

A 3D Characterization Framework for Intelligent Sequential Decision Making

Sadig Gojayeve, Carolina Fortuna

Department of Communication Systems, Jožef Stefan Institute, SI-1000 Ljubljana, Slovenia
 {sadig.gojayeve, carolina.fortuna}@ijs.si

Abstract—Puzzles are widely used to evaluate the reasoning capabilities of artificial intelligence (AI) systems for sequential decision making, yet approaches originating from different paradigms are rarely compared under unified conditions. To address this gap, we introduce a three-dimensional characterization framework that enables the analysts of AI methods by 1) projecting them to the Markov decision process (MDP) sequential decision making formalism, 2) degree of autonomy through human prior ranking of their designs and, 3) skill and computational cost. Using this framework, we analyze how representative graph-based, reinforcement learning, and large language model (LLM)-based approaches differ in their design choices and performance characteristics, instantiated respectively by Neurosolver, forward-backward reinforcement learning (FBRL), and automated thought-of-search (AutoToS), including a double-agent extension of thought-of-search (DA-ToS). The analysis relies on the Tower of Hanoi puzzle that provides a controlled benchmark with well-defined rules and scalable complexity, enabling consistent comparison across increasing problem sizes. The 3D characterization reveals that LLM-based methods, due to their weakly constrained action-space design, shift complexity from architecture to inference-time verification, leading to substantially higher memory and runtime costs than Neurosolver and FBRL.

I. INTRODUCTION

The rise of agentic artificial intelligence (AI) [1] is expected to significantly impact the economy and society as a whole. While agents relying on AI techniques have been around for several decades [2], the new generations of such agents are designed around the recent breakthrough in auto-regressive models, namely large language models (LLMs) [3]. These models, based on transformer architectures [4], essentially learn a probabilistic language model that, conditioned by an input generates an output. These models exhibit broad coverage over human-generated text because they are trained on massive web-scale corpora. Whether the training process enables them to go beyond memorization towards eliciting abstraction and generalization that would subsequently enable reasoning is still under investigation [5]. To improve reasoning, various training and prompting approaches such as reinforcement learning [6] and chain of thought [7] were proposed, however their reasoning capabilities are mostly unclear [8].

Reasoning capabilities of models are often evaluated on puzzles [9], [10], [8]. Some works focus more on logic puzzles [9] while others focus on text-based puzzles, classified into numeric, sequence, and physics categories [10]. Towers of Hanoi (ToH) is one recently used puzzle to investigate reasoning

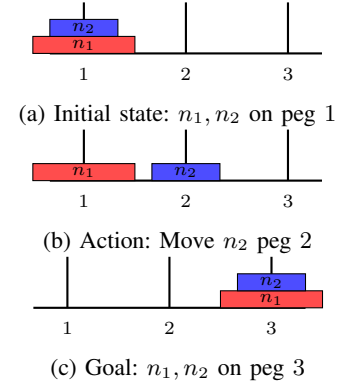


Fig. 1: ToH for $n = 2$ disks and $p = 3$ pegs.

differences between LLMs and LRMs [8]. ToH is also widely used in education to develop algorithmic thinking through inductive and recursive approaches [11], [12]. It also has wide applicability beyond education, being used in logistics and robotics, where weighted variants have been proposed to determine optimal move sequences with minimal cost [13], [14], [15].

ToH involves discs n and pegs p , where all the discs are initially placed on the first peg, and the goal is to move all the discs to the last peg. The key limitation is that only a smaller disc can be placed on top of a larger disc, as illustrated in Figure 1. To solve this puzzle, at each step, a human or AI player needs to understand the current state and take an action that will transition the puzzle to the next state until it is solved. This sequential decision making is referred to as a Markov decision process (MDP) and some AI approaches, especially the reinforcement learning ones, explicitly leverage the MDP formalism. Over time, several AI techniques, other than LLMs and LRMs, have been used to solve the ToH. For instance, [16] employed reinforcement learning, [17] approached it with logical neural networks, [18] proposed graph-based solvers, while [19] investigated neural cognitive modeling.

Currently, it remains unclear what the design trade-offs are when selecting one class of AI models over another for scalable sequential decision making problems such as ToH. State-of-the-art studies mostly focus on decision-quality comparisons between LLMs and RL [20], LLMs and AI planners [21]. Beyond performance metrics, very recent work

[22] provides theoretical formulations that cast LLM- and AI planners within MDP for optimization. However, while these efforts highlight differences across paradigms, they largely lack an explicit linkage between formulation choices, decision-making skill, computational cost, and the level of human-imposed biases.

To fully characterize an AI agent, one must consider three key aspects: (1) how the agent represents the world, (2) the degree of external guidance or supervision it receives, and (3) the computational cost required. In this paper, we propose a three-dimensional characterization framework that provides a structured and systematic basis for comparing different AI methods. The framework captures MDP formulation projecting to understand each method’s approach to sequential decision making, human prior to provide clearer insight into required guidance, and skill and computational cost to illustrate resource requirements. To illustrate the applicability of the proposed framework we employ it for comparing selected AI techniques. Our key contributions are as follows:

- We introduce a 3D characterization framework for sequential decision making methods that jointly analyzes (i) technique projection to the MDP-formalism, (ii) human prior across architecture/training/inference, and (iii) skill–cost trade-offs.
- We propose an operational human-prior rubric with explicit levels (High/Medium/Low) for representation/mechanism, knowledge/learning, and guidance/validation, enabling qualitative ranking across paradigms.
- We instantiate the framework on ToH by comparing a graph-based solver (Neurosolver) [18], a RL method Forward-backward reinforcement learning (FBRL) [16], and LLM-based planning via Automated Thought-of-Search (AutoToS) [23], including a new Double-Agent Thought-of-Search (DA-ToS) variant that modularizes code generation and repair.
- We show that weakly constrained LLM action spaces shift complexity from model design to inference-time verification loops, resulting in substantially higher run-time/memory than graph-based/RL approaches under the same task scaling.

This paper is organized as follows. Section II summarizes related work, Section III introduces the three dimensional characterization framework while Section IV elaborates on the methods. Sections V, VI and VII analyze the selected AI methods from the perspective of the proposed framework. Finally, Section VIII concludes the paper.

II. RELATED WORK

Comparisons of sequential decision making methods across different paradigms and backgrounds usually focus on outcome-based metrics, such as success rate or optimality. The UNO Arena [21] benchmark evaluates agents from reinforcement learning, heuristic, and LLM-based approaches by emphasizing decision-level quality, using metrics such as hit rate and decision rank rather than only win rate. For puzzle-solving

tasks, PUZZLES [9] provides a benchmark of logical puzzles for neural reasoning in reinforcement learning, evaluating various RL algorithms primarily through success rate. PUZZLEWORLD [24], in contrast, evaluates open- and closed-source LLMs by focusing on stepwise accuracy, where models are tasked with producing step-by-step reasoning toward a final solution. Recent work by Wang et al. [20] compares human decision-making with LLM and reinforcement learning agents, evaluating coordination behavior and path cost under identical environmental conditions. Overall, these comparisons adopt a largely single computational perspective, focusing primarily on result-based measures such as inference quality, even across different paradigms. They emphasize external performance metrics while lacking explicit analysis and direct linkage to representation-level action spaces, architecture-level transition mechanisms, and human guidance, leaving internal decision-making processes treated as black boxes.

