# Progress Reward Model for Reinforcement Learning via Large Language Models

Xiuhui Zhang, Ning Gao, Xingyu Jiang, Yihui Chen,
Yuheng Pan, Mohan Zhang, Yue Deng

Beihang University
NeurIPS 2025

December 12, 2025

Presented by Liwen Sun

# Overview

# The Challenge: Long-Horizon RL Tasks

**Traditional RL struggles with:**

- **Sparse rewards**: Only feedback at task completion
- **Long horizons**: Complex tasks require many steps
- **Multi-stage structure**: Need to accomplish subtasks in sequence

### Example (Robot Pick-and-Place Task)

1. Move gripper to object
2. Grasp object
3. Move to goal location
4. Release object

Sparse reward: $+1$ only when object reaches goal. All intermediate steps get 0.

**Why is this hard?** Without intermediate feedback, agent must explore blindly.

# Existing Solutions and Their Limitations

**Traditional Approaches**

- **Hierarchical RL**: Decompose autonomously
  - Requires expert data
  - Hard to scale
- **Reward shaping**: Design dense rewards
  - Needs domain expertise
  - Task-specific engineering

**LLM-Augmented Approaches**

- **LLM as Planner**: High-level decomposition
  - Lacks low-level guidance
  - Needs pre-trained policies
- **LLM as Rewarder**: Generate reward code
  - Hard for complex tasks
  - Requires extensive search

**Key Gap**: Current methods focus on *either* planning *or* reward, not both!

# LLM-Augmented RL: Two Paradigms


Figure: (a) LLM as Planner: Decompose task, but policy learns from sparse rewards. (b) LLM as Rewarder: Dense rewards, but no task decomposition. (c) PRM4RL: Combines both!

# Our Contribution: PRM4RL

**Progress Reward Model for RL (PRM4RL):** Unified framework integrating planning & reward shaping

## Key Idea

Use LLM to decompose task into subtasks, then construct a **progress function** that tracks execution. Apply this as **potential-based reward shaping** for theoretically-grounded dense rewards.

**Main Contributions:**

1. **High-level**: Subtask decomposition with automatic tracking (no repeated LLM calls)
2. **Low-level**: Progress Reward Model with optimality & convergence guarantees
3. **Empirical**: State-of-the-art on MetaWorld and ManiSkill benchmarks
4. **Theoretical**: Formal proofs connecting progress to potential-based shaping

# Background: Reinforcement Learning

**Markov Decision Process (MDP)**: $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma, \rho_0 \rangle$

- $\mathcal{S}$: State space
- $\mathcal{A}$: Action space
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$: Transition function
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: Reward function
- $\gamma \in [0, 1]$: Discount factor
- $\rho_0$: Initial state distribution

**Goal**: Find policy $\pi : \mathcal{S} \to \mathcal{A}$ maximizing expected return:

$$J(\pi) = \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[ \sum_{t=0}^{T} \gamma^t R(s_t, a_t) \right]$$

**Q-function**: Expected return from state-action pair:

$$Q^\pi(s, a) = \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[ \sum_{k=0}^{T} \gamma^k R(s_{t+k}, a_{t+k}) \mid s_t = s, a_t = a \right]$$

# Background: Large Language Models for RL

**Why use LLMs for RL?**

- Pre-trained on diverse human knowledge
- Strong reasoning and planning capabilities
- Can generate code/structured outputs

**Recent Approaches:**

- **SayCan** (Brohan et al., 2023): LLM plans, selects from pre-trained policy library
- **Plan-Seq-Learn** (Dalal et al., 2024): LLM decomposes task, uses motion planning
- **ELLM** (Du et al., 2023): LLM generates plans, compute similarity as reward
- **Text2Reward** (Xie et al., 2024): LLM writes reward functions in Python
- **Eureka** (Ma et al., 2023): Evolutionary search over LLM-generated rewards

**Gap**: Planning methods lack dense rewards; reward methods lack decomposition

# Method Pipeline

1. **Subtask Decomposition**
   - LLM breaks task into subtasks with "verb + noun" descriptions
   - Generates determination function $\Psi(s)$ to identify current subtask
2. **Progress Function Construction**
   - For each subtask, LLM designs fine-grained progress metric
   - Combine into overall progress $\Phi(s)$
   - Use as potential function for reward shaping
3. **Policy Training**
   - Augment state with subtask prior embedding
   - Train with Progress Reward Model (PRM)
   - Use standard RL algorithms (SAC/PPO)

