1}{N} \sum_{i=1}^{N} \tau_i \\ &\geq \frac{1}{N} \sum_{i=1}^{N} \frac{i}{q} \\ &= \frac{N+1}{2q}. \end{aligned} \tag{7}$$

The agent cannot reach the exit before revealing it, so

$$C_\pi(M) \geq \frac{N+1}{2q}.$$

For a randomized policy, the same result follows by conditioning on the policy's internal random seed and averaging over seeds. In particular,

$$C_{\mathrm{opt}}(M) \geq \frac{N+1}{2q}. \tag{8}$$

We next construct an admissible modular comparison policy. Let π^* be a globally Bayes-optimal policy for M. The comparison policy follows π^*, except that whenever it first enters a previously unexplored fragment U_i, it suspends π^*, performs a depth-first traversal of a spanning tree of U_i, and returns to the cell through which it entered. If the exit is encountered during this traversal, the policy terminates. Otherwise, it marks U_i as fully explored and resumes π^*.

The comparison policy can resume π^* using a virtual observation history that ignores the additional negative observations acquired during the detour. Since the detour returns the agent to the same physical location, this is a valid continuation of the simulated global policy. Ignoring additional information is permitted for the purpose of constructing an upper bound.

A spanning tree of U_i has $|U_i| - 1$ edges. A depth-first traversal rooted at the entry cell traverses each tree edge at most twice. Thus, the full exploration of U_i and return to the entry cell require at most

$$2\big(|U_i| - 1\big)$$

steps. This construction does not require the fragment to have only one gateway: during the comparison-policy detour, the policy simply does not cross the other gateways.

Each fragment is fully explored at most once. Therefore, the total additional cost of all fragment-exploration detours is at most

$$\begin{aligned}\sum_{i=1}^{m} 2\big(|U_i| - 1\big) &\leq 2\sum_{i=1}^{m} |U_i| \\ &\leq 2v.\end{aligned} \tag{9}$$

The modular-optimal policy is at least as good as this particular admissible modular policy. Hence,

$$C_{\mathrm{mod}}(M) \leq C_{\mathrm{opt}}(M) + 2v. \tag{10}$$

Because every traversable cell other than the starting cell is a valid potential exit location, and at most q such locations are visible initially,

$$v \leq N + q + 1.$$

Substituting this relation into Equation (10) gives

$$C_{\mathrm{mod}}(M) \leq C_{\mathrm{opt}}(M) + 2N + 2(q+1). \tag{11}$$

From Equation (8),

$$N + 1 \leq 2qC_{\mathrm{opt}}(M),$$

and therefore

$$2N \leq 4qC_{\mathrm{opt}}(M).$$

Using this in Equation (11),

$$C_{\mathrm{mod}}(M) \leq (1 + 4q)C_{\mathrm{opt}}(M) + 2(q+1).$$

For every nontrivial maze,

$$C_{\mathrm{opt}}(M) \geq 1,$$

so

$$2(q+1) \leq 2(q+1)C_{\mathrm{opt}}(M).$$

It follows that

$$\begin{aligned}C_{\mathrm{mod}}(M) &\leq \big(1 + 4q + 2(q+1)\big)\, C_{\mathrm{opt}}(M) \\ &= (6q+3)C_{\mathrm{opt}}(M).\end{aligned} \tag{12}$$

This establishes the pointwise constant-factor guarantee.

Finally, Theorem 3 gives an $O(n^2)$ upper bound for the globally optimal policy, while the spiral construction in Lemma 1 gives a matching $\Omega(n^2)$ worst-case lower bound within the fragmented-maze class. Thus,

$$\max_M C_{\mathrm{opt}}(M) = \Theta(n^2).$$

Because, for every maze,

$$C_{\mathrm{opt}}(M) \le C_{\mathrm{mod}}(M) \le (6q+3)C_{\mathrm{opt}}(M),$$

the modular-optimal policy also has worst-case expected cost

$$\max_M C_{\mathrm{mod}}(M) = \Theta(n^2).$$

□

Scope of the guarantee. The theorem applies to the optimal policy subject to the modular planning constraint. It does not by itself bound approximation error caused by finite POMCP rollouts, imperfect value estimates, or an incorrectly inferred decomposition. Accordingly, the theoretical guarantee concerns the modular-optimal policy rather than the finite-compute Gen-POMCP implementation.

C EXPERIMENTAL ENVIRONMENTS

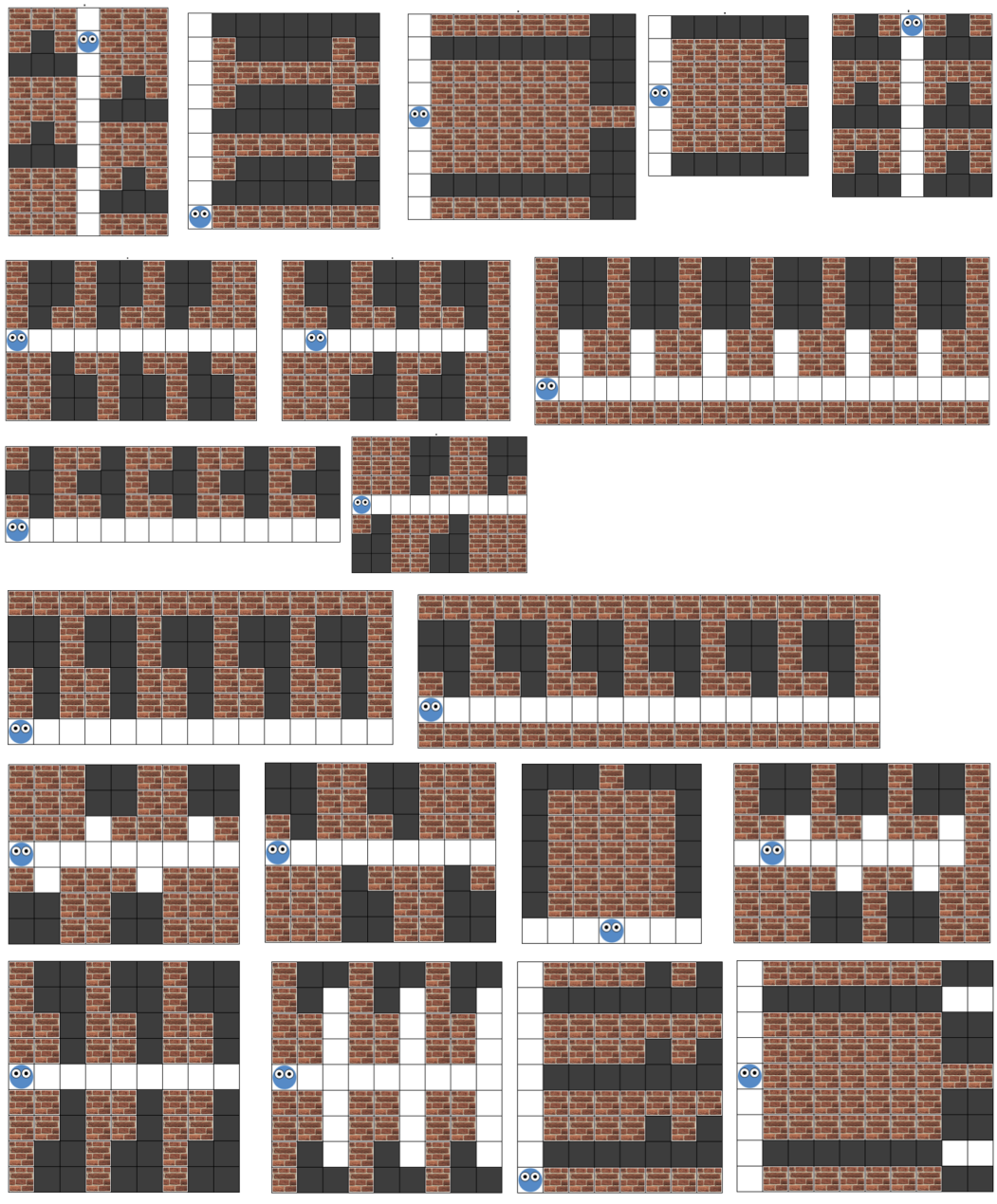


Figure 6: Environments used in Behavioral Experiment 1.

