A Language Model of Problem Solving in Humans and Macaque Monkeys

Qianli Yang[1*†], Zhihua Zhu[1†], Ruoguang Si[1,2], Yunwei Li[1,3], Jiaxiang Zhang[4,2], Tianming Yang ‡[1*]

[1] Institute of Neuroscience, Key Laboratory of Brain Cognition and Brain-inspired Intelligence Technology, Center for Excellence in Brain Science and Intelligence Technology, Chinese Academy of Sciences, Shanghai, China.

[2] Cardiff University Brain Research Imaging Centre, School of Psychology, Cardiff University, Maindy Road, Cardiff, CF24 4HQ, UK.

[3] University of Chinese Academy of Sciences, Beijing, China.

[4] School of Mathematics and Computer Science, Swansea University, Swansea, SA1 8DD, UK.

*Corresponding author(s). E-mail(s): qlyang@ion.ac.cn; tyang@ion.ac.cn;

†Co-first author(s).

‡Lead Contact

1 Summary

Human intelligence is characterized by the remarkable ability to solve complex problems by planning a sequence of actions that take us from an initial state to a desired goal state. Quantifying and comparing problem-solving capabilities across species and finding their evolutionary roots are critical for understanding how the brain carries out this intricate process. We introduce the Language of Problem-Solving (LoPS) model as a novel quantitative framework that investigates the structure of problem-solving behavior through a language model. We applied the model to an adapted classic Pac-Man game as a cross-species behavioral paradigm to test both humans and macaque monkeys. The LoPS model extracted the latent structure, or grammar, embedded in the agents' gameplay, revealing the non-Markovian temporal dependency structure of their problem-solving behavior and the hierarchical structures of problem-solving in both species. The complexity of LoPS grammar correlated with individuals' game performance and reflected the difference in problem-solving capacity between humans and monkeys. Both species evolved their LoPS grammars during learning, progressing from simpler to more complex ones, suggesting that the structure of problem-solving is not fixed but evolves to support more sophisticated and efficient problem-solving. Our study provides insights into how humans and monkeys break down problem-solving into compositional units and navigate complex tasks, deepening our understanding of human intelligence and its evolution, and establishing a foundation for future investigations of the neural mechanisms of problem-solving.

2 Introduction

Advanced problem-solving is a hallmark of human intelligence, enabling us to navigate complex tasks to reap greater rewards and avoid risks more effectively[1]. It has its roots in our primate ancestry and is likely shared by living nonhuman primates to varying degrees. Although many studies have investigated problem-solving capabilities in humans[2–5], quantifying and comparing these capabilities and their learning process across species remain a significant challenge. This is critical for us to understand the evolution of human intelligence and the underlying neural circuitry.

Problem-solving can be conceptualized as the process of establishing a sequence of operators or actions that link an initial state to a desired goal state. We hypothesize that this process, at its core, involves

a systematic and structured process akin to a language. This language encapsulates the rules and principles governing the composition and abstraction laws that construct the solution sequence[6–8] and guide our problem-solving efforts. The language may evolve to be more complex to support more sophisticated and more efficient problem-solving, both during evolution at the species level and during learning at the individual level.

To study this language of problem-solving quantitatively, we face two challenges. First, we need appropriate cross-species behavioral paradigms that can elicit problem-solving behavior with rich structures[9–11]. Second, we must develop methods to extract the structure of problem-solving from agents' behavior without relying on self-reports. This is critical, as subjective reports are impossible in animals and may be inconsistent with actual problem-solving processes in human subjects[12,13].

To address these challenges, we adapted the classic Pac-Man game as a cross-species behavior paradigm to test both humans and macaque monkeys. We developed a framework named the Language of Problem Solving (LoPS), using a grammar induction algorithm to extract the underlying grammars — the non-Markovian temporal dependency structure[14] — from the agents' gameplay. The complexity of the resultant structures offers a quantifiable and interpretable metric of players' problem-solving capacity[15].

Our results reveal notable differences in the complexity and hierarchical organization of problem-solving behavior between humans and monkeys, as well as among individuals. Grammar complexity positively correlated with players' performance. Human players, especially the expert players, exhibited more complex LoPS grammars with deeper hierarchies, reflected in a larger and more interconnected game state space. Furthermore, both humans and monkeys demonstrated an evolution in problem-solving capabilities during learning, as their grammars progressed from simpler to more complex structures.

Through the lens of a language model, our study offers a structured and systematic framework for understanding the complex cognitive process underlying problem-solving. It reveals how both humans and monkeys break down complex decision-making into compositional units, providing insights into how they navigate problem-solving[16–18]. This deepens our understanding of human intelligence and its evolution. Moreover, the quantitative nature of our study lays the foundation for future investigations of the neural mechanism of problem-solving.

3 Results

3.1 Pac-Man task

We adapted the classic Pac-Man game for a cross-species study involving humans and monkeys. The game is sufficiently complex to elicit rich and varied strategies, yet its core concepts – foraging, hunting, and escape – are intuitive even for monkeys to grasp.

In this game, players guide Pac-Man in a maze to eat pellets while avoiding and, at times, hunting ghosts. Two ghosts are programmed into the game. If caught by a ghost, players receive a time-out penalty, after which all characters are reset to their starting locations. However, Pac-Man may eat a special pellet, called an energizer, to temporarily turn the ghosts into a *scared* mode, allowing Pac-Man to eat them for extra rewards. Each game begins with randomly placed pellets and energizers. Players complete a game by clearing all the pellets in the maze.

We tested humans and monkeys using comparable game settings (See Methods 5.4.1), but with two notable differences. First, monkeys received real-time juice rewards corresponding to their in-game points, while humans were only shown a real-time cumulative tally of their current game points on the screen. Second, the game speed for the human version was set to be twice as fast as that for monkeys. These differences did not change the basic game mechanism but were necessary to maintain motivation and engagement for each species.

Throughout the game, we recorded all the game states and players' actions. Further details are available in Methods and our previous work[19]. In total, we collected behavioral data from a total of 34 humans and 2 monkeys.

3.2 Language of Problem-Solving

The Pac-Man game can be played at different levels. A player can react to immediate game states without doing any advanced planning. This requires either sufficient computational power to process all game information rapidly for a good performance or a compromise in decision quality due to limited processing capacity. Alternatively, with plenty of experience and a good understanding of the game, the player can plan their moves in advance. In this scenario, the player lays out a sequence of actions based on the anticipated game developments, thereby reducing the cognitive load of real-time decision-making. This scenario implies dependencies among the elements in the behavior sequence to play the game. Drawing on dependency grammar theory from linguistics[20], we employed a language model to investigate whether such

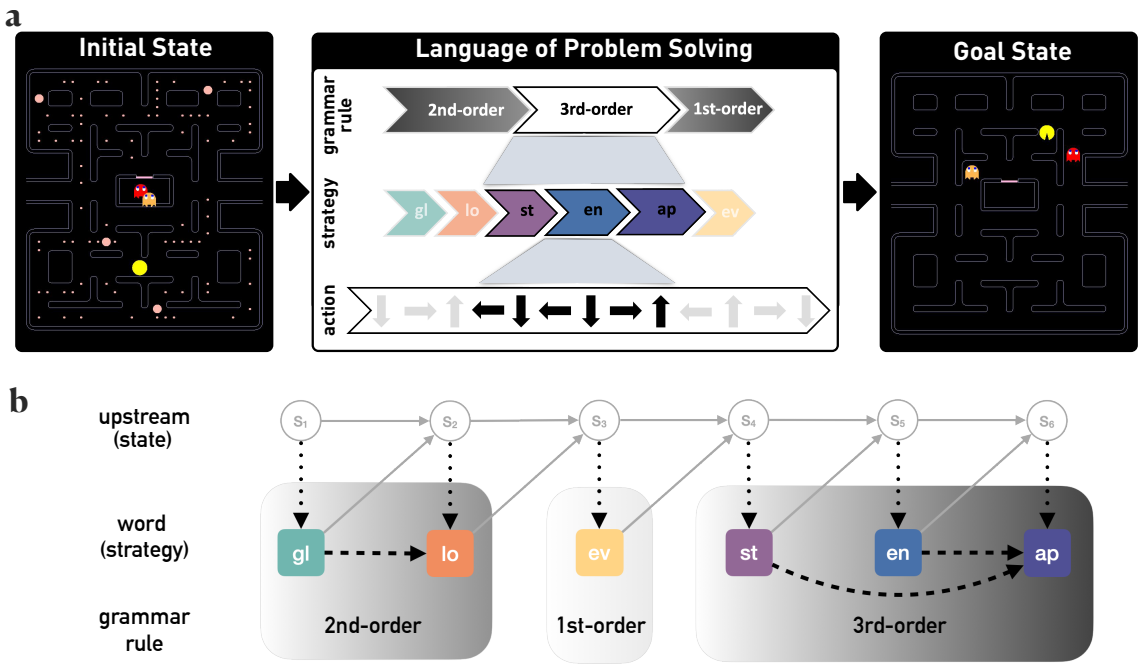

Figure 1: Language of Problem-Solving model. **(a)** The LoPS model conceptualizes problem-solving as a hierarchical structure of sequences that transform an initial state into a desired goal state. At the top of this hierarchy is a sequence of grammar rules, which describes game plans and governs how words (strategies) are concatenated. At the bottom level, these strategies are implemented by actual actions of joystick movements. Strategy abbreviations are lo: *local*, gl: *global*, ev: *evade*, st: *stay*, ap: *approach*, en: *energizer*. Detailed definitions of strategies can be found in Supplementary Table S4 and in our previous work[19]. **(b)** The LoPS is described by a probabilistic graphical model, where nodes represent variables and edges denote temporal dependencies between them. Upstream game states evolve according to the game mechanics and the strategies applied (grey arrows). Strong dependencies exist between strategies and game states (dotted arrows), whereas LoPS grammar rules describe temporal dependencies between strategies (dashed arrows) in addition to the upstream game states.

dependency structures were present in the player's problem-solving behavior and analyzed them for insights into the problem-solving process.

To achieve this goal, we first need to determine at which level we should build the language model. Although the Pac-Man game is eventually solved by a sequence of actions (e.g., '↑←↓→' clears a square maze counterclockwise), the basic components of the language model should be at a higher level of abstraction. Motor actions depend too much on specific game details, such as maze types and Pac-Man's exact location. Cognitively similar solutions may be associated with a wide variety of action sequences. Building the mode at the level of motor actions prevents us from understanding the cognitive principles of players' problem-solving.

Our previous work[19] offers a better alternative. In that study, we demonstrate that a set of heuristic strategies may explain monkeys' behavior in the Pac-Man game. Each strategy focuses on a subset of the game elements and a corresponding sub-goal, and the monkeys employed Take-The-Best heuristics, dynamically choosing a dominant strategy at any moment. We repeated the same analyses for human players and found that their gameplay can be captured similarly (Supplementary Figure S1). In addition, we identified two new strategies unique to human gameplay: *stay* (keeping Pac-Man near a specific location) and *save* (keeping Pac-Man away from the energizer). These two strategies allowed the human players to better exploit energizers and hunt ghosts more efficiently. With these seven strategies (Supplementary Table S4), we were able to accurately predict joystick actions for both the human and monkey players, achieving an average prediction accuracy of $90.6 \pm 3.0\%$ for human players and $93.9 \pm 2.2\%$ for monkey players. Thereby, we converted both humans' and monkeys' gameplay into sequences of dominant strategies.

3.3 Grammar Induction

At the level strategies, we developed the Language of Problem-Solving (LoPS) framework. In LoPS, the 'words' are strategies, while the 'grammar' consists of a set of dependency rules dictating how strategies are concatenated into problem-solving sequences (Figure 1a). A key aspect of LoPS is that its words, i.e., the strategies, depend on an upstream variable, the game states, which themselves have a temporal structure

(Figure 1b). The null hypothesis is that the strategy sequences exhibit no additional structures beyond what can be expected from game states alone, implying no higher-level planning beyond individual strategies.

At the core of the LoPS framework is a grammar induction algorithm that extracts the set of dependency grammar rules from an agent's behavior sequences. The algorithm begins with a set of basis strategies and computes all concatenations of the strategies that show significant dependency in the agent's gameplay, which measures whether the co-occurrence of two strategies can be attributed solely to the game states. Each valid concatenation forms a dependency rule. A 1st-order rule indicates that the strategy depends only on the current game states, whereas a 2nd-order rule signifies that the strategy depends not only on the game states but also on the previous strategy. This process continues recursively for higher-order rules. We also allow for the possibility of "skip-order" dependencies, where one or more strategies may intervene between two linked strategies. The procedure ends when no further significant dependencies are found. The result is a set of dependency rules of varying orders, which together constitute the agent's LoPS grammar. When applied to synthetic data, our grammar induction algorithm successfully recovers the grammar used to generate the data (See Methods 5.4.3 and Algorithm 1).

3.4 Players exhibit different levels of grammar complexity

The induced LoPS grammar provides insights into individual players' problem-solving structures. Both monkeys developed several 2nd-order rules, such as *local-global* and *global-local*, suggesting a divide-and-conquer scheme, and *energizer-approach*, indicating planning ahead for ghost-hunting. *Evade* was not combined with any other strategies, suggesting its impromptu nature. Other than *evade*, the monkeys' grammar rules can be divided into two categories: pellet-collection rules and ghost-hunting rules, connected by a sole 3rd-order rule, *local-energizer-approach*, signifying the emergence of higher-order hierarchical structures (Figure 2a).

Human players exhibited more sophisticated hierarchical structures (Figure 2). Example player 1 added a *save* strategy, leading to the skip-order rule *save...energizer-approach*, a more strategic approach. However, the grammar remained relatively modest (Figure 2b, left). In contrast, example player 2 developed a significantly more sophisticated grammar with many 3rd and 4th-order rules, many specifically for ghost-hunting, such as *stay-energizer-approach*. In addition, a large number of high-order rules, such as *energizer-approach-local-global*, string together pellet-collection and ghost-hunting rules, demonstrating a high level of planning that combines the two most important aspects of the game(Figure 2b, right).