# Step 1: Prompting the LLM

**Goal**: Decompose complex task into manageable subtasks

**Prompt Design** (Pythonic representation):

- **Environment Info**: State space definition as Python classes
  - Robot attributes: end-effector position, joint states
  - Object attributes: position, orientation, initial states
  - Goal specification
- **Task Description**: Natural language specification
- **Instructions**: Chain-of-Thought reasoning guide

**Output Format**: Python code with comments for reasoning

- Comments (#) for intermediate thoughts
- Executable code for final outputs

# Prompting Strategy: Overview

**Goal**: Generate code that tracks progress during task execution

**Prompt Structure** (following Text2Reward's Pythonic approach):

1. **Basic Prompt**: Establishes role and objective
2. **Environment Information**: Defines state space as Python classes
3. **Task Information**: Natural language task description
4. **Basic Instructions**: Code generation guidelines
5. **Reasoning Instructions**: Step-by-step procedure

**Output**: LLM generates `plan_list`, $\Psi(s)$, and $\Phi(s)$ in single call

**Key Innovation**: Program-aided Chain-of-Thought

- Reasoning in comments (#)
- Implementation in executable code

# Prompting: Basic Prompt

## Basic Prompt

You are an expert in robotics, reinforcement learning and code generation. We are going to use a robot arm to complete given tasks. Now I want you to help me write a python function named 'progress function' of reinforcement learning.

**Purpose**:

- Establishes LLM's role as expert
- Focuses on developing a `progress_function`
- Sets context: tracking progress to guide low-level policy learning

# Prompting: Environment Information

**Pythonic Class Representation**:

## Environment Classes

```
class BaseEnv(gym.env):
    self.robot: Robot  # the robot in the environment
    self.obj: RigidObject  # the first object in the environment
    self.goal_position: np.ndarray[(3,)]  # indicate the 3D position of the goal
    near_object = float(tcp_to_obj <= 0.3)
    success = float(obj_to_target <= 0.07)

class Robot:
    self.ee_position: np.ndarray[(3,)]  # indicate the 3D position of effector
    self.hand_init_pos: np.ndarray[(3,)]  # indicate the initial 3D position
    self.tcp = self.ee_position - [0, 0, 0.045]  # tool center point

class RigidObject:
    self.position: np.ndarray[(3,)]  # indicate the 3D position of object
    self.quaternion: np.ndarray[(4,)]  # indicate the quaternion of object
    self.obj_init_pos: np.ndarray[(3,)]  # indicate the initial 3D position
```

**Benefits**: Compact, informative, bootstraps Python code generation

# Prompting: Task Information

## Task Information

{ Fill in the task description here. }

*Example*: In pick-place, the robot need to pick up the object and move it to the goal position.

**Purpose**:

- Provides natural language specification of the goal
- User fills in specific task description
- Crucial for LLM to understand what to decompose

**Other Examples**:

- "Close a drawer by its handle"
- "Rotate the faucet handle counter-clockwise"
- "Press a button in y coordination"

# Prompting: Basic Instructions

## Basic Instruction

1. You are allowed to use any existing python package if applicable. But only use these packages when it's really necessary.
2. Do not invent any variable or attribute that is not given.
3. Think step by step, add comment for reasoning and thought when you write code.

**Rationale**:

- **Instruction 1**: Minimize dependencies, only import when needed (e.g., numpy)
- **Instruction 2**: Stay within defined environment - ensures code is executable
- **Instruction 3**: Explainability and debugging through comments

# Prompting: Reasoning Instructions (Overview)

## Reasoning Instructions(simplified)

Follow the steps below:

**a. Decompose the task and generate plan_list**

**b. Write a 'determination_function' for determine which subtask we are in**

**c. Write a 'progress_function'**

**Strategy**: Step-by-step guidance for LLM

# Step 2: LLM Generates Subtask List

**Output 1**: `plan_list` - ordered subtask descriptions

## Example (Pick-Place Task)

```
plan_list = ['move to object', 'grasp object', 'move to goal', 'release object']
```

**Design Choice**: "verb + noun" format

- **Consistency**: Same patterns across different tasks
- **Generalization**: Transfers to unseen tasks with similar structure
- **Compositionality**: Natural building blocks for complex behaviors

Common patterns: "move to X", "grasp X", "open X", "close X", "push X", etc.