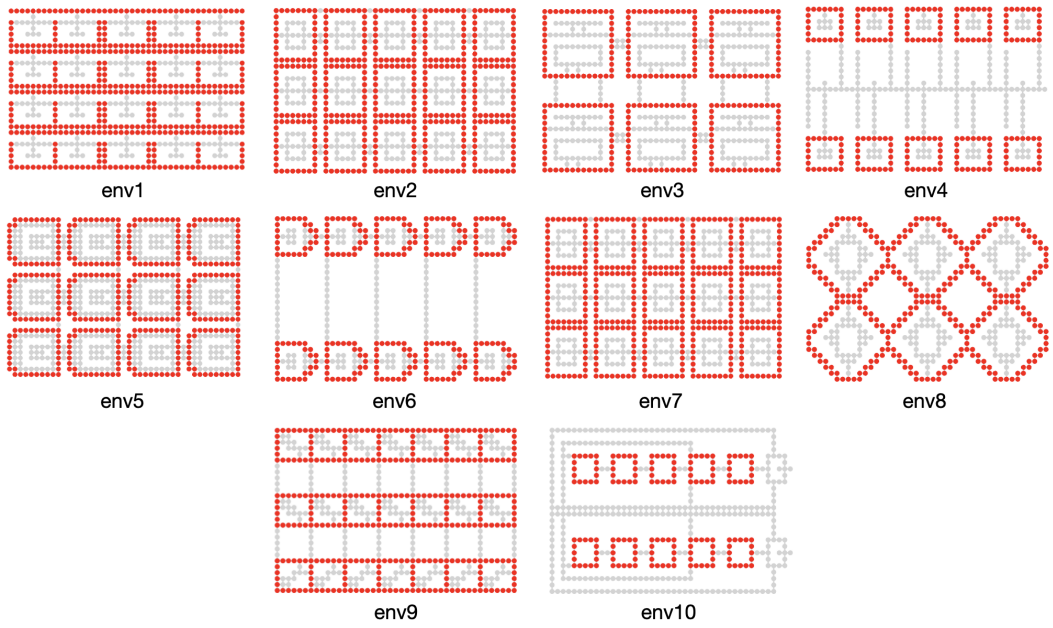


Figure 7: Environments used in Simulation Experiment 2.

D ADDITIONAL RESULTS - SIMULATION EXPERIMENT

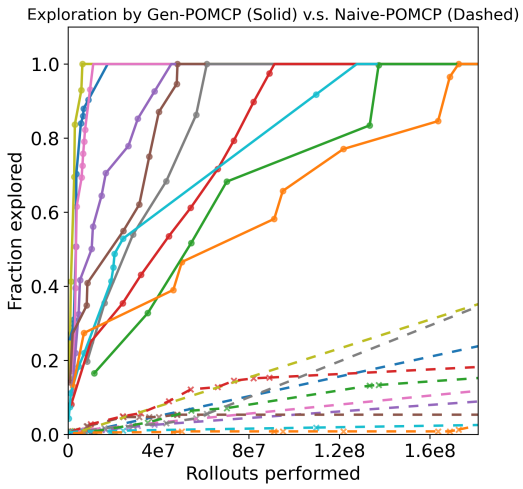


Figure 8: The fractions of each environment searched by Gen-POMCP and Naive-POMCP given an identical computational budget. Gen-POMCP requires fewer rollouts and reduces computational costs. Each environment is shown in a different color (see also Figure 4.)

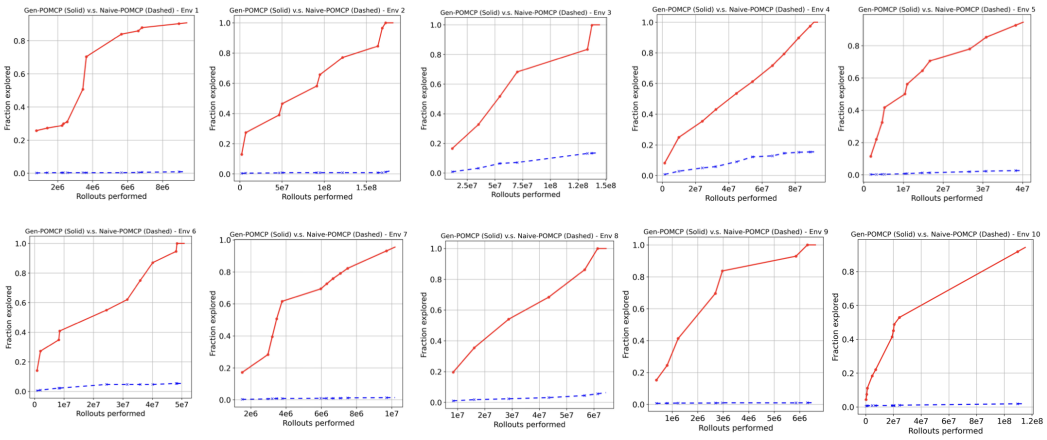


Figure 9: The fractions of each environment searched by Gen-POMCP and Naive-POMCP given identical computational budget. In each individual environment Gen-POMCP requires fewer rollouts and saves computing costs.

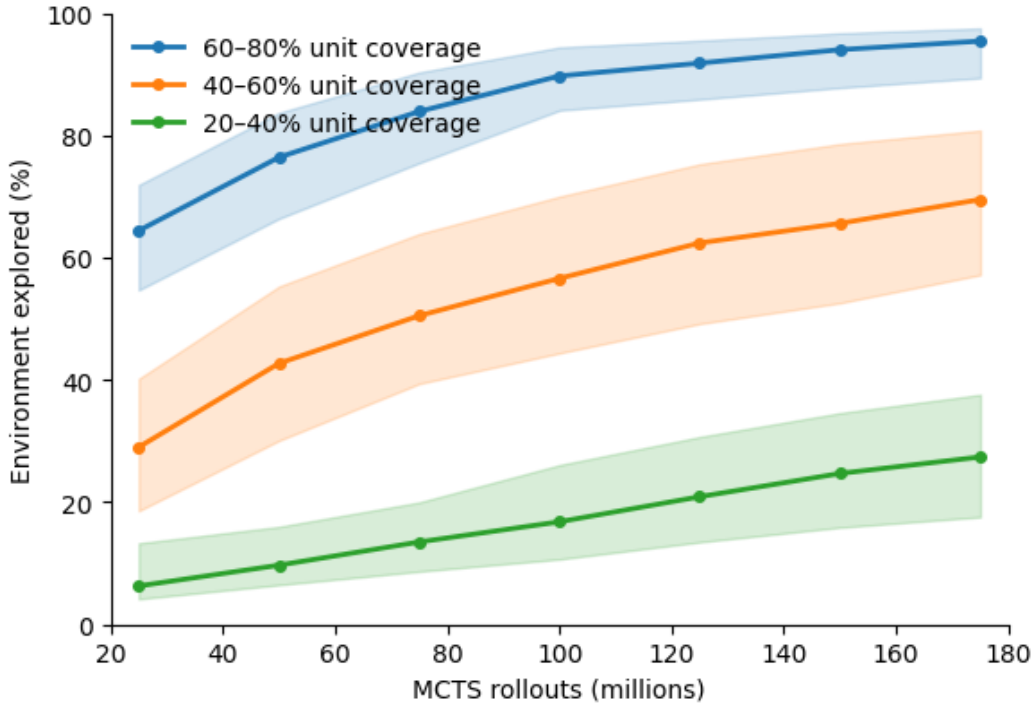


Figure 10: **Robustness of Gen-POMCP to partially perturbed compositional structure.** We progressively perturb repeated structural units while retaining a subset of exact repetitions, and group environments according to the fraction of the map covered by intact repeating units. Lines show the mean fraction of the environment explored by Gen-POMCP as a function of MCTS rollout budget for environments with 20–40%, 40–60%, and 60–80% intact unit coverage. Shaded regions indicate 95% confidence intervals. Gen-POMCP explores environments more efficiently when a larger fraction of the map can be represented using reusable structural units, with its advantage diminishing as reusable structure is removed.

