SafeNet: A Neural-Symbolic Network for Safe Planning in Robotic Systems using Formal Method-Guided LLM Fine-Tuning

Zifan Wang[1], Jialiang Fan[2], Rui Zuo[1], Qinru Qiu[1], Fanxin Kong[2]

[1]*Syracuse University* {zwang345, rzuo02, qiqiu}@syr.edu
[2]*University of Notre Dame* {jfan5, fkong}@nd.edu

***Abstract*— Robotic systems present unique safety challenges due to their complex integration of computational and physical processes and direct interaction with humans and environments. Traditional approaches to robot safety planning either rely on conventional methods, which struggle with the complexity of modern robotic systems, or on pure machine learning techniques, which lack formal safety guarantees. While recent advances in Large Language Models (LLMs) offer promising capabilities, pre-trained LLMs alone lack the specific domain expertise required for effective robotic safety planning. This paper introduces SafeNet, a novel neural-symbolic network architecture that enhances LLMs' safety planning capabilities through formal method-guided fine-tuning for robotic applications. Our approach integrates formal logical knowledge and reward machines into pre-trained LLMs by carefully designed fine-tuning, creating a neural-symbolic approach that combines the flexibility of neural networks with the precision of formal methods for robot trajectory generation and task planning. Experimental results demonstrate significant improvements in safe trajectory generation for robotic systems, with planning success rates increasing from 1.17% to 91.60% for the block manipulation task and from 7.23% to 90.63% for the robotic path planning task.**

I. Introduction

Robotic systems integrate computation and physical processes, where embedded computers monitor and control actuators, sensors, and end-effectors to interact with the real world. They are widespread in society, from autonomous robots and industrial manipulators to service and collaborative robots. The core challenge lies in bridging the discrete logic of computation with the continuous dynamics of mechanics and environments, which complicates system design [1], implementation [2], and validation [3]. Safety is a critical concern, as these systems often operate around humans. Failures can cause severe consequences, e.g., a malfunctioning mobile robot could lead to collisions and injuries [4], highlighting the paramount importance of safety [5].

Ensuring the safety of robotic systems necessitates a proactive approach, with safety planning conducted well in advance of deployment [6]. This preemptive strategy is effective because robotic systems operate within well-defined constraints and parameters. For example, robotic manipulators have specific operational ranges, such as joint limits, velocity constraints, and workspace boundaries, while autonomous mobile robots must adhere to predetermined acceleration limits, obstacle avoidance requirements, and navigation constraints. Similarly, these systems must operate within environments that have their own set of constraints, such as obstacle locations, goal-reaching objectives, and human interaction zones. These constraints can be both static (fixed obstacles, workspace boundaries) and dynamic (moving obstacles, changing environmental conditions, human presence). By leveraging this prior knowledge of robotic systems and environmental constraints, developers can implement robust safety measures that significantly mitigate risks associated with robot operation.

Historically, approaches to robotic safety planning have relied on conventional methods [7] or pure machine learning techniques [8]. Traditional methods, such as model predictive control and potential field approaches for robot navigation, often struggle with the complexity and dynamism inherent in modern robotic applications, while pure machine learning approaches suffer from training quality issues, including data bias, generalization limitations, and the inability to provide formal safety guarantees for robot behavior. The recent advancements in Large Language Models (LLMs) have opened new avenues for addressing these challenges in robotics. The remarkable capabilities of LLMs in natural language understanding and generation, coupled with their ability to learn complex patterns and relationships, have motivated researchers to explore their potential in the domain of safe planning for robotic systems [9]. However, pre-trained LLMs, despite their impressive general knowledge, lack the specific domain expertise required for effective robotic safety planning [10]. Their broad training data and general-purpose nature render them inadequate for the specialized task of ensuring safety in complex robotic environments.

To address this limitation, we propose a novel approach that leverages formal methods as domain knowledge to fine-tune pre-trained LLMs. This process results in the creation of neural-symbolic LLMs that possess a deep understanding of robotic systems and their associated environmental constraints. By incorporating formal logical knowledge into LLMs, we imbue the LLMs with the rigorous logical frameworks and mathematical precision necessary for effective safety planning. This integration allows the system to maintain the flexibility and adaptability of neural networks while ensuring compliance with formal safety requirements.