# Step 3: Determination Function

**Challenge**: How to know which subtask the agent is currently executing?

**Naive Solution**: Call LLM at every timestep

- Too slow and expensive
- Used by some prior work (ELLM)

**Our Solution**: Generate determination function $\Psi(s)$ once

## Determination Function

$$\Psi : \mathcal{S} \to \{0, 1, ..., N-1\}$$

Maps current state $s$ to index of current subtask in `plan_list`.

**Key Insight**: With good understanding of state space, can determine subtask from state alone!

# Example: Determination Function

## Generated Python Code

```
def determination_function(env, plan_list):
    # Subtask 1: 'move to object'
    if np.linalg.norm(env.robot.tcp - env.obj.position) > 0.3:
        return 0  # Still moving towards object
    # Subtask 2: 'grasp object'
    elif env.obj.position[2] <= env.obj.obj_init_pos[2]:
        return 1  # Object not yet grasped (still at initial height)
    # Subtask 3: 'move to goal'
    elif np.linalg.norm(env.obj.position - env.goal_position) > 0.07:
        return 2  # Still moving towards goal
    # Subtask 4: 'release object'
    else:
        return 3  # Task completed
```

**Logic**: Each subtask has completion condition based on state features

# Constructing the Progress Function

**Goal**: Fine-grained tracking of task execution progress

**For each subtask** $i$: Define sub-progress $\phi_i(s) \in [0, 1]$

- $0 =$ subtask just started
- $1 =$ subtask completed

**Common sub-progress metrics**:

- **Distance-based**: $\phi(s) = 1 - \frac{d_{\text{current}}}{d_{\text{initial}}}$
- **Height-based**: $\phi(s) = \frac{h_{\text{current}} - h_{\text{initial}}}{h_{\text{target}} - h_{\text{initial}}}$
- **Angle-based**: For rotation tasks

# Example: Progress Function

## Generated Python Code (Simplified)

```
def progress_function(env):
    subtask_idx = determination_function(env, plan_list)
    if subtask_idx == 0:  # 'move to object'
        init_dist = ||tcp_init - obj_init||
        curr_dist = ||tcp - obj||
        subprogress = 1 - curr_dist / init_dist
    elif subtask_idx == 1:  # 'grasp object'
        subprogress = (obj.z - obj_init.z) / 0.5
    elif subtask_idx == 2:  # 'move to goal'
        init_dist = ||obj_init - goal||
        curr_dist = ||obj - goal||
        subprogress = 1 - curr_dist / init_dist
    main_progress = subtask_idx + subprogress
    return main_progress, subtask_idx
```

# From Progress to Reward

**Key Idea**: Use progress function $\Phi(s)$ as potential function!

## Progress Reward Model (PRM)

$$R_t^{PRM} = \gamma \cdot \Phi(s_{t+1}) - \Phi(s_t) + I(s_{t+1}) \cdot r_{bonus}$$

**Components**:

- $\gamma \cdot \Phi(s_{t+1}) - \Phi(s_t)$: **Progress reward** (potential-based shaping)
  - $\gamma$: discount factor
  - Positive when making progress
  - Zero when stuck
  - Negative if regressing (rare)
- $I(s_{t+1}) \cdot r_{bonus}$: **Terminal reward** when task succeeds
  - Maintains original task objective
  - Typically $r_{bonus} = 10$ or $100$

**Result**: Dense feedback at every step while preserving optimal policy!

# Why Does This Work?

**Intuition**: Progress reward guides exploration toward completion

## Example (Pick-Place Scenario)

- Agent moves gripper toward object: $\Phi$ increases $\rightarrow$ positive reward
- Agent grasps object: $\Phi$ increases $\rightarrow$ positive reward
- Agent moves object toward goal: $\Phi$ increases $\rightarrow$ positive reward
- Agent moves away from goal: $\Phi$ decreases $\rightarrow$ negative reward

**Comparison to alternatives**:

- **Sparse reward only**: No guidance until completion
- **Distance reward only**: May not capture multi-stage structure
- **Progress directly as reward**: Can lead to unintended behaviors (see ablation)
- **PRM (potential-based)**: Theoretically grounded, guides efficiently

# Theorem 1: Policy Invariance

**Theorem (Policy Invariance)**

*Given MDP* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma, \rho_0 \rangle$ *with sparse reward* $R_t = I(s_{t+1}) \cdot r_{bonus}$, *and potential-based shaping:*

$$F = \gamma \cdot \Phi(s_{t+1}) - \Phi(s_t)$$

*Let* $\mathcal{M}' = \langle \mathcal{S}, \mathcal{A}, T, R + F, \gamma, \rho_0 \rangle$. *Then every optimal policy in* $\mathcal{M}'$ *is also optimal in* $\mathcal{M}$, *and vice versa.*