The two example human players illustrate the marked diversity in LoPS grammar among human players, ranging from relatively simple, monkey-like grammars to highly sophisticated ones. In Figure 2c, we plotted the usage ratios of rules of different orders for all human players in a 3D space. Agglomerative clustering reveals two distinct clusters. Example human player 1 belongs to the first cluster, characterized by a preference for simpler 1st-order and 2nd-order rules (1st-order: $55.0 \pm 3.8\%$, 2nd-order: $41.5 \pm 4.5\%$, order > 2: $3.5 \pm 1.9\%$), while the second cluster, including example human player 2, demonstrated a much higher usage ratio for higher-order rules (1st-order: $53.7 \pm 5.5\%$, 2nd-order: $26.2 \pm 5.2\%$, order > 2: $20.1 \pm 4.0\%$). We refer to the players in the two clusters as novice and expert players, respectively (see also Figure S2).

Interestingly, the two monkeys, when plotted in the same 3-D space (monkey O, 1st-order: $50.7 \pm 3.7\%$, 2nd-order: $44.1 \pm 3.5\%$, order > 2: $5.2 \pm 1.1\%$; monkey P, 1st-order: $50.6 \pm 6.5\%$, 2nd-order: $44.4 \pm 6.1\%$, order > 2: $5.0 \pm 1.5\%$), fall into the cluster of novice human players, suggesting that they share similar problem-solving structures. Notably, however, both monkeys had been playing the game for three years, while some novice human players were experiencing the game for the first time. Exemplar gameplay videos from different groups of humans and monkeys can be found in Supplementary Videos 1-3.

To further quantify the players' grammar complexity, we developed a summary statistic termed LoPS complexity, defined as the average rule order. The expert human players exhibited higher LoPS complexity compared to the novices (Figure 2d) and the monkeys (p ¡ 0.001, Mann-Whitney U test). No significant differences were observed between the novice human players and the monkeys in terms of LoPS complexity (all p ¿ 0.05, Mann-Whitney U tests).

3.5 Complex grammar, good performance

Complex grammars come at a computational cost, and their existence can only be justified if they provide tangible behavioral advantages. Indeed, we found a strong positive correlation between LoPS grammar complexity and problem-solving performance at the individual level among human subjects (Figure 3a). Leveraging more complex rules, the expert human players significantly outperformed the novice players ($U = 0.00$, $p < 0.001$, Mann-Whitney U test). When plotted on the same graph, the two monkeys were positioned at the lower end of both grammar complexity and performance, suggesting a continuum between humans' and monkeys' problem-solving capacities. Regression analysis revealed that LoPS complexity alone provides sufficient predictive power for game performance (Method 5.4.4).

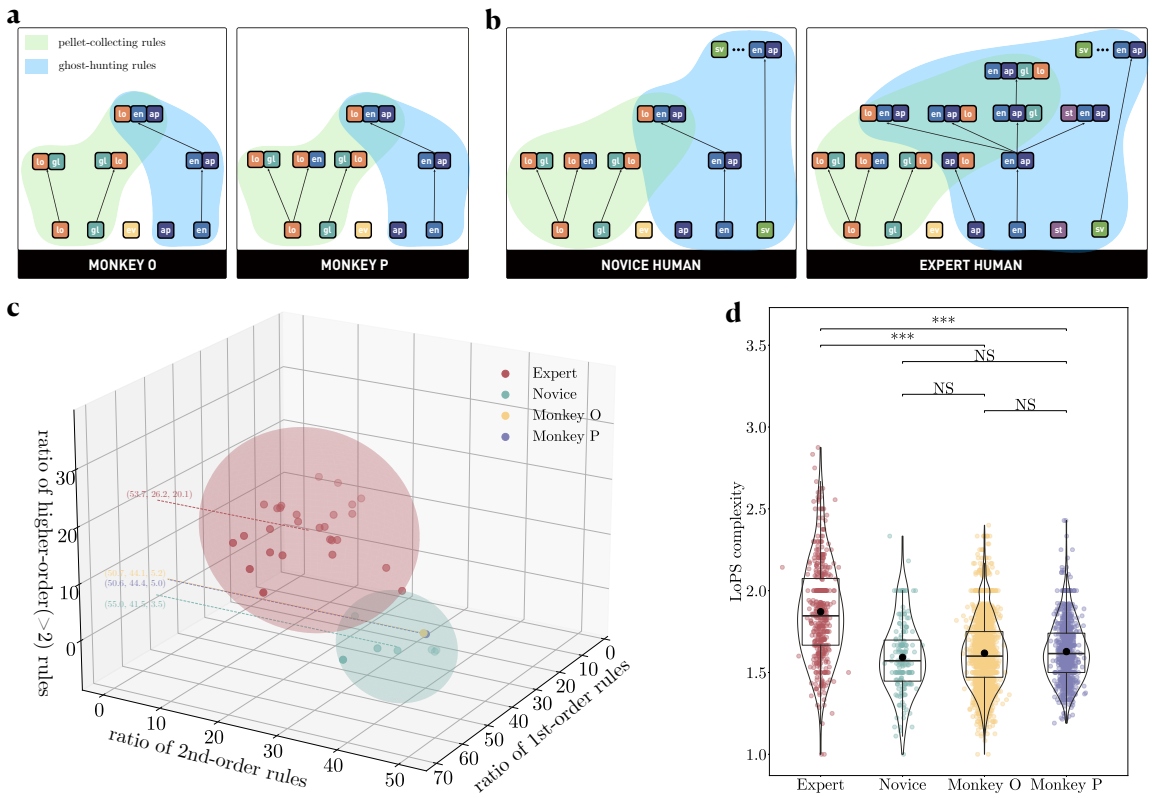

Figure 2: Participants' LoPS grammars. **(a)** LoPS grammars of the two monkeys. Left: monkey O; right: monkey P. Each block represents a basis strategy (lo: *local*, gl: *global*, ev: *evade*, ap: *approach*, en: *energizer*). High-order dependency rules are constructed by concatenating two sub-units. For clarity, only a single connection line is shown for each rule, linking either to the longer sub-unit or, in the case of equal-length sub-units, the first sub-unit. Green and blue shadings indicate pellet-collecting and ghost-hunting rules, respectively, and the overlapping region reflects rules that combine both objectives. **(b)** LoPS grammars of two example human players. Left: example player 1 (novice); right: example player 2 (expert). Same convention as **(a)**. Two additional basis strategies were used by the human players (sv: *save*, st: *stay*). **(c)** Rule usage ratio. Each subject's usage ratio of rules of order 1, 2, and higher ($n > 2$) is plotted in 3-D space. Using agglomerative clustering, we identified two distinct clusters among the human subjects: experts (red, $N = 27$) and novices (green, $N = 7$). The monkeys are also plotted in the same space: yellow (monkey O) and purple (monkey P). Example gameplay videos can be found in Supplementary Videos 1-3. **(d)** LoPS complexity for experts, novices, and monkeys. Each data point represents the average LoPS complexity in a completed Pac-Man game. *** $p < 0.001$, NS $p > 0.05$ (Mann-Whitney U tests). Analyses were based on the monkeys' behavior in the final year of the three-year study and the human participants' behavior in the second session. We did not report the statistical comparison on experts and novices, because their LoPS complexities are expected to be different from the clustering analysis.

In addition to game scores, we examined how fast an agent responded to strategy switches. Our hypothesis was that agents with more complex LoPS grammars could offload real-time decision making by planning strategy switches in advance, leading to faster reaction times. While it is difficult to measure the reaction times for strategy switches in general, we observed that the agents often changed Pac-Man's direction shortly after eating an energizer or a ghost due to a strategy switch. We measured the onset of these direction changes as reaction times, quantified as the number of tiles that Pac-Man travels, and examined whether they were correlated with grammar complexity. Consistent with this hypothesis, we found a significant negative correlation between LoPS complexity and reaction time (Figure 3b). A Kruskal-Wallis test comparing two human groups and two monkeys indicated a significant difference among the four ($H(3, 6212) = 517.62$, $p < 0.001$, Kruskal-Wallis test). Additional comparisons showed that expert humans exhibited shorter reaction times than novice humans and both monkeys (all $ps < 0.001$, Mann-Whitney U test, Bonferroni corrected).

These results suggest that complex LoPS grammars provide both performance and efficiency benefits in problem-solving.

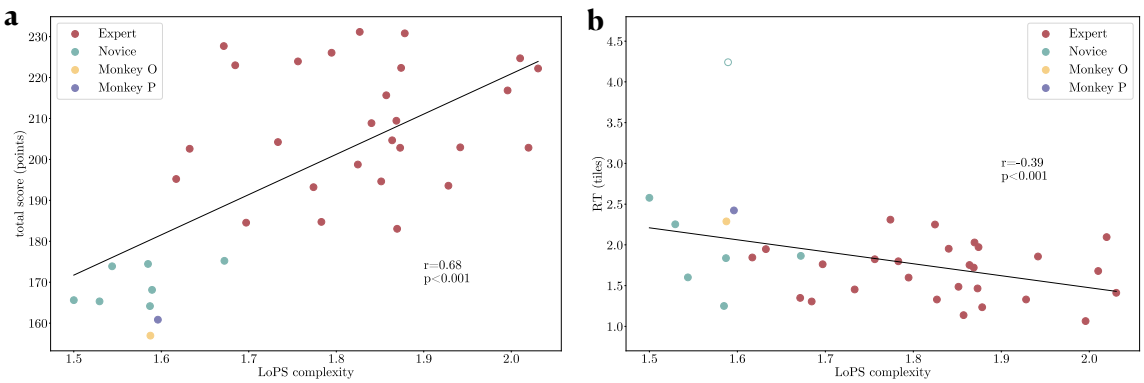

Figure 3: Grammar complexity correlates with performance. **(a)** Performance score. A significant positive correlation is observed between LoPS complexity and total score (Pearson's $r = 0.68$, bootstrap 95% CI $(0.55, 0.77)$, excluding the monkeys). The total score is the cumulative points earned by completing one game (detailed in Supplementary Table S1). Each data point represents an individual subject. **(b)** Reaction time. A significant negative correlation is observed between LoPS complexity and reaction time (Pearson's $r = -0.39$, bootstrap 95% CI $(-0.54, -0.21)$, excluding the monkeys; Pearson's $r = -0.24$, bootstrap 95% CI $(-0.43, -0.05)$), excluding the outlier human player (hollow circle). Reaction time (RT) is defined as the number of tiles between the occurrence of a specific event (eg. energizer or ghost consumption) and the subject's first Pac-Man direction change. The lines are from linear regressions using the human data. Red: expert players, green: novice players, yellow: monkey O, purple: monkey P.

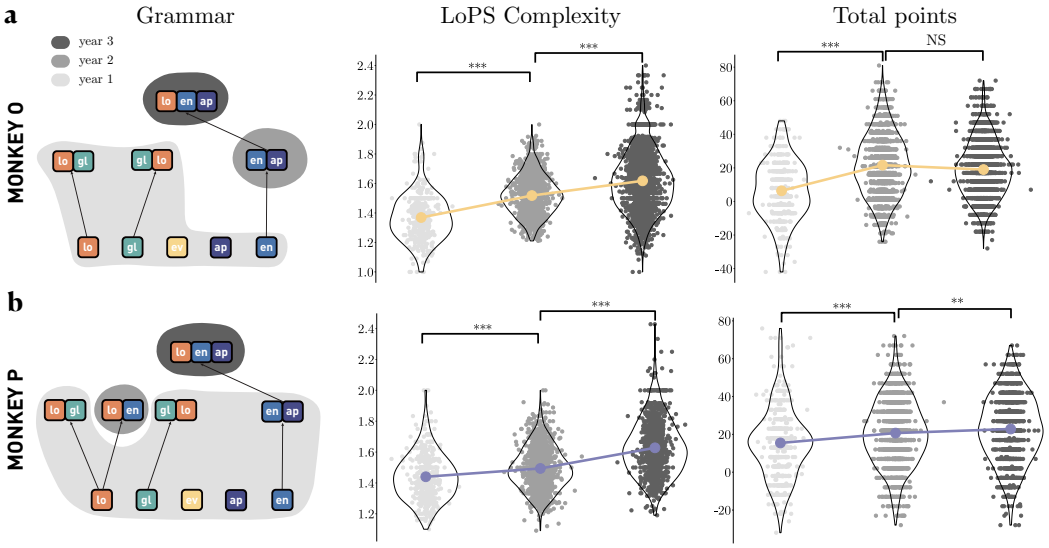

Figure 4: Evolving LoPS grammars with learning in monkeys. **(a)** Monkey O. Left: LoPS grammars extracted from the monkey O's gameplay. Different shades of gray indicate the addition of new rules each year. Middle: Progression of grammar complexity each year. Right: Progression of game scores each year. Each data point represents the mean in a complete Pac-Man game trial. The yellow dots represent the annual means. **(b)** Monkey P. Same convention as in **(a)**. *** $p < 0.001$, ** $p < 0.01$, NS $p > 0.05$ (Mann-Whitney U tests).

3.6 Evolving grammar with learning

Both monkeys and humans became better at the game with experience. By employing the LoPS induction algorithm at different stages of the experiment, we can assess whether these performance improvements were reflected in their LoPS grammars.

The monkeys were tested continuously over a span of three years. Through years of gaming, the monkeys gradually built their grammar set, incorporating more complex rules (Figure 4 left). The new rules, such as *energizer-approach* and *local-energizer-approach*, were primarily related to ghost-hunting. They helped monkeys take better advantage of energizers and made their gameplay more human-like. Consistently, their LoPS complexity also had a significant increase across the years (Figure 4 middle). This increase in grammar complexity was mirrored in their improved game performance (Figure 4 right).

