Learning to Prompt in Unknown Environments: A POMDP Framework with Compositional Actions for Large Language Models

Hiroshi Fujinoki, Eren Gultepe, and Alex Towell
Department of Computer Science
Southern Illinois University Edwardsville
Edwardsville, IL, USA
{hfujinoki, egultep, atowell}@siue.edu

Abstract—We present a novel framework that addresses the fundamental challenge of optimizing prompts for large language models (LLMs) whose internal dynamics are unknown and unobservable. Unlike approaches that assume knowledge of the LLM's behavior, we formulate prompting as a **Partially Observable Markov Decision Process (POMDP)** where the controller must learn effective strategies purely through interaction. Our framework makes three key contributions. First, we introduce **compositional action spaces** leveraging a rich prompting library that provides over 10 million possible prompt combinations through eight orthogonal dimensions: cognitive operations, focus aspects, reasoning styles, connection types, output formats, meta-strategies, confidence levels, and reasoning depths. Despite this massive action space, our Bayesian network factorization reduces complexity to just thousands of learnable parameters. Second, we develop a **belief-based Q-learning architecture** that maintains uncertainty about the LLM's hidden reasoning state using recurrent neural networks, enabling model-free learning without access to transition or observation dynamics. The controller learns which compositional prompts are valuable through trial and error, adapting to specific LLM behaviors and quirks. Third, we propose an **external controller design** that treats the LLM as a black-box environment, enabling deployment across different LLMs (GPT-4, Claude, Llama) without modification. This separation of prompting intelligence from reasoning intelligence allows for rapid adaptation and online learning. Our approach provides a principled alternative to both handcrafted prompting and opaque fine-tuning methods, offering interpretability while acknowledging the fundamental unknowability of LLM internals. Experimental protocols demonstrate how model-free reinforcement learning can discover effective prompting strategies without ever understanding how or why they work.

Index Terms—Large Language Models, POMDP, Model-Free Reinforcement Learning, Compositional Actions, Belief States, Q-learning, Unknown Environments, Prompting Strategies, Black-Box Optimization

1 Introduction

Large language models represent a unique challenge for optimization: they are powerful reasoning engines whose internal dynamics remain fundamentally unknown to their users. When we prompt an LLM, we are essentially interacting with a black-box environment—we can observe its outputs but cannot access or understand the complex transformations occurring within its billions of parameters. This creates a profound learning problem: how can we discover effective prompting strategies when we don't know how or why they work?

This paper addresses this challenge by formulating LLM prompting as a Partially Observable Markov Decision Process (POMDP) where the environment dynamics are unknown and must be learned through interaction. Unlike traditional planning problems where models are given, or even standard reinforcement learning where states are observable, we face a doubly difficult setting: the LLM's true reasoning state is hidden, and its response patterns must be discovered through trial and error.

Contribution 1: Compositional Action Spaces for Unknown Environments. We leverage a comprehensive compositional prompting framework that provides over 10 million possible prompt combinations through eight orthogonal dimensions. This rich action space is essential for exploring the unknown environment—since we don't know a priori which prompts will be effective, we need a diverse set of actions to discover successful strategies. Our implementation (`rlpilot` package) includes cognitive operations (ω), focus aspects (ϕ), reasoning styles (σ), connection types (κ), output formats (τ), meta-strategies (μ), confidence levels (ρ), and reasoning depths (δ). Despite this massive action space, our Bayesian network factorization makes learning tractable by reducing parameter complexity from millions to thousands.

Contribution 2: Model-Free Learning with Belief States. We develop a belief-based Q-learning architecture that explicitly acknowledges the LLM as an unknown environment. The controller maintains a belief state—a learned representation of uncertainty about the LLM's hidden reasoning configuration—using recurrent neural networks. This belief state integrates observations (extracted features from LLM outputs) with action history to form hypotheses about what prompts might work next. Crucially, our Q-learning approach learns the value of compositional ac-

tions without ever modeling the LLM's transition dynamics $P(s'|s,a)$ or observation function $\Omega(o|s',a)$. The controller discovers effective strategies purely through reinforcement signals, adapting to each LLM's specific behavior patterns, biases, and quirks.

Contribution 3: External Controller Architecture for Black-Box LLMs. We propose treating LLMs as black-box environments that controllers learn to interact with, rather than models to be fine-tuned. This external controller architecture has profound implications: (1) it works with any LLM accessible via API without needing model weights, (2) it can adapt online to new LLMs or domains through continued learning, (3) it maintains a clean separation between the learned prompting policy and the LLM's reasoning capabilities, and (4) it enables rapid experimentation since controllers can be trained in hours rather than the weeks required for LLM fine-tuning. The controller essentially learns a "user manual" for each LLM through experience, discovering which compositional prompts elicit desired behaviors.

This work makes both theoretical and practical contributions to the challenge of optimizing interactions with unknown, complex systems. Theoretically, we show how the POMDP formulation with compositional actions provides a principled framework for learning in environments where the dynamics cannot be modeled. Practically, we demonstrate that model-free reinforcement learning can discover effective prompting strategies without understanding why they work—the controller learns that certain compositional actions lead to better outcomes without needing to comprehend the LLM's internal reasoning processes. This approach offers a middle path between handcrafted prompting (which requires human expertise) and end-to-end fine-tuning (which requires massive computation and loses interpretability), providing a systematic way to learn effective prompting strategies for any LLM treated as an unknown environment.

2 Related Work

2.1 Latent Chain-of-Thought Methods

Models such as OpenAI's o1/o3 and DeepSeek-R1 employ reinforcement learning to encourage latent chains-of-thought that improve final outputs. These models optimize internal reasoning through end-to-end training on vast datasets with minimal manual intervention.

2.2 Reinforcement Learning in LLMs

Techniques like RLHF and policy optimization have been successfully applied to fine-tune LLMs for improved reasoning. However, these approaches require access to model weights and expensive fine-tuning. Our approach differs fundamentally by treating the LLM as an unknown environment in a POMDP formulation, learning prompting policies through model-free Q-learning without any access to the LLM's internals. This enables working with black-box, API-only models while adapting to their specific behaviors through interaction.

2.3 Structured Prompting Approaches

Recent work has explored various structured approaches to prompting:

Tree of Thoughts (ToT) extends chain-of-thought prompting by exploring multiple reasoning paths simultaneously, using tree search algorithms to navigate the space of possible solutions. While ToT provides systematic exploration, it relies on fixed search algorithms rather than learned policies.

ReAct combines reasoning with action, allowing models to interleave thought processes with external tool use. Our approach generalizes this by learning when and how to invoke different reasoning strategies and tools.

DSPy provides a framework for programming—rather than prompting—language models, with modules for different prompting techniques and optimizers for improving prompt selection. Unlike DSPy's gradient-free optimization, our Q-learning approach can leverage temporal difference learning and value function approximation.

Our compositional action framework can be viewed as learning to orchestrate these various prompting strategies, selecting and combining them based on the current state rather than using fixed heuristics.

2.4 Learning in Unknown Environments

Our work connects to the broader challenge of learning in unknown environments, a fundamental problem in reinforcement learning. The "bitter lesson" (Sutton, 2019) suggests that general learning methods ultimately outperform handcrafted approaches. We embrace this principle by using model-free Q-learning to discover prompting strategies without assuming knowledge of LLM dynamics. However, we structure the action space compositionally to make learning tractable—acknowledging that some inductive bias is necessary when learning in high-dimensional, partially observable environments. The compositional structure provides a rich yet manageable space for exploration, allowing the controller to discover effective strategies through experience rather than human engineering.

3 Contribution 1: Compositional Action Spaces

3.1 The Compositional Action Problem

Before presenting our specific approach, we develop a general framework for compositional action spaces in reinforcement learning. Consider an agent that must select actions from a space that naturally decomposes into n orthogonal components:

$$\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \ldots \times \mathcal{A}_n$$

where each $\mathcal{A}_i$ represents a distinct dimension of the action. In the context of LLM prompting, these might represent different linguistic or cognitive aspects of the prompt construction.

The fundamental challenge is how to model the policy $\pi(a|s)$ over this compositional space. We analyze three approaches:

3.1.1 Fully Joint Distribution

The most general approach treats the action as a monolithic unit:

$$\pi(a|s) = P(a_1, a_2, \ldots, a_n|s)$$

Parameters: $O\left(\prod_{i=1}^{n} |\mathcal{A}_i|\right)$

This approach can capture arbitrary correlations between components but suffers from the curse of dimensionality. For even modest component spaces (e.g., 10 choices per component with 5 components), this requires $10^5 = 100,000$ parameters per state.

3.1.2 Fully Factored Distribution

At the opposite extreme, we assume complete independence:

$$\pi(a|s) = \prod_{i=1}^{n} P(a_i|s)$$

Parameters: $O\left(\sum_{i=1}^{n} |\mathcal{A}_i|\right)$

While tractable, this assumes that action components are selected independently, which may produce invalid or incoherent combinations.

3.1.3 Structured Factorization via Bayesian Networks

We propose modeling the action distribution as a Bayesian network, where edges encode conditional dependencies:

$$\pi(a|s) = \prod_{i=1}^{n} P(a_i|s, \text{pa}(a_i))$$

where $\text{pa}(a_i)$ denotes the parents of component a_i in the dependency graph.

Parameters: $O\left(\sum_{i=1}^{n} |\mathcal{A}_i| \cdot \prod_{j \in \text{pa}(i)} |\mathcal{A}_j|\right)$

This framework allows us to:

- Encode domain knowledge about component relationships
- Dramatically reduce parameter complexity while preserving essential correlations
- Maintain interpretability of the learned policy
- Apply efficient inference algorithms from graphical model theory

3.2 Designing Dependency Structures

The choice of dependency structure should reflect semantic relationships between action components. We identify three principled approaches to structure design:

1. Semantic Coupling: Components that jointly determine meaning should be coupled. For instance, in language, the "how" (style) and "what" (content) of an utterance are often interdependent.

2. Sequential Dependencies: When actions unfold over time, later components may depend on earlier choices. This is natural in multi-step reasoning where each step conditions on previous decisions.

3. Hierarchical Organization: High-level strategic choices (e.g., reasoning strategy) may constrain lower-level tactical choices (e.g., output format).

3.3 Complexity Analysis

Given a Bayesian network structure $\mathcal{G}$ over n components, the parameter complexity depends on the graph topology:

- **Chain:** $O\left(\sum_{i=1}^{n} |\mathcal{A}_i| \cdot |\mathcal{A}_{i-1}|\right)$ - Linear in n
- **Tree:** $O\left(\sum_{i=1}^{n} |\mathcal{A}_i| \cdot \max_j |\mathcal{A}_j|\right)$ - Linear in n
- **Bounded in-degree** k**:** $O\left(n \cdot \max_i |\mathcal{A}_i|^{k+1}\right)$ - Exponential in k, linear in n

For example, with 5 components each having 10 choices:

- Fully joint: $10^5 = 100,000$ parameters
- Fully factored: $5 \times 10 = 50$ parameters
- Tree with binary coupling: ≈ 500 parameters

This represents a 200× reduction from the fully joint model while maintaining key dependencies.