The key contributions of this work are twofold:

- We propose a novel approach to safe robotic planning by integrating formal knowledge into fine-tuned LLMs, bridging the gap between safety-critical robotics engineering and AI-driven autonomy.

- We provide a comprehensive comparative analysis of pre-trained foundation models of varying scales, offering insights into performance–cost tradeoffs and evaluating their effectiveness in safe trajectory planning and robotic task execution.

The rest of the paper is formulated as follows. Section II presents related works. Section III presents the preliminaries of the proposed approach, covering essential concepts from both formal methods and machine learning domains. Section IV presents the design and implementation details of the proposed approach. Section V presents the experiments and results of the evaluation of the proposed approach.

II. Related Works

A. Formal Methods for Safe Planning

Due to the mathematical rigor, formal methods can be applied and integrated into planning frameworks to enable safety validation and to provide formal guarantees. For example, Linear Temporal Logic (LTL) has been employed to automatically verify temporal planning outcomes [11]. In addition, Cho *et al.* [12] proposed a cost-aware path planning algorithm that integrates LTL with RRT to generate low-cost and dynamically feasible trajectories. Zuo *et al.* [13] developed CommA*, which incorporates reliability as a safety constraint, enabling robust trajectory generation. Alqahtani *et al.* [14] introduced Robust A*, which leverages Metric Temporal Logic (MTL) to define dynamic safety margins. Furthermore, formal methods have been combined with reinforcement learning to support robotic planning, ensure safety, and guide policy synthesis through temporal logic and automaton-based approaches [15].

B. LLMs for Safe Planning

Existing studies leveraging LLMs for robotic task planning can be broadly categorized into prompt engineering–based approaches and fine-tuning–based approaches. For example, Yang *et al.* [16] proposed a framework that translates natural language task descriptions into automata, which are subsequently used to validate planning sequences. Similarly, Van *et al.* [17] employed LLMs to translate natural language into STL specifications and to validate their syntax, correctness, and safety in planning. Obi *et al.* [18] further leveraged formal logic and chain-of-thought reasoning to conduct multi-stage checking and generation, ensuring both the safety of user prompts and the executability of generated robotic task planning code. Pan *et al.* [19] introduced a data-centric method that synthesizes training data via rule-based LTL generation, enabling LLM fine-tuning for real-world robotic task planning. [20] proposed a framework in which formal methods are used to construct reward machines, where every state and state transition is explicitly defined and governed by formal method equations. While this approach demonstrates the promise of integrating formal methods into LLM fine-tuning, it requires manual decomposition of the formal specification into individual states and transitions, which can be labor-intensive and difficult to scale. In contrast, our approach automatically computes a scalar compliance score for each state without requiring explicit decomposition.

C. Neural symbolic

A neuro-symbolic network merges symbolic reasoning with neural network learning, leveraging the strengths of both approaches to create intelligent systems that are both accurate and interpretable. This integration addresses fundamental limitations of pure neural networks, such as their black-box nature and difficulty in incorporating domain knowledge, while overcoming the rigidity and scalability issues of purely symbolic systems. Based on [21], these hybrid approaches can be categorized into five distinct classes: Symbolic[Neuro], Neuro[Symbolic], Neuro—Symbolic, $\text{Neuro}_{\text{Symbolic}}$, and Neuro→Neuro. Our research focuses on the Neuro→Neuro paradigm, which incorporates symbolic rules as guidance mechanisms for neural network learning, enabling more structured and constrained learning processes.

The incorporation of symbolic rules into the learning process requires sophisticated methodology to effectively utilize symbolic methods in guiding neural network training. To address this challenge, we implement formal method-based reward machines as our primary mechanism for evaluating network-generated outputs. These reward machines provide a structured framework for calculating rewards that reflect both the accuracy of outputs and their adherence to symbolic rules and constraints, enabling efficient training guidance while maintaining computational tractability.

III. Preliminaries

This section introduces key concepts and definitions that form the foundation for our subsequent methodology.