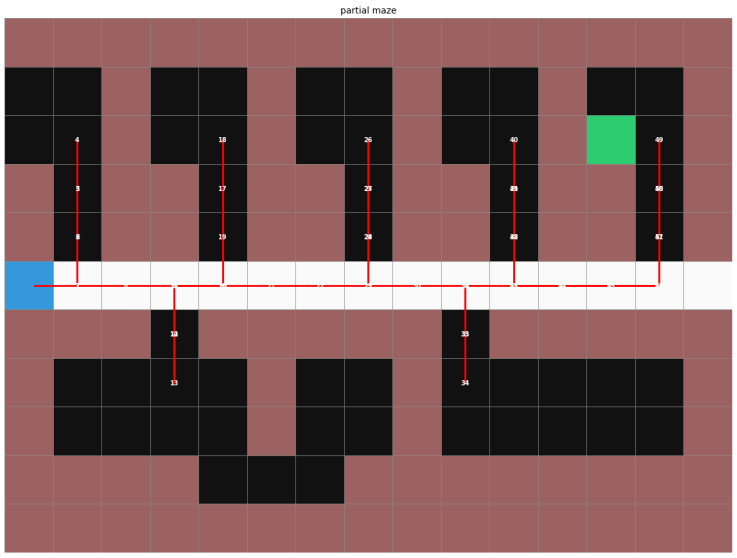


Figure 11: When searching a maze with partial structure, GenPlan treats structured units as options and solves a POMCP on the remaining unstructured map.

D.1 ADDITIONAL RESULTS - BEHAVIORAL

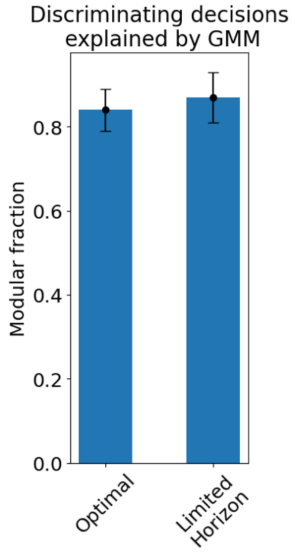


Figure 12: Model fit across participants. Each bar shows the fraction of discriminating decisions for each participant, in which human choice was correctly predicted by our model, but not by Naive-POMCP (Optimal) or DepthPlan (Limited Horizon) models. Error bars are 95% CI over people. To compute these fractions, for each person we first compute the fraction of a person's discriminating decisions that are predicted by GenPlan vs. another model. Then, we aggregate over these individual fractions to obtain 95% CIs.

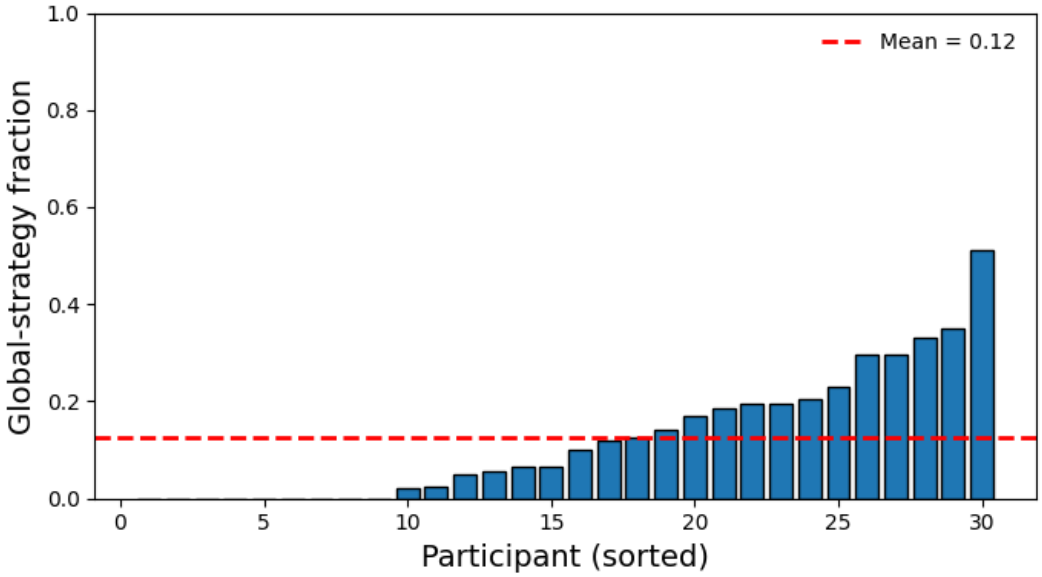


Figure 13: **Individual differences in global-strategy use.** For each participant, we compute the global-strategy fraction as one minus the modular fraction, i.e., the fraction of discriminating decisions better explained by the global planning strategy than by GenPlan. Participants are sorted by global-strategy fraction; the dashed red line indicates the mean across participants. Global-strategy use was generally low ($M = 0.124$, median $= 0.083$), although there was substantial individual variability, ranging from 0 to 0.51.

Figures 14 - 16 each show a sample of human search trajectories in the Maze Search Task. Each panel shows a single participant's path through a structured environment. The majority of sampled trajectories are consistent with GenPlan's predicted piecewise, structure-conditioned policy. For each environment, we also demonstrate a representative trajectory inconsistent with GenPlan (red outline) – showing globally-directed search, consistent with Naive-POMCP and DepthPlan predictions.

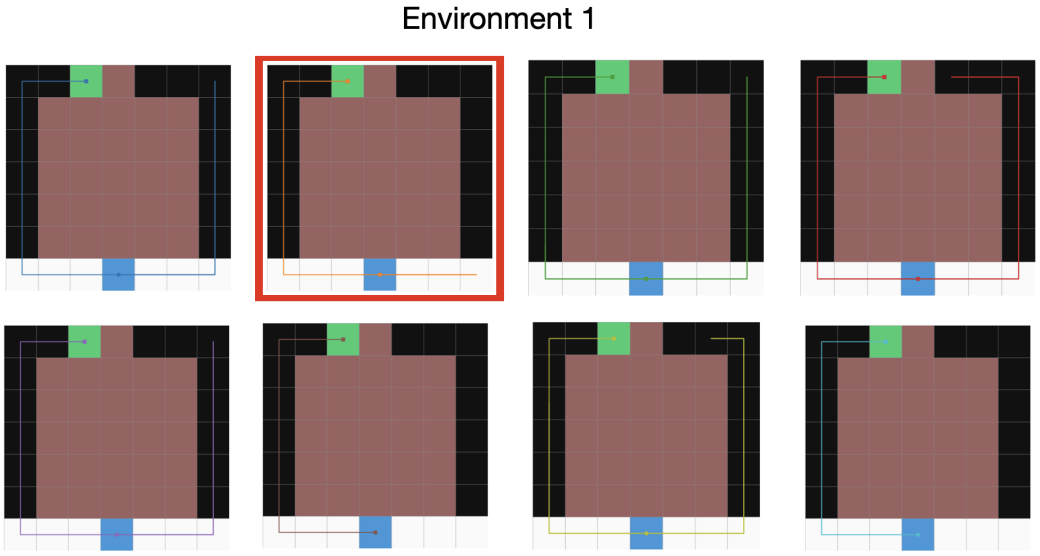


Figure 14: A sample of human search trajectories in the Maze Search Task. Red outline shows a globally-directed representative trajectory inconsistent with GenPlan.