3.4 Learning with Compositional Q-Functions

For Q-learning in compositional action spaces, we extend the standard Bellman equation:

$$Q(s, a) = Q(s, (a_1, \ldots, a_n)) = \mathbb{E}[r + \gamma \max_{a'} Q(s', a')]$$

The key insight is that the Q-function can be factored according to the same Bayesian network structure:

$$Q(s, a) = \sum_{i=1}^{n} Q_i(s, a_i, \text{pa}(a_i))$$

This decomposition allows us to:

- Learn component-specific value functions
- Share parameters across related components
- Perform efficient action selection via dynamic programming

4 Contribution 2: External Controller Architectures

Having established compositional action spaces, we now address the architectural question of how to implement controllers that leverage these actions. We explore the design space of learning systems that control LLM prompting, comparing integrated and external approaches.

4.1 Architectural Choice: External Controller vs. Policy Heads

A critical design decision is whether to learn the policy through fine-tuning the LLM itself (adding policy heads) or through an external neural network that controls prompting of a frozen LLM. Both approaches have demonstrated significant success, and understanding their trade-offs is essential.

Policy Heads on LLM (e.g., o1/o3 approach): The policy heads approach integrates reasoning and policy learning within the same model, enabling:

- **End-to-end optimization**: Direct optimization of reasoning traces that improve final outputs
- **Latent reasoning chains**: Rich internal representations that can capture complex reasoning patterns
- **Strong empirical performance**: Models like o3 demonstrate impressive capabilities across diverse tasks
- **Unified architecture**: Single model handles both reasoning and strategy selection

However, this approach also presents challenges:

- **Computational cost**: Fine-tuning 175B+ parameter models requires massive computational resources
- **Model specificity**: Trained policies are tied to specific model architectures and weights
- **Limited interpretability**: Policy decisions occur within the latent space, making them difficult to inspect
- **Deployment constraints**: Requires access to model weights and infrastructure for fine-tuning

External Controller Approach (Our Choice): We choose the external controller approach for several complementary reasons:

1. Model Agnosticism: An external controller can work with any LLM accessible via API (GPT-4, Claude, Llama, etc.) without requiring access to model weights or architecture details.

2. Computational Efficiency: Training a small controller network (10-100M parameters) is orders of magnitude cheaper than fine-tuning large language models (7-175B parameters):

- Fine-tuning cost: $10,000 - $1,000,000
- Controller training: $10 - $1,000

3. Online Learning: The controller can continuously learn from production interactions without expensive retraining cycles, enabling rapid adaptation to new domains or user preferences.

4. Separation of Concerns: The controller learns "how to prompt" while the LLM provides "how to reason," maintaining a clean abstraction boundary.

5. Rapid Experimentation: New prompting strategies can be tested in hours rather than weeks, accelerating research iteration.

This architectural choice fundamentally shapes our approach: we are learning a prompting policy, not a reasoning policy, which is why our compositional action space focuses on prompt construction rather than direct token generation.

Complementary Strengths: We view these approaches as complementary rather than competing:

- **Policy heads excel** when computational resources are available, model access is unrestricted, and end-to-end optimization is feasible
- **External controllers excel** in resource-constrained settings, when working with API-only models, or when interpretability and rapid iteration are priorities
- **Hybrid approaches** could potentially combine both: external controllers that guide policy-head-equipped models for maximum flexibility

Our framework's compositional action space could, in principle, be applied to both architectures. The key insight is that systematic decomposition of reasoning strategies benefits any approach to learning prompting policies, whether internal or external.

4.2 POMDP Formulation for LLM Prompting

4.2.1 The Fundamental Challenge: Unknown Environment Dynamics

A critical insight motivating our approach is that the LLM's internal reasoning process constitutes an **unknown environment** whose dynamics must be learned through interaction. Unlike traditional MDP settings where transition probabilities $P(s'|s, a)$ might be known or easily estimated, the LLM's response to prompts involves:

- **Latent reasoning states** that are fundamentally unobservable
- **Complex, non-stationary dynamics** that vary across problem domains
- **Partial observability** where we only see generated text, not internal computations
- **Stochastic transitions** due to temperature sampling and inherent model uncertainty

This necessitates a POMDP formulation with model-free learning, where the controller must simultaneously:

1) Learn the value of actions through trial and error (Q-learning)
2) Maintain beliefs about the current reasoning state
3) Adapt to the specific LLM's behavior patterns without access to its weights

4.2.2 POMDP Components

We model the prompting process as a POMDP defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, \Omega, R, \gamma, b_0)$:

- **Hidden States $s \in \mathcal{S}$:**
  The true state s_t represents the LLM's latent reasoning configuration, which is fundamentally unobservable. This includes:
  - The LLM's current attention patterns and activated knowledge
  - Internal problem representation and reasoning paths being considered
  - Confidence distributions over possible solutions
  - Contextual priming effects from the dialogue history

  Since we cannot directly observe s_t, we work with observations and belief states.
- **Observations $o \in \mathcal{O}$:**
  We observe only the LLM's generated text responses and can extract features from them:

  $$o_t = \langle e_c, f_{\text{struct}}, f_{\text{prog}}, f_{\text{hist}}, f_{\text{meta}}, f_{\text{trans}} \rangle \in \mathbb{R}^{948}$$

  where each component provides complementary information:
  - **Semantic Embedding** $e_c \in \mathbb{R}^{768}$: Dense vector representation of the current context from a pretrained encoder (e.g., Sentence-BERT), capturing semantic content and relationships.
  - **Structural Features** $f_{\text{struct}} \in \mathbb{R}^{20}$:
    - Lexical statistics: token count, average sentence length, vocabulary diversity
    - Content indicators: presence of equations ($\mathbb{1}$[has math]), code blocks ($\mathbb{1}$[has code]), lists
    - Question classification: softmax(type $\in$ {yes/no, MCQ, open, computational})
    - Complexity metrics: Flesch-Kincaid score, parse tree depth
  - **Progress Indicators** $f_{\text{prog}} \in \mathbb{R}^{10}$:
    - Normalized step count: $t/T_{\max}$
    - Token utilization: tokens_used/token_budget
    - Reasoning depth: number of completed compositional actions

  - ∗ Solution proximity: estimated completion $\in [0, 1]$
- **Historical Context** $f_{\text{hist}} \in \mathbb{R}^{100}$:
  - ∗ Previous action encoding: embed$(a_{t-1}) \in \mathbb{R}^{40}$
  - ∗ Compressed previous embedding: PCA$(e_{c,t-1}) \in \mathbb{R}^{50}$
  - ∗ Action history summary: mean$(\{a_{t-k}\}_{k=1}^{K}) \in \mathbb{R}^{10}$
- **Meta-cognitive Features** $f_{\text{meta}} \in \mathbb{R}^{20}$:
  - ∗ Domain classification: softmax(domain $\in$ {math, code, reasoning, creative})
  - ∗ Confidence scores from auxiliary networks
  - ∗ Coherence metric: coherence(c_t, c_{t-1})
  - ∗ Perplexity delta: change in LLM perplexity
- **Transition Features** $f_{\text{trans}} \in \mathbb{R}^{30}$:
  - ∗ Semantic shift: $\cos(e_c, e_{c,t-1})$
  - ∗ Progress delta: $f_{\text{prog},t} - f_{\text{prog},t-1}$
  - ∗ Complexity change: Δcomplexity
  - ∗ Information gain: KL divergence between consecutive states

This observation vector of 948 dimensions provides partial information about the hidden state.

- **Belief State** $b_t \in \Delta(\mathcal{S})$**:**
Since the true state is hidden, we maintain a belief distribution over possible states:

$$b_t(s) = P(s_t = s | o_{1:t}, a_{1:t-1})$$

In practice, we approximate the belief state using a recurrent neural network that processes the observation history:

$$b_t = \text{RNN}(o_t, a_{t-1}, b_{t-1}) \in \mathbb{R}^{256}$$

This learned belief representation captures:
- Uncertainty about the LLM's current reasoning state
- Historical context that influences future responses
- Patterns in how the LLM responds to different action sequences

- **Compositional Actions** $a \in \mathcal{A}$**:**
Our action space leverages the compositional prompting framework (`rlpilot` package) which provides a rich, extensible set of prompting operations. Each action a is a tuple of linguistic components:

$$a = (\omega, \phi, \sigma, \kappa, \tau, \mu, \rho, \delta) \in \Omega \times \Phi \times \Sigma \times K \times T \times M \times C \times D$$

The core components from our implementation include:
- $\omega \in \Omega$ - **Cognitive Operations**: {decompose, analyze, generate, verify, synthesize, abstract} plus extended operations
- $\phi \in \Phi$ - **Focus Aspects**: {structure, constraints, patterns, solution, correctness, efficiency, examples, relationships, alternatives, edge_cases}
- $\sigma \in \Sigma$ - **Reasoning Styles**: {systematic, creative, critical, formal, intuitive}
- $\kappa \in K$ - **Connection Types**: {therefore, however, building_on, alternatively, verify, continue}
- $\tau \in T$ - **Output Formats**: {steps, list, mathematical, narrative, code, solution, explanation, table}
- $\mu \in M$ - **Meta-Strategies**: {forward_chaining, backward_chaining, means_ends_analysis, case_based, analogical, constraint_satisfaction}
- $\rho \in C$ - **Confidence Levels**: {very_low, low, medium, high, very_high, certain}
- $\delta \in D$ - **Reasoning Depths**: {surface, shallow, moderate, deep, profound}

The compositional framework supports:
- **Fluid API construction**: Actions are built programmatically, enabling dynamic prompt generation
- **LLM-augmented actions**: Meta-reasoning capabilities for adaptive prompt refinement
- **Parallel execution**: Multiple prompting strategies can be explored simultaneously
- **RAG integration**: Retrieved examples can be dynamically incorporated

This yields a potential action space of over 10 million combinations, but our Bayesian network factorization (described below) makes learning tractable.

