HYPOTHESIS-GUIDED DISCOVERY OF COGNITIVE ALGORITHMS VIA PROGRAM REFINEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Developing cognitive models of algorithmic reasoning from behavioral data is a central problem in cognitive science that challenges current methods. Traditional techniques are interpretable and benefit from human expertise, but lack flexibility and scalability. Emerging techniques using large language models (LLMs) for de novo generation of cognitive models are scalable and flexible, but lack a role for human expertise and have mostly been applied to simpler choice tasks. We propose a hybrid system that treats discovery of cognitive algorithms as a program refinement problem. Human-created cognitive models are expressed as probabilistic programs and provided to a system of LLM agents with a mandate to: identify mismatches between model and behavior; propose code-level modifications within researcher-specified constraints; and verify structural fidelity. Revisions propagate to a probabilistic inference module that performs inference for latent variables and data likelihood computations. We evaluate the pipeline on human behavior in a problem-solving paradigm that exposes a variety of cognitive algorithms. Revised models consistently improve model fit relative to ancestral models and reveal a small set of recurring innovations that capture meaningful behavioral variability in this task.

1 INTRODUCTION

An overarching goal of modern cognitive science and artificial intelligence is to advance the science of cognitive modeling. Adaptive and scalable methods to induce interpretable cognitive models from behavioral data is a key frontier within this goal. In particular, advances that support induction of models for *algorithmically-structured* behaviors would enable innovation across many sub-areas, and substantively expand the reach of cognitive science. Recent advances in LLM-supported modeling (e.g. Rmus et al., 2025; Curtis et al., 2025) establish new potential for scalable generation of cognitive models. However, current methods (1) generally under-utilize human expertise; (2) often produce uninteresting models that reflect the LLM's prior; and (3) have mostly been applied at scale to simpler problems than algorithm recovery. We propose a solution and evaluate it in depth on a single well-suited problem-solving paradigm. Our approach views model discovery as *program refinement*: a scientist's best hypothesized models are expressed as probabilistic programs and provided to a system of LLM agents with a mandate to identify targeted revisions that improve the fit between these initial proposals and human data; revised programs propagate to a probabilistic inference module that performs inference for latent variables and data likelihood computations. Our approach offers systematic integration of human expertise into LLM-supported discovery of algorithmic cognitive models.

2 BACKGROUND

Standard approaches to cognitive modeling classically rely on model selection among a fixed set of models (Farrell and Lewandowsky, 2018). This approach benefits from human expertise, strong interpretability, and normative standards for likelihood-based inference (Wilson and Collins, 2019). It has been successful in domains such as multi-option choice or reward learning tasks, but involves fundamental limitations that become particularly apparent when applied to more complex settings; in particular, it is typically very challenging to capture the richness and variety of human thinking

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.

in *algorithmically-structured* behaviors that unfold over time such as problem-solving (Allen et al., 2020), planning (Van Opheusden et al., 2023), repeated games (Brockbank and Vul, 2024), or social interactions (Yu and Thompson, 2024; Jha et al., 2025). For example, imagine observing the actions of a skilled (Rubik's) Cuber, and trying to reverse-engineer a cognitive model of their underlying knowledge: the model would need to specify how the person's actions depend not only on a complex set of cube features, but also potentially on a history of actions and their outcomes; in other words, the model would need to identify the structure of a latent *cognitive algorithm*.

At least two challenges make this a very difficult problem. One fundamental challenge is that the space of plausible cognitive algorithms is large and richly structured, making it difficult to anticipate relevant model variants. Another fundamental challenge is that even when a model is generally a good hypothesis, human behavior is often not well-captured by a single algorithm; even in domains where people learn canonical algorithms, such as mathematical reasoning (Siegler and Shrager, 1984), people often exhibit systematic but idiosyncratic deviations from idealized procedures, including local execution errors, minor rule variations, and alternative tie-breaking conventions. Together, these challenges highlight a mismatch between classical approaches to model specification and the variability of real behavior.

3 Related Work

In response to the limitations of traditional cognitive modeling, efforts have focused on model *construction*. For example, an influential framework views model construction as *program induction* (Ellis et al., 2023; Jiang et al., 2024; Piantadosi et al., 2016). These methods typically specify a domain-specific language and perform statistical inference over expressible programs, extending a tradition of grammar-induction (O'Donnell et al., 2009), with successful application to, e.g., language (Yang and Piantadosi, 2022) and concept learning (Piantadosi et al., 2012; Lake et al., 2015; Rule et al., 2024) but often require specification of complex languages and sampling algorithms. These difficulties motivated more general systems for program induction, driving development of probabilistic programming languages such as WebPPL (Goodman and Stuhlmüller, 2014) that are natively enriched with probabilistic semantics and generic methods for statistical inference. Probabilistic programming frameworks resolve some difficulties of domain-specific methods, but remain primarily a tool for building hand-crafted models.

In parallel, advances have been made in automated model discovery. For example, one approach generates models by training artificial neural networks on very large datasets (Agrawal et al., 2020). This approach has been examined in decision-based problems (Peterson et al., 2021) as well as more algorithmic behaviors (Ji-An et al., 2025; Thompson et al., 2022); it is scalable and can learn complex or counterintuitive models, but is not easily interpretable and is data-intensive. Prior theories can be integrated via initialization of network weights by pre-training on synthetic data generated from structured cognitive models (Bourgin et al., 2019). More recently, emerging methods have leveraged the increasing capabilities of LLMs. One class of approaches simply fine-tunes LLMs on large volumes of human behavioral data (Binz et al., 2025); this is also data-intensive and the resulting models are difficult to interpret (Bowers et al., 2025).

Another class of methods leverages the capabilities of LLMs for program synthesis (Ellis et al., 2022; Pourcel et al., 2025). Under this approach, cognitive models can be expressed as programs in, e.g., Python. One stark proposal is to fully automate the larger enterprise of cognitive science (Jagadish et al., 2026a). Recent approaches have used LLMs to directly propose cognitive models (Rmus et al., 2025), integrating LLM proposals into model search or inference pipelines (Colas et al., 2025), using them to guide proposal generation (Curtis et al., 2025) or improve optimization over program spaces (Piriyakulkij et al., 2024), or leveraging natural language to bias synthesis toward more human-like solutions (Ellis, 2023). These approaches are most closely related to our framework, especially the approach of Rmus et al. (2025).

However, these methods focus primarily on de novo generation of candidate programs – a highly under-constrained problem that could benefit greatly from human expertise and the constraints of existing hypotheses or theories. In our experience (and commonly noted in the field), the programs proposed by automated discovery can often seem uninteresting or overly influenced by an LLM's prior over Python programs in general, being disconnected from the principles that researchers in the area would consider meaningful innovations, or e.g. simply enumerating contingencies.