Beyond purely computational evaluations, theoretical efforts such as FRAP [22] decompose AI planning and reinforcement learning algorithms into explicit dimensions (e.g., root states, trials, bootstrapping, backups) and analyze them through the lens of MDP optimization. Similarly, Burden et al. [25] classify AI evaluation approaches into paradigms such as benchmarking and exploratory evaluation. While these frameworks provide valuable formulations for comparing methods across paradigms, they do not offer insight into decision-making skill and computational cost in relation to MDP formulation. As a result, it remains unclear how action-space design, human guidance, and architectural constraints affect inference quality and computational cost.

There is a lack of unified frameworks across different paradigms, particularly between reinforcement learning and large language model-based approaches. Most comparisons focus on outcome metrics, rather than the formulation of sequential decision making (e.g., MDPs) and the role of human insight in shaping model behavior.

III. PROPOSED 3D CHARACTERIZATION FRAMEWORK

In this section, we introduce a three-dimensional framework for comparing AI methods used in sequential decision making. The framework focuses on three aspects: how each method projects to the elements of a MDP, what is the level of human prior introduced throughout the AI model development, and what skill vs computational trade-offs it involves.

A. Projecting through Markov Decision Process

The first dimension, projecting to MDP, clarifies how each method handles the core components of sequential decision making—such as state representation, transitions, rewards, and goal identification. Understanding this projection shows how a method reasons about progress and how it identifies correct or optimal next steps. Sequential decision making is typically formulated as a MDP [26], which is defined as $\mathcal{M} = \langle S, A, T, R, G \rangle$, where states (S) denote environment configurations, actions (A) specify possible moves, the transition function (T) gives the probability of reaching s'

after taking action a in state s , the reward function (R) assigns values to transitions, and goal states (G) indicate task completion.

State space representation refers to how the MDP state space S is specified in the sense of MDP. The levels of human prior are defined as follows:	
High	Humans explicitly design the state representation, fully specifying its structure and semantics, as in symbolic planners based on PDDL [27].
Medium	Humans prescribe a high-level representational structure or template, while allowing the system to learn internal parameters or details, as in logical neural networks or partially structured encodings [17].
Low	No explicit representational structure is imposed by humans; the model learns state representations end-to-end from interaction or data, as in fully end-to-end reinforcement learning systems trained via self-play [28].
Mechanism refers to the algorithmic structure responsible for reasoning and inference. The levels of human prior are defined as follows:	
High	Humans design the inference procedure, such as logical rules or search heuristics.
Medium	Humans select the general type of mechanism to employ, while the algorithm handles the details; for example, in Value Iteration Networks [29], humans prescribe a value-iteration-based planning structure, while the network learns the reward and transition representations end-to-end.
Low	Humans provide only high-level instructions, such as requesting an LLM to “generate code for running a DDQN,” without specifying internal structure.

TABLE I: Levels of Human Prior in Architecture Design.

B. Human Prior in AI for Sequential Decision Making

The second dimension, human prior, reflects how much human know-how is embedded in a method during its development: architecture design, training, and inference stages. This helps assess the degree of intrinsic intelligence that characterizes a method. We propose a categorization of human prior in AI techniques for sequential decision making focusing on three major aspects: architecture design, model training and model inference as follows.

a) Architecture Design: The architecture design involves the development of the structure of the model prior to learning or execution. We identify two components in the design: *Representation* and *Mechanism* as defined in Table I.

b) Training Stage: The training stage describes how a model acquires skill and how humans shape this process. It consists of two substages: *Knowledge Input* and *Learning* as defined in Table II.

c) Inference Stage: Inference describes how the model behaves once training is complete. It includes how humans guide the model, how decisions are produced, and how outputs are checked. It consists of three substages: *Guidance*, *Decision* and *Validation* as defined in Table III.

C. Performance and Computational Cost

The third dimension, focuses on assessing decision making skill versus computational costs required to provide that level of performance. By performance we refer to AI specific metrics measuring model performance at a single step, such as mean squared error distance or F1 or reward and regret

Knowledge Input concerns how training data, constraints, or reward structures are introduced, directly shaping the transition function T and reward function R in MDP formulation.	
High	Humans explicitly specify the training data, define relevant constraints, and design or shape reward structures, fully guiding how the model learns from examples, as in reinforcement learning with human-structured environments and hand-crafted training scenarios (e.g., Q-learning with expert-designed tasks).
Medium	Humans provide curated or loosely structured data without fully specified objectives, allowing the system to organize patterns and infer useful structure while still retaining partial human guidance, such as transformers trained on human-curated datasets.
Low	The model autonomously extracts training signals from large-scale interaction or self-generated experience, with minimal human specification beyond initial setup, as in self-supervised pretraining of large language models on unlabeled data [30].
Learning reflects how the model updates its parameters in response to the training signal and the extent of human involvement in this process.	
High	Humans explicitly shape the optimization process through curated objectives, staged training procedures, or tightly controlled learning phases, as in reinforcement learning from human feedback (RLHF) for large language models.
Medium	Humans specify general optimization settings—such as learning rates, network architectures, or update rules—while the system autonomously manages the internal learning dynamics, as in Double Deep Q-Networks (DDQN) [31].
Low	Learning proceeds end-to-end with minimal human involvement beyond initialization, with the system autonomously generating training signals, as in self-play reinforcement learning [28].

TABLE II: Levels of Human Prior at Training Stage.

characterizing the entire decision sequence. By computation cost we refer to metrics such as memory footprint, floating point operations, time and energy. This axis goes beyond measuring single domain specific performance metric towards a more holistic cost benefit analysis. This proposal is in line with prior work [20] that has emphasized the importance of understanding computational behavior across planning, reinforcement learning, and LLM-based reasoning, and broader overviews [22] of sequence models in sequential decision making that discuss how model architecture and training dynamics impact performance and cost. In this paper, we focus on time and memory consumption as the primary indicators of computational cost. In addition to measured time and memory, we also provide theoretical memory estimates for each selected method. For reproducibility purposes, we publish the code¹.

IV. METHODS

This section presents the selection of diverse AI techniques to be analyzed according to the three-dimensional framework introduced in Section III. We consider LLM-based agents, a graph-based learning method, and a reinforcement learning-based approach.