4.2.2.1 Retrieval and In-Context Learning Actions: Beyond the compositional prompting dimensions, our action space includes sophisticated retrieval mechanisms for incorporating external knowledge and examples:

1. Graph-Based Example Retrieval: Our framework models examples as nodes in a knowledge graph, enabling advanced retrieval strategies:
- **Hub identification**: Finding highly-connected examples that serve as conceptual anchors
- **Bridge detection**: Identifying examples that connect different knowledge communities
- **Community clustering**: Grouping related examples for coherent few-shot learning
- **Diverse sampling**: Selecting examples from different communities to maximize coverage

Each retrieval action a_{ret} includes:
- $n \in \{1, 2, ..., 10\}$ - Number of examples for ICL
- strategy $\in$ {similar, diverse, hub_aware, community, bridge}
- threshold $\in [0, 1]$ - Similarity threshold for relevance filtering

2. Classical Knowledge Base Lookup: Fast, deterministic retrieval from structured sources:
- **Wikipedia API**: Factual information injection with latency $<$ 100ms
- **Mathematical databases**: Theorem and formula lookup
- **Code repositories**: Algorithm and implementation patterns
- **Domain ontologies**: Hierarchical concept relationships

These lookup actions provide predictable sub-environments within the unknown LLM dynamics—we know exactly what will be retrieved, even if we don't know how the LLM will use it.

Bayesian Network Structure:
Rather than treating this as a fully joint or fully independent distribution, we impose a Bayesian network structure based on linguistic and cognitive principles:

$$\pi(a|s) = P(\omega, \phi, \sigma|s) \cdot P(\kappa|s, \omega, a_{t-1}) \cdot P(\tau|s, \omega, \phi)$$

This factorization reflects that:

- The cognitive operation (ω), focus (ϕ), and style (σ) form a semantic unit that should be chosen jointly
- The connection type (κ) depends on the current operation and previous action for coherence
- The output format (τ) is constrained by the operation and focus

Complexity Reduction:
This structure reduces parameters from:

- Fully joint: $30,720$ parameters per state (without retrieval actions)
- With retrieval: Over 10^7 when including all retrieval strategies
- Our factorization: $(10 \times 8 \times 6) + (8 \times 10) + (8 \times 10 \times 8) + 250 = 480 + 80 + 640 + 250 = 1,450$ parameters

Over 6,800× reduction from the full space while preserving essential correlations!

4.2.2.2 Graph-Based Example Network: The retrieval actions operate on a dynamically-maintained graph $G = (V, E)$ implemented through the `complex-network-rag` package. The graph construction proceeds as follows:

Graph Construction:

- **Vertices** V: Examples with embeddings $e_i \in \mathbb{R}^d$ (typically $d = 384$ for sentence transformers or $d = 768$ for BERT-based models) and metadata m_i
- **Edges** E: Weighted connections where $w_{ij} = \cos(e_i, e_j)$ if $\cos(e_i, e_j) \geq \tau_{\text{min}}$ (typically $\tau_{\text{min}} = 0.7$)
- **Strong edges**: Subset $E_s \subseteq E$ where $w_{ij} \geq \tau_{\text{strong}}$ (typically $\tau_{\text{strong}} = 0.8$)

Network Analysis Metrics:

1) **Hub Centrality (Degree)**: $h(v) = \deg(v) = |\{u : (v, u) \in E\}|$ - High-degree nodes serve as conceptual anchors with broad applicability
2) **Bridge Centrality (Betweenness)**: $b(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$ where σ_{st} is the number of shortest paths from s to t, and $\sigma_{st}(v)$ is the number passing through v. Bridge nodes connect disparate knowledge domains.
3) **Community Structure**: Louvain modularity maximization (Blondel et al., 2008) partitions V into communities $C = \{c_1, ..., c_k\}$ by optimizing:

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

where $m = |E|$, A_{ij} is the adjacency matrix, $k_i = \deg(v_i)$, and $\delta(c_i, c_j) = 1$ if nodes share a community.

4) **k-Hop Neighborhoods**: $N_k(v) = \{u : d(v, u) \leq k\}$ provides local context around retrieved examples

Retrieval Strategies: The Q-network learns to select among five retrieval strategies based on the current belief state:

- **Similarity**: Pure cosine similarity ranking - $\text{rank}(v) = \cos(e_q, e_v)$
- **Community-aware**: Find most similar node's community, then rank within that community for coherent examples
- **Bridge-focused**: Prioritize high-betweenness nodes when crossing domains
- **Hub-focused**: Prioritize high-degree nodes for general problem understanding
- **Hybrid**: Combine similarity with network position - $\text{score}(v) = \alpha \cos(e_q, e_v) + \beta \frac{h(v)}{\max_u h(u)} + \gamma \frac{b(v)}{\max_u b(u)}$

The policy learns which strategy to employ based on problem characteristics encoded in the belief state, discovering patterns such as using hub examples early in reasoning and bridge examples when integrating knowledge from multiple domains.

Prompt Construction Function:
Given an action tuple a sampled from this distribution and current context c, a deterministic function $g : A \times C \to P$ constructs the actual prompt:

$$p = g(a, c) = f_\kappa(c_{prev}) \circ f_{\omega,\phi,\sigma}(c_{current}) \circ f_\tau$$

where f_κ generates the connection phrase, $f_{\omega,\phi,\sigma}$ constructs the main directive, and f_τ specifies the output format.

- **Unknown Transition Function** $T(s, a, s')$**:**
The transition dynamics $P(s'|s, a)$ governing how the LLM's hidden state evolves are **unknown and must be learned through interaction**. This is a fundamental departure from classical planning where models are given. The transitions depend on:
  - The specific LLM's training data and architecture (which we don't have access to)
  - Complex interactions between the prompt and the LLM's learned representations
  - Stochastic sampling procedures within the LLM

  Since we cannot model T directly, we adopt a **model-free Q-learning approach** that learns the value of actions without explicitly modeling the transition dynamics.
- **Observation Function** $\Omega(o|s', a)$**:**
The observation function maps hidden states to observations (LLM outputs). We observe features extracted from the generated text:

$$P(o|s', a) = \text{LLM\_Generate}(s', \text{prompt}(a))$$

  The observation includes both the raw text and extracted features. Importantly, this function is also unknown—we don't know how the LLM will respond to a given prompt until we try it.
- **Reward** $R(s, a, s')$**:**
Our reward structure primarily focuses on terminal rewards:
  - **Terminal Rewards:** Significant reward upon reaching a state where the LLM produces an accurate solution, validated against ground-truth answers or evaluation metrics.
  - **Step Penalties:** Small negative rewards for each action taken, encouraging efficiency in the reasoning process.

4.2.2.3 Future Directions - Implicit Rewards:: While our current implementation relies on sparse terminal rewards, we envision extending our framework with implicit rewards inspired by program synthesis systems like DreamCoder (Ellis et al., 2021). These could include:

– **Compression Rewards:** Incentives for representing information more concisely without loss of utility.
– **Reuse Rewards:** Bonuses for leveraging previously stored reasoning traces or code snippets from memory.
– **Simplification Rewards:** Rewards for reducing complex reasoning paths or code into more elegant, generalizable forms.
– **Compositional Rewards:** Encouraging the decomposition of complex functions into compositions of simpler, reusable components.

Such intrinsic motivations could potentially lead to more efficient and generalizable policies, particularly for domains where explicit feedback is limited or costly to obtain. This aligns with the broader goal of developing systems that not only solve individual problems but also build increasingly powerful abstractions over time (Lake et al., 2017).

4.3 Belief State Representation and Updates

In the POMDP setting, we cannot directly observe the true state. Instead, we maintain a belief state that summarizes our uncertainty about the LLM's current reasoning configuration.

4.3.1 Belief State Update

The belief state update follows Bayes' rule:

$$b_{t+1}(s') = \eta \cdot \Omega(o_{t+1}|s', a_t) \sum_{s \in \mathcal{S}} T(s'|s, a_t) b_t(s) \tag{1}$$

where η is a normalization constant. However, since both T and Ω are unknown, we approximate the belief update using a learned recurrent model:

$$b_{t+1} = \text{GRU}(\phi(o_{t+1}), \text{embed}(a_t), b_t) \tag{2}$$

where:

- GRU is a Gated Recurrent Unit that learns to track belief dynamics
- $\phi(o_{t+1})$ is the observation feature vector (948 dimensions)
- $\text{embed}(a_t)$ is a learned embedding of the compositional action
- $b_t \in \mathbb{R}^{256}$ is the compressed belief representation

4.3.2 Information Preservation Through Observations

Our observation extraction preserves critical information about the hidden state:

1) **Multi-scale Feature Extraction:** We extract features at multiple granularities from the LLM's output:
   - Semantic embeddings (e_c) capture high-level meaning
   - Structural features (f_{struct}) identify problem characteristics
   - Progress indicators (f_{prog}) track solution advancement
   - Meta-cognitive features (f_{meta}) assess reasoning quality
2) **Temporal Context Integration:** The belief state maintains a compressed history:
$$b_t = f(o_t, o_{t-1}, ..., o_1, a_{t-1}, ..., a_1) \tag{3}$$
This allows the controller to recognize patterns in how the LLM responds to action sequences, even though the true state transitions are unknown.
3) **Uncertainty Quantification:** The belief state implicitly represents uncertainty about:
   - Which reasoning paths the LLM is considering
   - How close the LLM is to a solution
   - What types of prompts will be most effective next
4) **Adaptive Learning:** As the controller interacts with the LLM, the belief update model learns:
   - Patterns in how different LLMs respond to compositional actions
   - Domain-specific dynamics (e.g., mathematical vs. coding problems)
   - Individual LLM quirks and biases

4.3.3 Model-Free Learning in Unknown Environments

The POMDP formulation with unknown dynamics necessitates model-free reinforcement learning:

- **No Transition Model:** We cannot predict how the LLM will respond to prompts
- **No Observation Model:** We cannot predict what features we'll observe
- **Learning Through Interaction:** Q-values are learned purely from experience

This is why Q-learning is particularly suitable—it learns the value of actions without requiring a model:

$$Q(b, a) \leftarrow Q(b, a) + \alpha[r + \gamma \max_{a'} Q(b', a') - Q(b, a)]$$

where b and b' are belief states rather than true states. The Q-function learns which compositional actions are valuable in different belief contexts, without ever needing to know the true state or dynamics.

4.4 Compositional Q-Learning Architecture for POMDPs

We approximate the action-value function with a multi-head neural network that processes the belief state and decomposes the compositional action space:

$$Q_\theta(b, a) = Q_\theta(b_t, (\omega, \phi, \sigma, \kappa, \tau, \mu, \rho, \delta, a_{ret}, a_{lookup}))$$

where a_{ret} represents graph-based retrieval actions and a_{lookup} represents deterministic knowledge base queries.