A. Signal Temporal Logic

Signal Temporal Logic (STL) is a real-time temporal logic that extends LTL with real-time constraints and the ability to reason about dense-time, real-valued signals [22]. This extension is particularly valuable for robotic systems, where continuous dynamics and timing constraints play crucial roles in system behavior. An STL formula can be expressed using the following syntax:

$$\phi := \psi \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid F_{[t_1,t_2]}\phi \mid G_{[t_1,t_2]}\phi \mid \phi U_{[t_1,t_2]}\psi,$$

where ψ is an atomic predicate representing a basic proposition about the system state. The operators $\neg$, $\wedge$, and $\vee$ are the standard boolean operators (not, and, or) that allow for logical combinations of simpler formulas. The temporal operators $F_{[t_1,t_2]}$, $G_{[t_1,t_2]}$, and $U_{[t_1,t_2]}$ enable reasoning about temporal properties within specific time intervals: $F_{[t_1,t_2]}$ specifies that ϕ holds at some point during $[t_1, t_2]$, $G_{[t_1,t_2]}$ requires that ϕ holds at all points during $[t_1, t_2]$, and $U_{[t_1,t_2]}$ indicates that ϕ holds until ψ holds, with ψ occurring at some point during $[t_1, t_2]$.

The quantitative semantics [23] of STL, denoted as $\rho(\phi, S^{i,j}, t)$, measures how well a trajectory $S^{i,j}$ satisfies an STL formula ϕ at time t. This score provides rich feedback

about requirement satisfaction: $\rho \geq 0$ denotes satisfaction and $\rho < 0$ denotes violation, with the magnitude representing the degree of satisfaction or violation. The score is recursively calculated for each operator in the STL, allowing for systematic evaluation of complex specifications. Formally:

$$\rho(\mu > c, S^{i,j}, t) = \mu(S^{i,j}[t]) - c,$$
$$\rho(\mu < c, S^{i,j}, t) = c - \mu(S^{i,j}[t]),$$
$$\rho(\neg\phi, S^{i,j}, t) = -\rho(\phi, S^{i,j}, t),$$
$$\rho(\phi \wedge \psi, S^{i,j}, t) = \min(\rho(\phi, S^{i,j}, t), \rho(\psi, S^{i,j}, t)),$$
$$\rho(\phi \vee \psi, S^{i,j}, t) = \max(\rho(\phi, S^{i,j}, t), \rho(\psi, S^{i,j}, t)),$$
$$\rho(F_{[t_1,t_2]}\phi, S^{i,j}, t) = \max_{t' \in [t+t_1, t+t_2]} \rho(\phi, S^{i,j}, t'),$$
$$\rho(G_{[t_1,t_2]}\phi, S^{i,j}, t) = \min_{t' \in [t+t_1, t+t_2]} \rho(\phi, S^{i,j}, t'),$$

where μ is system status, c is a real-valued threshold to compare, and t is the time step to calculate rewards. The recursive definition ensures that complex formulas can be evaluated by combining the scores of their subformulas according to the semantics of the logical and operators.

B. Environment–Formula Mapping and System Trajectories

STL formulas can be used to define the desired behaviors and safety constraints of robotic systems across environments, ensuring that system requirements are accurately captured and verified. Let $\mathcal{E}$ denote the space of possible environments, each characterized by different configurations, operating conditions, and external factors. For each environment $e^i \in \mathcal{E}$, we associate an STL formula m^i that specifies both goals and safety requirements. If the system satisfies m^i, it achieves its objectives without entering unsafe states. To maintain clarity and tractability, we assume a one-to-one mapping between environments and STL formulas, where each environment is associated with a representative specification. This assumption is for exposition only; the framework readily extends to many-to-one or one-to-many relationships between specifications and environments.