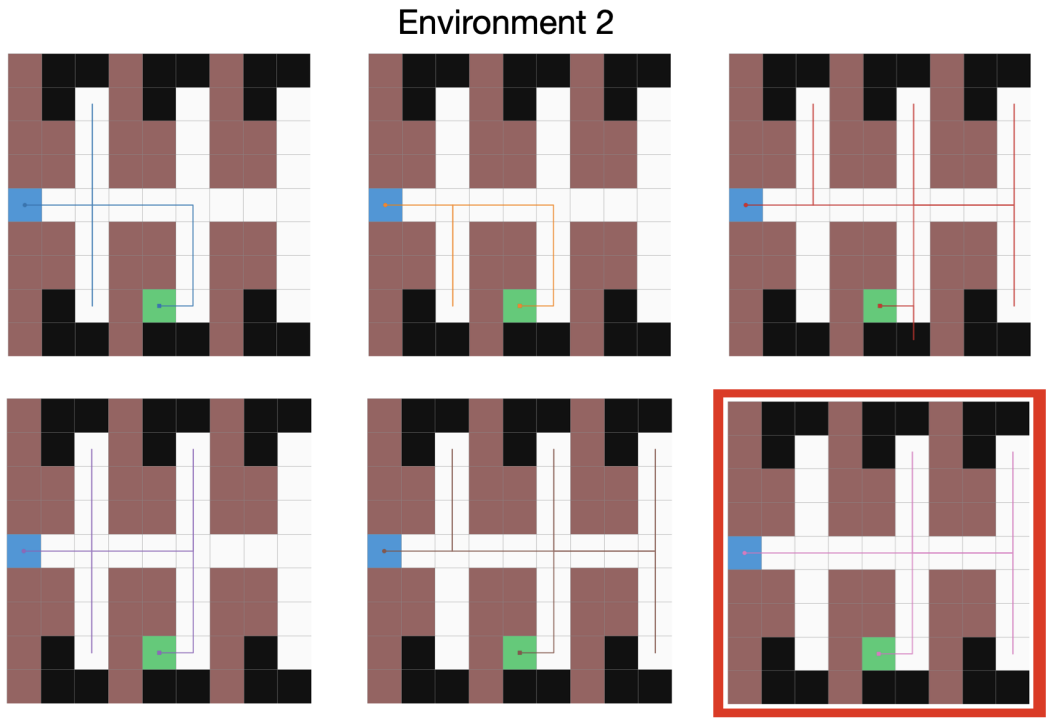


Figure 15: A sample of human search trajectories in the Maze Search Task. Red outline shows a globally-directed representative trajectory inconsistent with GenPlan.

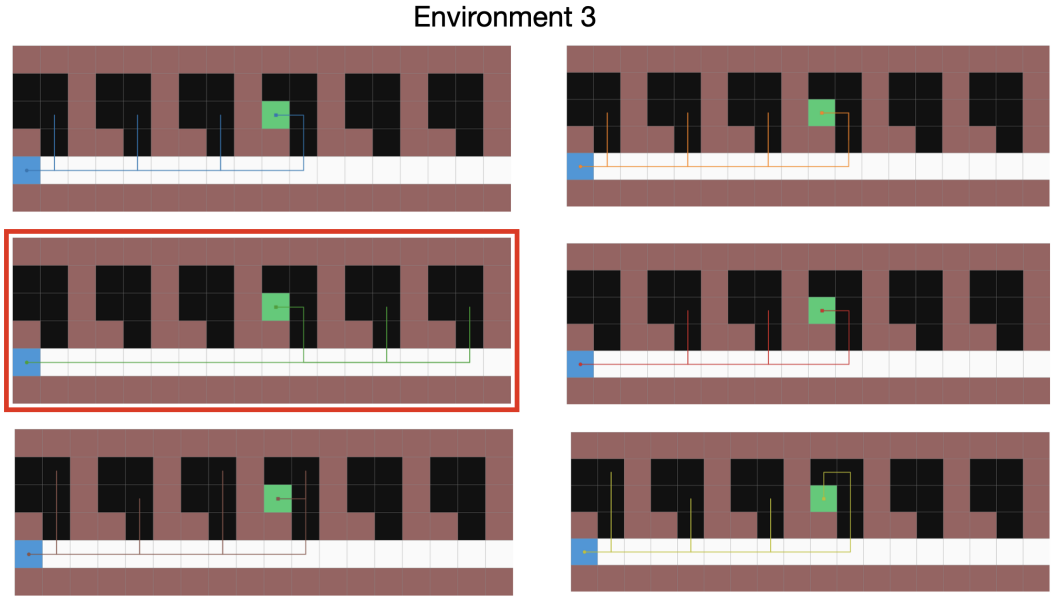


Figure 16: A sample of human search trajectories in the Maze Search Task. Red outline shows a globally-directed representative trajectory inconsistent with GenPlan.

E GENERALIZING ACROSS LLM ARCHITECTURES

Here we used GPT-4.1 as a GMM backend; however, the choice of LLM model and prompting strategy are not critical to our framework's results. Below we show that GMM can be implemented with different LLM architectures. To do this, we show experimental results producing similar reconstructions by using different LLMs as backends: GPT-4.1, Gemini-2.5-flash, Llama-3.3-70B, and Kimi-K2-Instruct-0905. Furthermore, we present results from two different prompting strategies (one-step prompt and multi-step prompts), showing that the exact prompt wording is not critical to producing the given results.

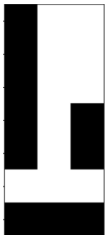
E.1 SINGLE PROMPT

Input map

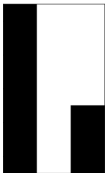


Top scoring unit candidate

GPT-4.1, Gemini-2.5-flash, Llama-3.3-70B



Kimi-K2-Instruct-0905



Reconstructed map

GPT-4.1, Gemini-2.5-flash, Llama-3.3-70B



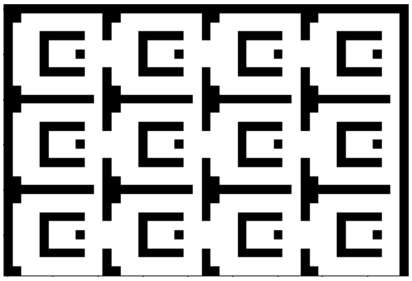
Kimi-K2-Instruct-0905



E.2 MULTI PROMPT

Input map

Complete input map

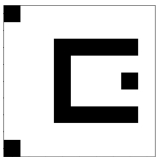


Partial input map



Top scoring unit candidate

GPT-4, Gemini-2.5-flash, Llama-3.3-70B

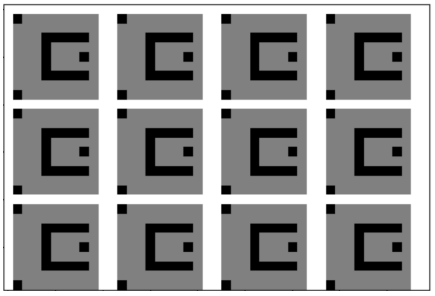


Kimi-K2-Instruct-0905

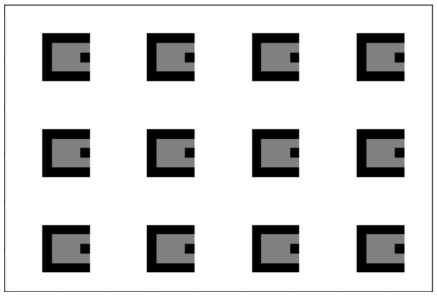


Reconstructed map

GPT-4, Gemini-2.5-flash, Llama-3.3-70B



Kimi-K2-Instruct-0905



F PROMPTS

F.1 ONE-STEP PROMPT FOR GMM

The Single Prompt GMM identifies the unit along with the reconstruction program using one prompt to the LLM. The prompt describes the task as a two-step procedure: first identify the repeating unit, then complete and return a runnable Python program that contains both the unit as a 2D array and the reconstruction function.

System prompt:

```
You are a designer's assistant, skilled in noticing patterns,
combining fragments into a pattern, and extrapolating them.  You
are skilled in identifying the underlying structure of a pattern
and generating new fragments that fit the pattern.  You are also
skilled at writing Python code.
```