Another limitation of existing approaches is that they have primarily been applied to canonical problems that often lack rich algorithmic structure, such as multi-option decision-making (e.g. Jagadish et al., 2026b) or reward-learning tasks (Rmus et al., 2026) or judgment problems (Prystawski et al., 2026). While there are many good reasons to begin with problems that have received much attention within cognitive science, this focus leaves open what we see as one of the most exciting opportunities presented by these methods: the opportunity to go *beyond* the problems we have focused on historically in part as a reflection of limited methods. Recent advances in e.g. program synthesis (Curtis et al., 2025) and evolutionary coding (Tang et al., 2026) suggest that LLM discovery of rich algorithmic structure can be very effective, and that this approach can be applied in behavioral contexts (Wang et al., 2026).

4 Key Contribution

We propose a system for *hypothesis-guided* discovery of algorithmically structured cognitive models. Our approach views model discovery as a program-refinement problem. We express existing models as probabilistic programs, which are provided along with human data to a system of LLM agents that iteratively adapt the programs to better account for observed behavioral trajectories. LLM agents are instructed to identify mismatches between a program and behavior, and to propose targeted modifications as executable code. In particular, the agents are given semantically rich instructions in natural language, such that their revisions can be precisely scoped. Here we used this scoping to constrain the agents' modifications to preserve the basic structure of the original model. We represent this structure explicitly through structured annotated program components and revision specifications, which constrain how agents can introduce new decision rules, internal state, and other components. An additional benefit is that agents can explain their revisions in natural language.

The pipeline consists of four modules that operate sequentially. Given a task specification, base model programs $(p_{t_1}, p_{t_2}, \ldots, p_{t_n})$, and a behavioral trajectory d, the system produces and evaluates a revised program $\hat{p}_t$. Three modules, the Game, Coding, and Audit modules, are implemented using LLMs (in this paper: OpenAI's GPT-5.4). A Game module interprets the trajectory and generates a structured update signal along with a natural language rationale; a Coding module maps this signal to program modifications; an Audit module verifies correctness and fidelity. In contrast, the Inference module is not LLM-based; it instead performs probabilistic program evaluation using standard methods for probabilistic inference, computing the likelihood of the observed behavior under the revised program. We evaluate the system on human behavior in a novel problem-solving task that admits multiple strategies and induces diverse behavioral patterns across experimental conditions. The pipeline reliably produces revised programs that improve model fit while remaining consistent with the original strategy space.

4.1 Cognitive models as probabilistic programs

Following previous work in cognitive science (Griffiths et al., 2024) and hierarchical reinforcement learning (Andre et al., 2002), we use probabilistic programs as a representation for cognitive models that capture structured problem-solving strategies. Probabilistic programs combine the expressivity of structured programs with the flexibility of probabilistic models to provide a general (indeed, universal) representation for sequential decision-making (van de Meent et al., 2021). For example, a probabilistic program can represent a hierarchical policy in which a high-level controller probabilistically decides between executing different algorithms instantiated as low-level policies (e.g., turning on colors in a blue-green-red order versus green-blue-red order). The current work uses the FlipPy Python library (Ho and Correa, 2026), which takes a subset of Python as a "host" language and combines it with sampling and conditioning statements to express probabilistic programs. One of the benefits of using FlipPy, as opposed to other probabilistic programming languages such as Church (Goodman et al., 2014) (LISP-based) or WebPPL (Goodman and Stuhlmüller, 2014) (JavaScript-based), is that LLMs demonstrate strong biases to program in Python (Twist et al., 2025).

4.2 Strategy Discovery and Evaluation System

Our system transforms base model programs into revised programs: given base model program (p_t), participant trajectory d, and a task specification, the system produces a revised program $\hat{p}_t$

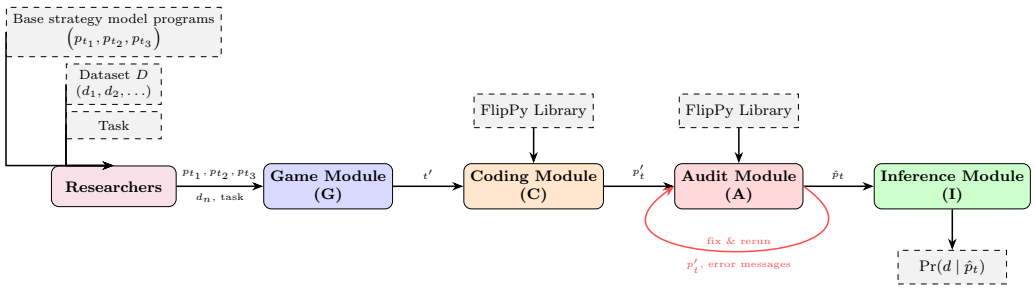

Figure 1: Strategy discovery and evaluation system

and evaluates its likelihood $\Pr(d \mid \hat{p}_t)$. The system consists of four modules: the **Game Module** (G), **Coding Module** (C), **Audit Module** (A), and **Inference Module** (I). These modules operate in a structured sequence: each stage transforms the output of the previous one, jointly producing validated revised programs. The **Game Module** and **Coding Module** are separated to distinguish behavioral reasoning from code generation. The Game Module has access to the task specification and participant trajectories. The Coding Module specializes in translating these revisions into executable FlipPy code using the FlipPy library. This separation also reduces context size for each module, improving output reliability. Fig. 1 provides a high-level overview of this system.

1. **Researchers' input to the system.** The system is initialized with a set of human-written cognitive models as inputs: base strategy model programs $(p_{t_1}, p_{t_2}, p_{t_3})$, participant data d from dataset D, and the task specification. These inputs define the initial state of the system and are provided to the Game Module G.
2. **Game Module (G).** For every participant data d and base strategy program p_t, the Game Module G generates a strategy revision signal t', representing a structured transformation of the base strategy program class induced by the observed behavior from this participant. The revision signal specifies targeted modifications to the decision policy, including rule-level changes, added latent variables and parameters, and operational constraints that preserve the original program structure while introducing behaviorally motivated revisions.
3. **Coding Module (C).** Conditioned on t' and the FlipPy library, C implements a program synthesis step that maps the updated strategy specification to an executable program p'_t.
4. **Audit Module (A).** The audit module A performs verification over p'_t under the FlipPy runtime environment. It checks for syntactic and runtime correctness. The output is a validated program $\hat{p}_t$. In case of failure, A applies a bounded repair procedure using execution error signals and re-verification.
5. **Inference Module (I).** The inference module evaluates $\hat{p}_t$ on participant data d and computes the likelihood $\Pr(d \mid \hat{p}_t)$ under the revised strategy model.