A. LLM-based Agents

1) Thought-of-Search (ToS): AutoToS is a framework that automates search-based planning by using a large language model (LLM) to generate two core components: a successor function (`succ`), which enumerates valid next states, and a

¹GitHub link to be inserted after double blind review completes.

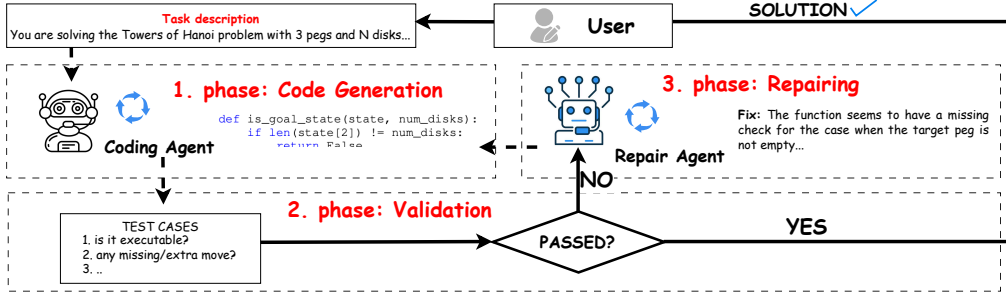


Fig. 2: DA-ToS Architecture Overview

Guidance captures the extent to which humans specify task details or constrain model behavior at runtime.	
High	Humans provide detailed task instructions, including explicit constraints, edge cases, and step-level guidance, often through fully specified prompts or rule descriptions, as in Least-to-Most prompting techniques [32].
Medium	Humans specify high-level task constraints or goals without defining the complete execution structure, such as providing domain rules (e.g., “a smaller disk cannot be placed beneath a larger one” in ToH) while leaving reasoning to the model.
Low	Humans provide only minimal task information—such as specifying the start and goal states in graph-based solvers—while the system autonomously determines the reasoning process and action sequence.
Decision describes how actions or outputs are selected by the system during inference and the extent of human involvement in this process.	
High	Humans explicitly drive or supervise the decision-making process, often through interactive or step-wise instructions. The model depends on human-provided reasoning steps, clarifications, or corrective feedback to determine each action, as in interactive planning with large language models (e.g., DEPS) [33].
Medium	Humans specify the decision framework or evaluation criteria—such as accuracy thresholds, selection rules, or preferred heuristics—while the system autonomously executes decisions within these constraints, as in safe reinforcement learning approach in constrained MDP [34].
Low	The system selects actions entirely autonomously based on its learned internal dynamics or probability distributions, without any human-defined decision logic at inference time, as in end-to-end transformer-based language models.
Validation describes how the correctness of model outputs is assessed and the extent of human involvement in this process.	
High	Humans manually inspect model outputs, identify errors, and provide corrective feedback after each execution.
Medium	Humans specify validation criteria or automated test suites, while the system autonomously executes these checks and uses the results to guide refinement or improving tests, as in TestGen-LLM framework [35].
Low	Validation and correction are performed entirely by the system itself, without human oversight, as in self-correcting large language models [36].

TABLE III: Levels of Human Prior at Inference Stage.

goal predicate (`isgoal`), which determines whether a state satisfies the task objective. In ToH domain, `succ` encodes legal disk moves, while `isgoal` identifies the configuration in which all disks are stacked on the target peg. The generated functions are embedded in an iterative refinement loop, where a Breadth-First Search (BFS) is used to test correctness and optimality. Detected errors, such as missing transitions or incorrect goal checks, are automatically converted into feedback

prompts for the LLM, which regenerates the functions until soundness and completeness are achieved.

2) *DA-ToS*: We extend AutoToS by introducing a dual-agent architecture that separates code generation from error interpretation. As illustrated in Figure 2, in DA-ToS *Coding agent* generates implementations of `succ` and `isgoal`, while a *Repair agent* analyzes compilation errors and test failures to produce targeted feedback. The system operates in a closed-loop workflow: generated code is validated against predefined test cases, failures are interpreted by the Repair agent, and corrective guidance is fed back to the Coding agent for refinement. This process repeats until all tests pass.

3) *Estimated Memory*: The total memory used by the model during inference mainly comes from two parts — the model weights and the key-value (KV) cache that stores hidden states for each token in the context. It can be expressed as $M_{\text{total}} \approx M_{\text{weights}} + M_{\text{KV}}$ where $M_{\text{weights}} = P \times b$ depends on the number of model parameters P and the precision b (bytes per parameter) and $M_{\text{KV}} \approx 2 \times L \times T \times d_{\text{model}} \times b$ is proportional with the number of L , hidden size d_{model} , and sequence length T (tokens in the conversation). The factor of 2 accounts for storing both keys and values. In practice, M_{KV} increases linearly with the token count, while M_{weights} remains constant, so the total memory grows slowly with longer interactions.

4) *Experimental Setting for LLM-based Agents*: We use standard (non-reasoning) LLMs instead of explicitly trained reasoning models, including low-parameter variants, to avoid introducing additional inductive biases from specialized reasoning supervision. In our approach, reasoning is externalized through search and verification, ensuring correctness independently of the model’s internal deliberation. This allows performance differences to be attributed to the planning framework rather than hidden reasoning priors, making standard LLMs a more neutral baseline.

In our implementation, we used DeepSeek Coder (deepseek-coder-v2-lite-instruct, 16B), Yi Coder (yi-coder-9b-chat, 9B), and Qwen Coder (qwen2.5-coder-32b-instruct, 32B) as Coding agents, while Gemma (8.5B) and Mistral (7.2B) served as Repair agents. By employing coder fine-tuned LLMs for the Coding agent and general-purpose LLMs for the Repair agent, we seek to improve the effectiveness of smaller-parameter coding models while reducing bias through cross-model vali-

dation.

Memory estimation is obtained by combining the model weight with an analytically computed key–value (KV) cache cost, using M_{KV} from Section IV-A3. The weight values are taken directly from the quantized GGUF models used in our experiments—DeepSeek (9.65 GB), Yi (4.96 GB), Qwen (11.47 GB), Gemma (4.67 GB), and Mistral (3.83 GB). The KV-cache is estimated assuming fp16 precision ($b = 2$ bytes), with architectural parameters $L = 40, d_{\text{model}} = 5120$ for DeepSeek, $L = 32, d_{\text{model}} = 4096$ for Yi and Mistral, $L = 64, d_{\text{model}} = 7168$ for Qwen, and $L = 28, d_{\text{model}} = 3072$ for Gemma, yielding a conservative upper-bound estimate where all tokens are assumed to remain in the context window.

B. Neurosolver

Neurosolver [18] is a graph-based planner that models puzzle state as a node with two divisions. States are encoded as strings, for example in ToH [123,000,000] for three disks stacked on the first peg. The upper division propagates goal-directed activity backward from the target state, tracing feasible transitions in reverse. The lower division encodes the current configuration and propagates signals forward to execute valid moves. Training builds the state space and transitions, the search phase spreads activation backward from the goal, and the solving phase reconstructs a valid path forward from the start state.