User prompt:

```
There are two steps to this task.  In Step 1, you will be given
an input (a map) and asked to identify its constituent units.
The input is a matrix, elements of which can take values 1 and
0.  Your task is to identify a repeating unit in this input.

To be considered a repeating unit, the unit does not have to tile
the space exactly, but it must appear at least twice.  The unit
instances may be flipped horizontally or vertically, translated
horizontally or vertically, and rotated by multiples of 90 degrees
(i.e.  0, 90, 180, 270).

IMPORTANT:
1.  Instances of the unit must NOT overlap in the original input.
2.  The height and width of the unit need not be equal

Example 1.
Given input:  {example input 1}
The repeating unit is:  {example unit 1}

Example 2.
Given input:  {example input 2}
The repeating unit is:  {example unit 2}

Example 3.
Given input:  {example input 3}
The repeating unit is:  {example unit 3}

In Step 2, you will write a function that attempts to identify all
occurrences of the unit in the input.  Return a list containing
the indexical locations of the top left corner for each copy,
along with whether to reflect the copy horizontally and the
number of 90 degree counter-clockwise rotations (these operations
together generate the dihedral group D4).

For instance, in the examples above, possible solutions include

Example 1.
Solution 1:  {example 1 program 1}
Solution 2.  {example 1 program 2}

Example 2.
Solution 1:  {example 2 program 1}
Solution 2:  {example 2 program 2}
```

```
Given your unit and partition, the user will attempt to
reconstruct the input using the following function:

def construct_copy(unit, reflect, rotations):
    if reflect:
      unit = np.flip(unit, 1)
   unit = np.rot90(unit, rotations)
   return unit

def regenerate_pattern(unit, copies, input_dims):
   pattern = -1 * np.ones(shape=input_dims)
   for copy in copies:
      transformed = construct_copy(
         unit,
         copy["reflect"],
         copy["rotations"],
      )
      height, width = transformed.shape
      tl_i, tl_j = copy["top left"]
      try:
         pattern[tl_i:tl_i+height, tl_j:tl_j+width] = transformed
      except:
         pass
   return pattern


We can test the success of this regeneration with

input_map = np.array(input_map)
output = regenerate_pattern(
   unit,
   partition(),
   input_map.shape,
)

Now is your turn.  Propose a unit that can be used to reconstruct
the given input.  Respond by completing the following Python code:

input_map = {input map}
# make sure to define all arrays as numpy arrays
import numpy as np
input_map = np.array(input_map)

unit = [ ...  ]
unit = np.array(unit)

def partition():
   copies = []
   # Place your code here.  Let's think step by step
   return copies

Please include only code in your response, no text."'
```

F.2 MULTI-PROMPT GMM

To improve GMM scalability on large maps, we introduce a two-step approach. The two steps are implemented in two separate prompts, which adapt the strategy described in the previous section. The

first prompt provides a part of the map and asks to identify a repeating unit. The second prompt asks the LLM to infer a reconstruction program for the complete map given the previously identified unit.

System prompt:

```
You are a designer's assistant, skilled in noticing patterns,
combining fragments into patterns, and extrapolating them.  You
are skilled in identifying the underlying structure of a pattern
and generating new fragments that fit the pattern.  You are also
skilled at writing Python code.
```

Unit Identification Prompt:

```
You will be given an input (a map) and asked to identify its
constituent units.  The input is a matrix, elements of which can
take values 1 and 0.  Your task is to identify a repeating unit in
this input, and ONLY output the unit as a 2D python array.  DO NOT
include anything else in the completion.

To be considered a repeating unit, the unit does not have to tile
the space exactly, but it must appear at least twice.  The unit
instances may be flipped horizontally or vertically, translated
horizontally or vertically, and rotated by multiples of 90 degrees
(i.e.  0, 90, 180, 270).

IMPORTANT:
1.  Instances of the unit must NOT overlap in the original input.
2.  The height and width of the unit need not be equal

Example 1.
Given input:  {example input 1}
The repeating unit is:  {example unit 1}

Example 2.
Given input:  {example input 2}
The repeating unit is:  {example unit 2}

Example 3.
Given input:  {example input 3}
The repeating unit is:  {example unit 3}

The input you are working with is the following map:  {input map}
```

Reconstruction Program Prompt:

```
You will write a function that attempts to identify all
non-overlapping occurrences of the unit in the input.  Return a
list containing the indexical locations of the top left corner
for each copy, along with whether to reflect the copy horizontally
and the number of 90 degree counter-clockwise rotations (these
operations together generate the dihedral group D4).


Example 1.
Given input:  {example input 1}
The unit is:  {example unit 1}
A solution is:  {example 1 program 1}
An alternative, more structured solution is:  {example 1 program
2}


Given input:  {example input 2}
The unit is:  {example unit 2}
A solution is:  {example 2 program 1}
An alternative, more structured solution is:  {example 2 program
2}


Given your unit and partition, the user will attempt to
reconstruct the input using the following function:

def construct_copy(unit, reflect, rotations):
    if reflect:
      unit = np.flip(unit, 1)
   unit = np.rot90(unit, rotations)                                    )
   return unit

def regenerate_pattern(unit, copies, input_dims):
   pattern = -1 * np.ones(shape=input_dims)
   for copy in copies:
      transformed = construct_copy(
         unit,
         copy["reflect"],
         copy["rotations"],
      )
      height, width = transformed.shape
      tl_i, tl_j = copy["top left"]
      try:
         pattern[tl_i:tl_i+height, tl_j:tl_j+width] = transformed
      except:
         pass
   return pattern


We can test the success of this regeneration with

input_map = np.array(input_map)
output = regenerate_pattern(
   unit,
   partition(),
   input_map.shape,
)
```

```
Now is your turn.  Respond by completing the following Python
code,
1.  include everything that is between START OF CODE and END OF
CODE
2.  include the entire input map provided, do not use ...  to omit
3.  ONLY fill partition(), do not use variables/functions that are
not defined
4.  In the returned copies, follow the exact key names in the
examples:  'top left', 'reflect', 'rotations'.
5.  DO NOT include anything else in the completion

# START OF CODE, make sure to define all arrays as numpy arrays
import numpy as np

input_map = {input map}
unit = {input unit}
input_map = np.array(input_map)
unit = np.array(unit)

def partition():
   copies = []
   # Place your code here.  Let's think step by step
   return copies
result = partition()

# END OF CODE
```

F.3 IN-PROMPT EXAMPLES

Example 1

Input map

```
[
  [1, 0, 1, 0],
  [0, 1, 0, 1],
  [0, 0, 0, 0],
  [1, 0, 1, 0],
  [0, 1, 0, 1]
]
```

Unit

```
[
  [1, 0],
  [0, 1]
]
```

Reconstruction program 1:

```
def partition():
  return [
    {"top left": (0,0), "reflect": False, "rotations": 0},
    {"top left": (0,2), "reflect": False, "rotations": 0},
    {"top left": (3,0), "reflect": False, "rotations": 0},
    {"top left": (3,2), "reflect": False, "rotations": 0},
  ]
```

Reconstruction program 2:

```
def partition():
   copies = []
   for tl_i, tl_j in [(0,0),(0,2),(3,0),(3,2)]:
      copies.append(
         {"top left": (tl_i, tl_j), "reflect": False, "rotations": 0},
      )
   return copies
```

Example 2

Input map

```
[
  [1,1,1,1,1,1],
  [0,0,1,0,0,1],
  [0,0,1,0,0,1],
  [1,0,1,1,0,1],
  [0,0,0,0,0,0],
  [1,1,1,1,1,1],
  [1,0,1,1,0,1],
  [0,0,1,0,0,1],
  [0,0,1,0,0,1]
]
```

Unit

```
[
  [1,1,1],
  [0,0,1],
  [0,0,1],
  [1,0,1]
]
```

Reconstruction program 1:

```
def partition():
   return [
      {"top left": (0,0), "reflect": False, "rotations": 0},
      {"top left": (0,3), "reflect": False, "rotations": 0},
      {"top left": (5,0), "reflect": True, "rotations": 2},
      {"top left": (5,3), "reflect": True, "rotations": 2},
   ]
```

Reconstruction program 2:

```
def partition():
   copies = []
   for tl_i, tl_j in [(0,0),(0,3)]:
      copies.append(
         {"top left": (tl_i, tl_j), "reflect": False, "rotations": 0},
      )
   for tl_i, tl_j in [(5,0),(5,3)]:
      copies.append(
         {"top left": (tl_i, tl_j), "reflect": True, "rotations": 2},
      )
   return copies
```

Example 3

Input map

```
[
  [1,1,1,1,1,1],
  [1,0,1,0,1,0],
  [1,0,1,0,1,0],
  [0,0,0,0,0,0],
  [0,1,0,1,0,1],
  [0,1,0,1,0,1],
  [1,1,1,1,1,1]
]
```

Unit

```
[
  [1,1],
  [1,0],
  [1,0]
]
```

G PSEUDOCODE

Algorithm 1 Single-prompt Generative Map Module

Require: I : Input map, t : Threshold, C : Number of completions, S : Likelihood function
Ensure: λ : Generative program, u : Unit

$S' \leftarrow 0$
$\lambda \leftarrow$ ""
$u \leftarrow$ ""
while $S' < t$ **do**
    Generate a prompt from I
    Send the prompt and receive C completions $(\lambda_1, u_1), \ldots, (\lambda_C, u_C)$
    Extract Python programs $\{\lambda_1, \lambda_2, \ldots, \lambda_C\}$
    for all $\lambda_i \in \{\lambda_1, \ldots, \lambda_C\}$ **do**
        if λ_i runs successfully **then**
            $S_i \leftarrow S(\lambda_i)$
        end if
    end for
    $(S', i) \leftarrow \max_i S_i$ ▷ highest scoring program based on likelihood
    $\lambda \leftarrow \lambda_i$
    $u \leftarrow u_i$
end while
return λ, u