4.3 Generation, Selection, and Validation of Revised Strategies

We ran the full system twice per participant per base strategy model, producing two independently generated revised programs for each participant-base strategy pair. Programs produced by the Coding Module (C) were passed through the Audit Module (A), which allowed up to five iterative repair attempts to resolve syntactic or runtime errors. Programs that remained non-executable were excluded from subsequent analyses. The resulting audited programs constitute the candidate revised-strategy set used in subsequent evaluation. Because different runs can produce different revisions of the same base strategy, we subsequently grouped candidate programs into broader classes of qualitatively similar revisions. This clustering and selection procedure was applied separately within each base-strategy family to obtain a compact set of representative revised strategy models for evaluation. After generating and selecting the representative revised strategy models, we evaluated whether they provided improved fit to participants' behavioral data. We first evaluated the representative programs on the original dataset and compared their fit with that of the corresponding base strategy models. We then conducted the same evaluation on a held-out set of trials that was not used during program

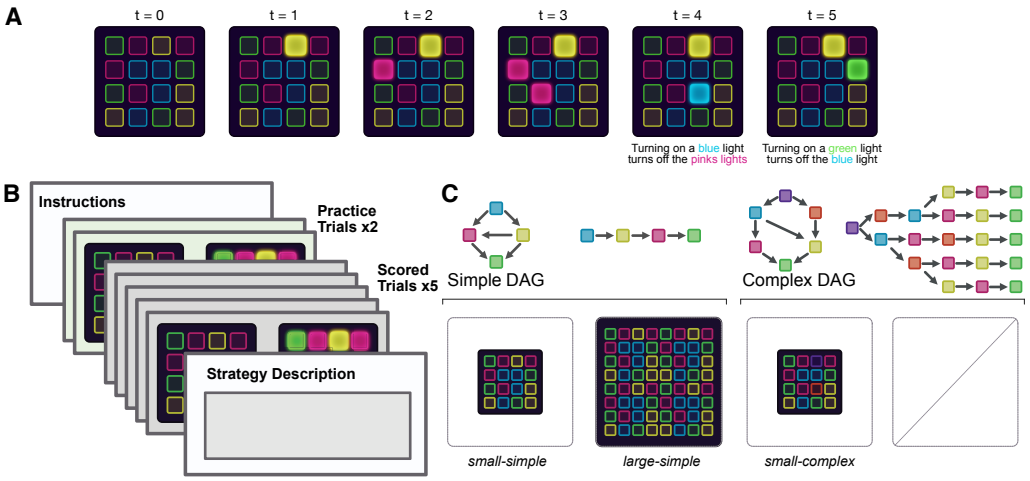

Figure 2: Behavioral experiment. (**A**) Task Structure: first five actions in an example trial. (**B**) Procedure: instructions, two practice trials, and five scored trials, followed by a strategy description. (**C**) Experimental conditions defined by board size (small vs. large) and task structure (simple vs. complex DAG), yielding three conditions. Representative grids and DAGs for each condition, along with a set of working orderings.

generation or selection, allowing us to assess whether improvements in fit generalized beyond the trajectories used to develop the revised strategies.

4.4 Human problem-solving algorithms

We applied our pipeline to human behavior in a problem-solving paradigm that examines how people develop algorithmic strategies to solve structured tasks (Yang et al., 2026). The dataset contains trial-level data from 150 participants across multiple task environments, providing an ideal testbed because participants developed a variety of strategies, producing heterogeneous behavior. Some participants' behavior is well captured by existing cognitive models of varying structural complexity, whereas other participants exhibit behavior that is not adequately captured by these models, providing an opportunity for our pipeline to discover more accurate algorithms.

4.4.1 Task Structure

Participants completed a problem-solving task involving a grid of colored lights (Fig. 2A). The goal is simple: turn on all the lights. Each light can be toggled by clicking. Clicking a light could, however, turn off other lights that were already on. The task was defined by a set of colors and their dependency relations, where a dependency from color i to color j meant that clicking any light of color i turned off all currently active lights of color j. These dependencies formed a directed acyclic graph (DAG) – e.g. blues turn off pinks, greens turn off blues. Grid size, number of colors, and the DAG structure varied across conditions. Participants were not told the underlying dependencies and had to discover an effective way to solve the task. See (Yang et al., 2026) for further detail.

Under these dynamics, the task can be solved by activating colors in an order consistent with the DAG. Specifically, any *topological sort* of the DAG, a permutation of its colors in which every source color precedes the colors it turns off, defines a valid color ordering. For example, if blue and yellow both turn off green, either blue–yellow–green or yellow–blue–green is a valid ordering. Participants therefore did not necessarily need to recover the full DAG; identifying a valid ordering was sufficient to solve the task efficiently. In the studies reported here, participants are not informed of the DAG's structure: they must find a way to solve the problem (turn on all the lights) efficiently, which could involve trying to establish the structure and using that knowledge to complete the task, or simply turning on lights at random, or any one of innumerable algorithms between these extremes; our goal as modelers is to characterize the structure of the algorithms developed by participants.

4.5 Base Strategy Models

We begin with the 3 strategy models developed in the original study to characterize participants' behavior in this task (Yang et al., 2026). These models were specified *a priori* to capture qualitatively distinct ways of approaching the task, ranging from surface-level spatial heuristics to strategies that infer and exploit the task's latent structure. In the present work, these models serve as the starting points for program refinement. Each model has a fixed program structure with latent parameters inferred during model fitting, and we implemented them using the probabilistic programming library FlipPy (Ho and Correa, 2026).

Spatial (SP) Lights are turned on using simple spatial patterns: e.g. row-by-row, column-by-column, or back-and-forth zigzag sequences. Pattern and lapse rate are latent parameters inferred during model fitting. Does not consider color or turn-off relations. This strategy reflects a failure to recognize or utilize the relevance of color-based structure altogether, instead relying on generic spatial heuristics that would be effective in tasks without latent dependencies.

Color Blocking (CB) Systematically turns on all lights of one color before moving to the next, without considering dependencies between colors. The color order and a probabilistic lapse rate are treated as latent parameters inferred during model fitting. This strategy reflects a partial discovery of task structure: participants recognize that color is a relevant organizing feature and exploit this regularity, but do not infer or utilize the underlying dependency relationships between colors.

Hierarchical Two-Phase Filling (TP) This model represents a hierarchical strategy in which a valid topological ordering is first inferred and then used to complete the board. In phase one, the model explores lights across colors, only testing lights not yet activated, thereby accumulating constraints on the set of possible orderings. In phase two, it commits to a single ordering drawn from the set of topological orders still consistent with the observed turn-offs. For example, if a yellow light is observed to turn off a green light, then any ordering placing yellow before green is eliminated from the feasible set. As more such constraints are observed in phase one, the set of admissible orderings becomes increasingly restricted, making correct orderings more likely. This strategy captures participants who infer color dependencies and attempt to exploit them in a heuristic, potentially imperfect manner. Two latent parameters are inferred: the transition point between phases and the phase-two ordering used to complete the board.

5 Results

5.1 Baseline Model Fit and Benchmark Quality

To establish a benchmark for evaluating refined programs, we first assessed fit for three base models. We computed data likelihood for each model on the final correctly completed trial for each participant. Participants who did not complete any trial correctly were excluded, leaving 49 trials in *small-complex*, 51 trials in *small-simple*, and 41 trials in *large-simple*. Tab. 2 (Appendix B) summarizes trial-level model fit using per-step log marginal likelihood. The baseline models produced numerically stable and broadly computable fits (finite-fit coverage: 95–100% across conditions), with relatively tight bootstrap confidence intervals around condition-level means and medians. Fig. 3A shows the distribution of best-fitting base-strategy log ML across conditions. The results reveal meaningful variation in fit across participants: some are well explained by the base strategies, while others are less well captured, leaving room for improvement through model refinement.