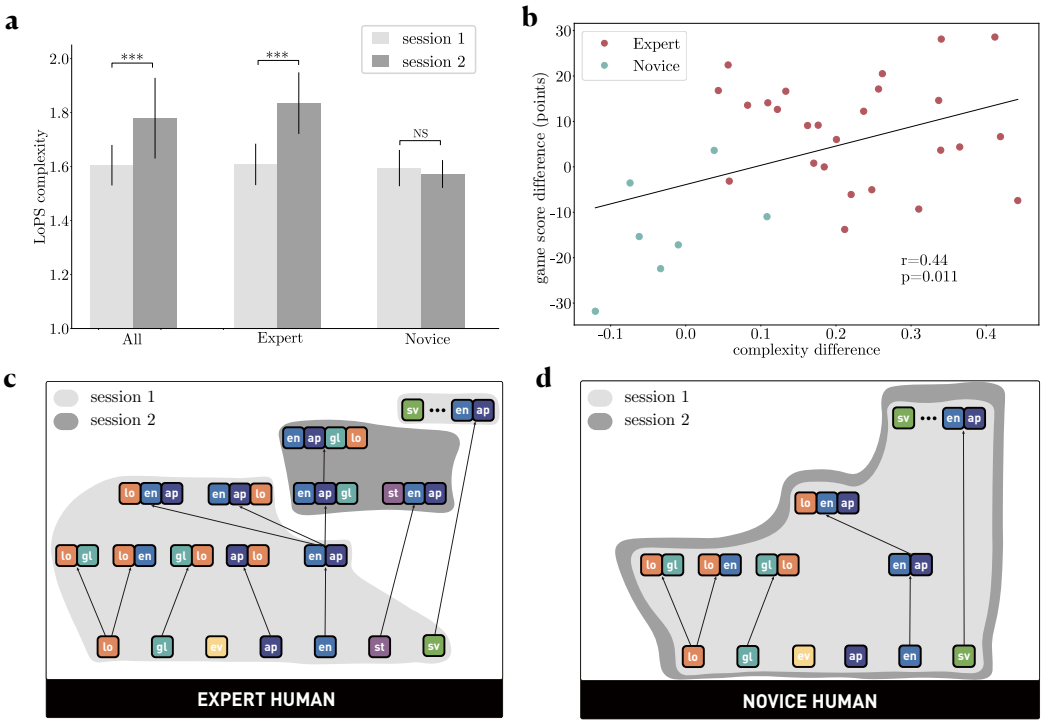

Figure 5: Evolving LoPS grammars with learning in human participants. **(a)** LoPS complexity for all human subjects and for the expert and novice groups separately across two sessions. Complexity increased significantly in both the overall group and the expert group, but not in the novice group. **(b)** Correlation between complexity difference and game score difference across sessions (Pearson's $r = 0.44$, $p = 0.011$). Experts: red; novices: green. **(c)** Evolution of LoPS grammar in experts. The newly developed rules were hybrid rules that combined pellet collecting and ghost hunting. **(d)** The grammar structure of novices did not change significantly between the two sessions. Different gray shadings indicate the two sessions. *** $p < 0.001$, NS $p > 0.05$ (paired sample t-test).

The human participants were tested in two sessions. We assessed the LoPS complexity for all subjects in both sessions and found a significant increase in LoPS complexity ($p < 0.001$, paired sample t-test, Figure 5a). At the group level, the increase was significant for the expert group ($p < 0.001$, paired sample t-test), but not for the novice group. In addition, a significant positive correlation was found between the complexity increase and the game score improvement between the sessions (Figure 5b), supporting the notion that greater grammar complexity enhances performance.

The new rules developed in the second session, such as *energizer-approach-global-local* and *stay-energizer-approach*, are hybrid rules that combine the objectives of pellet-collecting and ghost-hunting (Figure 5c). To reveal the effect of these new rules on game performance, we regressed the experts' performance improvement against two predictor variables: the difference in the frequency of using existing rules and the LoPS complexity increase caused by the three rules newly formed in the second session. Only the new rules had a significant effect on the performance improvement (regression weights: existing rules $w_1 = -3.25$, CI 95% $(-7.31, 0.83)$, new rules $w_2 = 5.38$, CI 95% $(1.33, 9.44)$). Therefore, the adoption of new rules was the primary driver of performance improvement in the second session for expert players. In contrast, the LoPS grammar of the novices did not change (Figure 5d).

Together, these results reveal the expansion of LoPS grammar in both monkeys and the expert humans during learning, indicating their problem-solving capacities evolved with experience. However, new grammar rules emerged in monkeys only after years of practice, and their LoPS grammars remained far less sophisticated than those of expert humans. There appears to be a performance ceiling that separates monkeys from humans, reflected in the complexity of their respective LoPS grammars.

3.7 State variable covariation graph

The game states in Pac-Man can be condensed into a few key game variables. We focused on six variables that are most critical to gameplay: the distance from Pac-Man to each ghost (g_1, g_2), the modes of the ghosts (m_1, m_2), Pac-Man's distance to the nearest energizer (e), and the count of local pellets (b) (See definitions in Supplementary Table S3). The basis strategies typically only deal with single variables. For example, strategy *local* mainly affects the number of local pellets, and strategy *approach* triggers a ghost mode change.

In contrast, higher-order rules often take into account multiple game variables. Therefore, examining how state variables co-varied can provide valuable insights into how grammars at different complexity levels influence gameplay.

Therefore, we computed the joint probabilities of these six key state variables, denoted as $P(\boldsymbol{s})$. Using the PC algorithm[21,22], we inferred the Markov network that depicts the correlational structure among these variables in the three groups of players —experts, novices, and monkeys— each characterized by distinct grammar complexities (See Methods 5.4.5). The resulting graph describes the inter-dependencies between these state variables as shaped by the execution of the players' strategies.

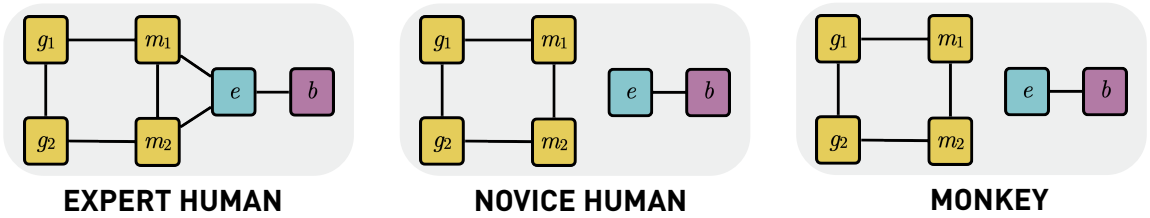

Figure 6: Markov networks of game state variables for experts (left), novices (mid), and monkeys (right). Nodes represent six key state variables: distances from Pac-Man to each ghost (g_1, g_2), modes of the ghosts (m_1, m_2), Pac-Man's distance to the energizer (e), and counts of local pellets (b). Edges between the nodes indicate conditional dependencies. The graphs were inferred with PC algorithm[21,22] (Methods 5.4.8).

We observed a key difference in the graph between the group of expert humans and the other two player groups (Figure 6). Both the novices and the monkeys formed two separate covariation clusters (Figure 6, middle and right). One cluster $\boldsymbol{s}_g$ consists of four ghost-related variables: each ghost's mode (m_1, m_2) and Pac-Man's distance to each ghost (g_1, g_2). It is disconnected from the second cluster, which consisted of e, the distance to the energizer, and b, the count of local pellets. In the expert group, however, the two clusters are linked via e, the distance-to-energizer (Figure 6, left), forming an integrated structure.

The structural differences in the covariation graphs indicate the players' distinct approaches of breaking down the task. Novice humans and monkeys treated the ghost-related and reward-related variables as independent, simplifying their decision-making by focusing on one component and disregarding the other. This reduction in state space may make it easier for them to navigate in state space, but it comes at the cost of game performance. In contrast, experts tracked all state variables simultaneously, allowing them to achieve superior performance, which indicates their superior problem-solving capacity.

3.8 State transition map

To further investigate the game state transition patterns driven by different LoPS grammars, we constructed transition maps between successor states caused by the grammar rules for each player group (Figure 7; see Methods 5.4.6 for more details). These transition maps can be viewed as cognitive maps[23] that help different subjects to navigate the game space, potentially explaining the exploratory advantages associated with more complex grammars.

Immediately, we can see that the number of effective states in the expert humans' gameplay greatly outnumbers that of the novices and monkeys (experts: 14 ± 0.1, novices: 7 ± 0.4, monkeys: 6 ± 0.6 (group mean $\pm$ standard error)), leading to a much more intricate transition map.

Upon closer inspection, state 9 stands out as a pivotal hub state for expert players. With two normal ghosts far away, an energizer and local pellets nearby available, Pac-Man can forage local pellets safely in this state. It serves as a strategic retreat state from many other states, including those with threatening ghosts (states 1-3) and those lacking local food or energizers (states 7, 8, 10, 13, 14). Expert players use a diverse set of grammar rules to navigate back to this safe state. Moreover, state 9 is involved in a state loop: $9 \rightarrow 12 \rightarrow 13 \rightarrow 9$. This loop reflects a clever game scheme in which the player hunts ghosts while staying close to an energizer ($9 \rightarrow 12$), saves this energizer ($12 \rightarrow 13$) and grazes until the ghosts return to their normal state ($13 \rightarrow 9$).

For novice players, state 9 is also a hub state, but with fewer connections and transitions using only lower-order rules. This leads to another interesting feature of the novices' effective states: in four states (states 15-18), the scared ghosts are too far from Pac-Man to be effectively hunted, limiting their ability to earn bonus points. In contrast, expert players avoided such situations with high-order rules that coordinate energizer consumption and ghost hunting.

State 9 is no longer a hub state for the monkeys. Their only 3rd-order rule (*local-energizer-approach*) takes the game from state 9 to state 13. Interestingly, experts used the same rule to navigate to state 12,

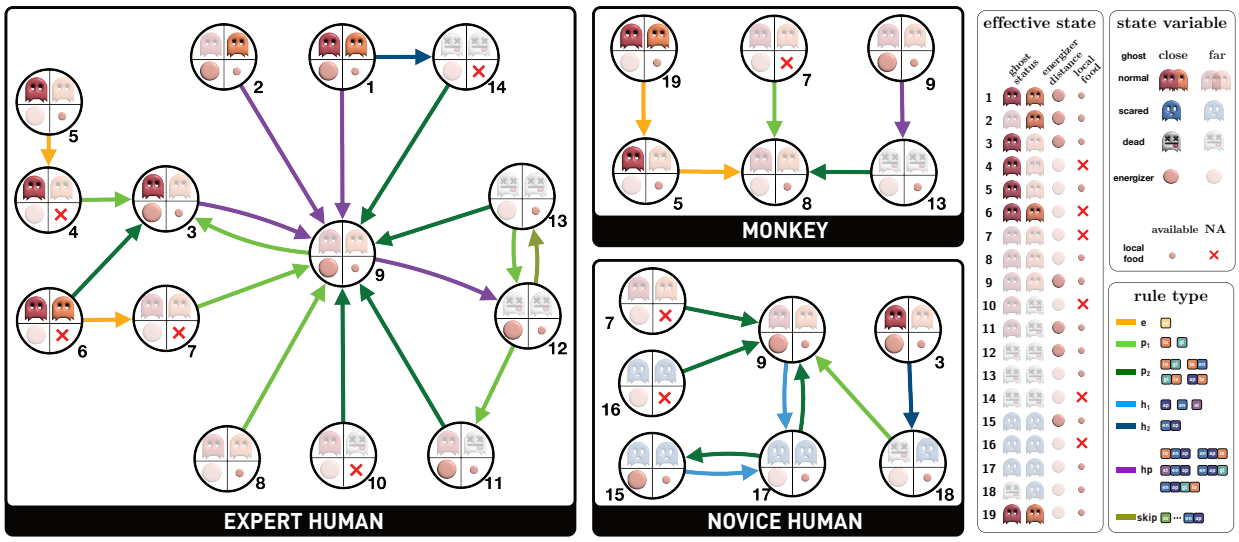

Figure 7: State transition maps for the experts, novices, and monkeys. Nodes represent effective game states, defined by six state variables (two ghosts' modes, their distances from Pac-Man, energizer distance, and local pellets availability). The effective states were selected such that their cumulative probability accounts for more than 70% of the total state space encountered by each group. Each state node is illustrated with four icons: two for the ghosts (three modes: normal, scared, or dead; dim: far, bright: close), one for energizer distance (dim: far, bright: close), and one for pellet availability (pellet: available, cross: NA). Directed edges denote the predominant rules that lead to statistically significant state transitions. The rules are grouped based on their purposes and complexity: pellet-collecting (p_1: 1st-order, p_2: 2nd-order); ghost-hunting (h_1: 1st-order, h_2: 2nd-order); hybrid rules combining pellet-collecting and ghost-hunting (hp); evasion (e); and skip-order rule ($skip$).

which differs from state 13 only in having a close-by energizer, reflecting better advanced planning. This suggests that the monkeys did not learn to proactively plan their hunting to end with nearby energizers, preventing them from forming efficient foraging-hunting loops like the experts. Instead, they often returned to state 8, which contains only local pellets, indicating their tendency to reduce the game to simple foraging.

Together, the state transition maps highlight key differences in how each player group navigates the game space. By adopting more sophisticated grammars, expert players accessed a larger set of advantageous game states that were unexplored by novices or monkeys. The high-order rules in experts' LoPS grammar enabled them to maneuver through their complex network of states, utilizing a central hub state (state 9) to efficiently alternate between collecting pellets, hunting ghosts, and evading threats, while the monkeys' simpler grammar reduced their game to mostly foraging. The complexity of a player's LoPS grammar reflects their capability to navigate the game space.

4 Discussion

Our investigation into the cognitive processes underpinning problem-solving through a language model has yielded intriguing insights into the differences between human and non-human primate intelligence. The adaptation of a classic video game like Pac-Man into a cross-species behavior paradigm allows us to explore these dynamic processes in a quantifiable manner. The model's ability to capture the temporal structure and complexity of problem-solving brings us closer to understanding the compositional nature of cognition, both in humans and macaque monkeys.

4.1 Language of Thought

The LoPS model provides a framework to analyze the 'grammar' of problem-solving and offers a novel approach to measuring and comparing compositional thought processes across species. It draws inspiration from the hypothesis of the language of thought (LoT), which conceptualizes mental representations using symbolic primitives and composition laws[6] and captures core principles of various human cognitive abilities[15,24,25].

Our grammar induction differs from previous program induction procedures that have been employed to capture the underlying structure or generative process of the data in LoT models[24,26–32]. It takes into account the temporal structure of upper stream states[33]. The dynamic environment in the Pac-Man game has a rich temporal structure, and our approach is essential for identifying the true temporal dependencies

between strategies, improving previous methods that focused primarily on action sequences while ignoring the temporal structure of the internal and external variables leading to the actions[34]. Neglecting the upstream variables and their temporal structures would result in false identifications of solution structures. In addition, Pac-Man's strategy transitions are probabilistic, requiring involve complex decision-making in a dynamic environment. This complexity necessitates modeling the grammar as probabilistic dependencies not only among strategies but also between strategies and game states, extending the previous definition of grammar in LoT.