Algorithm 2 Multi-prompt Generative Map Module

Require: I_c : Input map, I_p : Partial map, t : Threshold, C : Number of completions, S : Likelihood function
Ensure: λ : Generative program, u : unit

$S' \leftarrow 0$
$\lambda \leftarrow$ ""
$u \leftarrow$ ""
while $S' < t$ **do**
    Generate a prompt from I_p
    Send the prompt and receive C unit candidates $u_1, \ldots, u_C$
    for all $u_i \in \{u_1, \ldots, u_C\}$ **do**
        Generate a prompt from I_c and u_i
        Send the prompt and receive a program λ_i
        if λ_i runs successfully **then**
            $S_i \leftarrow S(\lambda_i)$
        end if
    end for
    $(S', i) \leftarrow \max_i S_i$ ▷ highest scoring program based on likelihood
    $\lambda \leftarrow \lambda_i$
    $u \leftarrow u_i$
end while
return λ, u

Algorithm 3 Structure-Based Planner

Require: I : Input map, u : Unit, C_i : Unit copies
Ensure: P : Agent path

$(r, c) \leftarrow$ Initial agent position
$C \leftarrow$ Empty set to track fully explored unit copies
$\pi \leftarrow$ Policy for unit exploration based on location of entrance
while C does not contain all copies **do**
    Run POMCP on I until reaching an unexplored unit C_i

    Identify the current entrance e into C_i
    if policy from e found in π **then**
        Explore C_i with policy from π
        Add C_i to C
    else
        Run POMCP on C_i to explore the unit
        Add new policy (e, π_e) to π
        Add C_i to C
    end if

    $o_1, \ldots, o_m \leftarrow$ Exit locations of C_i
    for o_j in $\{o_1, \ldots, o_m\}$ **do**
        $p_j \leftarrow$ compute penalty for escaping C_i from o_j
    end for
    Run MCTS on C_i until reaching an exit to escape
end while

Run POMCP on I to explore the rest of the map
return P

Algorithm 4 POMCP for In-Unit Planning

procedure SEARCH(h)
    if $B(h) = \emptyset$ **then**
        return
    else
        repeat
            $s_{\text{exit}} \sim B(h)$
            Simulate($s_{\text{exit}}, h, 0$)
        until Timeout
        return $\arg\max_a V(ha)$
    end if
end procedure

procedure ROLLOUT(s_{exit}, h, depth)
    if depth $>$ depth limit **then**
        return 0
    end if
    $a \sim \pi_{\text{random}}$
    $(o, r) \sim \mathcal{G}(h, a)$
    if o contains s_{exit} **then**
        return r
    else
        return $r +$ Rollout(s_{exit}, ha, depth$+1$)
    end if
end procedure

procedure SIMULATE(s_{exit}, h, depth)
    $N(h) \leftarrow N(h) + 1$
    if depth $>$ depth limit **then**
        return 0
    end if
    if $h \notin T$ **then**
        for $a \in \{\text{up}, \text{right}, \text{down}, \text{left}\}$ **do**
            $T(ha) \leftarrow (N_{\text{init}}(ha), V_{\text{init}}(ha), \emptyset)$
        end for
        return Rollout(s_{exit}, h, depth)
    end if
    $a \leftarrow \arg\max_a \left[V(ha) + c\sqrt{\frac{\log N(h)}{N(ha)}} \right]$
    $(o, r) \sim \mathcal{G}(h, a)$
    if o contains s_{exit} **then**
        $N(ha) \leftarrow N(ha) + 1$
    else
        $r \leftarrow r +$ Simulate(s_{exit}, ha, depth $+ 1$)
    end if
    $V(ha) \leftarrow V(ha) + \frac{r - V(ha)}{N(ha)}$
    return r
end procedure

H HUMAN EXPERIMENT - MAZE SEARCH TASK

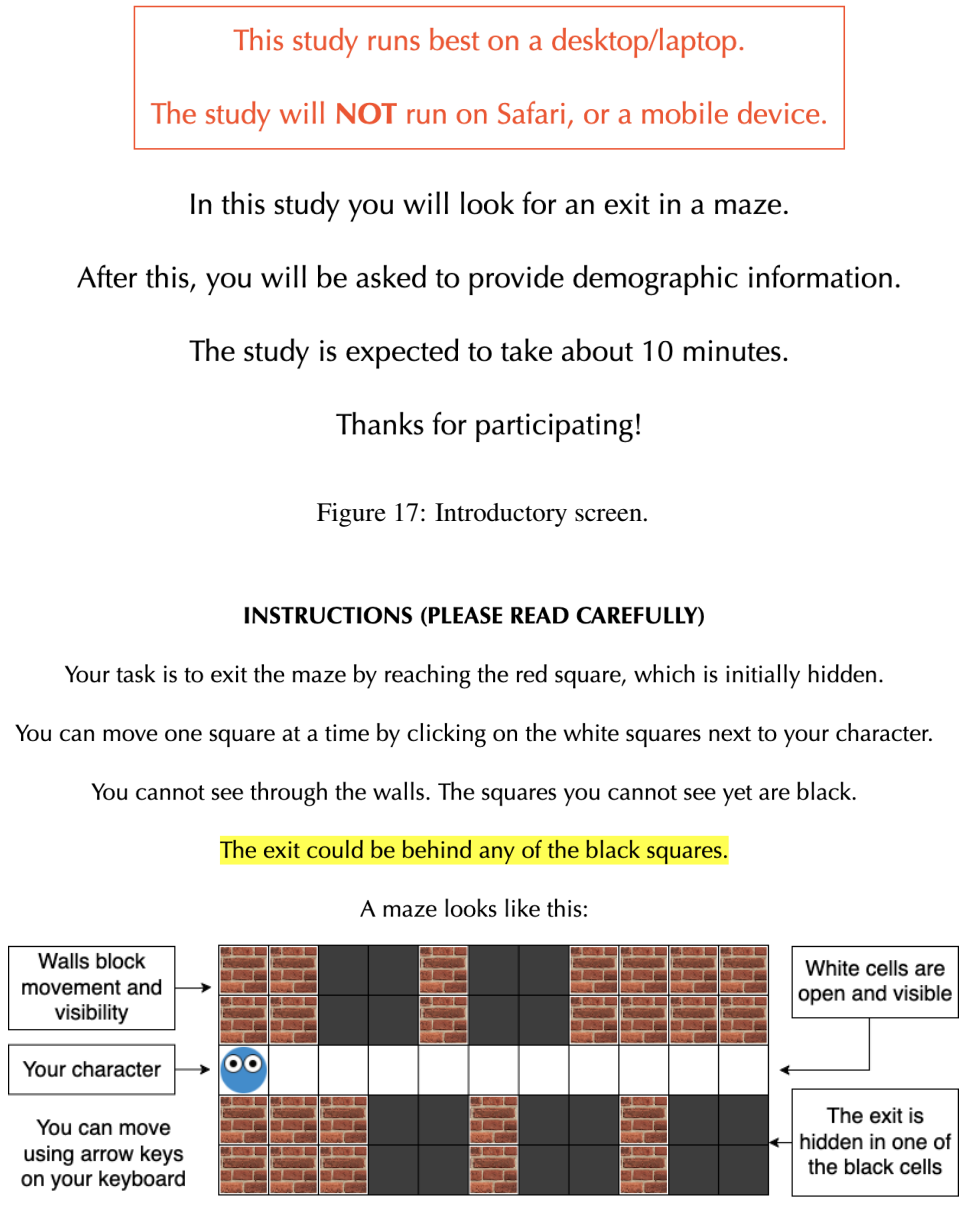


Figure 17: Introductory screen.

Figure 18: Instructions.

Practice Maze 1 of 5

Let's look at this map. There are some black squares, a brick wall, and your character.

The exit could be behind any of the black cells.

You can move your character by clicking one of adjacent white cells.

Find the exit and step on it to exit the maze.