The network architecture employs a hierarchical design to handle the belief state and observation features:

1) **Observation encoders** that preprocess each observation component:

$$h_e = \text{LayerNorm}(\text{Linear}_{768\to256}(e_c)) \quad (4)$$
$$h_s = \text{ReLU}(\text{Linear}_{20\to64}(f_{\text{struct}})) \quad (5)$$
$$h_p = \text{ReLU}(\text{Linear}_{10\to32}(f_{\text{prog}})) \quad (6)$$
$$h_h = \text{ReLU}(\text{Linear}_{100\to128}(f_{\text{hist}})) \quad (7)$$
$$h_m = \text{ReLU}(\text{Linear}_{20\to64}(f_{\text{meta}})) \quad (8)$$
$$h_t = \text{ReLU}(\text{Linear}_{30\to64}(f_{\text{trans}})) \quad (9)$$

2) **Belief state integration**:

$$h_{\text{belief}} = \text{GRU}(h_{\text{fused}}, b_t) \in \mathbb{R}^{512}$$

The GRU integrates current observations with the maintained belief state, capturing both immediate features and historical context.

3) **Shared reasoning layers**:

$$h_\theta(b) = \text{MLP}(h_{\text{belief}}) \in \mathbb{R}^{256}$$

where MLP consists of two hidden layers with residual connections and dropout for regularization.

4) **Compositional action heads** following our Bayesian network structure:
The network implements an extended factorization that includes meta-strategies and confidence levels:

$$\pi(a|b) = P(\omega,\phi,\sigma|b)\cdot P(\kappa|b,\omega)\cdot P(\tau|b,\omega,\phi)\cdot P(\mu|b,\omega)\cdot P(\rho,\delta|b,\mu)$$

- **Joint reasoning head:** Outputs joint distribution over (ω,ϕ,σ):

$$q_{\text{reason}} = \text{softmax}(W_{\text{reason}}h_\theta(b)+b_{\text{reason}}) \in \mathbb{R}^{|\Omega|\times|\Phi|\times|\Sigma|}$$

- **Connection head:** Conditions on operation and belief:

$$q_\kappa = \text{softmax}(W_\kappa[h_\theta(b);\text{embed}(\omega)] + b_\kappa) \in \mathbb{R}^{|K|}$$

- **Format head:** Conditions on operation and focus:

$$q_\tau = \text{softmax}(W_\tau[h_\theta(b);\text{embed}(\omega);\text{embed}(\phi)]+b_\tau) \in \mathbb{R}^{|T|}$$

- **Meta-strategy head:** Selects high-level reasoning approach:

$$q_\mu = \text{softmax}(W_\mu[h_\theta(b);\text{embed}(\omega)] + b_\mu) \in \mathbb{R}^{|M|}$$

- **Confidence and depth heads:** Assess reasoning requirements:

$$q_{\rho,\delta} = \text{softmax}(W_{\rho\delta}[h_\theta(b);\text{embed}(\mu)]+b_{\rho\delta}) \in \mathbb{R}^{|C|\times|D|}$$

- **Retrieval strategy head:** Selects graph-based example retrieval:

$$q_{ret} = \text{softmax}(W_{ret}[h_\theta(b); g_{\text{graph}}] + b_{ret}) \in \mathbb{R}^{5\times10}$$

where $g_{\text{graph}} \in \mathbb{R}^{32}$ encodes current graph state (hub distribution, community structure, bridge connectivity)

- **Knowledge lookup head:** Activates deterministic retrievals:

$$q_{lookup} = \text{sigmoid}(W_{lookup}[h_\theta(b);\text{embed}(\phi)]+b_{lookup}) \in \mathbb{R}^4$$

for Wikipedia, mathematical theorems, code patterns, and domain knowledge bases

The composite Q-value respects the extended Bayesian network structure:

$$Q_\theta(b,a) = \sum_{i\in\text{components}} \log q_i[a_i|\text{pa}(a_i),b] + \lambda_{ret}\log q_{ret} + \lambda_{lookup}\sum_j q^j_{lookup}$$

where $\lambda_{ret}, \lambda_{lookup}$ are learned weights balancing compositional prompting with retrieval actions.

This architecture processes belief states while maintaining the compositional action space's tractability. Despite the massive potential action space (10M+ combinations), the factorization reduces the parameter complexity to just a few thousand learnable weights.

4.4.1 Learning Without Known Dynamics

The Q-learning update in our POMDP setting is:

$$\theta \leftarrow \theta + \alpha[r + \gamma\max_{a'} Q_\theta(b',a') - Q_\theta(b,a)]\nabla_\theta Q_\theta(b,a)$$

Crucially, this update **does not require knowledge of the transition or observation functions**. The controller learns purely from:

- The sequence of observations (LLM outputs)
- The rewards received (task success/failure)
- The actions taken (compositional prompts)

This model-free approach enables the controller to adapt to any LLM without access to its internals, learning its specific behavior patterns through interaction.

4.5 Feature Extraction Pipeline

The enriched state representation requires a sophisticated feature extraction pipeline that operates in real-time during reasoning. We detail the key extraction components:

Structural Feature Extraction:

```
def extract_structural_features(context):
    return np.array([
        len(context.split()),
# token count
        np.mean([len(s) for s in sentences]),
# avg sentence length
        len(set(context.split())) / len(context.split()),
# vocab diversity
        1 if '$$' in context else 0,
# has math
        1 if ''''' in context else 0,
# has code
        classify_question_type(context),
# question type vector
        flesch_kincaid_score(context)
# readability
    ])
```

Progress and Transition Features: Progress features track reasoning advancement, while transition features capture dynamics:

$$f_{\text{trans}} = \begin{bmatrix} \cos(e_{c,t}, e_{c,t-1}) \\ \|f_{\text{prog},t} - f_{\text{prog},t-1}\|_2 \\ \text{KL}(p_t\|p_{t-1}) \end{bmatrix}$$

where p_t and p_{t-1} are token distributions from consecutive contexts.

Normalization Strategy: Given the heterogeneous nature of features, we apply feature-specific normalization:

- Embeddings: L2 normalization to unit sphere
- Counts and lengths: Log-scale transformation followed by z-score normalization
- Binary indicators: No normalization needed
- Probabilities: Clipping to [0,1] range

This ensures all features contribute meaningfully to the Q-network's decision-making without any single feature type dominating due to scale differences.

4.6 Theoretical Advantages of Compositional Actions

The compositional action formulation provides several theoretical advantages over atomic prompt templates:

1. Combinatorial Generalization: By learning independent distributions over action components, the policy can generalize to unseen combinations. If the agent learns that $(\omega = \text{verify}, \phi = \text{constraints})$ and $(\omega = \text{analyze}, \phi = \text{assumptions})$ are effective in different contexts, it can potentially discover that $(\omega = \text{verify}, \phi = \text{assumptions})$ is useful without explicit training.

2. Sample Efficiency: The factored representation reduces the effective action space from exponential to linear in the number of components. Instead of learning values for $|\Omega| \times |\Phi| \times |\Sigma| \times |K| \times |T|$ discrete actions, we learn $|\Omega| + |\Phi| + |\Sigma| + |K| + |T|$ component values, dramatically improving sample efficiency.

3. Interpretable Credit Assignment: When a reasoning step succeeds or fails, the compositional structure allows us to identify which components contributed. This enables more precise credit assignment during learning and better interpretability of the learned policy.

4. Coherence Through Conditioning: By conditioning the connection component κ on previous actions, we maintain coherence across the reasoning chain while still allowing flexibility in other components.

4.7 Convergence Properties

While Q-learning is known to converge to optimal policies in tabular settings under certain conditions (Watkins and Dayan, 1992), our use of function approximation introduces additional considerations. We leverage recent theoretical results on convergence bounds for deep Q-learning with continuous state spaces (Fan et al., 2020) to establish expected convergence properties.

Given that our state space (embedding vectors) is continuous but bounded, and our compositional action space is discrete and finite with factored structure, we expect convergence to a near-optimal policy under standard assumptions of sufficient exploration and appropriate learning rate schedules. The factorization further aids convergence by reducing the variance of gradient estimates. The use of experience replay further stabilizes the learning process by reducing correlation between consecutive updates (Mnih et al., 2015).

5 Design Trade-offs

Our explicit action space offers interpretability and modularity, allowing for direct intervention (e.g., selective retrieval or tool use). However, it introduces handcrafted components that may limit generality compared to latent chain-of-thought methods. The design challenge is to provide reward signals that encourage effective exploration without overly constraining the agent's learning.

5.1 Comparison with Latent Chain-of-Thought Methods

Latent chain-of-thought approaches such as OpenAI's o1/o3 and DeepSeek-R1 rely on letting the model implicitly generate internal reasoning chains through reinforcement learning on massive datasets. These methods benefit from scalability and minimal manual design. In contrast, our method:

- **Explicit Interpretability:**
  Provides a visible sequence of actions for easier debugging and analysis.
- **Modular Integration:**
  Allows for the targeted inclusion of external resources (e.g., retrieval and tool execution) as needed.
- **Controlled Trade-offs:**
  Offers a structured framework that may trade off some scalability for enhanced control, which can be advantageous in scenarios where understanding the reasoning process is critical.

5.2 Theoretical Comparison with End-to-End Methods

We can formally analyze the trade-offs between our discrete action approach and end-to-end latent reasoning methods:

- **Hypothesis 1:** Our approach will demonstrate better sample efficiency in scenarios where the optimal reasoning path involves a small set of general strategies applied in different combinations.
- **Hypothesis 2:** End-to-end methods will demonstrate superior performance on problems requiring novel reasoning techniques not captured in our discrete action space.
- **Hypothesis 3:** The performance gap between our approach and end-to-end methods will decrease as the granularity and diversity of our action space increases, at the cost of increased learning complexity.

These hypotheses derive from the inherent trade-off between structural constraints (our discrete actions) and flexibility (latent representations), analogous to the bias-variance trade-off in statistical learning theory (Geman et al., 1992).

5.3 Key Architectural Insight: Dual Factorization for Sample Efficiency

A critical insight underlying our framework's expected sample efficiency is the **dual factorization** of both state and action spaces. Unlike approaches that factorize only actions (compositional methods) or only states (representation learning), we systematically decompose both sides of the value function:

$$Q(b, a) \approx f_b(o_t, b_{t-1}) \times f_a(\omega, \phi, \sigma, \kappa, \tau, \mu, \rho, \delta)$$

This bilinear structure enables **bidirectional generalization**:

Semantic Generalization (State Factorization): The 948-dimensional enriched state representation allows the policy to recognize when two problems share similar reasoning requirements, even if their surface content differs completely. For instance, a geometric proof and an algebraic manipulation may have similar f_{prog} (reasoning progress) and f_{meta} (problem structure) components, enabling strategy transfer.

Compositional Action Generalization: The factored action space allows the controller to learn general patterns about what works in the unknown environment. For example, discovering that $\omega =$ decompose paired with $\sigma =$ systematic tends to produce better outputs generalizes across different problem types, even though the controller doesn't know why this combination is effective—it only knows that empirically, these actions lead to higher rewards.