Within each environment e^i, we define a state space $\mathcal{S}^i$ that includes all possible system states (e.g., position, velocity, acceleration). A trajectory is a finite sequence of states $S^{i,j} = [s_1^{i,j}, s_2^{i,j}, ..., s_\tau^{i,j}]$, where index i denotes environments and j indexes trajectories collected in environment e^i; τ denotes the time horizon. This discrete representation captures the system's evolution over time, enabling practical analysis and verification. To evaluate trajectories with respect to their STL specifications, we introduce a system trajectory reward based on the quantitative semantics of STL (see Section III-A), which provides a measure beyond binary satisfaction to assess, compare, and optimize system behaviors.

C. LLM Fine-tuning

Fine-tuning is a standard approach to adapt pre-trained LLMs to downstream tasks. Two common paradigms are supervised fine-tuning (SFT) and reinforcement learning fine-tuning (RLFT). SFT aligns the model outputs with task-specific data, ensuring that responses follow the desired format and semantics. RLFT further refines model behavior by incorporating preference or reward signals. Among RLFT methods, Direct Preference Optimization (DPO) has emerged as an efficient alternative to reward-model-based RL fine-tuning pipelines, as it directly optimizes relative preferences without requiring explicit reward modeling. In the following, we use SFT to enforce format/constraint adherence and apply DPO to align preferences to safety-compliant trajectories.

IV. System Design and Implementation

This section details the architecture and implementation of the proposed SafeNet framework. Fig. 1 provides an overview of our workflow.

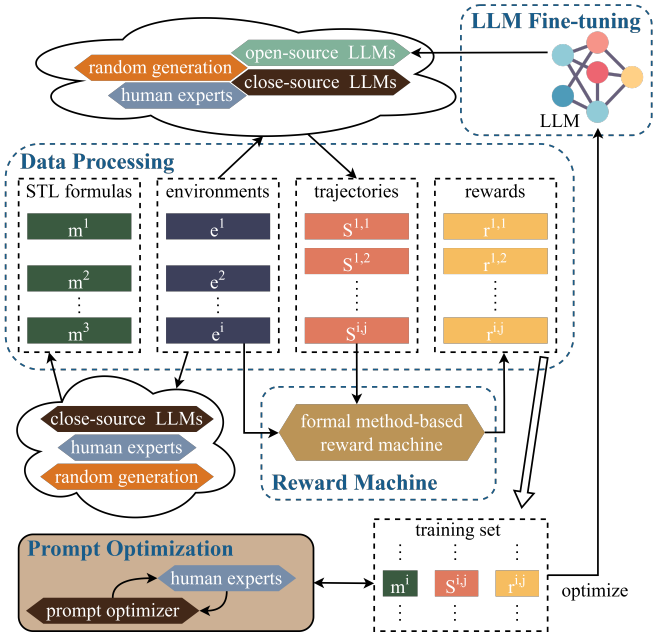


Fig. 1: An overview of our workflow.

A. Reward Machine

A core component of our approach is a formal method-based reward machine that automatically evaluates whether system trajectories satisfy specified formal logic constraints. Compared to manual evaluation, it is more consistent and efficient, reducing noise in training data and enabling the collection of sufficient high-quality trajectories for effective knowledge injection, which in turn improves learning reliability and robustness.

For combinatorial state spaces, we implement the reward machine described in Section III-A to check whether action sequences adhere to constraints. For numerical (continuous) state spaces, we develop a specialized reward machine based on Argus [24] and extend it with a time-constrained "'Finally" operator (absent in the original), which is essential in real-world settings where timing matters. For example, in path planning with continuous vector-valued locations, this operator can express temporal requirements such as reaching a goal within a time window or maintaining conditions for specified durations, expanding the constraint class enforceable in continuous spaces.

Both reward machines decompose complex constraints into sub-constraints by parsing along "And" operators. The total reward is the sum of sub-rewards; if any sub-constraint is violated, its sub-reward is set to the lower bound r_{lo} as a penalty, while trajectories that satisfy all constraints receive the upper bound reward r_{up}, avoiding unnecessary discrimination among different safe trajectories. As shown in Fig. 2, for a constraint with four sub-rules (ϕ_1–ϕ_4), trajectory $S^{i,1}$ violates ϕ_2, yielding $r_1 + r_{lo} + r_3 + r_4$ (substantially reduced due to the small r_{lo}), whereas $S^{i,2}$ satisfies all constraints and receives r_{up}.