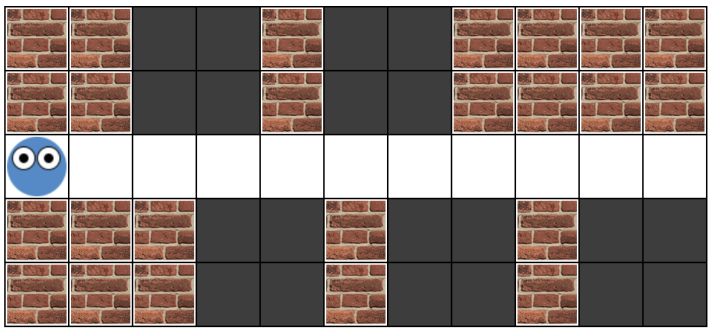


Figure 19: Practice (there are 5 practice trials).

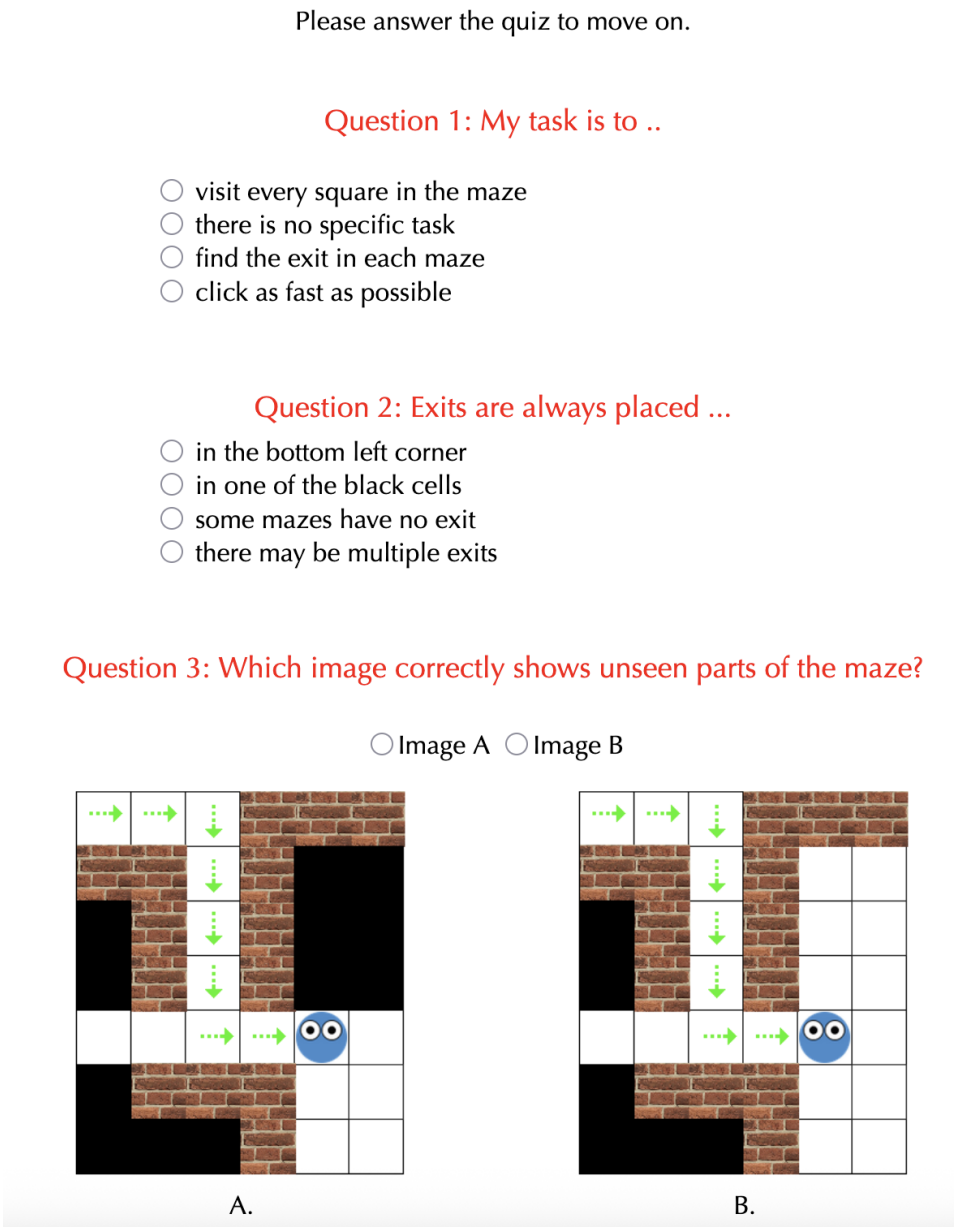


Figure 20: Comprehension Quiz.

Maze 1 of 21

Find the exit and step on it to exit the maze.

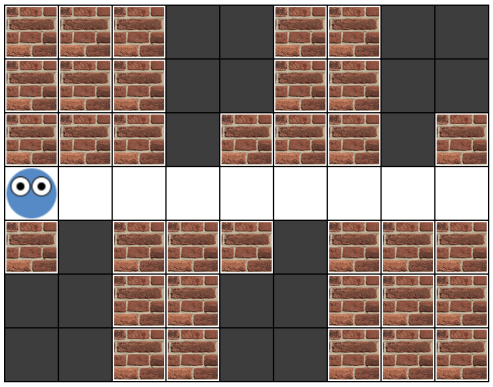


Figure 21: Experiment view.

I LLM-DIRECT PLANNER: EVALUATION AND FAILURE ANALYSIS

We evaluated LLM-Plan — a direct LLM planning baseline without program induction — to characterize what structured belief-state planning requires beyond general-purpose language model reasoning. The baseline received the same map representation and task description as GMM, but was asked to produce a search path directly, without an intermediate program synthesis step (the prompt is given in Section **Prompt** of this Appendix.).

MODELS AND CONDITIONS

We tested four configurations on a subset of N structured maps from Experiment 1 (maps with 2–6 structural units, used in the human study):

- GPT-4.1 (no reasoning)
- GPT-5.1 (default reasoning effort)
- GPT-5.2, low reasoning effort
- GPT-5.2, high reasoning effort

RESULTS

Table 1 summarizes the failure mode observed for each configuration. Section I shows representative planned paths for each condition on the same map.

Model	Valid moves	Reaches exit	Correct objective
GPT-4.1	X	X	X
GPT-5.1	V	X	X
GPT-5.2 low	V	X	X
GPT-5.2 high	V	X	X
GenPlan (ours)	V	V	V

Table 1: LLM-Plan failure modes by configuration. Valid moves: no attempted wall collisions. Reaches exit: path terminates at exit cell. Correct objective: maintains an observation model for the environment and plans against expected cost under uniform exit prior.

FAILURE ANALYSIS

GPT-4.1 GPT-4.1 failed to produce valid paths on all tested maps, resulting in paths that repeatedly attempted illegal moves into wall cells. Since the task requires tracking agent position and wall locations across a sequence of steps, this failure shows that GPT-4.1 lacks the spatial state maintenance required for basic grid-world navigation. Figures 22 and 23 show a representative attempted and executed path.

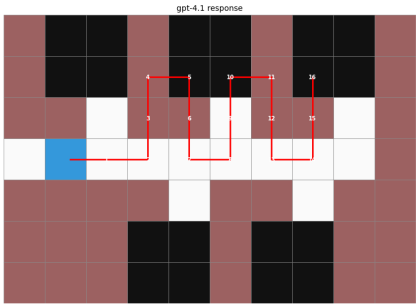


Figure 22: Representative path produced by GPT-4.1. The attempted path includes repeated illegal moves.

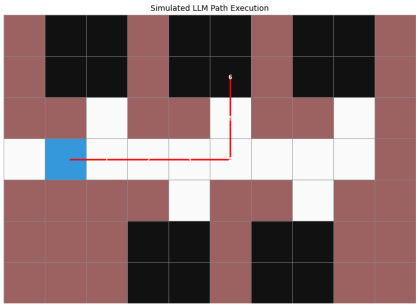


Figure 23: Representative path produced by GPT-4.1. The resulting path with only valid moves executed.