**What does this mean?**

- PRM doesn't change what the optimal policy should do
- No "reward hacking" or "deceptive" behaviors
- Agent still maximizes task success, just learns faster

**Proof sketch**: Show $Q^*_{\mathcal{M}'}(s, a) = Q^*_{\mathcal{M}}(s, a) - \Phi(s)$, so $\arg\max_a$ unchanged.

# Theorem 2: Convergence Efficiency

**Theorem (Convergence Efficiency)**

*Consider two MDPs: $\mathcal{M}$ with reward $R + F$ and Q-function $Q$, and $\mathcal{M}'$ with reward $R$ and Q-function $Q'$. If we initialize:*

$$Q'_0(s, a) = Q_0(s, a) + \Phi(s)$$

*Then for all $t \geq 0$:*

$$Q'_t(s, a) = Q_t(s, a) + \Phi(s)$$

**Practical implication**:

- PRM $\equiv$ good Q-value initialization
- Progress function provides prior knowledge: reduces initial error $|Q_0 - Q^*|$
- Accelerates convergence without changing optimum
- Shaping term acts as curriculum: guides exploration to high-value regions

# Theoretical Summary

**Corollary: Convergence Guarantee**

Potential-based reward shaping preserves convergence guarantees of RL algorithms.

**Why these results matter**:

1. **Safety**: Won't learn wrong behavior due to reward shaping
2. **Efficiency**: Provably faster convergence than sparse rewards
3. **Generality**: Applies to any RL algorithm (Q-learning, policy gradient, etc.)
4. **Design principle**: Progress is natural choice for potential function

**Contrast with LLM-reward baselines**:

- Text2Reward, Eureka: No theoretical guarantees
- May require extensive tuning or human feedback
- PRM: One-shot generation with formal properties

# Augmenting the MDP

**Two augmentations to improve learning**:

1. **State Augmentation**: Add subtask prior information
   - Compute current subtask: $i = \Psi(s)$
   - Get subtask description: desc = plan_list[$i$]
   - Embed description: $\mathcal{P}(s) = \text{Encoder}(\text{desc})$
     - Uses SimCSE sentence encoder
     - Produces fixed-size vector embedding
   - Augmented state: $s' = [s; \mathcal{P}(s)]$

2. **Reward Augmentation**: Use PRM reward

$$R_t^{PRM} = \gamma \cdot \Phi(s_{t+1}) - \Phi(s_t) + I(s_{t+1}) \cdot r_{bonus}$$

**Augmented MDP**:

$$\mathcal{M}' = \langle \mathcal{S} + \mathcal{P}, \mathcal{A}, T, R^{PRM}, \gamma, \rho_0 \rangle$$

# Training Procedure

**Algorithm 1** PRM4RL Training

```
Input: Environment, task description, RL algorithm
Offline Phase (once):
    Construct Pythonic prompt with environment info
    Call LLM to generate: plan_list, Ψ(s), Φ(s)
Training Loop:
for episode = 1 to N do
    Initialize state s_0
    for timestep t = 0 to T do
        Compute subtask prior: P(s_t) = Encoder(plan_list[Ψ(s_t)])
        Augment state: s'_t = [s_t; P(s_t)]
        Select action: a_t ~ π(s'_t)
        Execute action, observe s_{t+1}
        Compute reward: r_t = γ · Φ(s_{t+1}) − Φ(s_t) + I(s_{t+1}) · r_bonus
        Update policy with (s'_t, a_t, r_t, s'_{t+1})
    end for
end for
```

# Experimental Setup

**Environments**:

**MetaWorld**

- 7-DOF Sawyer robot
- Tabletop manipulation
- 10 tasks evaluated
- Examples: button-press, door-close, pick-place

**ManiSkill**

- 7-DOF Franka Panda
- High-fidelity physics
- 5 tasks evaluated
- Examples: LiftCube, PickCube, LiftPegUpright

**Baselines**:

- **ELLM**: LLM planner + subtask priors, sparse/subtask rewards
- **Text2Reward (T2R)**: LLM generates dense reward function
- **Oracle**: Expert-designed dense rewards (upper bound)

**Evaluation**: Success rate averaged over 5 random seeds

# MetaWorld

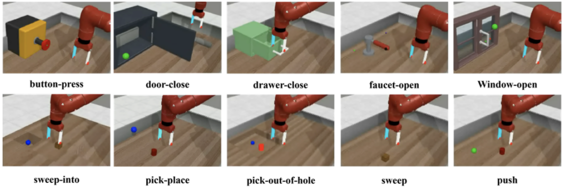



Figure 9: Tasks in Metaworld.