5.2 Discovery of Revised Strategies

We first generated candidate revisions by running the pipeline twice for each participant and each of the 3 base strategy models. This produced 846 candidate revised programs in total. The Audit Module validated 733 programs (86.6%); we excluded the rest because they remained non-executable after the allowed repair attempts. Within each base-strategy family, we clustered audited revisions according to the semantic similarity of their natural-language revision proposals. The largest clusters captured recurring types of modifications to the corresponding base strategies (Table 1).

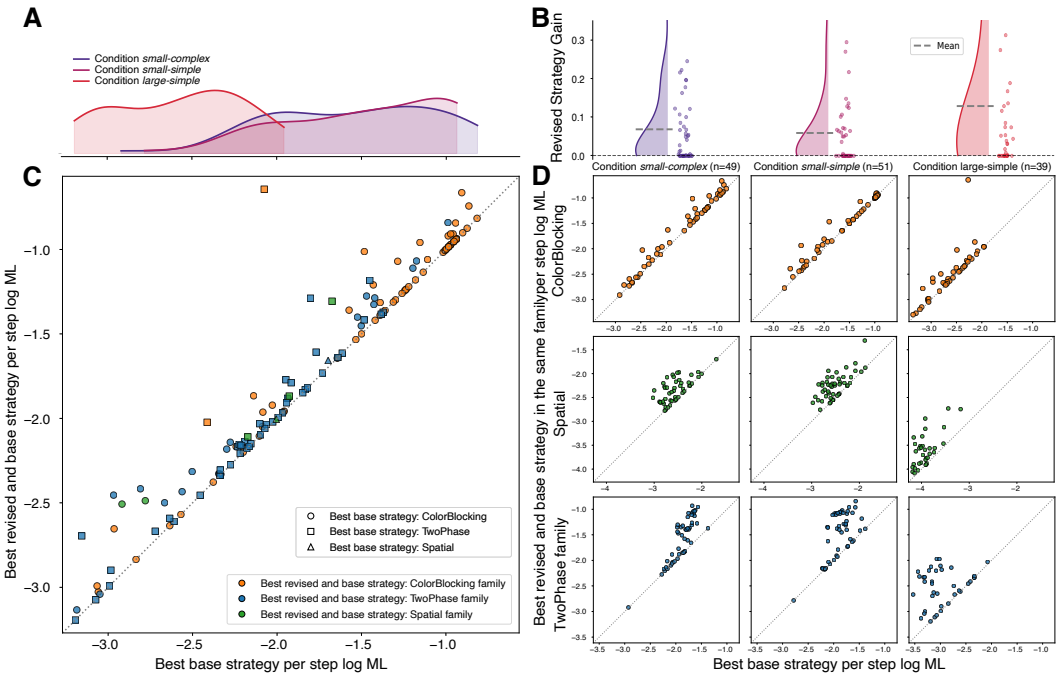

Figure 3: Base strategies provide adequate baseline fit, and revised strategies yield improved model fit for a substantial subset of participants. (**A**) Kernel density estimates of per-step log marginal likelihood (log ML) for the best-fitting base strategy, shown by condition (x-axis shared with panel C). (**B**) Improvement from revised strategies by condition, shown as per-step log ML gain (gain = revised - base). Half-violins show the distribution, points show individual participants, and dashed lines indicate condition means. The y-axis is truncated to [0, 0.35] for clarity (9/139 participants have higher gain and exceed this range). (**C**) Participant-level comparison of best base vs. best overall (base or revised) fit; points above the diagonal indicate improvement. Color denotes the family of the best revised strategy; shape denotes the best base strategy. (**D**) Same comparison as (C), broken down by strategy family and condition, with the y-axis showing the best revised variant within each family.

Across the 3 base strategy families, we identified 8 recurring revision clusters: 3 within Color Blocking, 2 within Spatial, and 3 within Two-Phase. Cluster size varied substantially, with some revisions recurring across many participants and others appearing more rarely. We selected 3 representative audited programs closest to the centroid of each cluster, yielding 24 prototypical revised models for the subsequent analyses.

5.3 Improvement in Model Fit from Revised Models

Fig. 3 B reports the improvement in per-step log marginal likelihood (log ML) for each participant. The improvement is computed as the difference between the per-step log ML of the best-fitting strategy (selected over both base and revised strategies) and the per-step log ML of the best-fitting base strategy alone. For a substantial subset of participants 53.2% (74/139), the inclusion of revised strategies yields meaningful improvements in fit. This indicates that, for these participants, the best-explaining model is not contained within the original base strategy set, but instead lies within the space of revised programs.

Fig. 3C,D further characterize these gains. Panel C shows that for 53.2% (74/139) of participants, a revised strategy provides the overall best fit. Panel D complements this analysis by evaluating improvements within each strategy family, rather than only the best-fitting strategy. As a result, improvements are observed across all families (ColorBlocking: 54%; Spatial: 76.9%; TwoPhase: 80.4%), including cases where a revised variant improves the fit of a non-winning base model without changing the participant's overall best-fitting model. Together, these results demonstrate that the

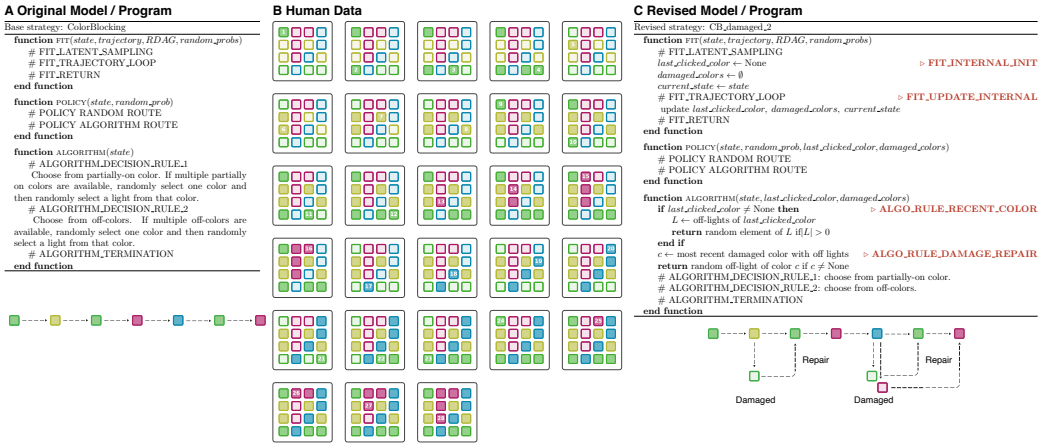

Figure 4: A representative case of a participant who is reasonably well fit by the base ColorBlocking strategy but exhibits improved fit under a revised variant. (**A**) Core logic of the base CB strategy, which predicts the behavior to complete one color at a time. (**B**) Participant behavior data. (**C**) Core logic of the revised strategy variant `CB_damaged_2`. Annotations highlight key modifications to the base strategy. Modifications capture the participant's behavior to repair previously damaged colors.

program refinement system reliably generates structured variants of base models that better account for observed behavior across participants.