Our algorithm has certain limitations. One key constraint is that it only factors in limited types of dependencies. In addition, the algorithm can suffer from data sparsity, as some rules might have relatively low frequency in the training datasets. One potential solution is to incorporate other higher-level properties of language models, such as role-filler independence, predicate-argument structures, logic operations[35], and And-Or graph[36,37]. State-of-the-art language models based on RNNs and transformers[38] present another avenue, as they allow for richer primitives and more adaptable compositional rules[39].

4.2 Human vs Monkey

Although the intelligence gap between humans and other animals, including monkeys, is evident, the extent and underlying causes of this separation remain an active area of research[40]. Our results clearly demonstrate that human players exhibit more complex problem-solving behaviors compared to monkeys. The hierarchical nature of human cognition[41–44] is reflected in their LoPS grammars, which are more intricate and possess deeper hierarchies. This aligns with the view that human intelligence is distinguished by its ability to use abstract thought processes and to construct and navigate complex state spaces — capabilities that are less developed in macaque monkeys. Moreover, humans display a capacity to quickly develop new and complex rules through experience, which cannot be compensated by extended training in monkeys. While there may be a shared ancestral cognitive language used for problem-solving, our findings suggest that the depth and complexity of these mental languages have evolved significantly in humans[45], giving rise to their sophisticated problem-solving abilities.

4.3 Neural substrate of LoPS

The LoPS reveals a hierarchical structure in the problem-solving of humans and monkeys. A rostrocaudal axis of the PFC has been proposed to underlie hierarchical cognitive control, with which the brain implements deeply structured, tree-like policies through nested corticostriatal gating loops, arranged from back to front within the frontal lobe[46,47]. Consistent with this idea, patients with lesions in the rostral PFC often exhibit impairments in problem-solving tasks that require planning and abstract reasoning[48]. In addition, the human brain's language circuitry and its counterparts in the monkey brain may also play an important role in LoPS.

The quantitative nature of the LoPS model opens up possibilities for future investigation into its underlying neural mechanisms, particularly in animal models. By recording and manipulating single-unit activities across the prefrontal cortex, we may explore how neurons in different PFC subregions are involved in dependency rules of different orders and how they coordinate to activate downstream motor regions to generate appropriate responses based on higher-order cognitive representations[49].

Author contributions

Q.Y. and T.Y. conceptualized the study, developed the methodology, conducted the investigation, conducted the formal analysis, developed the software, created the visualizations, validated the results, wrote the original draft, administered the project, and acquired funding. Z.Z. developed the methodology, developed the software, conducted the formal analysis, created the visualizations, and wrote the original draft. R.S. and Y.L. curated the data and conducted the investigation. J.Z. supervised the work and acquired funding. All authors reviewed and edited the manuscript.

Acknowledgments

We thank Yunxian Bai, Wei Kong, Jianshu Li, Zhongqiao Lin, Ruixin Su, Lu Yu, Yang Xie, Ruichang Sun, and Wenyi Zhang for their help in all phases of the study, and Thomas Akam for helpful discussions. This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDB1010301), National Science and Technology Innovation 2030 Major Program (Grant No. 2021ZD0203701) to T.Y., National Natural Science Foundation of China (Grant No. 32100832), Leading-edge Technology Program of Jiangsu Natural Science Foundation (BK20232004) and the Youth Innovation Promotion Association CAS to Q.Y., European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. [716321 - FREEMIND]) to J.Z., and

PhD studentship from China Scholarship Council to R.S. The funders had no role in study design, data collection and interpretation, or the decision to submit the work for publication.

Declaration of interests

The authors declare no competing financial interests.

5 STAR Methods

5.1 Key Resources Table

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
Monkey data (Year 2)	Previous Work	https://elifesciences.org/articles/74500
Monkey data (Year 1 and 3)	This Paper	https://github.com/lab-decision-making-mechanism/Language-of-Problem-Solving
Human data	This Paper	https://github.com/lab-decision-making-mechanism/Language-of-Problem-Solving
Software and Algorithms		
Python version 3.8	Python Software Foundation	https://www.python.org

5.2 Resource availability

5.2.1 Lead contact

Further information and requests for resources should be directed to and will be fulfilled by the lead contact, Tianming Yang (tyang@ion.ac.cn).

5.2.2 Materials availability

This study did not generate new unique reagents.

5.2.3 Data and code availability

The data and codes that support the findings of this study are provided at: https://github.com/lab-decision-making-mechanism/Language-of-Problem-Solving.

5.3 Experimental model and subject details

This study recruited 34 healthy participants (19 female, 14 male, and 1 undisclosed). The participants were aged from 18 to 40 with an average age of 24.79 ± 5.45. None of the participants had a previous history of neurological or psychiatric disorders. All participants gave written informed consent for the sessions they attended and received monetary compensation. The study was approved by the Cardiff University School of Psychology Research Ethics Committee.

Two male rhesus monkeys (Macaca mulatta) were used in the study (O and P). They weighed on average 6-7 kg during the experiments. All procedures followed the protocol approved by the Animal Care Committee of Shanghai Institutes for Biological Sciences, Chinese Academy of Sciences (CEBSIT-2021004).

5.4 Method details

5.4.1 Pac-Man task

The Pac-Man game for the monkey experiment has been described previously[19]. Briefly, monkeys control Pac-Man with a joystick through a maze to collect pellets. Each pellet consumed yields a juice reward. Completing the maze yields additional juice. Two ghosts roam the maze. If Pac-Man is caught, there is a time-out penalty. Eating special pellets, called energizers, switches ghosts to a temporary scared mode during which they can be eaten for extra juice rewards.

For the human task version, the human controls Pac-Man with a keyboard. The maze is sized 725×900 pixels and displayed at the resolution of 1920×1080. Same as the monkey task version, the maze is divided into square tiles of 25×25 pixels. The human game contains no fruits to simplify the game. Nine unconnected

pellet patches are defined in the maze. Each patch has 9 to 13 tiles, and in each game, only 7 randomly selected patches are filled. In total, there are 76 to 83 pellets in the maze, of which four randomly selected pellets are replaced in the upper left, upper right, lower left, and lower right areas of the maze with an energizer. Collecting a pellet and an energizer will gain 2 and 4 score points, respectively. In addition, humans get an extra 10 points for catching a scared ghost. A real-time cumulative point of the current game tally is shown on the screen. If caught by a ghost, humans receive a deduction of points. For better data collection, humans are allowed maximally two attempts to complete a game. Finally, the game speed for the human version is set to be twice as fast as that of the monkey version. The rest of the settings are identical between the human and the monkey versions. The human-specific settings aim to encourage more evenly distributed strategies used by human subjects to facilitate behavior analyzes with a limited amount of data.

The monkeys' behavioral data were collected over three years. During three years, monkeys played 820 (year 1), 3240 (year 2), and 571 (Year 3) game trials. The analyses presented here are based on the behavior in Year 3 unless otherwise mentioned. On average, monkeys completed 33 ± 9 games per session, with each game requiring 4.9 ± 1.8 attempts. Human participants were involved in two separate sessions. We used the behavior data in the second session to do the analyses in the paper unless otherwise mentioned. On average, each human participant completed 36 ± 6 games per session, with each game taking an average of 1.2 ± 0.4 attempts.

5.4.2 Language of Problem-Solving

The Language of Problem-Solving (LoPS) model employs a language model to analyze and describe the structure of an agent's problem-solving behavior.

Primitive words in LoPS are a set of decision-making schemes which are a selection of basis strategies in the context of the Pac-Man game. The grammars in LoPS describe the temporal dependencies between the sequences of strategies, in accordance with the dependency grammar formalism in natural language processing[20].

Mathematically, it can be formulated in the framework of Probabilistic Graphical Models (PGMs). A PGM $G = \{\mathcal{V}, \mathcal{E}\}$ contains nodes $\mathcal{V}$ representing random variables and edges $\mathcal{E}$ indicating conditional dependencies. The edges can be directed ($\mathcal{V}_k \rightarrow \mathcal{V}_l$) or undirected ($\mathcal{V}_k - \mathcal{V}_l$), used in Bayesian networks and Markov networks, respectively. A dependency grammar in LoPS can be described by a Bayesian network $\mathcal{G}$:

$$\mathcal{G} = \{G_{g_1}, G_{g_2}, G_{g_3}, ..., G_{g_{skip}}\} \tag{1}$$

where G_{g_1} (1st-order), G_{g_2} (2nd-order), G_{g_3} (3rd-order), ..., and $G_{g_{skip}}$ (skip-order) denote different types of dependency grammar rules. The naming convention is similar to that used in Markov chains: The order of the dependency grammar rule refers to the number of previous strategies considered when determining the probability of transitioning to the next strategy, in addition to the upstream variables, i.e., the game states. Mathematically, they can be defined as

$$\begin{aligned} G_{g_1} &= \{\boldsymbol{s}^t \rightarrow \pi^t\} \\ G_{g_2} &= \{(\pi^{t-1}, \boldsymbol{s}^t) \rightarrow \pi^t\} \\ G_{g_3} &= \{(\pi^{t-2}, \pi^{t-1}, \boldsymbol{s}^t) \rightarrow \pi^t\} \\ &... \\ G_{g_{skip}} &= \{(\pi^{t-t'}, \boldsymbol{s}^t) \rightarrow \pi^t\} \end{aligned} \tag{2}$$

where $\boldsymbol{s}^t$ represents the game state at time t, π^t represents the strategy at time t, and $t' > 1$ denotes the temporal dependency of non-consecutive preceding strategies.

All concepts in the LoPS model are summarized in Supplementary Table S2.

5.4.3 LoPS gammar induction

Grammar induction is an inverse problem in which we commence with a subject's gameplay and deduce the LoPS grammar. Below, we describe the entire procedure in details. There are three steps: feature extraction, strategy fitting, and grammar induction. The pseudocode can be found in Algorithm 1.

Feature extraction

We first extract location information about each game element from the visual input time series. From each frame of the Pac-Man game, we obtain the location of Pac-Man l, the locations of the two ghosts l_{g1}, l_{g2}, the locations of the energizers l_e, and the locations of the pellets l_p.

We compute the utility values for all directions ($\mathcal{A} = \text{left}, \text{right}, \text{up}, \text{down}$) under each of the seven strategies included in LoPS. Note that not all directions are always available. For unavailable directions,

utility values are set to negative infinity. The moving direction is computed according to the largest average utility value for each strategy.

We then determine the utility associated with each direction and its possible trajectories. Let l represent Pac-Man's position and $\tau(l, a)$ represent a path starting from l and moving in the direction of a with a length of 10. We compute the utility of the path $\tau(l, a)$ under each strategy as follows (for simplicity, $\tau(l, a)$ is denoted as τ):

- *local* strategy: $u_{lo}(\tau) = \sum_{x \in \tau \cap l_p} \text{Reward}(x)$
- *energizer* strategy: $u_{en}(\tau) = \sum_{x \in \tau \cap l_e} \text{Reward}(x)$
- *save* strategy: $u_{sv}(\tau) = \sum_{x \in \tau \cap l_e} \text{Penalty}(x)$
- *approach* strategy: $u_{ap}(\tau) = \sum_{x \in \tau \cap l_{g_1}, l_{g_2}} \text{Reward}(x)$
- *evade* strategy: $u_{ev}(\tau) = \sum_{x \in \tau \cap l_{g_1}, l_{g_2}} \text{Penalty}(x)$, if g_1, g_2 are normal, else 0.

The rewards and penalty utilities for each element of the game in the model can be found in Table S1.

For each direction $a \in \mathcal{A}$, its utility $\boldsymbol{U}_\pi(l, a)$ is obtained by averaging all the path utilities $u_\pi(\tau)$ in that direction:

$$\boldsymbol{U}_\pi(l, a) = \begin{cases} \dfrac{1}{|\mathcal{T}|} \displaystyle\sum_{\tau \in \mathcal{T}} u_\pi(\tau), & \text{if } a \text{ is available} \\ -\infty, & \text{otherwise} \end{cases} \tag{3}$$

Two strategies, *global* and *stay*, do not depend on utilities associated with Pac-Man location l. The utility of the *global* strategy $\boldsymbol{U}_{gl}$ counts the total number of pellets throughout the maze at a specific direction, excluding those within 10 steps of the current location. The utility of the *stay* strategy $\boldsymbol{U}_{st}$ is infinitely negative in all directions, causing Pac-Man to stay in place.

Another set of features includes a series of relational-based state variables, which capture the necessary information for strategy initiation and arbitration. These variables include s_e (Dijkstra distance from Pac-Man to the closest energizer), s_{g1} (Dijkstra distance from Pac-Man to Blinky), and s_{g2} (Dijkstra distance from Pac-Man to Clyde), s_b (Local pellet number within 10 steps away from Pac-Man), s_{m1} (Blinky's mode), and s_{m2} (Clyde's mode). To simplify calculations, we binarize the distance and the pellet number variables. More information on state variables can be found in Table S3.

The pseudocode for feature extraction can be found in Algorithm 2.

Strategy fitting

The strategy fitting follows the same procedure described in our previous work[19]. The strategy fitting process is based on the assumption that the strategy weights remain stable for a period of time. This period is defined by fine-grained time windows $\Delta = \delta_1, \delta_2, ..., \delta_k$, separated by essential game events, including changes in Pac-Man's direction, ghost consumption, energizer consumption, and ghost mode change.

Within each time segment, a softmax policy is employed to linearly combine utilities under each basis strategy, with the strategy weights as model parameters. The probability of choosing a certain action a is defined as:

$$P(a|\boldsymbol{w}_\pi, \boldsymbol{U}) = \frac{\exp(\sum_\pi w_\pi U_\pi^t(l, a))}{\sum_{a' \in \mathcal{A}} \exp(\sum_\pi w_\pi U_\pi^t(l, a'))} \tag{4}$$

where $\boldsymbol{w}_\pi \in \mathbb{R}^7$ represents the strategy weight.

Given the utility and action time series in each time segment, $D_{\boldsymbol{u}}^\delta$, and D_a^δ, the likelihood function can be formulated as:

$$\mathcal{L}(D_{\boldsymbol{u}}^\delta, D_a^\delta \mid \boldsymbol{w}_\pi) = \prod_{t \in \delta} P(a_t|\boldsymbol{w}_\pi, \boldsymbol{U}_t) \tag{5}$$

A genetic algorithm (GA) is then used to estimate the weights by maximizing the likelihood function. This implementation of the GA algorithm uses the GA class from the scikit-opt library in Python with the following parameters — Population size: 100, mutation probability: 0.1, crossover probability: 0.8, and maximum iteration number: 500.