Cross-Product Generalization: Most powerfully, the combination enables learning from sparse experiences to generalize to the full $|\mathcal{S}|$
$times|\mathcal{A}|$ space. If we learn that:

- State features $[e_c^{(math)}, f_{\text{meta}}^{(complex)}]$ benefit from systematic reasoning
- Action component
  $omega =$ decompose works well for multi-step problems

Then the controller can hypothesize that $Q(b_{math,complex}, a_{decompose,*,systematic,*,*})$ will have high value, even without direct experience. This hypothesis isn't based on understanding the LLM's reasoning—it's based on patterns discovered through trial and error in the unknown environment.

This dual factorization is why we expect our approach to be particularly sample-efficient compared to:

- Atomic action methods (no action generalization)
- Raw text state methods (no semantic generalization)
- Single-factorization approaches (limited cross-domain transfer)

The observation features extracted from LLM outputs provide signals about the hidden state, while the belief state integrates these observations over time to form hypotheses about the LLM's behavior. This combination of immediate observations and temporal integration is crucial for navigating unknown environments where single observations provide only partial information about the true state.

5.4 Integration with Search-Based Reasoning

Our compositional action framework naturally extends to search-based reasoning methods, connecting to a rich literature on Monte Carlo Tree Search (MCTS) and planning in reasoning (Yao et al., 2023; Besta et al., 2024; Hao et al., 2023). Unlike traditional MCTS that uses random rollouts, our learned policy can guide exploration through the reasoning tree.

5.4.1 Policy-Guided Tree Search

Our framework can integrate with MCTS in several ways, each with distinct advantages:

Option 1: Policy-Prior MCTS If we train our network to output action probabilities $\pi(a|s)$, we can use these as priors in MCTS:

$$\text{UCB}(s,a) = Q_{\text{MCTS}}(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} + \beta\pi(a|s)$$

Option 2: Value-Function MCTS Alternatively, we could train the network to directly estimate $V(s)$ or $Q(s,a)$, using these as initialization for MCTS node values. This provides better starting estimates but loses the explicit action guidance.

Option 3: Hybrid Approach We could learn both: $Q(s,a)$ for value estimation and $\pi(a|s)$ for action priors, though this requires more complex training.

The choice depends on what signal is easier to learn from limited data. $Q(s,a)$ is most universal since $V(s) = \max_a Q(s,a)$ and $\pi(a|s) \propto \exp(Q(s,a)/\tau)$, but direct policy learning might be more sample-efficient for our structured action space.

MCTS Policy vs. Learned Policy: A Fundamental Trade-off

An important insight is that MCTS naturally produces policies through search without explicit learning:

$$\pi_{\text{MCTS}}(a|s) = \frac{N(s,a)}{\sum_{a'} N(s,a')}$$

where $N(s,a)$ is the visit count of action a from state s after running the full search tree. This creates a fundamental speed-accuracy trade-off:

- **MCTS Policy**: Computationally expensive (full tree search per decision) but adapts precisely to each state through targeted exploration
- **Learned Policy**: Fast inference (single forward pass) but relies on generalization from training data, potentially missing state-specific nuances

Interestingly, both approaches involve "learning" - MCTS learns through visit count accumulation during search, while neural policies learn through parameter updates during training. The key difference is *when* the learning occurs: MCTS learns at inference time through search, while neural networks learn at training time through experience replay.

This suggests several hybrid strategies:

- **Policy Distillation**: Train the learned policy to mimic MCTS policies by using $\pi_{\text{MCTS}}(a|s)$ as training targets
- **Adaptive Deployment**: Use fast learned policy for routine problems, expensive MCTS for challenging ones
- **Search-Augmented Training**: Use MCTS-discovered action sequences as high-quality training data

Critical Challenge: Terminal Node Evaluation A key challenge in applying MCTS to reasoning is recognizing and evaluating terminal nodes. Unlike game trees with clear win/loss outcomes, reasoning chains require:

1. **Terminal Detection**: When has the reasoning process concluded? This could be: - Fixed depth limit - LLM-based termination decision: "Is this reasoning chain complete?" - Confidence threshold on solution quality

2. **Terminal Evaluation**: How good is the final reasoning chain? Options include: - Ground truth comparison

(when available) - LLM-as-judge evaluation: "How correct/convincing is this solution?" - Multi-metric evaluation (correctness, clarity, efficiency) - Human evaluation (expensive but high-quality)

Our compositional actions *may* help with terminal evaluation by making the reasoning process more interpretable, but this advantage requires empirical validation.

5.4.2 Compositional Search Strategies

Our factored action space *potentially* enables more structured search strategies, though their effectiveness compared to well-designed atomic action spaces remains an empirical question:

1. Component-wise Beam Search: Instead of expanding the full action space, we could beam search through components sequentially:

- Beam search over ω (cognitive operations)
- For each ω^*, beam search over ϕ given $P(\phi|s, \omega^*)$
- Continue through σ, κ, τ in dependency order

This could reduce search complexity from $O(|\mathcal{A}|^d)$ to $O(\sum_i |\mathcal{A}_i|^d)$ where d is search depth, but only if the factorization assumptions hold and the sequential dependencies are meaningful.

2. Hierarchical Reasoning Trees: We can structure search hierarchically: - **Strategic level**: Search over high-level reasoning approaches (ω, ϕ) - **Tactical level**: For each strategy, search over implementation details (σ, κ, τ)

This mirrors human reasoning: first deciding "how to approach the problem" then "how to execute that approach."

3. Multi-Path Consistency Checking: Our compositional actions enable exploring multiple reasoning paths simultaneously and checking for consistency:

"' Path 1: (decompose, structure, systematic, ...) Path 2: (analyze, constraints, formal, ...) Path 3: (generate, solution, direct, ...) "'

If multiple paths converge to the same answer with high confidence, we gain additional validation.

5.4.3 Adaptive Search Budget Allocation

The finite context window W creates natural budget constraints for search. We can adaptively allocate search budget based on:

- **Problem complexity**: Estimated from f_{meta} features
- **Current uncertainty**: Entropy of policy distribution $H(\pi(\cdot|s))$
- **Progress indicators**: Change in f_{prog} features
- **Remaining budget**: Available tokens in context window

$$\text{SearchDepth}(s) = \min\left(\frac{W - |c_t|}{E[\text{tokens per action}]}, f_{\text{complexity}}(s) \cdot \alpha\right)$$

5.4.4 LLM-Guided Meta-Reasoning Enhancement

A particularly promising direction is using LLMs not just as reasoning engines, but as meta-reasoners about their own prompting strategies. Our compositional action space enables several forms of LLM-guided enhancement:

1. Context-Adaptive Action Modification: Instead of using fixed compositional templates, we can ask the LLM to adapt actions to specific contexts:

> *"Given this mathematical problem: [context]*
> *Here are potential approaches: (decompose, structure, systematic, therefore, steps)*
> *How should this be modified for this specific problem type?"*

This allows dynamic prompt engineering based on problem characteristics.

2. LLM-Based Action Evaluation: Rather than relying solely on visit counts, we can use LLM judgment to evaluate action quality:

$$\text{UCB}_{\text{LLM}}(s, a) = Q(s, a) + C\sqrt{\frac{\ln N(s)}{N(s, a)}} + \beta \cdot \text{LLMScore}(s, a)$$

where $\text{LLMScore}(s, a)$ asks the LLM: *"Rate how appropriate this prompting strategy is for the current reasoning state (1-10)."*

3. Compositional Strategy Suggestion: The LLM can propose entirely new action combinations:

> *"Given the current context, what values of $(,,,,)$ would be most effective? Explain your reasoning."*

4. Three-Layer Reasoning Architecture: This suggests a hybrid system with complementary strengths:

- **MCTS Layer**: Strategic exploration through principled search, handles long-term planning
- **Neural Network Layer**: Fast pattern recognition and learned priors, provides efficiency
- **LLM Meta-Reasoning Layer**: Context-specific adaptation and strategy refinement, adds flexibility

The compositional action space serves as a common vocabulary enabling all three layers to communicate. The neural network provides fast priors, MCTS provides systematic exploration, and the LLM provides contextual adaptation.

Potential Implementation: 1. NN suggests initial action probabilities based on state features 2. LLM refines these suggestions based on full context understanding 3. MCTS uses both as priors for systematic search 4. Visit counts and outcomes update both NN and LLM guidance strategies

This architecture could combine the best aspects of each approach: NN speed, MCTS systematicity, and LLM contextual intelligence.

5.4.5 Connection to Tree of Thoughts and ReAct

This search integration connects our work to recent advances in search-based reasoning:

Tree of Thoughts (Yao et al., 2023): Our approach generalizes ToT by: - Using learned policies instead of handcrafted heuristics - Providing finer-grained action spaces beyond "continue/backtrack" - Enabling compositional combination of reasoning strategies

ReAct (Yao et al., 2023): Our ω component can include "tool$_use"operationssimilartoReAct'sactionspace, butwithinamorege$
$actingasspecialcases.$

Graph of Thoughts (Besta et al., 2024): Our compositional actions can construct arbitrary reasoning graphs, not just trees, by allowing actions that reference and combine previous reasoning steps.

5.4.6 Search-Training Synergy

Importantly, search and policy learning create a virtuous cycle:

- **Search → Better Training Data**: MCTS exploration discovers high-reward action sequences that provide better training targets for the policy
- **Better Policy → More Efficient Search**: Improved policy provides better priors, reducing search budget needed for good performance
- **Compositional Transfer**: Learning about component effectiveness in one search tree transfers to other trees with shared components

This suggests a training curriculum: 1. **Bootstrap Phase**: Learn basic policy from simple, direct reasoning 2. **Search-Augmented Phase**: Use policy-guided search to discover complex reasoning chains 3. **Distillation Phase**: Train policy to directly predict successful search outcomes 4. **Deployment Phase**: Use lightweight policy for easy problems, full search for hard problems

The compositional structure makes this particularly effective because component-level learning transfers across all phases, unlike end-to-end approaches that must relearn everything for each level of sophistication.

6 Experimental Evaluation

We now describe our experimental protocol for evaluating model-free learning in unknown LLM environments. The key research questions are: (1) Can a POMDP controller learn effective prompting strategies without understanding the LLM? (2) How does performance compare to approaches with model access? (3) Can controllers trained on one LLM transfer to others?

6.1 Datasets and Benchmarks

We will evaluate our framework on a range of standard tasks: -

- **Mathematical Reasoning:** Datasets such as AIME and MATH-500.
- **Code Generation and Debugging:** Benchmarks like Codeforces and LiveCodeBench.
- **Factual Question Answering:** Standard QA datasets covering general knowledge.
- **Out-of-Distribution Tasks:** To assess robustness and generalization to unseen contexts.