1) *Estimated Memory*: The memory requirements of Neurosolver can be understood from the way the puzzle’s state space grows as discussed in this section. Because Neurosolver stores both states and transitions (in forward and reverse directions), the memory footprint grows in proportion to both of these quantities. If we let α be the average storage cost per state and β the average storage cost per edge, the overall memory can be approximated as $M_{\text{neuro}}(n) \approx \alpha \cdot 3^n + \beta \cdot n \cdot 3^n = (\alpha + \beta n) 3^n$. This formula shows that memory usage increases *exponentially* with the number of disks (3^n), with an additional *linear factor* in n due to the growing number of possible moves.

2) *Experimental Setting for Neurosolver*: Neurosolver was implemented such that each disk-number configuration was trained using 100 iterations of random exploration. To ensure robustness, each configuration was trained and solved in ten independent runs, with results averaged across runs reported for metrics such as step count and execution time.

To estimate memory usage, M_{neuro} , each state is assigned a constant storage cost of $\alpha = 64$ bytes, corresponding to a compact fixed-size encoding (e.g., $4 \times 4 \times 4$ bytes per peg). Each transition is assigned $\beta = 16$ bytes, representing a stored reference (source–target identifiers). Since both forward and reverse transitions are stored, this cost is doubled using a duplication factor $\kappa = 2$. We further assume an average out-degree of $d = 3$ valid moves per state. These assumptions are combined to estimate the total memory footprint, which is reported in megabytes.

C. Forward-Backward Reinforcement Learning

FBRL [16] is a reinforcement learning method that combines forward and backward exploration during training. Built on the Double Deep Q-Network (DDQN) framework with a replay buffer, FBRL exploits both real and imagined experiences to accelerate learning in sparse-reward environments. In the forward step, the agent updates Q-values using real trajectories collected from the start state, while in the backward step it generates imagined trajectories by predicting reverse transitions from known goal states. Replay buffer is prioritized where transitions with higher temporal-difference errors are sampled more frequently [37].

1) *Estimated Memory*: The total memory consumption of FBRL can be decomposed into three principal components: the Q-network, the backward (reverse) model, and the replay buffer. The Q-network learns a mapping $Q(s, a; \theta)$, and its memory cost is determined by the number of parameters in that network. If the Q-network has L_Q hidden layers each of width h_Q , with input dimension $3n$ and output dimension $|A|$, then its memory cost can be approximated as $M_Q = \alpha_Q(3n \cdot h_Q + (L_Q - 1)h_Q^2 + h_Q \cdot |A|)$, where α_Q is the average cost per parameter (in bytes or memory units).

Similarly, the backward model—which predicts reverse transitions or goal-conditioned dynamics—has an analogous architecture. Let it have L_B hidden layers of width h_B . Then its memory cost is $M_B = \alpha_B(3n \cdot h_B + (L_B - 1)h_B^2 + h_B \cdot |A|)$ where α_B is its parameter storage cost.

The replay buffer stores past transitions (s, a, r, s') and thus scales proportionally with the number of stored entries. Denoting the buffer capacity by N and the average memory per stored tuple by γ , we write $M_R = \gamma N$. Adding up the three components, the total memory consumed by FBRL is $M_{\text{FBRL}}(n) \approx (\alpha_Q + \alpha_B)(3n h + (L - 1)h^2 + h n) + \gamma N$. This expression makes clear that while the network components scale roughly linearly with n (given fixed architecture), the replay buffer term γN is independent of n and may dominate memory usage for sufficiently large N .

2) *Experimental Setting for FBRL*: We implemented FBRL for ToH using a string-based state representation. Each disk is encoded according to its peg location, expressed as a binary tuple. For example, the state [(100), (100), (100)] denotes the initial configuration of a three-disk problem, where all disks are placed on the first peg. In the experimental setup, the replay buffer was set to a capacity of 10,000 transitions, and the DDQN learning rate was fixed at 5×10^{-4} .

To estimate the memory consumption of FBRL, we use M_{FBRL} introduced in this section, which decomposes total memory into the Q-Network, Backward Model, and Replay Buffer components. The estimation was based on the architecture parameters: number of disks $n = 4$, input dimension $d_{\text{in}} = 3n = 12$, action space $|A| = 3n = 12$, hidden widths $h_Q = 64$ and $h_B = 100$, and two linear layers per network ($L_Q = L_B = 2$). The replay buffer capacity was set to $N = 10,000$ transitions, with all values stored in 32-bit floating-point format (4 bytes each).

TABLE IV: Comparison of selected ToH models in terms of their State, Action, Transition, Reward, and Goal Representations.

Model	State (S)	Action (A)	Transition (T)	Reward (R)	Goal (G)
DA-ToS	$\langle \text{task prompt, code } c, \text{ generated feedback } F(c, e) \text{ (token sequences)} \rangle$	Invoke coding or repair LLM once	Probabilistic token generation: $P(y_{t+1:t+m} x_{1:t})$	1 = working code, 0 = non-working code	Verified <code>succ</code> and <code>isgoal</code> functions
AutoToS [23]	$\langle \text{task prompt, function, verifier feedback} \rangle$ (token prefixes)	Single LLM generation step	Probabilistic token generation: $P(y_{t+1:t+m} x_{1:t})$	1 = working code, 0 = non-working code	Verified <code>succ</code> and <code>isgoal</code> functions
Neurosolver [18]	$s \in \{1, 2, 3\}^n$ (symbolic disk-peg assignments)	Legal disk moves	Deterministic: $s' = T(s, a)$, guided by goal-conditioned value signal	Goal reward; penalties for illegal or redundant moves	All disks stacked on target peg (optimal $2^n - 1$ steps)
FBRL [16]	Bit string (disk positions)	Valid disk moves (bit changes)	Deterministic: $Q(s_t, A_t) \approx \mathbb{E}[R + \gamma \max Q(s_{t+1}, A)]$	+1 valid, -1 invalid, +10 for goal	Bit string for disks on Peg C (binary pattern)

V. ANALYSIS FROM THE MDP PERSPECTIVE

Along the first dimension of the proposed framework in Section III, in this section, we project the methods from Section IV to the elements of the MDP. The columns represent State, Action, Transition, Reward, and Goal, while each row corresponds to a method, as shown in Table IV.