Fig. 4 illustrates a representative case. This participant is reasonably well fit by the base ColorBlocking (CB) algorithm, but exhibits improved fit under a revised variant, `CB_damaged_2`. While the base CB algorithm captures the participant's tendency to complete one color at a time, it fails to account for behavior following a *damage event* (i.e. a turn-off event, where previously-on lights are turned off by an action that violates the DAG). Specifically, after observing that activating a yellow light turns off the greens, the participant completes the yellow lights and then immediately returns to repair the previously 'damaged' green lights. The revised strategy `CB_damaged_2` captures this damage-aware repair behavior, resulting in improved fit for this participant by proposing a meaningful qualitative innovation to the base algorithm.

5.4 Generalization and Synthetic Recovery

To assess whether these improvements generalize beyond the data used for model refinement, we additionally evaluated the 24 prototypical revised strategy programs on the held-out dataset. In the held-out data, 52.6% (72/137) of trials showed improved fit under a revised program. Thus, the improvement observed during refinement was not restricted to the data used to generate and select revisions. The persistence of these improvements on held-out data suggests that the revised programs capture behavioral regularities that extend beyond the trajectories on which they were developed. Details on held-out analyses are provided in Appendix E.

As a complementary synthetic recovery test, we generated synthetic behavioral trajectories from the `CB_damaged_2` strategy and applied our pipeline to individual trajectories. Of the 215 valid revised programs generated by our pipeline, 111 (51.6%) achieved a higher mean held-out log likelihood across 40 held-out trials than the base Color Blocking program, while 18 (8.4%) recovered at least 75% of the predictive gap between the two models. 104 (48.4%) did not achieve an improvement in mean held-out log likelihood over the base model over held-out trials. The variability in recovery may in part reflect that the pipeline sometimes produces revised programs that overfit to idiosyncratic features of the single behavioral trajectory used to generate each revision, rather than recovering the underlying strategy. Overall, these results provide evidence that the procedure can recover predictive consequences of a richer strategy, while also illustrating the limitations of trajectory-level strategy modeling. Further synthetic recovery analyses are provided in Appendix F.

Table 1: Top three revision clusters per strategy family, ranked by size (# revisions within the cluster). Cluster descriptions are LLM-generated for interpretation only and are not used in modeling or analysis. Full results are in Appendix C.

Strategy Family	Size	LLM-generated cluster description
ColorBlocking	161	Prioritize recent-damage repair before resuming default strategy
ColorBlocking	44	Damage-aware color-blocking with flexible partial completion
ColorBlocking	8	Within-color tie-breaking with consistent index order
Spatial	8	Spatial scanning with temporary damage-repair mode
Spatial	3	Probabilistic local repair and repeat after events
TwoPhase	106	Two-phase damage-aware repair and priority policy
TwoPhase	74	Short-term repair prioritization after observed damage
TwoPhase	29	Stateful color-focused exploration with immediate repair

5.5 Revised models reveal a small set of recurring algorithmic innovations

To better understand the structure of the revisions produced by the system, we analyze their organization in semantic space. Clustering identifies recurring themes and yields a compact set of representative variants, Tab. 1 reveals two robust regularities in the structure of revisions. First, revisions are highly clustered in semantic space: across 462 total revision proposals, behavior concentrates into a small number of dominant groups (6 clusters for ColorBlocking, 3 for Spatial, and 6 for TwoPhase), with a small subset of clusters accounting for the majority of revisions within each strategy family. Second, the dominant clusters are strikingly consistent across strategy families and reflect an unanticipated feature: *locally grounded behavioral adjustments*. Themes related to damage repair, local corrective actions, and deterministic tie-breaking appear as the largest cluster(s) in all three strategy families. These account for approximately 73% of ColorBlocking revisions (161/222), 67% of Spatial revisions (8/12), and 79% of TwoPhase revisions (180/228).

6 Limitations and future work

Our analysis focused on a single problem-solving paradigm, using just a single iteration of the propose-revise loop and a single underlying LLM. This *narrow and deep* approach revealed important new findings relative to *broad and shallow* approaches, but it is clearly important to explore whether other algorithmically-structured problem settings reveal further insight and to explore potential gains from alternative models. In future work, we envisage extending our approach to a larger multi-agent process or other canonical sampling algorithms such as MCMC or SMC procedures, and further exploring aspects of the harness that could support improved exploration in program space. Perhaps the most important issue not addressed here or in other work concerns the potential for program revisions to over-fit individual participants, trials, or tasks: methods for effective partial pooling and program compression across contexts are a clear priority for future work. Similarly, no current approach (including ours) offers a systematic way to assess how discovered programs relate to broader theoretical frameworks beyond lower-level procedural constraints; improved data-likelihood alone is an insufficient indicator of meaningful meaningful innovation, which here we assessed only heuristically–this broader question will be central as the area advances.

7 Conclusion

We proposed a program refinement framework for cognitive modeling that combines structured probabilistic programs with an LLM-based program revision system. We applied this system to the more challenging domain of algorithmically-structured sequential decision-making models. By enabling targeted revisions while preserving hypothesized structure and interpretability, our framework offers one principled way to integrate the increasing capabilities of frontier LLMs with (1) human expertise and (2) probabilistic statistical inference, and shows that these methods successfully extend to recovery of algorithmic models, a key frontier in cognitive science.

AI USE STATEMENT

In this work, we used generative AI tools as part of the proposed cognitive model refinement pipeline to generate and revise cognitive models as probabilistic programs, and to produce natural-language descriptions of clusters of revised programs for interpretive analysis. We have not used generative AI tools to generate or alter the human behavioral data, perform statistical analyses, or determine the reported experimental results.

We have reviewed all AI-assisted work, and we take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The experimental protocol was reviewed and approved by the institutional review board (IRB) at our institution, and all participants provided informed consent prior to participation. Participants received a base payment of $3 and a bonus of $0.4 per correctly completed trial.

REPRODUCIBILITY STATEMENT

All code, data, and experimental materials associated with this work are available in the supplementary materials and in the accompanying anonymous OSF repository. The main paper and appendix provide detailed descriptions of the model refinement pipeline, base and revised strategy programs, model-fitting and inference procedures, and held-out evaluation. The appendix further provides complete descriptions of the revision clusters and additional analyses. The OSF repository contains the source code, experimental data, model programs, and materials necessary to reproduce the analyses and results reported in this work.

A ZIP archive containing all code, data, and experimental materials associated with this work is included with the supplementary materials, and is available in the accompanying anonymous OSF repository: `https://osf.io/dcj54/overview?view_only=f8ef9ed4b6a84510b89a5a354118f753`.

REFERENCES

Mayank Agrawal, Joshua C Peterson, and Thomas L Griffiths. Scaling up psychology via scientific regret minimization. *Proceedings of the National Academy of Sciences*, 117(16):8825–8835, 2020.

Kelsey R Allen, Kevin A Smith, and Joshua B Tenenbaum. Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. *Proceedings of the National Academy of Sciences*, 117(47):29302–29310, 2020.

David Andre, Stuart J Russell, et al. State abstraction for programmable reinforcement learning agents. In *Aaai/iaai*, pages 119–125, 2002.