6.2 Baselines

We will compare our framework against:

- **Standard Prompting Techniques:** Manual prompt design and basic chain-of-thought prompting.
- **Latent Chain-of-Thought Models:** Published results from OpenAI's o1/o3 and DeepSeek-R1, and our in-house implementations where applicable.

6.3 Evaluation Metrics

Key metrics include:

- **Final Output Accuracy:** Correctness of the terminal solution.
- **Reasoning Efficiency:** Number of steps (actions) required to reach the final output. -
- **Computational Cost:** Resource and time usage per task.
- **Interpretability:** Qualitative assessment of the chain of actions.
- **Generalization:** Performance on unseen or out-of-distribution prompts.

6.4 Training Methodology and Reward Engineering

Training our external controller presents unique challenges: unlike fine-tuning approaches that can leverage next-token prediction, we must rely entirely on reinforcement learning with outcome-based rewards. This aligns with the principle that "reward is enough" (Silver et al., 2021) - that complex behaviors emerge from optimizing simple reward signals - while acknowledging the practical challenges of credit assignment in long reasoning chains.

6.4.1 The Fundamental Training Challenge

Our external controller cannot access the LLM's internal representations or gradients, making traditional pre-training approaches like minimizing perplexity infeasible. Instead, we must evaluate complete reasoning chains based on their outcomes. This constraint, while limiting, has an important benefit: it forces us to optimize for what actually matters - correctness and efficiency - rather than proxy metrics like token likelihood.

This approach resonates with Sutton's "Bitter Lesson" (Sutton, 2019): rather than encoding specific reasoning strategies, we let the controller discover effective prompting patterns through trial and error, guided only by reward signals.

6.4.2 Comprehensive Reward Design

The key to successful training lies in rich, multi-scale reward signals that address the credit assignment problem inherent in sequential decision-making:

1. Terminal Rewards (Correctness Evaluation):

$$R_{\text{terminal}}(s_T, a_T) = \begin{cases} r_{\text{exact}} & \text{if answer matches ground trut} \\ r_{\text{partial}} \cdot \text{sim}(y, y^*) & \text{if answer partially correct} \\ r_{\text{fail}} & \text{if answer incorrect or incompl} \end{cases}$$

Evaluation methods depend on the domain:

- **Mathematics:** Symbolic math solvers (SymPy) for verification
- **Code:** Unit test execution and output matching
- **Reasoning:** LLM-as-judge with calibrated confidence scores
- **Factual QA:** Exact match or semantic similarity metrics

2. Intermediate Progress Rewards (and the Goodhart's Law Dilemma):

To address the credit assignment problem, we provide dense rewards throughout the reasoning chain. However, this introduces a fundamental tension: we need intermediate signals to guide learning, but any proxy metric we optimize can be gamed (Goodhart's Law). Our approach is to use a diverse portfolio of weakly correlated signals:

$$R_{\text{progress}}(s_t, a_t, s_{t+1}) = \sum_i w_i(t) \cdot r_i$$

where components include:

- **Semantic coherence:** $r_{\text{coh}} = \cos(e_{c,t}, e_{c,t+1})$ - Maintains logical flow
- **Information gain:** $r_{\text{info}} = H(s_t) - H(s_{t+1})$ - Rewards uncertainty reduction
- **Complexity reduction:** $r_{\text{simp}} = \text{complex}(s_t) - \text{complex}(s_{t+1})$
- **Goal proximity:** $r_{\text{goal}} = \text{dist}(s_{t+1}, s_{\text{goal}}) - \text{dist}(s_t, s_{\text{goal}})$

Critically, the weights $w_i(t)$ decrease over training time, gradually shifting emphasis from these proxies to terminal correctness. This curriculum prevents the model from overfitting to intermediate metrics while still providing early training signal.

3. Efficiency Penalties:

$$R_{\text{efficiency}} = -c_{\text{token}} \cdot \text{tokens}(a) - c_{\text{step}} - c_{\text{time}} \cdot t$$

These penalties encourage concise, efficient reasoning paths.

6.4.3 Evaluation Infrastructure

Building reliable evaluation infrastructure is crucial for generating consistent reward signals:

1. Synthetic Data Generation:

- Generate problems with known solutions using rule-based systems
- Create compositional tasks by combining simpler verified components
- Use data augmentation to increase diversity while maintaining correctness guarantees

2. LLM-as-Judge Framework:

For tasks without deterministic evaluation, we employ carefully calibrated LLM judges:

$$R_{\text{judge}} = \mathbb{E}_{M \in \text{Models}}[\text{score}_M(y, y^*, \text{criteria})]$$

where we aggregate scores from multiple models to reduce bias.

3. Verification Pipelines:

- Mathematical: SymPy for symbolic verification, numerical solvers for approximation
- Code: Sandboxed execution with test suites
- Logic: SAT/SMT solvers for formal verification
- Factual: Knowledge base lookups and cross-referencing

6.4.4 Training Algorithm

Our training employs a modified Q-learning algorithm with prioritized experience replay:

Algorithm 1 Compositional Q-Learning for Prompt Control

```
Initialize Q-network Q_θ with random weights
Initialize target network Q_θ⁻ ← Q_θ
Initialize prioritized replay buffer D
for episode = 1 to M do
    Sample problem p ~ P
    Initialize state s_0 = ψ(p)
    for t = 0 to T_max do
        Sample action a_t ~ π_ε(a|s_t) using Bayesian network structure
        Execute prompt, observe r_t, s_{t+1}
        Calculate TD error: δ = r_t + γ max_a Q_θ⁻(s_{t+1}, a) − Q_θ(s_t, a_t)
        Store (s_t, a_t, r_t, s_{t+1}, δ) in D with priority |δ|
        if terminal state reached then
            Break
        end if
    end for
    if |D| > batch_size then
        Sample batch B ~ D weighted by priority
        Update Q_θ using gradient descent on TD error
    end if
    if episode mod target_update = 0 then
        Q_θ⁻ ← Q_θ
    end if
end for
```

6.4.5 Addressing the Credit Assignment Problem

The credit assignment problem - determining which actions contributed to eventual success or failure - is particularly acute in our setting where reasoning chains can span many steps. We address this through:

1. Reward Shaping: Carefully designed intermediate rewards that align with progress toward the goal while avoiding reward hacking. However, we must remain cognizant of Goodhart's Law (Goodhart, 1975): "When a measure becomes a target, it ceases to be a good measure." For instance, optimizing solely for semantic coherence might lead to verbose but ultimately incorrect reasoning chains. To mitigate this:

- Use multiple uncorrelated reward signals that are difficult to game simultaneously
- Maintain a large weight on terminal correctness rewards relative to intermediate signals
- Periodically audit learned behaviors for degenerate strategies
- Employ reward curriculum that shifts emphasis from intermediate to terminal rewards over training

2. Hindsight Experience Replay: When a trajectory fails, we can still learn by relabeling it with alternative goals it did achieve.

3. Monte Carlo Returns: For stable tasks, we can use full episode returns:

$$G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$$

4. Temporal Difference Learning: For faster learning, we bootstrap from value estimates:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

6.5 Compression, Complexity, and the Foundations of Intelligent Prompting

Our framework's effectiveness ultimately rests on a fundamental principle: intelligence is compression (Solomonoff, 1964; Hutter, 2005). This connection, formalized through Solomonoff induction and Kolmogorov complexity, provides theoretical grounding for our compositional approach and informs critical design decisions around context management and action selection.

6.5.1 Theoretical Foundations: Intelligence as Compression

Solomonoff's theory of universal induction establishes that optimal prediction requires finding the shortest program that generates observed data. In our context, effective prompting means finding minimal representations that preserve task-relevant information:

$$\pi^* = \arg\min_{\pi} K(\pi) + \mathbb{E}_{p\sim\mathcal{P}}[-\log P(y^*|p,\pi)]$$

where $K(\pi)$ is the Kolmogorov complexity of the policy and the second term measures prediction accuracy. This formulation reveals why our compositional action space is theoretically motivated: by factorizing actions into reusable components, we achieve compression through modularity.

The connection to the Minimum Description Length (MDL) principle (Rissanen, 1978) suggests that our policy should learn to:

- Identify regularities in reasoning patterns
- Compress these patterns into reusable action components
- Apply compressed representations to novel problems

6.5.2 Optimal Context Compression

Given finite context windows, optimal prompting requires principled compression that preserves task-relevant information while discarding redundancy. We formalize this as:

$$c_{t+1} = \arg\min_{c'\in\mathcal{C}} K(c') \quad \text{s.t.} \quad I(c';y^*) \geq I(c_t;y^*)$$

where $I(\cdot;\cdot)$ denotes mutual information with the target output. This constraint ensures compression doesn't lose critical information.

In practice, our context management operator ψ approximates this ideal through learned compression strategies:

$$\psi(c_t, r_t, \omega) = \begin{cases} \text{abstract}(r_t) & \text{if } \omega = \text{synthesize} \\ \text{extract\_key}(c_t) \oplus r_t & \text{if } \omega = \text{abstract} \\ \text{compress}(c_t \cup r_t) & \text{if } |c_t \cup r_t| > W \end{cases}$$

Each strategy represents a different compression philosophy:

- **Abstractive compression**: Identifies high-level patterns, discarding implementation details
- **Extractive compression**: Selects most informative segments
- **Lossy compression**: Accepts controlled information loss for greater compression

6.5.3 Compression-Aware Q-Learning

We modify our Q-learning objective to explicitly reward compression:

$$Q^{\text{comp}}(s,a) = Q(s,a) + \lambda \cdot \frac{I(s';y^*)}{K(a)}$$

This augmented Q-value rewards actions that maximize information gain per unit of complexity. The hyperparameter λ controls the trade-off between task performance and compression efficiency.

During training, we estimate $K(a)$ using practical proxies:

- Token count: $K_{\text{tokens}}(a) = |\text{tokens}(a)|$
- Template complexity: $K_{\text{template}}(a) = -\log P(a|\mathcal{A})$
- Semantic novelty: $K_{\text{semantic}}(a) = \text{dist}(a, \bar{a})$

6.5.4 Information-Theoretic Action Selection

Our compositional action space enables information-theoretic action selection that maximizes expected information gain:

$$a^* = \arg\max_{a} \mathbb{E}_{s'\sim P(\cdot|s,a)}[H(Y|s) - H(Y|s')]$$

where $H(Y|s)$ is the conditional entropy of the target given state s. This formulation naturally balances exploration (high uncertainty reduction) with exploitation (moving toward low-entropy states).

The Bayesian network structure of our action space provides computational advantages:

$$I(s';Y) \approx \sum_{i=1}^{n} I(s';Y|a_1,...,a_{i-1})$$

This decomposition allows us to estimate information gain for each component independently, dramatically reducing computational complexity.