Table 3: Task list of Metaworld

| **Task** | **Task Description** |
|---|---|
| button-press | Press a button in y coordination. |
| door-close | Close a door with a revolving joint by pushing the door's handle. |
| drawer-close | Close a drawer by its handle. |
| faucet-open | Rotate the handle counter-clockwise. |
| window-open | Push and open a sliding window by its handle. |
| sweep-into | Sweep a puck from the initial position into a hole. |
| pick-place | Pick up an object and move it to the goal location. |
| pick-out-of-hole | Pick an object out of a hole and move it to the goal location. |
| sweep | Sweep a puck off the table. |
| push | Push an object to the goal location. |

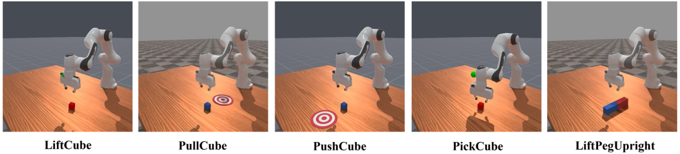



Figure 10: Tasks in Maniskill.

Table 4: Task list of Maniskill

| Task | Task Description |
|---|---|
| LiftCube | Pick up cube A and lift it up by 0.2 meters. |
| PullCube | Pull a cube to the goal position. |
| PushCube | Push a cube to the goal position. |
| PickCube | Pick up cube A and move it to the 3D goal position. |
| LiftPegUpright | Move a peg laying on the table to any upright position on the table. |

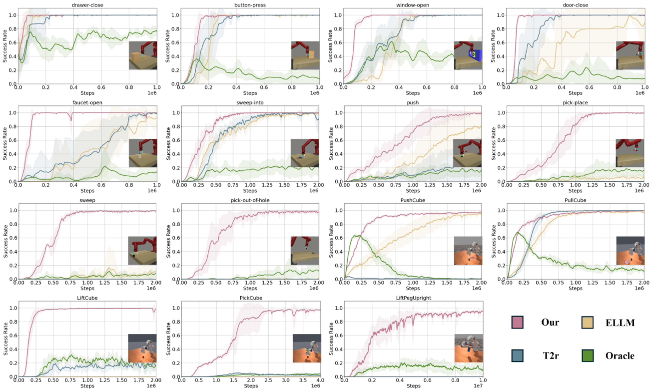



Figure 3: The learning curves of comparison methods on Metaworld and Maniskill measured by success rate. All experiments are conducted with 5 random seeds.

Figure: Training curves across tasks. PRM4RL (red) shows faster convergence and higher final performance than all baselines.

# Main Results: Final Performance

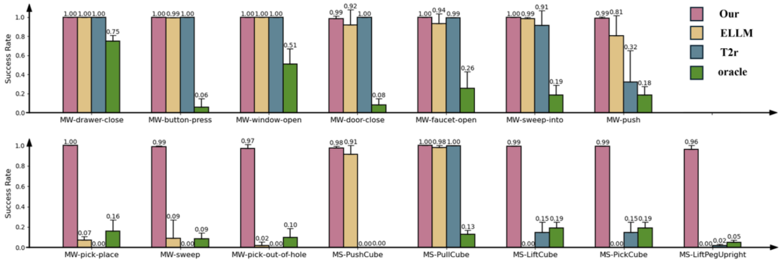


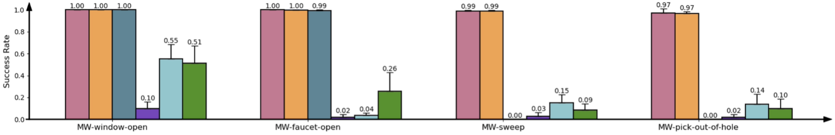

Figure 4: The evaluation results of comparison methods on Metaworld(With prefix 'MW') and Maniskill(with prefix 'MS') measured by success rate. All experiments are conducted with 5 random seeds.

Figure: Final success rates after training. PRM4RL achieves near-perfect performance on complex tasks where baselines fail.