Marcel Binz, Elif Akata, Matthias Bethge, Franziska Brändle, Fred Callaway, Julian Coda-Forno, Peter Dayan, Can Demircan, Maria K Eckstein, Noémi Éltető, et al. A foundation model to predict and capture human cognition. *Nature*, 644(8078):1002–1009, 2025.

David D Bourgin, Joshua C Peterson, Daniel Reichman, Stuart J Russell, and Thomas L Griffiths. Cognitive model priors for predicting human decisions. In *International conference on machine learning*, pages 5133–5141. PMLR, 2019.

Jeffrey S Bowers, Guillermo Puebla, Sushrut Thorat, Konstantinos Tsetsos, and Casimir JH Ludwig. Centaur: A model without a theory. *OSF*, 2025.

Erik Brockbank and Edward Vul. Repeated rock, paper, scissors play reveals limits in adaptive sequential behavior. *Cognitive Psychology*, 151:101654, 2024.

CŽdric Colas, Tracey Mills, Ben Prystawski, Michael Henry Tessler, Noah Goodman, Jacob Andreas, and Joshua B Tenenbaum. Language and experience: A computational model of social learning in complex novel tasks. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 47, 2025.

Aidan Curtis, Hao Tang, Thiago Veloso, Kevin Ellis, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Llm-guided probabilistic program induction for pomdp model estimation. In *Conference on Robot Learning*, pages 3137–3184. PMLR, 2025.

Kevin Ellis. Human-like Few-Shot Learning via Bayesian Reasoning over Natural Language, September 2023. URL http://arxiv.org/abs/2306.02797. arXiv:2306.02797 [cs].

Kevin Ellis, Adam Albright, Armando Solar-Lezama, Joshua B. Tenenbaum, and Timothy J. O'Donnell. Synthesizing theories of human language with Bayesian program induction. *Nature Communications*, 13(1):5024, August 2022. ISSN 2041-1723. doi: 10.1038/s41467-022-32012-w. URL https://www.nature.com/articles/s41467-022-32012-w.

Kevin Ellis, Lionel Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lore Anaya Pozo, Luke Hewitt, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: growing generalizable, interpretable knowledge with wake–sleep bayesian program learning. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 381(2251), 2023.

Simon Farrell and Stephan Lewandowsky. *Computational modeling of cognition and behavior*. Cambridge University Press, 2018.

Noah Goodman, Vikash Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models, 2014. URL https://arxiv.org/abs/1206.3255.

Noah D Goodman and Andreas Stuhlmüller. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org, 2014. Accessed: 2026-5-6.

Thomas L. Griffiths, Nick Chater, and Joshua B. Tenenbaum. *Bayesian Models of Cognition: Reverse Engineering the Mind*. MIT Press, November 2024. ISBN 978-0-262-38104-8. Google-Books-ID: FEb6EAAAQBAJ.

Mark K. Ho and Carlos G. Correa. FlipPy: Pythonic probabilistic programming. https://github.com/codec-lab/flippy, 2026. GitHub repository.

Akshay K. Jagadish, Milena Rmus, Kristin Witte, Marvin Mathony, Marcel Binz, and Eric Schulz. Can we automatize scientific discovery in the cognitive sciences?, March 2026a. URL http://arxiv.org/abs/2603.20988. arXiv:2603.20988 [cs].

Akshay K Jagadish, Younes Strittmatter, Nori Jacoby, George Kachergis, Eric Schulz, Nathaniel Daw, Suyog H Chandramouli, and Thomas L Griffiths. Closing the loop to discover psychological theories with an automated cognitive scientist. *arXiv preprint arXiv:2606.26448*, 2026b.

Kunal Jha, Aydan Yuenan Huang, Eric Ye, Natasha Jaques, and Max Kleiman-Weiner. Modeling others' minds as code. *arXiv preprint arXiv:2510.01272*, 2025.

Li Ji-An, Marcus K Benna, and Marcelo G Mattar. Discovering cognitive strategies with tiny recurrent neural networks. *Nature*, 644(8078):993–1001, 2025.

Guangyuan Jiang, Matthias Hofer, Jiayuan Mao, Lionel Wong, Joshua B Tenenbaum, and Roger P Levy. Finding structure in logographic writing with library learning. *arXiv preprint arXiv:2405.06906*, 2024.

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

Timothy J O'Donnell, Joshua B Tenenbaum, and Noah D Goodman. Fragment grammars: Exploring computation and reuse in language. 2009.

Joshua C Peterson, David D Bourgin, Mayank Agrawal, Daniel Reichman, and Thomas L Griffiths. Using large-scale experiments and machine learning to discover theories of human decision-making. *Science*, 372(6547):1209–1214, 2021.

Steven T Piantadosi, Joshua B Tenenbaum, and Noah D Goodman. Bootstrapping in a language of thought: A formal model of numerical concept learning. *Cognition*, 123(2):199–217, 2012.

Steven T Piantadosi, Joshua B Tenenbaum, and Noah D Goodman. The logical primitives of thought: Empirical foundations for compositional cognitive models. *Psychological review*, 123(4):392, 2016.

Wasu Top Piriyakulkij, Cassidy Langenfeld, Tuan Anh Le, and Kevin Ellis. Doing Experiments and Revising Rules with Natural Language and Probabilistic Reasoning, October 2024. URL http://arxiv.org/abs/2402.06025. arXiv:2402.06025 [cs].

Julien Pourcel, Cédric Colas, and Pierre-Yves Oudeyer. Self-improving language models for evolutionary program synthesis: A case study on arc-agi. *arXiv preprint arXiv:2507.14172*, 2025.

Ben Prystawski, Kushin Mukherjee, Daniel Wurgaft, Linas Nasvytis, Michael Y Li, Noah D Goodman, and Michael C Frank. auto-psych: Automating the science of mind using agent-driven theory discovery and experimentation. *arXiv preprint arXiv:2606.26460*, 2026.

Milena Rmus, Akshay K Jagadish, Marvin Mathony, Tobias Ludwig, and Eric Schulz. Generating computational cognitive models using large language models. *arXiv preprint arXiv:2502.00879*, 2025.

Milena Rmus, Akshay Kumar Jagadish, Marvin Mathony, Tobias Ludwig, and Eric Schulz. Generating computational cognitive models using large language models. *Advances in Neural Information Processing Systems*, 38:87796–87833, 2026.

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.

Robert S. Siegler and Jeff Shrager. A model of strategy choice. In Catherine Sophian, editor, *Origins of Cognitive Skills*, pages 229–293. Lawrence Erlbaum Associates, Hillsdale, NJ, 1984.

Ziyi Tang, Xuexiong Yin, Weixing Chen, Zechuan Chen, Yongsen Zheng, Wenxuan Ye, Keze Wang, and Liang Lin. Alphaagentevo: Evolution-oriented alpha mining via self-evolving agentic reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Bill Thompson, B Van Opheusden, T Sumers, and TL Griffiths. Complex cognitive algorithms preserved by selective social learning in experimental populations. *Science*, 376(6588):95–98, 2022.

Lukas Twist, Jie M. Zhang, Mark Harman, Don Syme, Joost Noppen, and Detlef D. Nauck. Llms love python: A study of llms' bias for programming languages and libraries. *CoRR*, abs/2503.17181, 2025. doi: 10.48550/arXiv.2503.17181. URL https://arxiv.org/abs/2503.17181.