DA-ToS: In DA-ToS, MDP is formulated over sequential language-model interactions between a Coding agent and a Repair agent as illustrated in Figure 3. Let t denote the iteration step. The initial state is defined as $s_0 = \langle \text{task prompt} \rangle$, where the task prompt specifies the problem instance (Boxes 1 and 2 from the Annex). At step t , the state is defined as $s_t = \langle \text{task prompt, } c_t, F(c_t, e_t) \rangle$, where c_t denotes the current candidate code implementation, e_t represents the corresponding evaluation signal (e.g., unit-test or verifier output), and $F(c_t, e_t)$ denotes the feedback generated by the Repair agent based on the code and its evaluation. The action space A consists of invoking a language model once. An action corresponds either to generating updated code (coding agent) or generating feedback $F(c_t, e_t)$ (repair agent). The transition function is induced by autoregressive generation. Let the internal token representation of s_t be denoted by $(x_1, \dots, x_n)$, where n is the current token length. The transition probability is defined as:

$$T(s_t, a_t) = P(y_{n+1:n+m} | x_{1:n}) = \prod_{i=n+1}^{n+m} P(y_i | x_{1:n}, y_{<i}), \quad (1)$$

where $y_{n+1:n+m}$ corresponds to the newly generated tokens. The next state is obtained by concatenation, $s_{t+1} = s_t \parallel (y_{n+1}, \dots, y_{n+m})$. The reward function is binary, $R(s_t) \in \{0, 1\}$, assigning 1 only when the generated planning functions satisfy soundness and completeness constraints, and 0 otherwise.

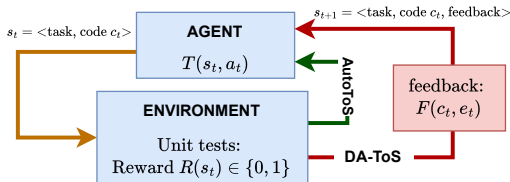


Fig. 3: LLM-based Agents Projection to MDP.

wise. Iteration proceeds until $R(s_t) = 1$, at which point the goal state is reached. The goal state G is defined as the state in which both `succ` and `isgoal` are verified to be sound and complete, ensuring that the ToH problem instance can be solved correctly with 100% validity.

AutoToS: In AutoToS, the MDP follows the same formulation as in DA-ToS (see Figure 3), but without a separate Repair agent providing feedback. Let t denote the iteration step. The initial state is $s_0 = \langle \text{task prompt} \rangle$. At step t , the state is defined as $s_t = \langle \text{task prompt, } c_t, e_t \rangle$, where c_t denotes the current candidate implementation (e.g., `succ`, `isgoal`), and e_t denotes the verification feedback obtained directly from predefined unit tests and BFS soundness checks. Unlike DA-ToS, no secondary model generates feedback; the verifier output e_t is directly appended to the state. The action space consists of a single language-model invocation conditioned on s_t . The transition dynamics follow the same autoregressive formulation as in Eq. 1, and the next state is obtained via token concatenation, $s_{t+1} = s_t \parallel (y_{n+1}, \dots, y_{n+m})$. The reward function remains binary, $R(s_t) \in \{0, 1\}$, with termination when $R(s_t) = 1$. The goal state G is reached when both `succ` and `isgoal` are verified to be sound and complete, ensuring correct solution of the ToH problem.

Neurosolver: In Neurosolver, the problem is modeled as a discrete MDP. The state space is defined as $s_n = \{1, 2, 3\}^n$, where a state $s = (s_1, \dots, s_n)$ encodes the peg position of each disk, and the initial state is $s_0 = \{123, 000, 000\}$, corresponding to all disks stacked on the source peg. The action space consists of all legal disk moves, and the transition function is deterministic, $T: s_n \times A \rightarrow s_n$, $s' = T(s, a)$, with transitions defined only for legality-preserving actions. Action selection is guided by a goal-conditioned value signal, such that $\pi(a | s, g) \propto \exp(V_g(s'))$, biasing exploration toward states closer to the goal. The reward function assigns maximal reward when the goal is reached, penalizes illegal or redundant moves, and otherwise reflects progress toward the target state. The goal state G is defined as the configuration where all disks are stacked on the destination peg, yielding the optimal solution length $2^n - 1$.

FBRL: In FBRL, the state space s is encoded as a bit string of length $3n$, where each group of three bits specifies the peg location of a disk. For example, with two disks, the initial state is represented as $[1, 0, 0, 1, 0, 0]$ (both disks on Peg

TABLE V: Human Prior ranking across Architecture, Training, and Inference Stages.

Method	Model	Architecture		Training		Inference		
		Representation	Mechanism	Knowledge	Learning	Guidance	Decision	Validation
DA-ToS	DeepSeek + Mistral	Medium	High	High	High	Medium	Low	Medium
	DeepSeek + Gemma			Medium	Medium			
AutoToS [23]	DeepSeek	Medium	High	High	High	High	Low	Medium
	Qwen			Medium	Medium			
	Yi			Medium	Medium			
Neurosolver [18]	Graph-based model	Low	High	Low	Low	Medium	Low	N/A
FBRL [16]	Reinforcement Learning	Low	High	High	Medium	Low	Low	N/A

A), and the goal state as $[0, 0, 1, 0, 0, 1]$ (both disks on Peg C). The action space A consists of legal disk moves, expressed as updates to the relevant 3-bit segment of the bit string while ensuring only valid transitions occur. The transition function T is deterministic; approximated with a DDQN, where transitions are sampled from a replay buffer and updated using the Bellman equation $Q(s_t, A_t) \approx \mathbb{E}[R + \gamma \max Q(s_{t+1}, A)]$. The reward function (R) assigns +1 for valid moves, -1 for illegal moves (e.g., placing a larger disk on top of a smaller one), and a terminal reward of +10 when the goal is reached. The goal state G is the bit-string configuration where all disks are successfully moved to Peg C.

Summary: The projecting of methods to the MDP formalism enables in depth analysis and understanding of their details. From the analysis it is possible to notice that the state space for the ToS/LLM-based methods consists of a sequence of tokens and it's significantly larger than in the case of Neurosolver and FBRL. The state, transition and action structure of the ToS/LLM rely on probabilistic distributions (input, output tokens and the LLM) while in the case of RL and graph based they rely on graphs encoding states in nodes and transitions on edges that dictate the action. The reward in ToS is binary realized through the verifier loop, while the other two rely on maximizing continuous reward.

VI. ANALYSIS FROM THE HUMAN PRIOR PERSPECTIVE

In this section, we analyze the degree of human prior embedded in each method, as defined in Section III-B. Table V reports a stage-wise comparison across Architecture, Training, and Inference, highlighting how human knowledge and design choices shape each system.

a) Architecture: In AutoToS and DA-ToS, representation is implicit and function-based (Table I): the model generates successor and goal-checking functions without an explicitly defined symbolic state encoding. This corresponds to a Medium level of human prior, since the task formulation and verification criteria are human-defined, while the internal state representation is learned implicitly by the language model. In contrast, Neurosolver and FBRL use low-level encodings that capture only disk positions on pegs, without higher-level semantics, resulting in a Low level of human prior.