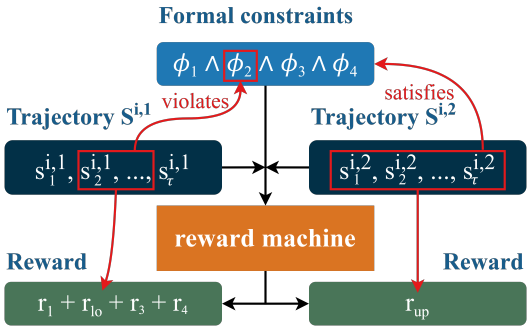


Fig. 2: Reward Machine

The reward machines serve two primary functions within our framework, each critical to the system's overall effectiveness. During SFT, they act as automated validators, meticulously verifying whether generated outputs satisfy the formal logic constraints. This verification process ensures that the model learns to generate outputs that consistently meet the specified requirements. During DPO, the reward machines enable quantitative comparison between trajectory pairs, providing a numerical basis for determining which trajectories better satisfy the given constraints.

B. Prompt Optimization

To generate effective prompt templates for safe planning, we utilize the Claude Prompt Generator, inspired by [25]. This advanced tool transforms our initial prompt drafts into well-structured task prompts optimized for safe planning. Our initial drafts included six input fields: 1) task specification, 2) formal constraints, 3) notations for the formal logic, 4) initial state, 5) specific details to attend, and 6) an example for reference. The resulting output is a clear, step-by-step prompt encapsulated in HTML tags. As an example, below is the prompt template for Block World scenario.

Prompt Template

You will be given a bag of non-duplicated blocks and a set of rules written in Signal Temporal Logic (STL). Your task is to arrange these blocks on a table in a sequence that satisfies all the given rules. Here are the details:
<bag> {{BAG}} </bag> <rules> {{RULES}} </rules> <notation> - **G**: Globally, - **F**: Eventually, - **U**: Until, - **X**: Next, - $\wedge$: And, - $\vee$: Or, - $\neg$: Not, - $\Rightarrow$: Implies - $\leq$, $\geq$, $<$, $>$, $=$, $\neq$ </notation> Your goal is to determine a valid sequence for placing the blocks on the table that satisfies all the given rules. There may be multiple valid solutions, but you only need to provide one. To solve this problem, follow these steps: 1. Analyze the rules to understand the relationships between the blocks. 2. Identify any blocks that must be placed before others. 3. Build up the sequence, ensuring each block is placed after the blocks it needs to be on top of. 4. One block can only be placed once. 5. Order starts from 1. 6. Double-check that your sequence satisfies all rules. Once you have determined a valid sequence, provide your answer in the following format: <sequence>[List the blocks in the order they should be placed on the table]</sequence>
An example you may refer to: <example> <bag> C, E, F, M, O, R </bag> <rules> $\mathbf{G}[1,2](E = 0) \wedge \mathbf{F}[3,5](E \neq 0) \wedge (R = 0)\ \mathbf{U}\ (F \neq 0) \wedge \mathbf{F}[1,2](M \neq 0)$ </rules> <sequence>[M, C, E, F, R, O]</sequence> </example>
Remember, there might be multiple valid solutions, but you only need to provide one that satisfies all the rules. Do not attempt to explain or justify your answer. Simply output the answer.

The designed prompt template encodes both the task description and the logical constraints in a structured way. In practice, placeholders are used for the initial state (i.e., the available blocks) and the formal STL rules. For each task scenario, the system retrieves the corresponding optimized prompt and fills in the environment-specific information, including the block set and the formal constraints, to construct the complete input. This input is then provided to the model to generate a valid sequence satisfying the given rules.

C. Data Processing

1) Environment Generation: For each task scenario, N distinct environments are randomly generated, with each associated with a synthetically generated STL formula. These formulas explicitly define target states that the system should reach and unsafe states that must be avoided. The environments are partitioned into three datasets: D_{SFT} (size N_{SFT}), D_{DPO} (size N_{DPO}), and D_{test} (size N_{test}), used for SFT, DPO, and evaluation.