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming, 2021. URL https://arxiv.org/abs/1809.10756.

Bas Van Opheusden, Ionatan Kuperwajs, Gianni Galbiati, Zahy Bnaya, Yunqi Li, and Wei Ji Ma. Expertise increases planning depth in human gameplay. *Nature*, 618(7967):1000–1005, 2023.

Caroline Wang, Daniel Kasenberg, Kim Stachenfeld, and Pablo Samuel Castro. Discovering differences in strategic behavior between humans and llms. *arXiv preprint arXiv:2602.10324*, 2026.

Robert C Wilson and Anne GE Collins. Ten simple rules for the computational modeling of behavioral data. *elife*, 8:e49547, 2019.

Huiwen Alex Yang, Mark K Ho, and Bill D Thompson. Spontaneous meta-learning of efficient problem-solving algorithms. In *Proceedings of the 47th Annual Conference of the Cognitive Science Society*, 2026. Accepted for publication.

Yuan Yang and Steven T Piantadosi. One model for the learning of language. *Proceedings of the National Academy of Sciences*, 119(5):e2021865119, 2022.

Dhara Yu and Bill Thompson. People balance joint reward, fairness and complexity to develop social norms in a two-player game. In *Proceedings of the annual meeting of the cognitive science society*, volume 46, 2024.

A Participant Instructions

All participants received the same instructions prior to beginning the task, reproduced below:

> On each trial, you will play a game. You will see a grid of lights like the one below.
> Your goal is simple: turn on all the lights!
> Every square on the grid is a light. You can turn a light on or off by clicking on it.
> You will complete 2 practice trial(s). In these trials, every trial is the same game.
> After you complete the practice trials, you will complete 5 scored trial(s). In these trials, every trial will be a different game.
> You will get a bonus payment for every trial you complete ($bonus per trial).
> If you really cannot complete a trial, you can skip it (but you will not receive the bonus for that trial).
> Good luck!

B Base strategy model fit summary

Table 2 summarizes the performance of the best-fitting base strategy model across conditions using per-step log-likelihoods. Overall, the base models produce stable and well-defined fits, with high finite-fit coverage (95–100%) across all conditions, indicating that inference is numerically reliable for the vast majority of trials. The relatively tight bootstrap confidence intervals further suggest that condition-level estimates are well-constrained. There are also systematic differences across conditions: Conditions *small-complex* and *small-simple* exhibit similar fit quality, whereas Condition *large-simple* shows substantially more negative per-step log-likelihood. This pattern is consistent with the increased trajectory length in Condition *large-simple*, reflecting the increased task scale and associated demands. Taken together, these results indicate that the base models provide a stable but imperfect account of behavior, establishing a meaningful baseline against which improvements from revised strategies can be evaluated.

Table 2: Descriptive summary of best-fitting base model performance by condition. Values reported here are per-step log-likelihoods, computed as the log marginal likelihoods of the best-fitting base strategy model normalized by trajectory length. 95% confidence intervals are bootstrap percentile intervals ($B = 2000$).

Condition	N	Finite fits (%)	Mean [CI_{95}]	SD	Avg. traj. length
small-complex	49	100.0	-1.544 [-1.680, -1.414]	0.489	39.7
small-simple	51	100.0	-1.464 [-1.598, -1.335]	0.479	38.4
large-simple	41	95.1	-2.577 [-2.693, -2.465]	0.373	125.6

C Clusters of revisions

Table 3 provides a comprehensive summary of revision clusters across all strategy families. Several regularities emerge. First, revisions are highly concentrated: within each strategy family, a small number of dominant clusters account for the majority of proposals. For example, in the ColorBlocking family, the largest cluster (size 161) substantially outweighs all others, and a similar skew is observed in the TwoPhase family, where the top two clusters (sizes 106 and 74) dominate. In contrast, the Spatial family contains fewer and smaller clusters overall, reflecting both lower revision volume and a more compact set of variation patterns.

Second, the dominant clusters across families reflect closely related functional themes. In particular, revisions centered on damage detection and repair, as well as short-term prioritization of recently affected lights, appear as the largest clusters in all three families. Deterministic tie-breaking and ordering strategies also recur, though typically at smaller scales. This suggests that, despite differences in the underlying base strategies, revisions are organized around a shared set of locally grounded behavioral adjustments.

Table 3: Summary of revision clusters across strategy families. For each family, all clusters, ranked by size (number of revisions in the cluster), are shown. Descriptions are LLM-generated for interpretive purposes and are not used in the system or analysis.

Size	Strategy Family	LLM-generated cluster description
	ColorBlocking Strategy Family	
161	Prioritize recent-damage repair before resuming default strategy	After a turn-off/damage event, temporarily focus on repairing or avoiding the most recently damaged color(s) using a stable within-color click order, then resume the default color-blocking policy.
44	Damage-aware color-blocking with flexible partial completion	A damage-aware color-blocking strategy that maintains memory of recently damaged colors and flexibly switches between continuing a color block and repairing or reordering priorities.
8	Within-color tie-breaking with consistent index order	When selecting within a color, ties among same-color OFF lights are resolved using a deterministic spatial/index sweep instead of random selection.
4	Occasional post-completion or late-stage exploratory actions	Rare exploratory deviations from strict termination, such as additional clicks after completion or near-completion, to capture behavioral variability.
3	Selection stickiness — persist with chosen color until ON_COLOR	A persistence bias that keeps selecting the same color until it is fully ON, only switching when external state changes require it.
2	Optional exploratory lapse augmenting ColorBlocking policy	A low-probability lapse mechanism that occasionally overrides the main policy, producing rare exploratory or erroneous clicks.
	Spatial Strategy Family	
8	Spatial scanning with temporary damage-repair mode	Spatial scanning with a conditional repair mode that activates after damage events, prioritizing restoration before resuming scanning.
3	Probabilistic local repair and repeat after events	A stochastic mechanism that occasionally repeats or locally repairs recent actions after specific outcome events.
1	Structured-prior bias toward deterministic ordering	A strong bias toward deterministic spatial ordering and reduced randomness in action selection.
	TwoPhase Strategy Family	
106	Two-phase damage-aware repair and priority policy	A two-phase strategy that prioritizes repairing damaged colors before proceeding with the exploit phase.
74	Short-term repair prioritization after observed damage	Temporary prioritization of recently damaged colors before returning to the main policy.
29	Stateful color-focused exploration with immediate repair	Stateful exploration that maintains color focus while inserting immediate repair steps when damage is observed.
15	Deterministic spatial tie-breaking within color selection	Deterministic spatial ordering replaces random selection within color choices during execution.
2	Turn-off-triggered repair prioritization in exploitation	Exploitation phase is temporarily interrupted after damage events to prioritize repair.
2	Deterministic within-color spatial ordering strategy	A deterministic within-color ordering rule applied during the exploit phase.