For all methods, the reasoning mechanism, as defined in Table I, carries a High level of human prior. AutoToS and

DA-ToS rely on Transformer-based language models, whose architectures are fully human-designed. FBRL is built on the Double Deep Q-Network framework, which explicitly specifies the learning and decision-making mechanism. Neurosolver employs a graph-based planning mechanism with manually defined propagation rules. In all cases, the core reasoning process is explicitly designed by humans rather than discovered autonomously.

b) Training: For LLM-based methods (AutoToS and DA-ToS), the level of human prior in knowledge input as defined in Table II, depends on the underlying language model. DeepSeek, Yi, and Mistral are trained primarily on large-scale, human-curated datasets, which directly embed substantial human knowledge into the models, resulting in a High level of human prior [38], [39], [40]. Qwen and Gemma, in contrast, incorporate a significant amount of synthetic data and automated reinforcement learning signals during training, reducing direct human involvement in knowledge specification and leading to a Medium level of human prior [41], [42]. FBRL exhibits a High level of human prior, as humans explicitly define the reward structure, penalties, goal conditions, and environment constraints that directly shape the learning signal. In contrast, in Neurosolver, humans do not provide training data, rewards, or explicit transition constraints; the model explores valid moves autonomously based on basic puzzle rules, corresponding to a Low level of human prior.

In AutoToS and DA-ToS, the learning as defined in Table II, process depends on the underlying LLM. DeepSeek is trained primarily through large-scale supervised learning on human-curated code data, resulting in a High level of human prior. Qwen incorporates automated reinforcement learning with synthetic data generation, reducing direct human involvement and leading to a Medium level of human prior [41]. Yi relies on continued pretraining from an existing base model using curated data [39], which also corresponds to a Medium level. Among repair agents, Mistral is trained via supervised instruction tuning with human-curated datasets, yielding High human prior, whereas Gemma explicitly employs reinforcement learning from human feedback [42] (RLHF), introducing partial automation and resulting in a Medium level. In contrast, Neurosolver and FBRL do not use human feedback-based learning. Neurosolver learns autonomously through explo-

ration over valid state transitions, and FBRL updates its policy via predefined reinforcement learning rules. Since no human feedback is involved during learning, both correspond to a Low level of human prior.

c) Inference: Guidance, as defined in Table III, differs across methods in how task constraints are provided at inference time. AutoToS exhibits a High level of guidance, as human-provided test cases and error types are directly used to steer the refinement loop during inference. In DA-ToS, this guidance is partially automated: although human-defined test cases remain, error interpretation and correction are handled by a secondary language model, resulting in a Medium level of guidance. Neurosolver also exhibits Medium guidance, since human-defined legality rules constrain admissible moves during execution without interactive input. In contrast, FBRL operates with Low guidance, executing a learned policy without task-specific input or constraints at inference time.

For all evaluated methods, the level of human prior in decision-making as defined in Table III, is Low. Action selection and planning are performed by the systems themselves during inference, without direct human involvement across AutoToS, DA-ToS, Neurosolver, and FBRL.

Validation, as defined in Table III, differs substantially across methods. In AutoToS and DA-ToS, humans explicitly define test cases that are used to verify solution correctness, resulting in a Medium level of human prior in validation. In contrast, Neurosolver and FBRL do not employ an explicit validation stage: correctness is determined implicitly by goal satisfaction within the environment, making validation not applicable for these methods.

Summary: The human prior ranking of the methods makes a significant view on understanding autonomous level. In general, it can be clearly seen that ToS/LLM-based approaches need more human insight compare to FBRL/Neurosolver. Due to constrain-based architectural design, FBRL and Neurosolver don't need verification after final decision, this shows clear superiority compare to ToS/LLM-based methods. While all four paradigm include high human prior in their architectural design, FBRL and Neurosolver don't need verification after final decision, due to stricter constraining at architectural stage.

VII. ANALYSIS FROM THE SKILL AND COMPUTATIONAL COST PERSPECTIVE

In this section, we analyze the skill and computational cost of each method, as defined in Section III-C by focusing on the cost to solve the puzzle correctly. We consider the methods and experimental settings outlined in Section IV. In the analysis on ToH problem depicted in Figure 1 we assume three pegs and an number of disks $n \in 2, 3, \dots, 7$. Table VI presents a comprehensive comparison of experimental results across DA-ToS, AutoToS, FBRL, and Neurosolver, with all values reported as mean $\pm$ Median Absolute Deviation. The table reports the model configuration, including the LLM combinations used for LLM-based agents, the number of disks, the number of attempts or steps, token usage for LLM-based methods, total

execution time (including both training and solving), and both estimated and observed memory consumption.

A. LLM-based Agents

For LLM-based approaches, the reported results correspond to the combined generation of the Successor and Goal functions. As a result, the number of attempts (LLM call) reflects successful generation for both components, with one step required for the Goal function and one for the Successor function in the optimal case.

DA-ToS: For DA-ToS, it is evident that the double-agent design incurs higher computational cost in terms of both token usage and execution time. Although the DeepSeek+Gemma and DeepSeek+Mistral configurations exhibit comparable memory consumption, the Mistral-based setup is markedly more efficient, requiring fewer steps, fewer tokens, and shorter execution time, while also exhibiting lower variance across runs. Specifically, the DeepSeek+Gemma configuration consumes over 17k tokens and approximately 209 seconds, whereas DeepSeek+Mistral completes the task with roughly 6.5k tokens in about 32 seconds. The estimated memory consumption exceeds the observed usage, as the estimation assumes simultaneous high-precision execution of both agents. In practice, quantized deployments such as DeepSeek+Gemma (17.73 GB $\rightarrow$ 15.13 GB) and DeepSeek+Mistral (15.12 GB $\rightarrow$ 14.00 GB) use compressed weights and lighter key-value caches, resulting in a lower actual memory footprint.

Auto-ToS: For AutoToS, the results indicate a substantially lower computational overhead compared to DA-ToS, as the approach relies on a single LLM without an explicit repair agent. Across all configurations, AutoToS typically converges in one to three attempts, with lower token usage and shorter execution times. In particular, Qwen and Yi achieve the most efficient performance, requiring fewer than 1k tokens on average, completing execution in approximately 25–35 seconds, and consistently producing the optimal step size, whereas DeepSeek exhibits higher token usage and additional refinement cycles in the Successor function, leading to increased attempts and longer execution times. In terms of memory consumption, AutoToS follows the expected trend based on model size, with DeepSeek exhibiting the largest footprint (≈ 11.3 GB observed), followed by Qwen (≈ 12.0 GB) and Yi (≈ 6.0 GB). As with DA-ToS, estimated memory exceeds observed usage due to conservative precision and cache assumptions, while quantized execution reduces the actual footprint during inference.