The strategy with the highest weight is used to create a strategy time series $D_\pi = \{\pi^t\}_{t=1}^T$, where $\pi^t \in \mathbb{Z}^7$ represents the dominant strategy at time t. More details about this procedure can be found in Algorithm 3.

Grammar induction

Given the time series of strategy D_π and state $D_{\boldsymbol{s}}$, we induce the dependency grammar $\mathcal{G}$ using the structure learning method. The process begins with a grammar graph that only contains 1st-order rules, $G_{g_1} = \{\boldsymbol{s}^t \to \pi^t\}$. We apply a hybrid network learning algorithm (session 5.4.10 and Algorithm 8) to infer an initial graph containing all 1st-order rules.

With this initial graph $G_1 = G_{g_1}$, we compute the likelihood score of an alternative graph by concatenating one pair of strategies, G_2. If the likelihood score for the alternative graph is higher than that for the initial graph, we accept the alternative graph as the hypothetical graph. Upon each update of the hypothetical graph, we parsed the strategy time series again to generate a new rule time series D_g based on the

updated graph. This new time series is used to compute the likelihood score in the subsequent iteration. This iterative process continues until the likelihood score for the hypothetical graph reaches a maximum. At this point, the induced graph structure is defined as the subject's LoPS grammar $\mathcal{G} = \max_{G_g} P(D \mid G_g)$. This pseudocode for this procedure can be found in Algorithm 4.

The parsing algorithm used in the grammar induction follows the principle of Minimum Description Length (MDL). Given a specific strategy time series D_π, we parse it into a rule time series D_g that uses the minimal number of grammar rules according to a set of rules defined by the grammar G_g. Thus, we compress the original strategy time series into the shortest possible description. The pseudocode for the parsing algorithm can be found in Algorithm 5.

Validation of LoPS grammar

To validate the LoPS induction algorithm, we construct artificial agents with flexible ground-truth grammar, varying in their grammar complexity ($\mathcal{G}_1$: 1st-order; $\mathcal{G}_2$: 2nd-order; $\mathcal{G}_3$: 3rd-order). For an agent with a defined grammar complexity, their grammar rules are randomly selected from the library we induced from human and monkey subjects. The artificial agent then interacts with the Pac-Man game environment. We collecte data on the game of $M = 500$ different artificial agents. We then apply the LoPS induction algorithm (Algorithm 1) to infer each agent's LoPS grammar. In Supplementary Figure S3, we plot the accuracy of the estimated agent type and grammar complexity, along with the increasing sample size of the strategy sequences (D_π). Both estimates can reach high accuracy below the quantity of experimental data we collected in humans (~ 4000) and monkeys (~ 20000).

5.4.4 LoPS analyses

Although we define the grammar with a set of dependency rules, we can organize them according to their generative process hierarchically, ranked vertically based on their orders (Figure 2a). Each higher-order rule node is built from two simpler rule nodes. For better visualization, only one link is shown. If a complex rule is created from daughter nodes of equal complexity, only the edge to the first node is shown. For cases where a complex rule breaks down into two daughter nodes of differing complexity, we retain the edge to the more complex rule node. This approach provides a clear and concise visual representation, emphasizing the inherent hierarchical and compositional nature of the grammar rules.

Given each subject's grammar book, we apply the parsing algorithm (Algorithm 5) to each individual's sequence of strategies to obtain their respective rule sequences. Based on these rule sequences, we can compute the probability of each rule, denoted as $P(g_n)$. The ratio of all participants' 1st-order rule, 2nd-order rule, and 3rd-order rule usage can be computed accordingly. Employing the agglomerative clustering method on these three statistical measures across 34 human subjects leads to the identification of two clusters, named experts and novices according to their grammar rule usage preference. The clustering algorithm uses the Euclidean distance metric and 'ward' linkage criterion to minimize the variance of the clusters being merged.

We define the complexity of each dependency grammar rule g_n as its order n. The complexity of a skip-order corresponds to the length of the subsequent rule sequence up to the targeted rule element. The LoPS complexity can then be computed as the average of n with respect to the probability of the rules, represented as $C_\mathcal{G} = \langle n \rangle_{P(g_n)}$. This measure provides a concise summary statistic to quantify grammar complexity.

To investigate the relationship between LoPS complexity and game performance, we regress total game score against LoPS complexity along with two confounding variables, total number of actions, and total game time, all normalized to their respective maximum (excluding monkeys). The regression weight of LoPS complexity ($w_c = 93.20$, 95% CI $(87.35, 98.74)$) is significantly higher than the two confounders (number of actions: $w_a = 0.0022$, 95% CI $(0.0003, 0.0051)$; game time: $w_t = -0.0006$, 95% CI $(-0.0013, -0.0001)$). This result demonstrates that LoPS complexity alone provides sufficient predictive power for game performance.

5.4.5 State variable covariation graph

We use a Markov network to illustrate the correlational structure among the game state variables (Figure 6). The nodes on the network graph are the six game state variables, denoted as $\boldsymbol{s} = (s_{m1}, s_{m2}, s_{g1}, s_{g2}, s_e, s_b)$, and the undirected edges indicate conditional dependencies between the nodes. The definition of these states can be found in Supplementary Table S3. To learn the graph structure, we use the PC algorithm (Algorithm 6,[21,22]), which is a constraint-based structure learning approach. The PC algorithm starts with a fully connected undirected graph and iteratively removes edges based on conditional independence tests. It tests the conditional independence of each pair of variables given the set of all other variables using a Bayesian approach. If two variables are found to be conditionally independent, the edge between them is removed from the graph. This process is repeated until no more edges can be removed, resulting in the final Markov network structure that represents the conditional dependencies among the game state variables. More details about the PC algorithm can be found in Methods 5.4.8.

5.4.6 State transition map

The game state is defined as the ensemble of the six state variables described above. The distance and pellet count variables are binarized for simplicity. Effective states are defined as the set of most commonly encountered states at grammar rule transitions whose cumulative probability accounts for more than 70% of the total state space encountered by each group. This approach ensures that the selected effective states capture the most representative and frequently visited states during gameplay.

To simplify the map, we reduce all grammar rules into seven types, $\phi \in \{e, p_1, p_2, h_1, h_2, hp, skip\}$, based on their purpose (pellet-collecting, ghost-hunting, or evade) and complexity. The rule types are as follows: e: evading rules; p_1: 1st-order pellet-collecting rules; p_2: 2nd-order pellet-collecting rules; h_1: 1st-order ghost-hunting rules; h_2: 2nd-order ghost-hunting rules; hp: rules involving both pellet-collecting and ghost-hunting; $skip$: skip-order rules (Listed in the legend of Figure 7).

We select statistically significant transitions that navigate the player from the current state $\boldsymbol{S}$ to the successor state $\boldsymbol{S}'$ according to the criterion that the transition probability is above the chance level, $P(\boldsymbol{S}'|\boldsymbol{S}) > 1/|S'|$. We label these transitions according to the most frequently used grammar type.

In Figure 7, we visualize the state space of each group as a map, with the effective states as the nodes and the predominant grammar rule type as the edges linking the nodes. Each node can be identified with four icons: two ghosts with three modes (normal, scared, or dead) and two distance statuses (dim: far, bright: close), energizer distance (dim: far, bright: close), and food availability (pellet: available, cross: NA).

5.4.7 Structure learning in Probabilistic graphical model

To compute the marginal likelihood of a dataset D given a hypothetical graph structure $G = \{\mathcal{V}, \mathcal{E}\}$, we integrate over all possible parameter configurations Θ under the given graph structure:

$$P(D \mid G) = \int P(D \mid G, \Theta) P(\Theta \mid G) d\Theta, \tag{6}$$

where $P(D \mid G, \Theta)$ is the likelihood of the data given the graph structure and a specific parameter configuration, and $P(\Theta \mid G)$ is the prior probability of the parameter given the graph structure. The integration over Θ is necessary to avoid favoring more complex models which are characterized by larger parameter sets.

To identify the best network structure given the data, we use a scoring method that seeks to maximize this marginal likelihood over all possible graph structures:

$$\mathcal{G} = \max_{G} P(D \mid G) \tag{7}$$

In this setup, $\mathcal{G}$ denotes the optimal graph structure, which is the one that maximizes the likelihood of the data.

5.4.8 PC algorithm for learning a Markov network

For a Markov network with n variables $\mathcal{V} = \{\mathcal{V}_1, ...\mathcal{V}_n\}$, the joint probability is defined as a product of potentials on the subsets of variables (clique) $\mathcal{V}_c \in \mathcal{V}$:

$$P(\mathcal{V}_1, ..., \mathcal{V}_n) = \frac{1}{Z} \prod_{c=1}^{C} \phi_c(\mathcal{V}_c) \tag{8}$$

where Z is the normalization constant, ϕ_c is potential function for the clique $\mathcal{V}_c$.

The PC algorithm is a constraint-based structure learning approach[21,22]. It starts with a fully connected undirected graph and iteratively removes edges based on conditional independence tests. Here is a brief summary of the method:

1. Start with a fully connected graph.
2. For each pair of variables $\mathcal{V}_k$ and $\mathcal{V}_l$, test the conditional independence of $\mathcal{V}_k$ and $\mathcal{V}_l$ given the set of all other variables $\mathcal{V}_{-k,-l}$.
3. If $\mathcal{V}_k$ and $\mathcal{V}_l$ are conditionally independent given $\mathcal{V}_{-k,-l}$, remove the edge between $\mathcal{V}_k$ and $\mathcal{V}_l$.
4. Repeat 2-3 until no more edges can be removed, and output the structure of the resulting graph.

The conditional independence test used in the PC algorithm is performed using a Bayesian approach, where the likelihoods under the independence and dependence hypotheses are evaluated with the data. The independence hypothesis assumes that the joint distribution of $\mathcal{V}_k$ and $\mathcal{V}_l$ can be factored into separate

distributions conditioned on $\mathcal{V}_{-k,-l}$:

$$P(\mathcal{V}_k, \mathcal{V}_l, \mathcal{V}_{-k,-l} \mid \mathcal{H}_{indep}) = P(\mathcal{V}_k \mid \mathcal{V}_{-k,-l}, \theta_{\mathcal{V}_k|\mathcal{V}}) P(\mathcal{V}_l \mid \mathcal{V}_{-k,-l}, \theta_{\mathcal{V}_l|\mathcal{V}_{-k,-l}}) P(\mathcal{V}_{-k,-l} \mid \theta_{\mathcal{V}_{-k,-l}}), \tag{9}$$

whereas the joint distribution in the dependence hypothesis cannot be factorized:

$$P(\mathcal{V}_k, \mathcal{V}_l, \mathcal{V}_{-k,-l} \mid \mathcal{H}_{dep}) = P(\mathcal{V}_k, \mathcal{V}_l, \mathcal{V}_{-k,-l} \mid \theta_{\mathcal{V}_k, \mathcal{V}_l, \mathcal{V}_{-k,-l}}). \tag{10}$$

By treating variables as categorical and assuming Dirichlet priors for all parameters, we can compute the marginal likelihood with the dataset $D = \{\mathcal{V}_k^m, \mathcal{V}_l^m, \mathcal{V}_{-i,-j}^m\}_{m=1}^{m=M}$ for the independent hypothesis and the dependent hypothesis as

$$P(D \mid \mathcal{H}_{indep}) = \frac{B(N_{\mathcal{V}_{-k,-l}} + \alpha_{\mathcal{V}_{-k,-l}})}{B(\alpha_{\mathcal{V}_{-k,-l}})} \prod \frac{B(N_{\mathcal{V}_k|\mathcal{V}_{-k,-l}} + \alpha_{\mathcal{V}_k|\mathcal{V}_{-k,-l}})}{B(\alpha_{\mathcal{V}_k|\mathcal{V}_{-k,-l}})} \frac{B(N_{\mathcal{V}_l|\mathcal{V}_{-k,-l}} + \alpha_{\mathcal{V}_l|\mathcal{V}_{-k,-l}})}{B(\alpha_{\mathcal{V}_l|\mathcal{V}_{-k,-l}})} \tag{11}$$

$$P(D \mid \mathcal{H}_{dep}) = \frac{B(N_{\mathcal{V}_k,\mathcal{V}_l,\mathcal{V}_{-k,-l}} + \alpha_{\mathcal{V}_k,\mathcal{V}_l,\mathcal{V}_{-k,-l}})}{B(\alpha_{\mathcal{V}_k,\mathcal{V}_l,\mathcal{V}_{-k,-l}})}, \tag{12}$$

where B is beta function, N_x is the number of times that event x is presented in the data, α_x is the corresponding hyperparameter for each event x[21,22].

The pseudocode for the conditional independence test and PC algorithm can be found in Algorithm 10 and Algorithm 6.

To validate the PC algorithm, we first randomly generate Markov graph structures and specifiy the parameters for their local pairwise potentials. However, it does not mean that the joint distribution fulfills the Markov property that the probability of each variable is dependent only on its immediate neighbors. We compute the following two distributions: $P(\mathcal{V}_i|\mathcal{V}_{-i})$ and $P(\mathcal{V}_i|ne(\mathcal{V}_i))$ according to the joint probability of the specified local structures and parameters and ensured that they were equal for all variables. In this way, we generate M Markov networks that fulfill local Markov properties, each characterized by its unique structure and parameters, to serve as the ground truth to validate the PC algorithm.

For each generated Markov network, we create N samples under the marginal distributions of each variable, computed according to the defined joint distributions. We then recover the ground-truth network structure by applying the PC algorithm to the synthetic data. Without loss of generality, we set $n = 3$, $M = 50000$, and varied the sample size from $N = 20$ to $N = 2000$. The average accuracy of the PC algorithm was shown in Supplementary Figure S4, where the accuracy is based on the congruence between the edge sets specified by the ground truth network and the induced network. The validation results indicate that the PC algorithm may recover the network structure well even with a small sample.