6.5.5 Hierarchical Compression and the Solomonoff Prior

We can view our entire framework through the lens of Solomonoff's universal prior, which assigns higher probability to simpler (more compressible) hypotheses:

$$P(\pi) = 2^{-K(\pi)}$$

This prior naturally induces a hierarchy of prompting strategies:

1) **Level 0**: Direct answering (minimal complexity)
2) **Level 1**: Single-step decomposition
3) **Level 2**: Multi-step reasoning with tool use
4) **Level 3**: Recursive decomposition with backtracking

Our policy learns to select the minimum complexity level sufficient for each problem, embodying Occam's Razor in automated reasoning.

6.5.6 Practical Implications for Training

The compression perspective reveals several critical training challenges:

1. Exploration-Compression Trade-off: Random exploration in compositional spaces is inefficient due to the $|\Omega| \times |\Phi| \times |\Sigma| \times |K| \times |T|$ action space. Instead, we employ compression-guided exploration:

$$P(a|s) \propto \exp\left(\frac{Q(s,a)}{\tau} - \beta K(a)\right)$$

where β penalizes complex actions during exploration.

2. Meta-Learning Compression Operators: The optimal compression strategy is task-dependent. We employ meta-learning to discover compression operators:

$$\psi^* = \arg\min_{\psi} \mathbb{E}_{\tau\sim p(\tau)}[\mathcal{L}(\tau,\psi)]$$

where τ represents task distributions and $\mathcal{L}$ measures compression-performance trade-off.

3. Curriculum Learning via Complexity: We structure training curricula by Kolmogorov complexity:

- Begin with problems requiring minimal compression
- Gradually increase complexity, forcing sophisticated compression
- Transfer learned compression strategies across domains

This approach aligns with the "Bitter Lesson" while maintaining theoretical grounding: we're not hand-coding compression strategies but learning them through complexity-aware optimization.

7 Contribution 3: Multiple Implementation Strategies

Our compositional action spaces and external controller architecture enable multiple implementation approaches, each with distinct advantages and use cases. We present three primary strategies and their hybrid combinations.

7.1 Neural Network Controllers (POMDP Approach)

The first implementation uses a neural network external controller trained via reinforcement learning in a POMDP formulation.

7.1.1 State Representation

We employ an enriched 948-dimensional state representation:

$$s_t = \langle e_c, f_{\text{struct}}, f_{\text{prog}}, f_{\text{hist}}, f_{\text{meta}}, f_{\text{trans}} \rangle$$

This captures semantic content (768D embedding), problem structure, reasoning progress, interaction history, metadata, and transition features.

7.1.2 Q-Learning with Compositional Actions

The neural network learns Q-values for compositional actions using the Bayesian network factorization:

$$Q(s, a) = \sum_i Q_i(s, a_i, \text{pa}(a_i))$$

Training uses prioritized experience replay and carefully designed reward signals including terminal correctness rewards and intermediate progress signals.

7.2 Tree Search Controllers (MCTS Approach)

The second implementation uses Monte Carlo Tree Search with compositional actions, requiring no neural network training.

7.2.1 Policy-Guided Exploration

Unlike random rollouts, we can use learned or heuristic priors over compositional actions:

$$\text{UCB}(s, a) = Q_{\text{MCTS}}(s, a) + C\sqrt{\frac{\ln N(s)}{N(s, a)}} + \beta\pi(a|s)$$

7.2.2 Component-wise Search Strategies

The factored action space enables structured exploration: - Beam search through components sequentially - Hierarchical search (strategy then tactics) - Multi-path consistency checking

7.3 LLM Meta-Reasoning Controllers

The third implementation uses the LLM itself to reason about optimal compositional strategies.

7.3.1 Context-Adaptive Strategy Selection

We prompt the LLM to evaluate and modify compositional actions based on problem context:

> *"Given this problem context, what values of $(,,,,)$ would be most effective for the next reasoning step?"*

7.3.2 Dynamic Action Refinement

The LLM can adapt fixed compositional templates to specific problem characteristics, providing contextual intelligence that learned policies might miss.

7.4 Hybrid Combinations

The three approaches can be combined in various ways:

1. NN-Guided MCTS: Neural network provides priors for tree search, combining learned patterns with systematic exploration.

2. MCTS-Trained NN: Tree search discovers high-quality action sequences that serve as training data for neural network distillation.

3. LLM-Adaptive NN: LLM refines neural network suggestions based on full context understanding.

4. Three-Layer Architecture: Neural network provides fast priors, MCTS provides systematic search, LLM provides contextual adaptation.

7.5 Extensible Prompt Composition via Fluid APIs

A key implementation insight is that compositional actions can serve as the foundation for an extensible prompt building system. Rather than limiting ourselves to fixed $(,,,,)$ combinations, we can create a fluid API that builds prompts incrementally:

```
prompt = (
    ComposingPrompt()
    .cognitive_op("decompose")
    .focus("structure")
    .style("systematic")
    .connect("therefore")
    .format("steps")
    .llm_augment("make this specific to geometry")
    .llm_coherence_check()
    .llm_add_examples(n=2)
    .build()
)
```

This creates a **hybrid discrete/continuous action space**:

Discrete Backbone: Core compositional components provide structured, learnable building blocks that can be systematically explored and optimized.

Continuous Extensions: LLM meta-reasoning operations ('.llm$_augment()$', '$.llm_coherence_check()$', '$.llm_add_examples()$', '$.llm_add_constraints()$', '$.llm_specificrefinement$.

This architecture offers several advantages:

- **Scalable Action Spaces:** New LLM-based operations can be added without retraining base policies
- **Compositional Flexibility:** Operations can be chained in arbitrary orders
- **Gradual Complexity:** Simple problems use basic compositional actions, complex problems add LLM meta-reasoning layers
- **Interpretable Pipelines:** Each step in the prompt construction is explicit and debuggable

The learning challenge becomes training policies over *when* to apply different operations rather than just *which* fixed action to take. This enables truly adaptive prompt engineering that combines structured reasoning with contextual intelligence.

7.5.1 Adaptive Deployment

Different strategies can be deployed based on problem complexity and computational budget: - Simple problems: Pure NN with basic compositional actions (fast, cheap) - Medium complexity: NN + shallow MCTS + selective LLM augmentation - Complex problems: Full MCTS + extensive LLM meta-reasoning pipeline - Novel domains: LLM suggests new compositional strategies and operation chains

7.6 Additional Training Challenges and Considerations

Beyond the compression-theoretic perspective, our framework faces several practical challenges that affect training efficiency and final capabilities:

7.6.1 Exploration in High-Dimensional Compositional Spaces

The compositional action space, while providing exponential expressiveness with linear parameters, presents unique exploration challenges. With $|\Omega| \times |\Phi| \times |\Sigma| \times |K| \times |T| = 30,720$ possible actions, naive ϵ-greedy exploration would require millions of episodes to discover effective action combinations.

Our Bayesian network structure offers a principled solution through factored exploration:

$$\epsilon_i = \epsilon_0 \cdot \sqrt{\frac{\log(N+1)}{n_i + 1}}$$

where n_i is the visit count for component i. This allows components to be explored at different rates based on their individual uncertainties, dramatically improving sample efficiency.

7.6.2 Non-Stationary LLM Dynamics

Unlike traditional RL environments, the underlying LLM represents a non-stationary MDP:

- API models update without notice, changing $P(r|s,a)$
- Temperature and sampling parameters affect stochasticity
- Context-dependent behavior varies with prompt history

We address this through:

- **Periodic re-evaluation**: Regularly test learned policies on validation sets to detect distribution shift
- **Online adaptation**: Continue learning with lower learning rates during deployment
- **Robust training**: Train against multiple LLM versions/parameters when possible

7.6.3 Context Window as Resource Constraint

The finite context window W creates a knapsack problem: how to optimally allocate tokens across reasoning steps. This connects to our compression framework but adds a sequential decision aspect:

$$\text{maximize} \sum_{t=1}^{T} r_t \quad \text{s.t.} \sum_{t=1}^{T} |c_t| \leq W$$

Optimal policies must learn to:

- Predict future token requirements
- Compress aggressively when approaching limits
- Trade off exploration depth vs. breadth

7.6.4 Catastrophic Forgetting in Continual Learning

As our controller encounters new problem domains, it faces the stability-plasticity dilemma. Standard Q-learning can catastrophically forget previously learned strategies when adapting to new tasks.

Mitigation strategies include:

- **Experience replay**: Maintain buffer of past experiences across domains
- **Elastic weight consolidation**: Protect important parameters identified through Fisher information
- **Progressive neural networks**: Add new capacity for new domains while freezing old

7.6.5 Delayed Feedback and Credit Assignment

While we've discussed reward engineering, the temporal credit assignment problem deserves special attention. In reasoning chains spanning 10-50 steps, determining which early actions enabled final success is non-trivial.

Beyond standard solutions (eligibility traces, n-step returns), we employ:

- **Counterfactual reasoning**: What if we had taken different actions at step t?
- **Attention-based credit**: Use attention weights to propagate credit
- **Hierarchical credit assignment**: Credit high-level strategies separately from low-level tactics

7.6.6 Adversarial Robustness and Distribution Shift

Our controller must handle adversarial or out-of-distribution inputs gracefully. Unlike supervised learning where we can detect OOD inputs, in RL we must act regardless.

Robustness measures:

- **Uncertainty quantification**: Ensemble Q-networks to estimate epistemic uncertainty
- **Safe exploration**: Constrain actions when uncertainty is high
- **Fallback strategies**: Revert to simple, reliable actions in unfamiliar states

7.6.7 Computational Efficiency and Deployment Constraints

While our external controller is lightweight compared to fine-tuning, inference still requires:

- State embedding computation (768d + features)
- Q-network forward pass
- Potential multiple LLM calls for exploration

Optimization strategies:

- **Caching**: Memoize state embeddings and Q-values
- **Distillation**: Train smaller student networks for deployment
- **Adaptive computation**: Use simpler policies for easy problems

These challenges, while significant, are not insurmountable. The compression-theoretic foundation provides principled ways to address many of them, while the compositional structure offers flexibility in solution design. As we gather experimental results, we expect to identify which challenges are most critical and develop targeted solutions.

7.7 Experimental Protocol

1) **Training Phase:**
   - Train the Q-learning agent using the methodology above on a diverse set of prompts.
   - Use both sparse terminal rewards and dense intermediate rewards to guide learning.
2) **Evaluation Phase:**
   - Test the learned policy on a held-out dataset.
   - Perform ablation studies to analyze the impact of different action subsets and reward components.
   - Measure performance against each baseline.
3) **Reproducibility:**
   - Utilize standardized hardware and publicly available datasets.
   - Release code, training logs, and hyperparameters upon final submission.