B. Neurosolver

Across all configurations, Neurosolver consistently achieves the optimal number of steps ($2^n - 1$), while runtime and memory consumption increase with puzzle size. Runtime grows gradually from 12.06 seconds for two disks to 60.40 seconds for seven disks, remaining below one minute even for the largest instance. In contrast, memory consumption increases more sharply, from 0.005 MB to 0.321 MB, reflecting the exponential growth of the state space. While

TABLE VI: Comprehensive Performance and Memory Comparison across DA-ToS, AutoToS, FBRL, and Neurosolver. All values are reported as mean $\pm$ MAD*.

Model Setup	Disk	Att. / Steps	Tokens	Time (s)	Total E.Mem (MB)	Total O.Mem (MB)
DA-ToS						
DeepSeek + Gemma	n	6.8 ± 1.5	17998.2 ± 3552.5	209.21 ± 47.84	1.82×10^4	$1.55 \times 10^4 \pm 0$
DeepSeek + Mistral	n	5.0 ± 1.5	6718.4 ± 1571.0	32.69 ± 10.37	1.55×10^4	$1.43 \times 10^4 \pm 0$
AutoToS [23]						
DeepSeek	n	4.2 ± 1.0	5257.5 ± 975.0	23.61 ± 8.80	1.40×10^4	$1.16 \times 10^4 \pm 0$
Yi	n	2.0 ± 0.0	1162.3 ± 64.5	26.37 ± 5.06	5.47×10^3	$6.16 \times 10^3 \pm 0$
Qwen	n	2.0 ± 0.0	831.4 ± 3.5	33.56 ± 1.16	1.27×10^4	$1.23 \times 10^4 \pm 0$
Neurosolver [18]						
Graph-based	2	3	—	12.06 ± 0.04	1.37×10^{-3}	$5.41 \times 10^{-3} \pm 9.77 \times 10^{-7}$
	3	7	—	19.75 ± 0.18	4.12×10^{-3}	$5.62 \times 10^{-3} \pm 8.30 \times 10^{-6}$
	4	15	—	28.13 ± 0.17	1.24×10^{-2}	$1.40 \times 10^{-2} \pm 0$
	5	31	—	38.00 ± 0.35	3.71×10^{-2}	$2.51 \times 10^{-2} \pm 0$
	6	63	—	49.76 ± 0.45	1.11×10^{-1}	$1.05 \times 10^{-1} \pm 0$
	7	127	—	60.40 ± 0.15	3.34×10^{-1}	$3.30 \times 10^{-1} \pm 5.47 \times 10^{-5}$
FBRL [16]						
Reinforcement Learning	2	—	—	2.42 ± 0.07	6.65×10^{-1} (0.018 + 0.037 + 0.61)	$3.29 \times 10^{-1} \pm 6.25 \times 10^{-3}$ (0.004 + 0.005 + 0.32)
	3	—	—	7.68 ± 0.27	9.03×10^{-1} (0.020 + 0.043 + 0.84)	$7.03 \times 10^{-1} \pm 0$ (0.005 + 0.008 + 0.69)
	4	—	—	31.20 ± 2.02	1.14×10^0 (0.022 + 0.048 + 1.07)	$9.37 \times 10^{-1} \pm 0$ (0.007 + 0.010 + 0.92)

* MAD denotes the Median Absolute Deviation, defined as $\text{MAD} = \text{median}(|x_i - \text{median}(x)|)$, computed over repeated runs.

observed memory usage remains within the same order of magnitude as theoretical estimates, deviations are expected due to random exploration and higher-level state representations in the implementation. Overall, this behavior aligns with the theoretical scaling law from Section IV-B1, highlighting the trade-off between guaranteed optimality and rapidly growing memory requirements.

C. Forward-backward Reinforcement Learning

For FBRL, computational cost increases rapidly with the number of disks, primarily due to its dual exploration mechanism that combines forward trajectories from the initial state with backward rollouts from the goal. Runtime grows from approximately 2.4 seconds for two disks to over 31 seconds for four disks, reflecting the increased training effort required as the search space expands. In terms of memory consumption, the breakdown shown in the table highlights contributions from the Q-Network, Backward Model, and Replay Buffer. While the Q-Network and Backward Model remain lightweight and scale linearly with the input and output dimensions of $3n$, their memory usage is negligible compared to the replay buffer. In contrast, experience storage dominates the overall footprint, increasing from about 0.32 MB for two disks to approximately 0.92 MB for four disks. Both observed and estimated results indicate that FBRL’s memory behavior is driven primarily by replay buffer growth rather than network complexity, resulting in a predictable but increasingly costly memory profile.

Summary: The skill to solve the ToH puzzle and computational comparison illustrates quantitative insight about pros and cons in terms of memory and time. From the table, it is

clear that ToS/LLM-based approaches need large amount of memory resource compared to Neurosolver and FBRL. On the other hand the ToS/LLM-based ones are not limited by about disk numbers. Focusing on FBRL vs Neurosolver, it seems that Neurosolver has superior scalability performance since time and memory does not increase as much as for FBRL.

VIII. CONCLUSION

In this paper we addressed the challenge that existing evaluations of sequential decision making methods largely rely on isolated pros-cons analyses, which fail to reveal how design choices redistribute complexity across system components. To address this gap, we proposed a three-dimensional characterization that jointly captures MDP formulation, human prior, and skill-cost trade-offs. Using the proposed framework, three representative graph-based, reinforcement learning, and large language model (LLM)-based approaches and the Tower of Hanoi puzzle we found that in LLM-based methods, where the action space is weakly constrained by design, : 1) correctness is enforced through increased human insight at inference time, primarily via verification and correction loops and 2) complexity is shifted to computation, resulting in significantly higher memory and runtime costs compared to graph-based and reinforcement learning models. Our findings enable structured cross-paradigm comparison linking theoretical formulation with empirical performance. The framework provides a principled basis for analyzing such trade-offs across AI techniques even beyond the ToH.