5.4.9 Network scoring algorithm for learning a Bayesian network

For a Bayesian network with n variables $\mathcal{V} = \{\mathcal{V}_1, ...\mathcal{V}_n\}$, the joint probability is defined as

$$P(\mathcal{V}_1, ..., \mathcal{V}_n) = \prod_k P(\mathcal{V}_k \mid \mathcal{V}_{pa(k)}) \prod_{k'} P(\mathcal{V}_{k'}) \tag{13}$$

where $\mathcal{V}_{pa(k)}$ represents the parents of $\mathcal{V}_k$ in the network, $\mathcal{V}_{k'}$ are the most upstream variables in the network who have no parents.

The method for learning the structure of a Bayesian network involves using network scoring techniques. Calculating marginal likelihood (Eq. 6) generally involves an intractable integration. However, by treating variables as categorical and assuming Dirichlet priors for parameters, the marginal likelihood can be analytically calculated[21,22].

Specifically, these factorized local distributions can be expressed as

$$P(\mathcal{V}_k = i \mid \mathcal{V}_{pa(k)} = j) = \mathrm{Cat}(\theta_{ki|j}) \tag{14}$$

$$P(\mathcal{V}_{k'} = i) = \mathrm{Cat}(\theta_{k'i}) \tag{15}$$

where Cat represents the probability density function of a categorical distribution, $\theta_{ki|j}$ indicates the probability of $\mathcal{V}_k$ being in state i given its parents are in state j, and $\theta_{k'i}$ specifies the probability of $\mathcal{V}_{k'}$ being in state i.

Given data $D = \{\mathcal{V}_1^m, ..., \mathcal{V}_n^m\}_{m=1}^M$, the likelihood of these local distributions can be expressed as:

$$P(D \mid \{\mathcal{V}_k \leftarrow \mathcal{V}_{pa(k)}\}, \Theta_{\mathcal{V}_k|\mathcal{V}_{pa(k)}}) = \prod_j \prod_i \theta_{ki|j}^{N_{ki|j}} \tag{16}$$

$$P(D \mid \{\mathcal{V}_k\}, \Theta_{\mathcal{V}_k}) = \prod_i \theta_{k'i}^{N_{ki}}, \tag{17}$$

where $N_{ki|j}$ is the number of times that $\mathcal{V}_k$ is in state i and its parents in state j in the data, $N_{k'i}$ is the number of times that $\mathcal{V}_k$ is in state i, $\Theta_{\mathcal{V}_k|\mathcal{V}_{pa(k)}}$ and $\Theta_{\mathcal{V}_{k'}}$ are parameter sets.

We apply the Bayesian Dirichlet likelihood equivalent uniform (BDeu) prior for the parameters:

$$P(\theta_{ki|j}) = \mathrm{Dir}(\alpha_{ki|j}) P(\theta_{k'i}) = \mathrm{Dir}(\alpha_{k'i}), \tag{18}$$

where $\alpha_{ki|j} = \frac{\alpha}{\dim(\mathcal{V}_k)\dim(\mathcal{V}_{pa(k)})}$, $\alpha_{k'i} = \frac{\alpha}{\dim(\mathcal{V}_{k'})}$, $\alpha > 0$. We can compute the marginal local likelihood as

$$P(D \mid \{\mathcal{V}_k \leftarrow \mathcal{V}_{pa(k)}\}) = \mathrm{score}(\mathcal{V}_k, \mathcal{V}_{pa(k)}) = \prod_j \prod_i \frac{B(N_{ki|j} + \alpha_{ki|j})}{B(\alpha_{ki|j})} \tag{19}$$

$$P(D \mid \{\mathcal{V}_{k'}\}) = \mathrm{score}(\mathcal{V}_{k'}) = \prod_i \frac{B(N_{k'i} + \alpha_{k'i})}{B(\alpha_{k'i})}, \tag{20}$$

where B is the beta function.

Based on these marginal local likelihoods, we can then combine them to compute the likelihood of the whole graph as

$$\mathrm{score}(G) = P(D \mid G) = \prod_k \mathrm{score}(\mathcal{V}_k, \mathcal{V}_{pa(k)}) \prod_{k'} \mathrm{score}(\mathcal{V}_{k'}). \tag{21}$$

Based on this likelihood score, we can search for graph structures that yield the highest scores. This network scoring approach can be more efficient than the PC algorithm, as it allows for the comparison of differences in local structures between two networks, thanks to the score function's compositionality into local likelihoods. Therefore, search heuristics that involve local addition or removal of edges can be particularly effective[22].

The pseudocode for the network scoring algorithm can be found in Algorithm 7.

To validate the network scoring algorithm, we generate Bayesian networks with n upstream variables and m downstream variables. For each of the downstream nodes, we independently choose one of the 2^n possible combinations of the upstream nodes as its parent nodes. We parameterize the conditional probability distribution underlying this local structure as a categorical distribution. Parameters are randomly generated from a uniform distribution with support between 0 and 1. In this way, we generate M Bayesian networks, each characterized by its unique structure and parameters, to serve as the ground truth for the network scoring algorithm.

For each Bayesian network, we generate N samples under defined conditional probability distributions. We then attempt to recover the structure of the ground-truth network by applying the network scoring algorithm to these synthetic data. Without loss of generality, we set $n = 3$, $m = 2$, $M = 50000$, and vary the sample size from $N = 20$ to $N = 2000$. The average accuracy of the network scoring algorithm is shown in the Supplementary Figure S5, where the accuracy is calculated according to the congruence between the edge sets specified by the ground-truth network and the induced network.

5.4.10 Hybrid network learning

Here, we explore a special case in which upstream variables (states) are modeled through a Markov network, which is an undirected network, while the interconnections between the upstream and the downstream (strategies) variables are captured by a Bayesian network, which is directed. To tackle the challenge of structure learning in such a setting, we developed a hybrid network learning algorithm, described as follows.

First, we apply the PC algorithm to induce the structure of the Markov network (G_m) amongst the upstream variables. This structure allows for the identification of neighboring nodes, $ne(\mathcal{V}'_l)$, for any given upstream node $\mathcal{V}'_l$. These neighbors form the Markov blanket: the smallest set of nodes that make $\mathcal{V}'_l$ conditionally independent to all other nodes.

Subsequently, we apply the network scoring algorithm to induce the structure of the Bayesian network (G_b). The computation of the marginal likelihood entails a directed edge from upstream node $\mathcal{V}'_l$ to a downstream node $\mathcal{V}_k$, necessitates the additional conditioning on the neighboring nodes $ne(\mathcal{V}'_l)$:

$$P(D \mid \mathcal{V}_k \leftarrow \mathcal{V}'_l) = \mathrm{score}(\mathcal{V}_k, \{\mathcal{V}'_l, ne(\mathcal{V}'_l)\}) \tag{22}$$

This score function is computed according to Eq 19. Consequently, the revised likelihood score function is:

$$\mathrm{score}(G_b) = \prod_k \mathrm{score}(\mathcal{V}_k, \{\mathcal{V}'_l, ne(\mathcal{V}'_l)\}) \prod_{k'} \mathrm{score}(\mathcal{V}_{k'}) \tag{23}$$

With this likelihood score, we search for the graph structure that yields the highest score.

The hybrid network learning algorithm exploits the conditional independence properties of the Markov network to reduce computational complexity, transitioning from conditioning on all upstream state variables to conditioning solely on the Markov blanket.

The pseudocode for the hybrid network learning algorithm can be found in Algorithm 8.

To validate the hybrid network learning algorithm, we generate synthetic data similar to the scenario in the Bayesian network validation. However, here, the upstream variables are not independent. Instead, they are generated from the Markov network defined in Session 5.4.8. Without loss of generality, we set $n = 3$, $m = 2$, $M = 50000$, and vary the sample size from $N = 20$ to $N = 2000$. The average accuracy of the hybrid network learning algorithm is depicted in the Supplementary Figure S6, where the accuracy is computed according to the congruence between the edge sets specified by the ground truth network and the induced network. Our hybrid network learning algorithm can accurately capture the true network structure used to create the simulated data.

5.5 Quantification and statistical analysis

Statistical analyses in the Results focuses mainly on LoPS grammar complexity and game performance. Due to the large discrepancy in the total number of games played by the two human groups and the two monkeys, non-parametric statistics (Kruskal-Wallis test) were used to test the differences among the four groups. If a significant difference was detected, Mann-Whitney U tests were employed for further pairwise comparisons. When comparing the learning effect between groups using summarized statistics from each session, paired-sample t-tests were utilized (Figure 5a). Pearson's correlation coefficient was calculated to assess the relationship between two summarized statistics. The significance level for all statistical tests was set at $p < 0.05$.

6 Video Legends

Video 1: example game video from example human 1 (novice).
Video 2: example game video from example human 2 (expert).
Video 3: example game video from Monkey P.
Video 4: example game video for a skip-order grammar rule from example human 2
Video 5: example game video for a 3rd-order rule *Stay-Energizer-Approach* from example human 2.

References

1 Mattar, M. G. and M. Lengyel (2022). Planning in the brain. Neuron *110*, 914–934.

2 Opheusden, B. van, I. Kuperwajs, G. Galbiati, Z. Bnaya, Y. Li, and W. J. Ma (2023). Expertise increases planning depth in human gameplay. Nature, 1–6.

3 Zylberberg, A. (2021). Decision prioritization and causal reasoning in decision hierarchies. PLoS computational biology *17*, e1009688.

4 Donnarumma, F., D. Maisto, and G. Pezzulo (2016). Problem solving as probabilistic inference with subgoaling: explaining human successes and pitfalls in the tower of hanoi. PLoS computational biology *12*, e1004864.

5 Callaway, F., B. van Opheusden, S. Gul, P. Das, P. M. Krueger, T. L. Griffiths, F. Lieder, and T. L. Griffiths (2022). Rational use of cognitive resources in human planning. Nature Human Behaviour *6*, 1112–1125.

6 Fodor, J. A. (1975). *The language of thought*. Vol. 5. Harvard university press.

7 Lake, B. M., T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman (2017). Building machines that learn and think like people. Behavioral and brain sciences *40*, e253.

8 Dehaene, S., F. Al Roumi, Y. Lakretz, S. Planton, and M. Sablé-Meyer (2022). Symbols and mental programs: a hypothesis about human singularity. Trends in Cognitive Sciences.

9 Yoo, S. B. M., B. Y. Hayden, and J. M. Pearson (2021). Continuous decisions. Philosophical Transactions of the Royal Society B *376*, 20190664.

10 Rajalingham, R., A. Piccato, and M. Jazayeri (2022). Recurrent neural networks with explicit representation of dynamic latent variables can mimic behavioral patterns in a physical inference task. Nature Communications *13*, 5865.

11 Lakshminarasimhan, K. J., E. Avila, E. Neyhart, G. C. DeAngelis, X. Pitkow, and D. E. Angelaki (2020). Tracking the mind's eye: Primate gaze behavior during virtual visuomotor navigation reflects belief dynamics. Neuron *106*, 662–674.

12 Dang, J., K. M. King, and M. Inzlicht (2020). Why are self-report and behavioral measures weakly correlated? Trends in cognitive sciences *24*, 267–269.

13 Eisenberg, I. W., P. G. Bissett, A. Zeynep Enkavi, J. Li, D. P. MacKinnon, L. A. Marsch, and R. A. Poldrack (2019). Uncovering the structure of self-regulation through data-driven ontology discovery. Nature communications *10*, 2319.

14 Tinbergen, N. (2020). *The study of instinct*. Pygmalion Press, an imprint of Plunkett Lake Press.

15 Al Roumi, F., S. Marti, L. Wang, M. Amalric, and S. Dehaene (2021). Mental compression of spatial sequences in human working memory using numerical and geometrical primitives. Neuron *109*, 2627–2639.

16 Panichello, M. F. and T. J. Buschman (2021). Shared mechanisms underlie the control of working memory and attention. Nature *592*, 601–605.

17 Nieh, E. H., M. Schottdorf, N. W. Freeman, R. J. Low, S. Lewallen, S. A. Koay, L. Pinto, J. L. Gauthier, C. D. Brody, and D. W. Tank (2021). Geometry of abstract learned knowledge in the hippocampus. Nature *595*, 80–84.

18 Fascianelli, V., A. Battista, F. Stefanini, S. Tsujimoto, A. Genovesio, and S. Fusi (2022). Neural representational geometries correlate with behavioral differences in monkeys and recurrent neural networks. bioRxiv, 2022–10.

19 Yang, Q., Z. Lin, W. Zhang, J. Li, X. Chen, J. Zhang, and T. Yang (2022). Monkey plays Pac-Man with compositional strategies and hierarchical decision-making. Elife *11*, e74500.

20 Jurafsky, D. and J. H. Martin (2023). *Speech and Language Processing (Third Edition)*. Prentice-Hall, Inc.

21 Koller, D. and N. Friedman (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

22 Barber, D. (2012). *Bayesian reasoning and machine learning*. Cambridge University Press.

23 Behrens, T. E., T. H. Muller, J. C. Whittington, S. Mark, A. B. Baram, K. L. Stachenfeld, and Z. Kurth-Nelson (2018). What Is a Cognitive Map? Organizing Knowledge for Flexible Behavior. Neuron *100*, 490–509.

24 Piantadosi, S. T. and R. A. Jacobs (2016). Four Problems Solved by the Probabilistic Language of Thought. Current Directions in Psychological Science *25*, 54–59. DOI: 10.1177/0963721415609581.

25 Lake, B. M., R. Salakhutdinov, and J. B. Tenenbaum (2015). Human-level concept learning through probabilistic program induction. Science *350*, 1332–1338.

26 Goodman, N. A., J. B. Tenenbaum, J. Feldman, and T. L. Griffiths (2008). A rational analysis of rule-based concept learning. Cognitive Science *32*, 108–154. DOI: 10.1080/03640210701802071.

27 Planton, S., T. van Kerkoerle, L. Abbih, M. Maheu, F. Meyniel, M. Sigman, L. Wang, S. Figueira, S. Romano, and S. Dehaene (2021). *A theory of memory for binary sequences: Evidence for a mental compression algorithm in humans*. Vol. 17, 1–43. ISBN: 1111111111. DOI: 10.1371/journal.pcbi.1008598.

28 Sablé-Meyer, M., K. Ellis, J. Tenenbaum, and S. Dehaene (2022). A language of thought for the mental representation of geometric shapes. Cognitive Psychology *139*, 101527.

29 Mills, T., J. Tenenbaum, and S. Cheyette (2024). Human spatiotemporal pattern learning as probabilistic program synthesis. Advances in Neural Information Processing Systems *36*.