D HETEROGENEITY OF BEST-FITTING REVISED STRATEGY IMPLEMENTATIONS

Table 4 shows how best-fitting revised strategies are distributed across strategy families, clusters, and specific variants. Several patterns emerge. First, even within a given cluster, multiple variants are selected across participants, indicating that no single implementation uniformly dominates. For example, within the TwoPhase "damaged" cluster, all 3 variants are preferred by some participants, and similar splits appear in ColorBlocking clusters. This provides direct evidence of within-cluster heterogeneity in how shared high-level revision ideas are instantiated. Second, while certain variants account for a larger fraction of participants (e.g., TP_repair_1 with 22/75), the overall distribution remains diffuse, with a substantial portion of participants assigned to lower-frequency variants. This long-tailed pattern indicates that model fit is distributed across many implementations rather than concentrated on a single dominant strategy. Finally, these patterns are consistent across strategy families. In all three families, clusters such as repair and damage handling contain multiple viable variants, reinforcing the idea that common functional theoretical revisions admit multiple realizations on a program level. Together, these results support the conclusion that, although revisions are organized around shared cluster-level principles, the specific implementations that best explain behavior vary systematically across participants.

Table 4: Number of participants best-fitted by each revised strategy, grouped by base strategy family, cluster type, and specific variant (N = 75).

Strategy Family	Cluster	Variant	Count	%
TwoPhase (TP)				
TP	repair	TP_repair_1	22	29.3%
		TP_repair_3	2	2.7%
TP	damaged	TP_damaged_2	9	12.0%
		TP_damaged_3	3	4.0%
		TP_damaged_1	3	4.0%
TP	local	TP_local_1	7	9.3%
ColorBlocking (CB)				
CB	damaged	CB_damaged_2	11	14.7%
		CB_damaged_1	3	4.0%
CB	repair	CB_repair_3	2	2.7%
		CB_repair_1	2	2.7%
CB	index	CB_index_3	5	6.7%
		CB_index_1	1	1.3%
Spatial (SP)				
SP	repair	SP_repair_3	2	2.7%
		SP_repair_1	2	2.7%
SP	damaged	SP_damaged_3	1	1.3%

E HELDOUT ANALYSES

E.1 HELDOUT DATASET

Each participant completed five scored trials in the experiment. In the model refinement process and analyses reported in the main text, we used only the final correctly completed trial from each participant. The held-out dataset consists of the second-to-last correctly completed trial for each participant. Participants who did not complete any trial correctly, or who had only one correctly completed trial, were therefore excluded from the held-out analysis. This resulted in 49 trials in the

small-complex condition, 50 trials in the *small-simple* condition, and 39 trials in the *large-simple* condition.

E.2 Held-out Evaluation

We evaluated the held-out trials using the same model-fitting procedure as in the main analysis. First, we fit each of the 138 held-out trials to the 3 base strategies and selected the base strategy with the highest per-step log marginal likelihood for each trial. We then fit the same trials to the 24 prototypical revised strategies identified from the 3 strategy families and selected the strategy with the highest per-step log marginal likelihood among all base and revised strategies.

One held-out trial could not be evaluated because all 27 strategies produced a log marginal likelihood of $-\infty$. This trial contained 270 behavioral steps, but we could not determine whether the non-finite likelihoods were attributable to sequence length or to another numerical or model-specific issue. We therefore excluded this trial from the comparison, leaving 137 evaluable held-out trials.

For each held-out trial, we quantified improvement as the difference between the per-step log marginal likelihood of the best-fitting strategy among the base and revised strategies and that of the best-fitting base strategy alone. A positive value therefore indicates that at least one revised strategy provides a better fit to the held-out trial than any of the three base strategies. Across the 137 held-out trials, 72 (52.6%) showed positive improvement under a revised strategy.

Similar to the main analysis, held-out improvements are observed across all three strategy families (ColorBlocking: 52.6%; Spatial: 86.7%; TwoPhase: 83.2%), including cases in which a revised variant improves fit relative to the corresponding base strategy without becoming the best-fitting strategy overall.

F Synthetic Analyses

F.1 Synthetic Dataset

To evaluate recovery in a setting where the underlying behavioral strategy is known, we generated a synthetic dataset from the `CB_damaged_2` strategy. This strategy variant was selected because the original behavioral study identified damage-repair behavior as a common extension of the ColorBlocking strategy, with participants often returning to repair colors that had been damaged by subsequent actions (Yang et al., 2026). Thus, `CB_damaged_2` provides a concrete example of a behavioral regularity observed in participants and allows us to ask whether the discovery pipeline can recover such a strategy when its generating process is known.

To make the synthetic dataset resemble the empirical data as closely as possible, we used 30 board configurations that participants encountered in the behavioral experiment and generated 5 independent synthetic behavioral trajectories for each configuration, yielding 150 trajectories in total. Within each board configuration, the 5 trajectories were generated with lapse probabilities of 0.0, 0.0, 0.1, 0.1, and 0.2, respectively. These values were chosen to approximate the distribution of lapse probabilities estimated when fitting the original ColorBlocking model to participants: most participants had low estimated lapse rates, and approximately half had an estimated lapse rate of 0.0. This design therefore preserves the empirical task configurations and approximates the level of stochasticity observed in the original behavioral data while allowing the underlying strategy to be controlled.

F.2 Generating Revised Programs

For each synthetic behavioral trajectory, we independently ran our program refinement pipeline twice, resulting in two candidate revised programs per synthetic trial. Of the 300 generated programs, 287 were validated by the Audit Module. Among those programs, 215 had the expected program interface and could be evaluated using the same likelihood-based inference procedure as the base CB model. The remaining programs were excluded because their generated interfaces required additional arguments or otherwise could not be evaluated under the standardized procedure. Thus, the synthetic analysis reports results for 215 refined programs.

F.3 SYNTHETIC ANALYSIS RESULTS

We evaluated each discovered program on 40 independently generated held-out trajectories from the same `CB_damaged_2` strategy and task configurations that were not included in the generation of revised programs. For each program, we compared its mean held-out log likelihood with those of the original ColorBlocking (CB) model and the generating `CB_damaged_2` model. Across the 215 evaluated programs, 111 (51.6%) achieved higher mean held-out log likelihood than CB, while 2 (0.9%) matched or exceeded the generating model. To quantify partial predictive recovery, we computed the fraction of the predictive gap between CB and CB-damaged recovered by each revised program:

$$R = \frac{\overline{\mathrm{LL}}\mathrm{revised} - \overline{\mathrm{LL}}\mathrm{CB}}{\overline{\mathrm{LL}}\mathrm{CB\text{-}damaged\text{-}2} - \overline{\mathrm{LL}}\mathrm{CB}}.$$

The median recovery was 0.334. Overall, 109 programs (50.7%) recovered a positive fraction of the predictive gap, 18 (8.4%) recovered at least 75%, and 2 (0.9%) matched or exceeded the generating model. These results indicate that the discovery procedure can recover predictive consequences of a richer strategy, but recovery is heterogeneous: because revisions are generated from individual trajectories, they can also overfit trajectory-specific features and fail to generalize even when the underlying data-generating strategy is known. Importantly, predictive recovery does not imply exact recovery of the generating computational mechanism; multiple distinct implementations of damage-repair behavior were produced by the discovery procedure.