REFERENCES

- [1] S. Murugesan, “The rise of agentic ai: implications, concerns, and the path forward,” *IEEE Intelligent Systems*, vol. 40, no. 2, pp. 8–14, 2025.
- [2] C. Castelfranchi, “Modelling social action for ai agents,” *Artificial intelligence*, vol. 103, no. 1-2, pp. 157–182, 1998.
- [3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [5] X. Wang, A. Antoniadis, Y. Elazar, A. Amayuelas, A. Albalak, K. Zhang, and W. Y. Wang, “Generalization v.s. memorization: Tracing language models’ capabilities back to pretraining data,” in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: <https://openreview.net/forum?id=IQxBDLmVpT>
- [6] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, “Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,” *arXiv preprint arXiv:2501.12948*, 2025.
- [7] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.
- [8] P. Shojaei, I. Mirzadeh, K. Alizadeh, M. Horton, S. Bengio, and M. Farajtabar, “The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity,” *arXiv preprint arXiv:2506.06941*, 2025.
- [9] B. Estermann, L. Lanzendörfer, Y. Niedermayr, and R. Wattenhofer, “Puzzles: A benchmark for neural algorithmic reasoning,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 127 059–127 098, 2024.
- [10] J. Hawkins, “Enigma: Generative text puzzles for evaluating reasoning in language models,” *arXiv preprint arXiv:2505.04914*, 2025.
- [11] P. Merriotsy, “The tower of hanoi and inductive logic,” *Australian Senior Mathematics Journal*, vol. 29, no. 1, pp. 16–24, 2015.
- [12] O. D. Anderson, “Induction, recursion, and the towers of hanoi,” *International Journal of Mathematical Education in Science and Technology*, vol. 23, no. 3, pp. 339–343, 1992.
- [13] J. A. H. Aguilar, A. R. P. Mayo, S. Y. Ortuño, A. Ochoa-Zezzatti, and J. C. P. Gallegos, “The application of hanoi towers game in logistics management,” in *Handbook of Research on Military, Aeronautical, and Maritime Logistics and Operations*. IGI Global Scientific Publishing, 2016, pp. 422–433.
- [14] E.-M. Mehiri and H. Belbachir, “The weighted tower of hanoi,” *Discrete Mathematics, Algorithms and Applications*, vol. 16, no. 05, p. 2350051, 2024.
- [15] G. Havur, K. Haspalamutgil, C. Palaz, E. Erdem, and V. Patoglu, “A case study on the tower of hanoi challenge: Representation, reasoning and execution,” in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 4552–4559.
- [16] A. D. Edwards, L. Downs, and J. C. Davidson, “Forward-backward reinforcement learning,” *arXiv preprint arXiv:1803.10227*, 2018.
- [17] D. J. Agravante, D. Kimura, and M. Tatsubori, “Learning neuro-symbolic world models with logical neural networks,” in *PRL Workshop Series Bridging the Gap Between AI Planning and Reinforcement Learning*, 2023.
- [18] A. Bieszcza and S. Kuchar, “Neurosolver learning to solve towers of hanoi puzzles,” in *2015 7th International Joint Conference on Computational Intelligence (IJCCI)*, vol. 3. IEEE, 2015, pp. 28–38.
- [19] T. Stewart and C. Eliasmith, “Neural cognitive modelling: A biologically constrained spiking neuron model of the tower of hanoi task,” in *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 33, no. 33, 2011.
- [20] L. Wang, Z. Jiang, C. Hu, J. Zhao, Z. Zhu, X. Chen, Z. Wang, T. Liu, G. He, Y. Yin *et al.*, “Comparing ai and human decision-making mechanisms in daily collaborative experiments,” *iScience*, vol. 28, no. 6, 2025.
- [21] Z. Qin, H. Wang, D. Liu, Z. Song, C. Fan, Z. Lv, J. Wu, Z. Lei, Z. Tu, D. Chu *et al.*, “Uno arena for evaluating sequential decision-making capability of large language models,” *arXiv preprint arXiv:2406.16382*, 2024.
- [22] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, “A unifying framework for reinforcement learning and planning,” *Frontiers in Artificial Intelligence*, vol. 5, p. 908353, 2022.
- [23] M. Katz, H. Kokel, K. Srinivas, and S. Sohrabi Araghi, “Thought of search: Planning with language models through the lens of efficiency,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 138 491–138 568, 2024.
- [24] H. Li, B. Jiang, A. Naehu, R. Song, J. Zhang, M. Tjandrasuwita, C. Ekbote, S.-S. Chen, A. Balachandran, W. Dai *et al.*, “Puzzleworld: A benchmark for multimodal, open-ended reasoning in puzzle hunts,” *arXiv preprint arXiv:2506.06211*, 2025.
- [25] J. Burden, M. Tešić, L. Pacchiardi, and J. Hernández-Orallo, “Paradigms of ai evaluation: Mapping goals, methodologies and culture,” *arXiv preprint arXiv:2502.15620*, 2025.
- [26] D. Z. Chen, P. Verma, S. Srivastava, M. Katz, and S. Thiébaux, “Ai planning: A primer and survey (preliminary report),” *arXiv preprint arXiv:2412.05528*, 2024.
- [27] T. Silver, V. Hariprasad, R. S. Shuttlesworth, N. Kumar, T. Lozano-Pérez, and L. P. Kaelbling, “Pddl planning with pretrained large language models,” in *NeurIPS 2022 foundation models for decision making workshop*, 2022.
- [28] Y. Bai and C. Jin, “Provable self-play algorithms for competitive reinforcement learning,” in *International conference on machine learning*. PMLR, 2020, pp. 551–560.
- [29] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, “Value iteration networks,” *Advances in neural information processing systems*, vol. 29, 2016.
- [30] R. R. Hegde and S. Sharma, “Self supervised llm customizer (sslc): Customizing llms on unlabeled data to enhance contextual question answering,” in *Proceedings of the 4th International Conference on AI-ML Systems*, 2024, pp. 1–11.
- [31] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.
- [32] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. Le *et al.*, “Least-to-most prompting enables complex reasoning in large language models,” *arXiv preprint arXiv:2205.10625*, 2022.
- [33] Z. Wang, S. Cai, G. Chen, A. Liu, X. S. Ma, and Y. Liang, “Describe, explain, plan and select: interactive planning with llms enables open-world multi-task agents,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 34 153–34 189, 2023.
- [34] A. Wachi and Y. Sui, “Safe reinforcement learning in constrained markov decision processes,” in *International Conference on Machine Learning*. PMLR, 2020, pp. 9797–9806.
- [35] N. Alshahwan, J. Chheda, A. Finogenova, B. Gokkaya, M. Harman, I. Harper, A. Marginean, S. Sengupta, and E. Wang, “Automated unit test improvement using large language models at meta,” in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, 2024, pp. 185–196.
- [36] S. Welleck, X. Lu, P. West, F. Brahman, T. Shen, D. Khashabi, and Y. Choi, “Generating sequences by learning to self-correct,” *arXiv preprint arXiv:2211.00053*, 2022.
- [37] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.
- [38] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Li *et al.*, “Deepseek-coder: When the large language model meets programming—the rise of code intelligence,” *arXiv preprint arXiv:2401.14196*, 2024.
- [39] A. Young, B. Chen, C. Li, C. Huang, G. Zhang, G. Zhang, G. Wang, H. Li, J. Zhu, J. Chen *et al.*, “Yi: Open foundation models by 01. ai,” *arXiv preprint arXiv:2403.04652*, 2024.
- [40] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, “Mistral 7b,” *arXiv preprint arXiv:2310.06825*, 2023.
- [41] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu *et al.*, “Qwen2. 5-coder technical report,” *arXiv preprint arXiv:2409.12186*, 2024.
- [42] G. Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love *et al.*, “Gemma: Open models based on gemini research and technology,” *arXiv preprint arXiv:2403.08295*, 2024.