30 Ellis, K., C. Wong, M. Nye, M. Sablé-Meyer, L. Morales, L. Hewitt, L. Cary, A. Solar-Lezama, and J. B. Tenenbaum (2021). "Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning". *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*, 835–850.

31 Overlan, M. C., R. A. Jacobs, and S. T. Piantadosi (2017). Learning abstract visual concepts via probabilistic program induction in a Language of Thought. Cognition *168*, 320–334.

32 Ho, M., S. Sanborn, F. Callaway, D. Bourgin, and T. Griffiths (2018). Human Priors in Hierarchical Program Induction. 2018 Conference on Cognitive Computational Neuroscience.

33 Ligneul, R., Z. F. Mainen, V. Ly, and R. Cools (2022). Stress-sensitive inference of task controllability. Nature Human Behaviour *6*, 812–822.

34 Wu, S., N. Élteto, I. Dasgupta, and E. Schulz (2022). Learning Structure from the Ground up—Hierarchical Representation Learning by Chunking. Advances in Neural Information Processing Systems *35*, 36706–36721.

35 Quilty-Dunn, J., N. Porot, and E. Mandelbaum (2022). The best game in town: The reemergence of the language-of-thought hypothesis across the cognitive sciences. Behavioral and Brain Sciences *46*, e261.

36 George, D., W. Lehrach, K. Kansky, M. Lázaro-Gredilla, C. Laan, B. Marthi, X. Lou, Z. Meng, Y. Liu, H. Wang, et al. (2017). A generative vision model that trains with high data efficiency and breaks text-based CAPTCHAs. Science *358*, eaag2612.

37 Zhu, S.-C. and D. Mumford (2007). A stochastic grammar of images. Foundations and Trends in Computer Graphics and Vision *2*, 259–362.

38 Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). Attention is all you need. Advances in neural information processing systems *30*.

39 Tang, J., A. LeBel, S. Jain, and A. G. Huth (2023). Semantic reconstruction of continuous language from non-invasive brain recordings. Nature Neuroscience, 1–9.

40 Laland, K. and A. Seed (2021). Understanding human cognitive uniqueness. Annual Review of Psychology *72*, 689–716.

41 Eckstein, M. K. and A. G. Collins (2020). Computational evidence for hierarchically structured reinforcement learning in humans. Proceedings of the National Academy of Sciences *117*, 29381–29389.

42 Botvinick, M. and A. Weinstein (2014). Model-based hierarchical reinforcement learning and human action control. Philosophical Transactions of the Royal Society B: Biological Sciences *369*, 20130480.

43 Botvinick, M. M., Y. Niv, and A. G. Barto (2009). Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. Cognition *113*, 262–280.

44 Ribas-Fernandes, J. J., A. Solway, C. Diuk, J. T. McGuire, A. G. Barto, Y. Niv, and M. M. Botvinick (2011). A neural signature of hierarchical reinforcement learning. Neuron *71*, 370–379.

45 Beck, J. (2017). Do nonhuman animals have a language of thought? The Routledge handbook of philosophy of animal minds, 46–55.

46 Badre, D. and D. E. Nee (2018). Frontal cortex and the hierarchical control of behavior. Trends in cognitive sciences *22*, 170–188.

47 Collins, A. G. and M. J. Frank (2013). Cognitive control over learning: creating, clustering, and generalizing task-set structure. Psychological review *120*, 190.

48 Szczepanski, S. M. and R. T. Knight (2014). Insights into human behavior from lesions to the prefrontal cortex. Neuron *83*, 1002–1018.

49 Genovesio, A. and S. P. Wise (2007). *The Neurophysiology of Abstract Response Strategies*. Oxford University Press. ISBN: 9780195314274.

Supplemental Information

Supplemental Figures

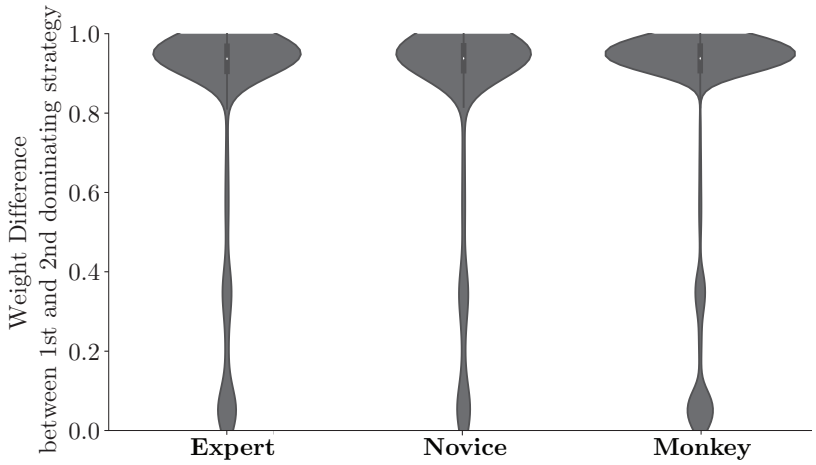

Figure S1: Take-The-Best strategy heuristic. The distribution of the weight difference between the most and the second dominating strategies. We assumed that decisions for Pac-Man's moving directions were based on a linear combination of the basis strategies. We estimated the strategy weights using time windows of flexible length, by assuming that the relative strategy weights are stable for a period. The strategies were ranked according to their fitted weights at each time segment. The weight difference between the first and the second most dominating strategies was heavily skewed toward one. The result is consistent with our previous findings[19], indicating that both the humans and the monkeys adopted Take-The-Best heuristics in which action decisions were formed with a single strategy that was heuristically and dynamically chosen.

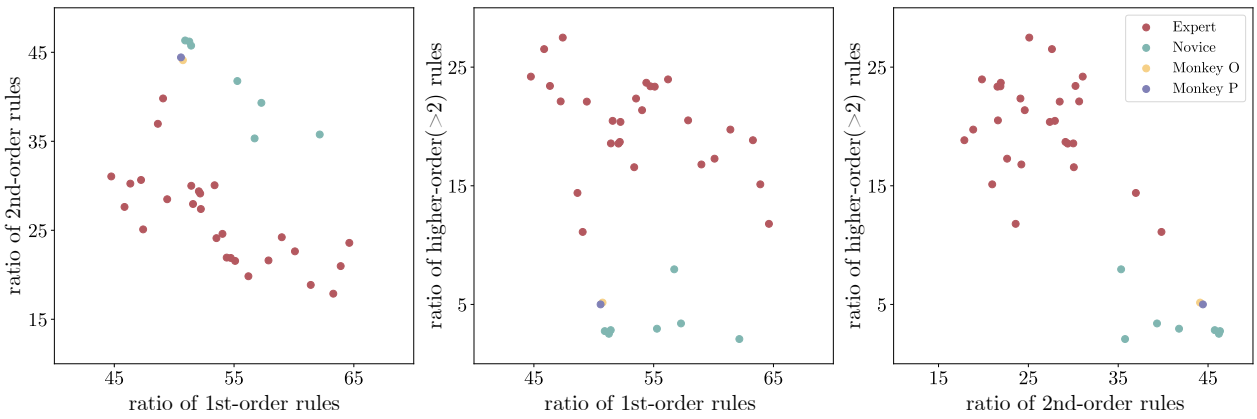

Figure S2: Rule usage ratio. Each subject's usage ratio of rules at order 1, order 2, and higher ($n > 2$) is plotted in three 2-D spaces. By applying the Agglomerative clustering method, we could identify two distinct clusters among the human subjects: experts (red, $N = 27$) and novices (green, $N = 7$). The monkeys are indicated by the yellow (monkey O) and the purple dot (monkey P).

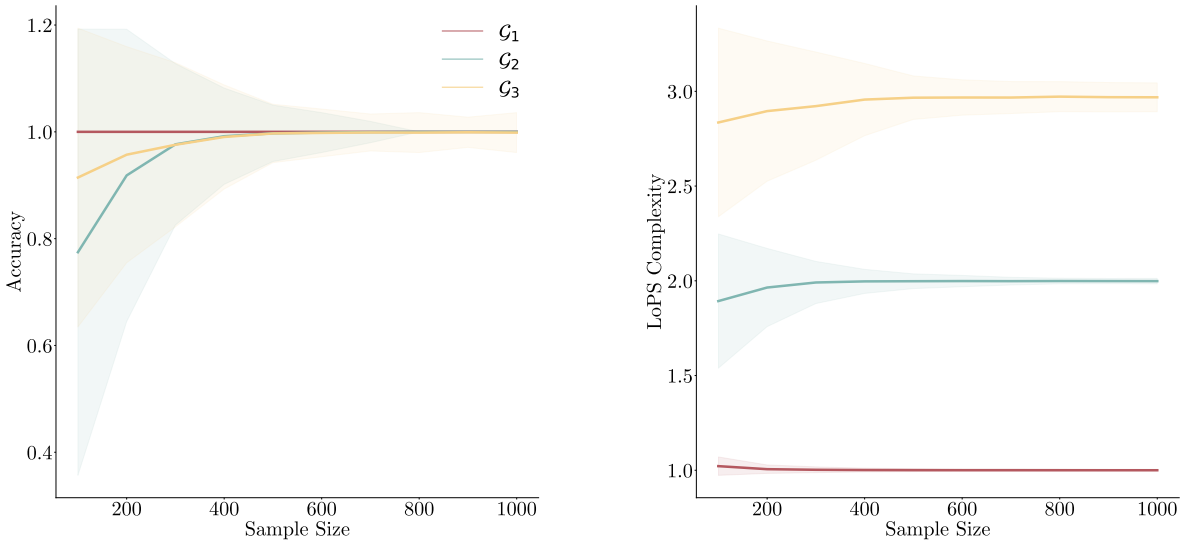

Figure S3: Validation of the LoPS induction algorithm. Three sets of simulated data are generated with different complexity levels, indicated with different colors. The solid line denotes the average estimate. The shade denotes the standard error. The accuracies of estimated agent type (left) and grammar complexity (right) reach high accuracy with sample sizes well below that of the experiment data we used for humans (~ 4000) and monkeys (~ 20000).

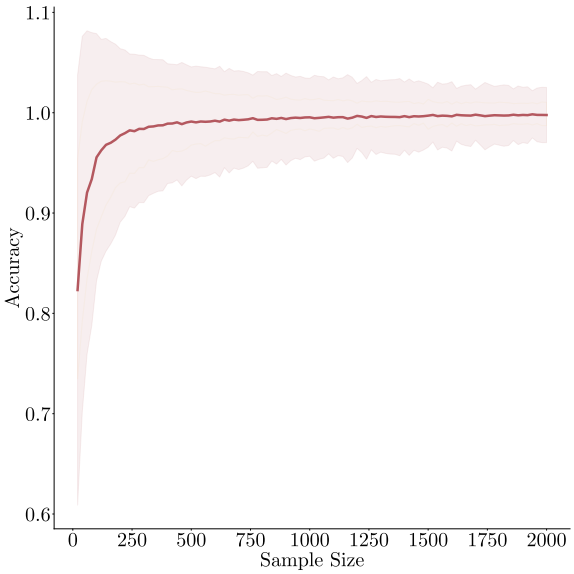

Figure S4: Validation of the PC algorithm on synthetic data generated from the ground-truth Markov network. The solid line denotes the average accuracy. The shade denotes the standard error.

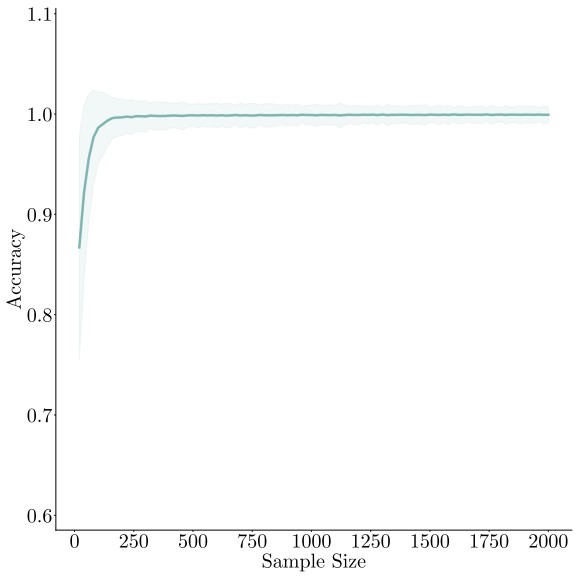

Figure S5: Validation of the network scoring algorithm on synthetic data generated from the ground-truth Bayesian network. The solid line denotes the average accuracy. The shade denotes the standard error.

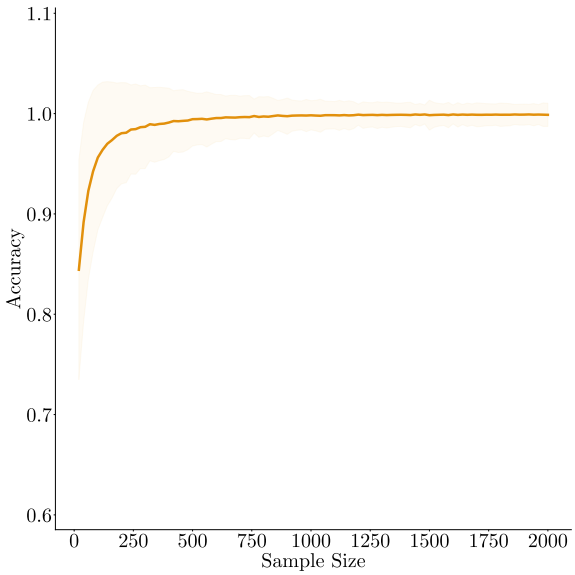

Figure S6: Validation of the hybrid network learning algorithm on synthetic data generated from the ground-truth hybrid network. The solid line denotes the average accuracy. The shade denotes the standard error.