GPT-5.1 In addition to the same failure modes as 4.1, GPT-5.1 occasionally produced paths that respected movement constraints, but failed to pursue the search objective. In these instances, the model navigated to a location in the maze and terminated without having revealed the exit or exhausted unobserved cells, suggesting that it did not maintain a goal-directed search strategy. This failure is consistent with an inability to reason about the belief state over exit locations. Figure 24 shows a representative planned path in one such case.

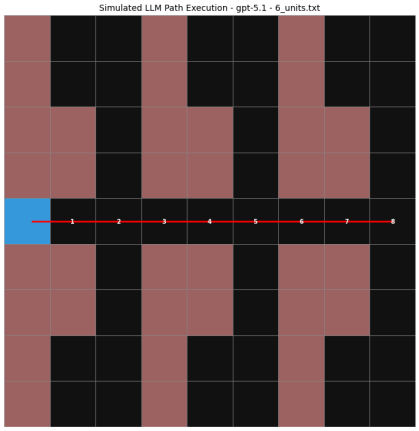


Figure 24: Representative path produced by GPT-5.1. The agent navigates within task constraints, but terminates prematurely without locating the exit.

GPT-5.2, low reasoning effort GPT-5.2 at low reasoning effort produced the most effective paths of all tested configurations when given small mazes. However, in such cases it consistently substituted a full-coverage objective for the correct expected-cost objective: rather than planning against a uniform prior over exit locations, it attempted to visit every unobserved cell in the maze. This behavior reflects a failure to internalize the observation model and maintain uncertainty over the exit location.

On larger mazes, GPT-5.2 terminated prematurely – a behavior already present in GPT-5.1. Figures 25 and 26 show representative paths on a small and large map respectively.

GPT-5.2, high reasoning effort On small maps, GPT-5.2 at high reasoning effort exhibited the same full-coverage behavior as the low reasoning condition. On larger maps, the model timed out, making it unsuitable for evaluation across the full map set. Taken together, these results indicate that the expected-cost planning problem is not reducible to general-purpose language model reasoning, even at the highest reasoning budgets tested.

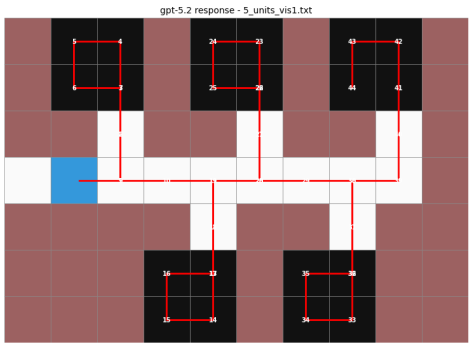


Figure 25: Representative paths produced by GPT-5.2 (low or high reasoning effort) on a small map. In both low and high reasoning cases the agent pursues full coverage rather than expected-cost minimization.

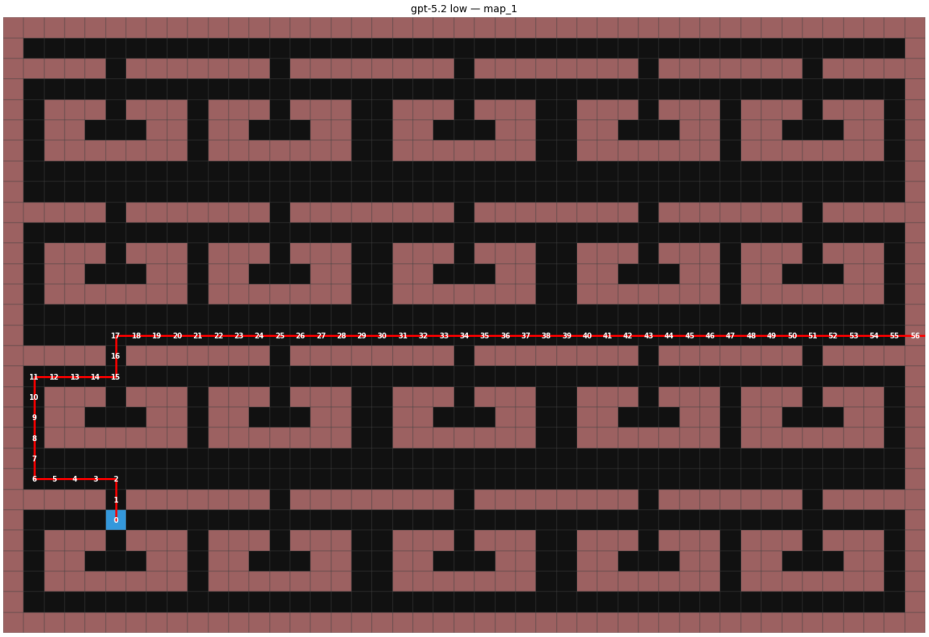


Figure 26: Representative paths produced by GPT-5.2 (low reasoning effort) on a small map (left) and large map (right). In both cases the agent pursues full coverage rather than expected-cost minimization.

PROMPT

```
You are a grid-world path planner.  Your task is to navigate a
partially observed maze to find a hidden exit in as few steps as
possible.

GRID LEGEND:
  3 = wall (impassable)
  0 = unobserved floor (passable; content hidden)
  6 = observed floor (passable; confirmed empty)
  5 = agent start position (treat as observed floor)
  2 = exit (revealed when it enters line of sight)

GRID FORMAT:
  Each row is a string of digits.  Each character is one cell.
  Row 0 is the top row, col 0 is the leftmost character.

MOVEMENT RULES:
  - 4-directional only (up, down, left, right).  No diagonal
movement.
  - The agent cannot move into or through wall cells (3).
  - Attempting to move into a wall leaves the agent in place
    (do not include failed moves in the path).
  - The agent can revisit observed floor cells (6) freely.

LINE OF SIGHT (ISOVIST):
  At each position, the agent reveals cells within its isovist --
  the region fully visible from the agent's current cell,
  accounting for walls as occluders.

  VISIBILITY RULE:
    A candidate cell is visible if and only if its entire area can
    be seen from somewhere within the agent's cell.  Treat each
cell
    as a unit square in continuous 2D space.  A candidate cell is
NOT
    visible if any part of its area is fully occluded -- that is,
    if every line of sight from within the agent's cell to that
part
    is intercepted by a wall cell.

  PRACTICAL TEST:
    Check the four corners of the candidate cell.  Cast rays from
    the agent's cell toward each corner.  If a corner lies behind
a
    wall edge such that no ray from anywhere inside the agent's
cell
    can reach it without passing through a wall, the candidate
cell
    is not visible.

  MAZE OBSERVABILITY EXAMPLE (agent at (0,2), marked 5):
    6653
    6360
    6333
    Cell (1,3) is NOT visible from (0,2).
    Reason:  the wall occupying (0,3) blocks the upper-right
corner
    of cell (1,3).  No ray from within the agent's cell at (0,2)
can
    reach that corner without passing through the wall at (0,3).
    Since the entire area of (1,3) cannot be confirmed visible,
    it remains unobserved.
```

```
  NOTE: This is stricter than cardinal ray casting.  Cells that
share
  only a diagonal with the agent's current cell are typically not
  visible, and cells around wall corners may be partially occluded
  even if they are adjacent.

GOAL:
  Navigate from the start position to reach the exit (2) in as few
  steps as possible.  Because the exit location is unknown, the
agent
  must plan a search path that exposes unobserved cells
efficiently.
  Once the exit becomes visible (revealed as 2), the agent travels
  directly to it via the shortest available path.  The path ends
when
  the agent steps onto the exit cell.

STRATEGY:
  Plan as if the exit could be behind any unobserved (0) cell with
  equal probability.  Assume
  that the plan will be terminated as soon as the exit is
revealed,
  and the agent will route directly to it.

OUTPUT FORMAT:
  Return a coordinate sequence of (row,col) cells visited in
order,
  including the start.
  Format:  (row, col)(row, col)...
  Do not include any other explanation or commentary.

For example, in this maze:
  0003
  0003
  3033
  5666
The start position is given at:  (row=3, col=0)
The agent at (3,0) has already revealed the entire bottom corridor
(row 3).
Cells in rows 0-2 remain unobserved.  Note that (2,0) is a wall,
so the
unobserved cells in columns 0-1 of rows 0-2 are not yet visible
despite
the agent's proximity -- they are occluded from (3,0).
The exit (2) is hidden behind one of the unobserved (0) cells.
The expected response is:  (3,0)(3,1)(2,1)(1,1)

Now is your turn:
```