7.8 Results

Final Output Accuracy:
Reasoning Efficiency:
Computational Cost:
Interpretability (qualitative):
Generalization Performance:

Tables and figures summarizing these metrics will be included once experiments are complete.

8 Case Study

8.1 Mathematical Reasoning with Compositional Actions

Problem: Find the number of integer solutions to $|x| + |y| \leq 5$.

Compositional Action Sequence:

1) **Step 1:** $a_1 = (\omega = \text{decompose}, \phi = \text{structure}, \sigma = \text{systematic}, \kappa = \text{None}, \tau = \text{steps})$
   - **Constructed Prompt:** "Let me systematically break down the structure of this problem. Step-by-step:"
   - **LLM Response:** "The inequality $|x| + |y| \leq 5$ describes a diamond shape centered at origin..."
2) **Step 2:** $a_2 = (\omega = \text{analyze}, \phi = \text{constraints}, \sigma = \text{formal}, \kappa = \text{continue}, \tau = \text{list})$
   - **Constructed Prompt:** "Building on that, let me formally analyze the constraints. List the key points:"
   - **LLM Response:** "1. For each fixed value of $|x|$ from 0 to 5, $|y|$ can range from 0 to $5 - |x|$..."
3) **Step 3:** $a_3 = (\omega = \text{generate}, \phi = \text{solution}, \sigma = \text{systematic}, \kappa = \text{therefore}, \tau = \text{mathematical})$
   - **Constructed Prompt:** "Therefore, I'll systematically generate the solution:

$$\text{"}\textbf{LLM Response:}\text{"}Total = \sum_{k=0}^{5} 4(5-k) + 1 = 4(5+4+3+2+1) + 6 = 61\text{"}$$

4) **Step 4:** $a_4 = (\omega = \text{verify}, \phi = \text{correctness}, \sigma = \text{critical}, \kappa = \text{verify}, \tau = \text{solution})$
   - **Constructed Prompt:** "Let me verify that by critically checking correctness. The solution is:"
   - **LLM Response:** "Verified: 61 integer points satisfy the inequality."

Analysis:

- **Compositional Advantages:** The policy learned to combine formal analysis ($\sigma = \text{formal}$) with constraint focus ($\phi = \text{constraints}$) for mathematical problems
- **Coherence:** The connection components (κ) maintained logical flow throughout the reasoning chain
- **Interpretability:** Each component choice can be explained and adjusted if needed

9 Limitations and Future Work

Our theoretical framework has several limitations that present opportunities for future research:

- **Action Space Constraints:** Our discrete action space may not capture all effective reasoning strategies, potentially limiting performance on problems requiring novel approaches.
- **Embedding Limitations:** As discussed in our theoretical framework, embeddings necessarily lose information when compressing the full context into a fixed-dimensional vector. This information loss becomes particularly problematic for complex reasoning chains where nuanced details may be critical for optimal decision-making.
- **Reward Sparsity:** Terminal rewards provide limited guidance for credit assignment across long reasoning chains.
- **Computational Efficiency:** The iterative nature of our approach may require more computational resources than end-to-end methods for certain problems.

Future work will explore hierarchical action representations, improved embedding techniques specifically designed for reasoning contexts, and more sophisticated reward shaping approaches that preserve the advantages of our explicit framework while addressing these limitations.

10 Discussion

Our framework presents an interpretable, modular approach to prompting by explicitly modeling reasoning as a sequence of discrete actions. While experimental results are pending, we can discuss several important theoretical implications and potential applications of our approach.

10.1 Interpretability vs. Performance Trade-offs

The discrete action space provides transparency that latent chain-of-thought methods like o1/o3 and DeepSeek-R1 lack. This transparency serves multiple purposes:

- **Debugging and Analysis:** The explicit action sequence makes it possible to pinpoint exactly where reasoning goes wrong, potentially allowing for targeted improvements.
- **Trust and Verification:** In high-stakes domains such as medical diagnosis or financial analysis, the ability to audit reasoning steps could be critical for establishing trust.
- **Educational Applications:** The explicit reasoning steps could be leveraged for explaining complex problem-solving techniques to students, with the model demonstrating metacognitive strategies.

We hypothesize that these benefits may come with performance costs in some domains, and our experiments will help quantify this interpretability-performance trade-off.

10.2 Compositional Generalization

A key theoretical advantage of our discrete action framework is its potential for compositional generalization. By learning when to apply different prompting templates and tools based on state embeddings, our policy may develop the ability to tackle novel problems by composing familiar actions in new ways. This contrasts with end-to-end methods that might struggle with systematic generalization to problems requiring unfamiliar combinations of reasoning steps.

The degree to which this compositional advantage materializes in practice remains an empirical question that our experimental protocol is designed to evaluate.

10.3 Integration with External Systems in Unknown Environments

When treating LLMs as unknown environments, external tools become crucial anchors of predictability. Unlike the opaque LLM, tools have known, deterministic behaviors:

- **Retrieval as Exploration:** The controller learns to use RAG not because it understands what the LLM knows or doesn't know, but because it empirically observes better rewards when certain problem types trigger retrieval actions.
- **Tools as Reliable Sub-Environments:** While the LLM's behavior is unknown, tools like calculators or code interpreters have predictable input-output mappings. The controller learns to leverage these islands of certainty within the broader unknown environment.
- **Human Expertise Without Model Knowledge:** Domain experts can guide the controller by suggesting action sequences or modifying rewards, even without understanding the LLM's internals. This enables practical deployment where humans provide domain knowledge while the controller handles the unknown LLM dynamics.

The key insight is that the controller doesn't need to understand why tools help—it only needs to learn when using them leads to better outcomes in the unknown LLM environment.

10.4 Implications for LLM Training and Architecture

Our approach suggests that adding explicit reasoning modules to LLMs—rather than relying solely on latent mechanisms—may be beneficial for certain applications. This raises interesting questions about hybrid architectures that combine:

- Latent, end-to-end components for fluency and general knowledge
- Explicit, discrete components for complex reasoning and tool integration

Such hybrid approaches might offer a middle path between the "Bitter Lesson" emphasis on general methods and the need for interpretable, reliable systems in high-stakes domains.

10.5 Broader Implications

The broader challenge our work addresses is how to maintain human oversight and understanding of increasingly powerful reasoning systems. As LLMs grow in capability, ensuring that their reasoning processes remain interpretable becomes more important yet more difficult. Our MDP framework represents one approach to this challenge, providing a structured way to guide and interpret LLM reasoning without fully constraining its capabilities.

As our experimental results become available, we will be able to more concretely evaluate whether this approach effectively balances performance with interpretability across different domains and tasks.

11 Conclusion

We have presented a POMDP framework for learning to prompt large language models treated as unknown environments. Our approach addresses the fundamental challenge of optimizing interactions with powerful but opaque systems whose internal dynamics cannot be modeled or predicted. By combining compositional action spaces with belief-based Q-learning, we enable controllers to discover effective prompting strategies through pure trial-and-error learning, without ever understanding why these strategies work.

This framework makes three fundamental contributions:

First, it provides a principled formalization of the LLM prompting problem as a POMDP with unknown dynamics, acknowledging that we cannot observe the LLM's true state or predict its behavior. This framing makes explicit what is often implicit: that prompt engineering is fundamentally about learning to interact with an unknown environment.

Second, it demonstrates that rich compositional action spaces (over 10 million possible prompts through our framework) are essential for exploring unknown environments. Since we don't know a priori which prompts will work, diversity in the action space enables discovery through exploration.

Third, it shows that model-free reinforcement learning can discover effective strategies without model access. The controller learns which compositional actions lead to success purely from rewards, adapting to each LLM's specific quirks and biases through experience rather than understanding.

The compositional formulation addresses a critical challenge in unknown environments: how to explore a vast space of possible strategies efficiently. By factoring the action space through Bayesian networks, our approach can express millions of possible prompts while learning only thousands of parameters. This makes it feasible to discover effective strategies even when we cannot predict which ones will work.

Our work embodies the "Bitter Lesson" in its purest form: we make no assumptions about optimal prompting strategies, instead letting controllers discover what works through interaction with unknown LLM environments. This approach is particularly valuable as LLMs become increasingly complex and opaque. Rather than trying to understand these massive models, we can learn to use them effectively through systematic exploration and reinforcement learning.

As AI systems become more powerful but less interpretable, frameworks for learning to interact with them as unknown environments will become increasingly important. Our POMDP formulation provides a principled foundation for this challenge, demonstrating that we can achieve effective control without understanding—a crucial capability for the age of large, opaque AI systems.

References

Arora, S., Khandeparkar, H., Khodak, M., Plevrakis, O., & Saunshi, N. (2019). A theoretical analysis of contrastive unsupervised representation learning. In *International Conference on Machine Learning* (pp. 5628–5637).

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), 1798–1828.

Ellis, K., Wong, C., Nye, M., Sable-Meyer, M., Cary, L., Morales, L., Hewitt, L., Solar-Lezama, A., & Tenenbaum, J. B. (2021). DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning. *Proceedings of the National Academy of Sciences*, 118(9), e2015759118.

Fan, J., Wang, Z., Xie, Y., & Yang, Z. (2020). A theoretical analysis of deep Q-learning. In *Learning for Dynamics and Control* (pp. 486–489).

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1), 1–58.

Goodhart, C. A. E. (1975). Problems of monetary management: The UK experience. In *Papers in Monetary Economics*, Reserve Bank of Australia.

Hutter, M. (2005). Universal artificial intelligence: Sequential decisions based on algorithmic probability. *Springer*.

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

Li, L., Walsh, T. J., & Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs. In *International Symposium on Artificial Intelligence and Mathematics*.

Li, M., & Vitányi, P. (1997). An introduction to Kolmogorov complexity and its applications. *Springer-Verlag*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems* (pp. 3111–3119).

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14(5), 465-471.

Schmidhuber, J. (2010). Formal theory of creativity, fun, and intrinsic motivation. *IEEE Transactions on Autonomous Mental Development*, 2(3), 230-247.

Silver, D., Singh, S., Precup, D., & Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, 299, 103535.

Solomonoff, R. J. (1964). A formal theory of inductive inference. *Information and Control*, 7(1-2), 1-22, 224-254.

Sutton, R. (2019). The bitter lesson. *Incomplete Ideas (blog)*, March 13, 2019. http://www.incompleteideas.net/IncIdeas/BitterLesson.html

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4), 279–292.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023). Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. *International Conference on Learning Representations*.

Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Gianinazzi, L., Gajda, J., ... & Hoefler, T. (2024). Graph of thoughts: Solving elaborate problems with large language models. *AAAI Conference on Artificial Intelligence*.

Hao, S., Gu, Y., Ma, H., Hong, J., Wang, Z., Wang, D., & Hu, Z. (2023). Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10), P10008.