# Key Findings

1. **Solves Complex Tasks**:
    - pick-out-of-hole: PRM4RL 100%, baselines $<$15%
    - LiftPegUpright: PRM4RL 98%, baselines $<$10%
    - Tasks with intricate multi-stage structure
2. **Faster Convergence**:
    - Reaches high performance 2-3$\times$ faster than ELLM
    - PRM rewards accelerate learning significantly
3. **Outperforms Direct Reward Design**:
    - Beats Text2Reward on most tasks
    - Subtask decomposition simplifies reward specification
    - More reliable than evolutionary search
4. **Approaches Oracle**:
    - Competitive with expert-designed rewards
    - Sometimes even better (better exploration guidance)

# Ablation Study: Both Components Matter




**(b) Evaluation results of ablation experiment.**



**Variants tested**:

- **w.o. PRM**: Subtask prior + oracle reward → Poor performance
- **w.o. prior**: PRM reward only → Slower convergence
- **w.o. potential**: Direct progress as reward → Unstable, degrades over time

**Conclusion**: Synergistic "1+1 >2" effect when combining both augmentations

# Generalization to Unseen Tasks

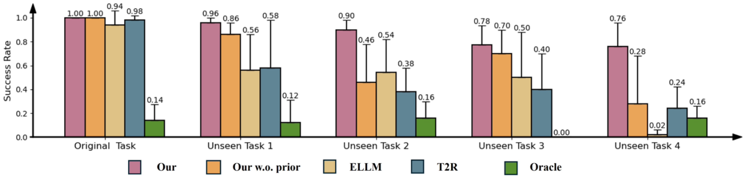



Figure 7: The evaluation results of generalization experiment. The results are averaged on 5 random seeds and 100 evaluations per seed.

Figure: Transfer performance: Train on one task, evaluate on related unseen tasks

**Setup**: Train on faucet-open, test on button-press, drawer-close, etc.

**Results**:

- PRM4RL shows smallest performance drop
- "verb + noun" pattern captures shared structure
- Subtask priors crucial for generalization (compare w.o. prior)

# Strengths

1. **Theoretical Foundation**
   - Rare among LLM-RL papers to have formal guarantees
   - Connects to established theory (potential-based shaping)
2. **Unified Framework**
   - Integrates planning and reward shaping naturally
   - Each component addresses specific limitation
3. **Efficiency**
   - Single LLM call vs. iterative search
   - No per-timestep LLM invocation
4. **Strong Empirical Results**
   - Consistent improvements across diverse tasks
   - State-of-the-art on benchmarks
5. **Thorough Ablations**
   - Clearly demonstrate necessity of both components
   - Show importance of potential-based formulation

# Limitations

1. **LLM Hallucination Risk**
   - Method fails if LLM generates incorrect decomposition/progress
   - No validation or error-handling discussed
   - May need human verification in practice
2. **Domain Specificity**
   - Only tested on robotics manipulation
   - Requires structured state representations
   - Unclear how to extend to other RL domains (games, dialogue, etc.)
3. **Prompt Engineering**
   - Success depends on prompt quality
   - Pythonic prompts require domain expertise
   - Different environments need different templates
4. **Limited Baseline Comparison**
   - Text2Reward uses mixed zero/few-shot settings
   - Could potentially be stronger with more tuning

# Open Questions

**Practical Implementation**:

- How to validate LLM-generated code automatically?
- What to do when determination function is inaccurate?
- How sensitive to choice of sentence encoder?

**Generalization**:

- Can this work for non-robotic RL domains?
- How to handle tasks without clear stage structure?
- What about partially observable environments?

**Scaling**:

- Does it work with smaller/cheaper LLMs?
- Can we learn to refine progress functions from experience?
- How to handle very long task horizons ($>$10 subtasks)?

# Relevance to Healthcare

**Clinical Decision-Making Applications**:

1. **Treatment Planning**
   - Decompose protocols: "stabilize patient" $\rightarrow$ "administer treatment" $\rightarrow$ "monitor recovery"
   - Progress: Track patient state toward clinical milestones
   - More interpretable than black-box RL
2. **Discharge Recommendation**
   - Subtasks: "assess stability", "medication reconciliation", "arrange follow-up"
   - Progress function: Multi-dimensional readiness score
   - PRM framework for sequential decision support
3. **Surgical Workflow**
   - Natural stage structure in procedures
   - Progress tracking for real-time guidance
   - Safety-critical: need theoretical guarantees!

**Challenge**: Medical LLMs need careful validation and domain-specific prompting