Supplemental Tables

Table S1: Rewards and penalties in the Pac-Man game

Reward (pellet)	Reward (energizer)	Reward (scared ghost)	Penalty (death)	Penalty (save energizer)
2	4	10	−10	−4

Table S2: Concepts in Language of Problem-Solving

I	Visual inputs
a	Joystick movement actions, $a = \{\text{up}, \text{down}, \text{left}, \text{right}\}$
$\boldsymbol{U}$	Utilities associated with each action under a specific strategy
$\boldsymbol{s}$	Game states, $\boldsymbol{s} = \{s_e, s_b, s_{g1}, s_{g2}, s_{m1}, s_{m2}\}$, categorical variable denoting the condition of the current game states. Detailed definitions can be found in Table S3
π	Strategy, selected from a set of seven strategies. Detailed definitions can be found in Table S4
$\mathcal{G}$	Grammar, specifies a set of probabilistic concatenation rules, with each rule defining a temporal dependency between the strategies. $\{g_1, g_2, g_3, ..., g_{skip}\} \in \mathcal{G}$, where g_1 is the 1st-order rules, g_2 is the 2nd-order rules, g_3 is the 3rd-order rules, etc. g_{skip} is the skip-order rules. The order of the dependency grammar rule refers to the number of previous strategies considered when determining the probability of transitioning to the next strategy, in addition to the upstream variables, i.e., the game states.
$C_{\mathcal{G}}$	LoPS Complexity, defined as average rule order, $C_{\mathcal{G}} = \langle n \rangle_{P(g_n)}$, where $P(g_n)$ is the probability of each rule based on the parsed rule sequence according to the subjects' grammar $\mathcal{G}$

Table S3: Game state variables

s_b	Local pellet number within 5 steps, binarized (1 if > 0, 0: if=0)
s_e	Dijkstra distance to the closest energizer, binarized (1 if > 10; 0 if $\leq$10)
s_{g1}	Dijkstra distance to Blinky, binarized (1 if > 10; 0 if ≤ 10)
s_{g2}	Dijkstra distance to Clyde, binarized (1 if > 10; 0 if ≤ 10)
s_{m1}	Blinky ghost's mode, $s_{m1} \in \{\text{normal}, \text{dead}, \text{scared}\}$
s_{m2}	Clyde ghost's mode, $s_{m2} \in \{\text{normal}, \text{dead}, \text{scared}\}$

Table S4: Strategies

π_{lo}	*local* strategy moves Pac-Man in the direction that offers the largest reward within a 10-tile radius
π_{gl}	*global* strategy moves Pac-Man in the direction that has the largest total number of pellets in the whole maze, excluding those within a 10-tile radius
π_{en}	*energizer* strategy moves Pac-Man toward the closest energizer
π_{ap}	*approach* strategy moves Pac-Man toward the ghosts, regardless of the ghosts' mode
π_{ev}	*evade* strategy directs Pac-Man away from the nearest ghost
π_{st}	*stay* strategy keeps Pac-Man near a specific location
π_{sv}	*save* strategy keeps Pac-Man away from the energizer

Table S5: Probabilistic Graphical Model

G	a graph including nodes and edges, $G = \{\mathcal{V}, \mathcal{E}\}$.
$\mathcal{V}$	Nodes in a graph, indexed by k, l. The states of a node is indexed by i, j
$\mathcal{E}$	Edges in a graph, categorized into directed ($\mathcal{V}_k \rightarrow \mathcal{V}_l$) or undirected ($\mathcal{V}_k - \mathcal{V}_l$)
$\mathcal{V}_{-k,-l}$	the subset of variable nodes except for $\mathcal{V}_k$ and $\mathcal{V}_l$
pa, ch	Parent, child nodes in a Bayesian network
nb	Neighboring nodes in a Markov network
Θ	Parameter set for a graph
D	Data, $D = \{\mathcal{V}_1^m, ..., \mathcal{V}_K^m\}_{m=1}^M$ contains M samples of K variables in a graph
N_x	the number of times that some event x is presented in the data
α_x	hyperparameter for each event x when using the Dirichlet priors

Supplemental Texts

Psudocodes for algorithms

Algorithm 1 LoPS induction algorithm

Require: Visual input image time series $D_I = \{I^t\}_{t=1}^T$, joystick action time series $D_a = \{a^t\}_{t=1}^T$
Ensure: LoPS Grammar $\mathcal{G}$

Feature extraction (Algorithm 2): Extract state $D_{\boldsymbol{s}}$ and utility $D_{\boldsymbol{u}}$ from D_I.
Strategy fitting (Algorithm 3): Generate strategy time series D_π from $D_{\boldsymbol{u}}$ and D_a
Grammar induction (Algorithm 4): Induce LoPS grammar $\mathcal{G}$ from $D_{\boldsymbol{s}}$ and D_π
return $\mathcal{G}$

Algorithm 2 Feature extraction algorithm

Require: Visual input image time series $D_I = \{I^t\}_{t=1}^T$
Ensure: State time series $D_{\boldsymbol{s}}$, Utility time series $D_{\boldsymbol{U}}$

$D_{\boldsymbol{s}} \leftarrow \{\}$
$D_{\boldsymbol{U}} \leftarrow \{\}$
for $t = 1$ to T **do**
    Extract the location of the Pac-Man l, two ghosts (l_{g1}, l_{g2}), energizers l_e, and pellets l_p from Image I^t
    Compute all possible trajectories from Pac-Man's current location in four moving actions.
    Compute the utility for each strategy $\boldsymbol{U}^t = (\boldsymbol{U}_{gl}, \boldsymbol{U}_{lo}, \boldsymbol{U}_{en}, \boldsymbol{U}_{ap}, \boldsymbol{U}_{ev_1}, \boldsymbol{U}_{ev_2}, \boldsymbol{U}_{ne})$
    Compute $s_b, s_e, s_{g_1}, s_{g_2}$ with all locational variables
    Identify the mode variables s_{m_1}, s_{m_2} according to the ghosts' color
    Discretinize state variables $\boldsymbol{s}^t = (s_b, s_e, s_{g_1}, s_{g_2}, s_{m_1}, s_{m_2})$ according to Table S3
    Add $\boldsymbol{s}^t$ to $D_{\boldsymbol{s}}$
    Add $\boldsymbol{U}^t$ to $D_{\boldsymbol{U}}$
end for
return $D_{\boldsymbol{s}}$, $D_{\boldsymbol{U}}$

Algorithm 3 Strategy fitting algorithm

Require: Visual input image time series $D_I = \{I^t\}_{t=1}^T$, Utility time series $D_{\boldsymbol{U}} = \{\boldsymbol{U}^t\}_{t=1}^T$, joystick action time series $D_a = \{a^t\}_{t=1}^T$
Ensure: Strategy time series $D_\pi = \{\pi^t\}_{t=1}^T (\pi^t \in \mathbb{Z}^7)$

$D_\pi \leftarrow \{\}$
Formulate fine-grained time windows $\Delta = \delta_1, \delta_2, ...\delta_K$ according to events get from D_I
for k = 1 to K **do**
    Get the utility u corresponding to δ_k from $D_{\boldsymbol{U}}$
    Get the action a^* corresponding to δ_k from D_a
    Get the strategy weight $\boldsymbol{w}_\pi$ and prediction error $Error$ by minimizing Eq 5
    if all a_t^* is empty **then**
        $\pi_k \leftarrow stay$
    else
        $\pi_k \leftarrow \text{argmax}(\boldsymbol{w}_\pi)$
    end if
    If $\pi_k \neq \pi_{k-1}$, Add π_k to D_π
end for
return D_π

Algorithm 4 Grammar induction algorithm

Require: State time series $D_{\boldsymbol{s}} = \{\boldsymbol{s}^t\}_{t=1}^{T} (s^t \in \mathbb{Z}^{|s|})$, strategy time series $D_\pi = \{\pi^t\}_{t=1}^{T} (\pi^t \in \mathbb{Z}^{|\pi|})$
Ensure: LoPS grammar $\mathcal{G}$

$G_1 =$ Hybrid network learning(D_s, D_π)
initialize G_g and G'_g with all 1st-order rules given by G_{g_1}
repeat
    $G_g \leftarrow G'_g$
    $D_g \leftarrow Parse(D_\pi, G_g)$
    for all g_i, g_j such that $g_i, g_j \in G_g$ and $g_i \neq g_j$ **do**
        $D_{s_i}, D_{s_j} \leftarrow$ extract strategy-related states data from D_g according to G_{g_1}
        score$(g_j) \leftarrow$ BDS(D_{π_j}, D_{s_j})
        score$(g_i, g_j) \leftarrow$ BDS$(D_{\pi_j}, \{D_{\pi_i}, D_{s_j}\})$
        if score$(g_i, g_j) >$ score(g_j) **then**
            construct a new rule $g' = \{g_i \rightarrow g_j\}$
            Add rule g' into G_g as a new graph G'_g
        end if
    end for
until $G_g = G'_g$
$\mathcal{G} \leftarrow G_g$
return $\mathcal{G}$

Algorithm 5 Parsing algorithm $Parse$

Require: Strategy time series $D_\pi = \{\pi^t\}_{t=1}^{T}$, Grammar $\mathcal{G}$
Ensure: Parsed rule time series D_g

$t \leftarrow 0$
$D_g \leftarrow \{\}$
while $t < T$ **do**
    $l_{max} \leftarrow 0$
    $g^t \leftarrow \pi^t$
    for all $g_i \in \mathcal{G}$ **do**
        calculate the depth of a rule $l \leftarrow len(g_i)$
        if $\pi^{t:t+l} = g_i$ and $l > l_{max}$ **then**
            $g^t \leftarrow g_i$
            $l_{max} \leftarrow l$
        end if
    end for
    Add g^t to D_g
    $t \leftarrow t + l_{max}$
end while
return D_g

Algorithm 6 PC Algorithm

Require: Data for K nodes with M samples, $D = \{\mathcal{V}_1^m, \mathcal{V}_2^m, ..., \mathcal{V}_K^m\}_{m=1}^{M}$
Ensure: Markov graph G

Initialize undirected graph G with edges between all pairs of variables
$i \leftarrow 0$
while all nodes have $> i$ neighbours **do**
    for $\mathcal{V}_k \in \mathcal{V}$ **do**
        for $\mathcal{V}_l \in nb(\mathcal{V}_k)$ **do**
            for all subset $\mathcal{V}_{-k,-l}$ of size i of the neighbours of $\mathcal{V}_k$ **do**
                if $CDT(\{\mathcal{V}_1^m, \mathcal{V}_2^m, \mathcal{V}_{-k,-l}^m\}_{m=1}^{m=M})$ is False **then**
                    Remove the edge between $\mathcal{V}_k$ and $\mathcal{V}_l$ from G
                end if
            end for
        end for
    end for
    $i \leftarrow i + 1$
end while
return G

Algorithm 7 Bayesian network learning

Require: Upstream nodes data $D' = \{\mathcal{V}_1^{'m}, \mathcal{V}_2^{'m}, ..., \mathcal{V}_L^{'m}\}_{m=1}^{M}$, downstream nodes data $D = \{\mathcal{V}_1^m, \mathcal{V}_2^m, ..., \mathcal{V}_K^m\}_{m=1}^{M}$, upstream nodes graph G'
Ensure: Bayesian network G
$\mathcal{V} \leftarrow \{\mathcal{V}_1^{'}, ..., \mathcal{V}_L^{'}, \mathcal{V}_1, ..., \mathcal{V}_K\}$
$\mathcal{E} \leftarrow \{\}$
$G \leftarrow \{\mathcal{V}, \mathcal{E}\}$
for $\mathcal{V}_k \in \mathcal{V}$ **do**
  for $\mathcal{V}_l^{'} \in \mathcal{V}^{'}$ **do**
    score$(\mathcal{V}_k, \mathcal{V}_{nb(l)}^{'}) \leftarrow BDS(\{\mathcal{V}_k^m\}_{m=1}^{M}, \{\mathcal{V}_{nb(l)}^{'m}\}_{m=1}^{M})$
    score$(\mathcal{V}_k, \{\mathcal{V}_l^{'}, \mathcal{V}_{nb(l)}^{'}\}) \leftarrow BDS(\{\mathcal{V}_k^m\}_{m=1}^{M}, \{\mathcal{V}_l^{'m}, \mathcal{V}_{nb(l)}^{'m}\}_{m=1}^{M})$
    if score$(\mathcal{V}_k, \{\mathcal{V}_l^{'}, \mathcal{V}_{nb(l)}^{'}\}) >$ score$(\mathcal{V}_k, \mathcal{V}_{nb(l)}^{'})$ **then**
      Update graph G by Adding a directed edge from node $\mathcal{V}_l^{'}$ to node $\mathcal{V}_k$
    end if
  end for
end for
return G

Algorithm 8 Hybrid network learning

Require: Upstream nodes data $D' = \{\mathcal{V}_1^{'m}, \mathcal{V}_2^{'m}, ..., \mathcal{V}_L^{'m}\}_{m=1}^{M}$, downstream nodes data $D = \{\mathcal{V}_1^m, \mathcal{V}_2^m, ..., \mathcal{V}_K^m\}_{m=1}^{M}$
Ensure: Bayesian network G
$G' \leftarrow MarkovNetworkLearning(D')$
$G \leftarrow BayesianNetworkLearning(D', D, G')$
return G

Algorithm 9 Bayesian Dirichlet Score BDS

Require: Child nodes data $D_{\mathcal{V}_k} = \{\mathcal{V}_k^m\}_{m=1}^{m=M}$, parent nodes data $D_{\mathcal{V}_{pa(k)}} = \{\mathcal{V}_{pa(k)}^m\}_{m=1}^{m=M}$, BDeu prior α
Ensure: Bayesian Dirichlet Score
Get the number of times that $\mathcal{V}_k$ is in state i and its parents in state j, $N_{ki|j}$, from the data
if $D_{\mathcal{V}_{pa(k)}}$ is empty **then**
  Compute score$(\mathcal{V}_k)$ according to Eq 20
else
  Compute score$(\mathcal{V}_k, \mathcal{V}_{pa(k)})$ according to Eq 19
end if
return score

Algorithm 10 Conditional dependence test CDT

Require: $D = \{\mathcal{V}_k^m, \mathcal{V}_l^m, \mathcal{V}_{-k,-l}^m\}_{m=1}^{m=M}$
Ensure: If node $\mathcal{V}_k$ and node $\mathcal{V}_l$ are independent conditioned on all other variables $\mathcal{V}_{-k,-l}$
Compute $P(D \mid \mathcal{H}_{indep})$ accroding to Eq 11
Compute $P(D \mid \mathcal{H}_{dep})$ accroding to Eq 12
return $\frac{P(D|\mathcal{H}_{dep})}{P(D|\mathcal{H}_{indep})} > 1$