2) Training Data Preparation: Fig. 3 shows the procedure to prepare training data. For environments in D_{SFT}, we employ a two-stage data preparation process: golden trajectory generation and data augmentation. In the first stage, human annotators create initial golden trajectories, where each trajectory is verified against STL requirements using the reward machine. The second stage is designed to address performance limitations and prevent severe distribution drift. The augmentation includes variation of STL formula representations, diversification of state descriptions (e.g., integers to floating-point), modification of output format

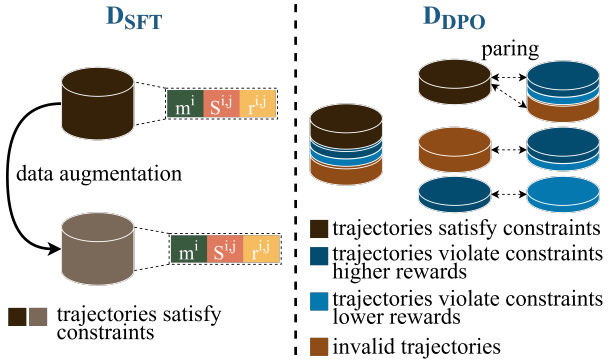


Fig. 3: Training Data Preparation.

requirements, and enhancement of prompt structures.

For D_{DPO}, we implement a trajectory generation and pairing process. First, we employ multiple sources to generate diverse trajectories, which are categorized into safe, unsafe, and invalid by referring to rewards calculated by the reward machine. Invalid trajectories are the ones that do not follow the required format or contain invalid decisions. To ensure better performance, we take the best advantages of all the trajectories and match them to pairs. Specifically, (1) safe trajectories are paired with unsafe and invalid ones, (2) invalid trajectories are paired with unsafe ones, and (3) unsafe trajectories are paired with each other based on their reward scores.

D. Fine-tuning for Safe Robotic Planning

We adopt a two-phase fine-tuning strategy to adapt pre-trained LLMs for safety-critical robotic planning: SFT followed by RL fine-tuning. Both phases employ Low-Rank Adaptation (LoRA), which enables efficient parameter updates while preserving the general knowledge of the base model. This dual-phase methodology has proven effective in specialized applications that demand both accuracy and computational efficiency. In our setting, SFT provides the model with a foundational understanding of the task-specific requirements, while the subsequent RL fine-tuning phase further refines performance through preference-based optimization. Compared to single-phase approaches, this design achieves superior results in domains requiring precise output formatting and strict adherence to safety constraints.

a) Supervised Fine-Tuning: In the SFT phase, each training instance pairs a prompt with a corresponding safe trajectory S that satisfies the STL specification m. Prompts are constructed by filling template placeholders with formal constraints, initial states, and trajectories, ensuring the model learns to generate structured, constraint-compliant outputs. The training objective is to maximize the likelihood of producing the ground-truth safe trajectory given its prompt:

$$\mathcal{L}_{\text{SFT}}(\pi_\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{SFT}}} \log \pi_\theta(y \mid x).$$

This objective calibrates the model's input–output mapping and enforces consistent formatting of trajectory responses.

b) Reinforcement Learning Fine-Tuning: To further align the model with safety requirements, we employ DPO. Unlike $\mathcal{D}_{SFT}$, which provides a single trajectory per prompt, $\mathcal{D}_{DPO}$ contains paired responses (y^+, y^-), where y^+ is a safe trajectory satisfying the STL specification, and y^- is an unsafe trajectory violating it. This binary preference structure naturally captures the critical distinction between valid and invalid trajectories in robotic systems. DPO is trained to maximize the relative likelihood of y^+ over y^-:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta) = -\mathbb{E}_{(x,y^+,y^-)\sim\mathcal{D}_{\text{DPO}}}\Big[\log\sigma\Big(\beta\log\frac{\pi_\theta(y^+|x)}{\pi_{ref}(y^+|x)} - \beta\log\frac{\pi_\theta(y^-|x)}{\pi_{ref}(y^-|x)}\Big)\Big],$$

where π_{ref} is the reference LLM, and β controls the deviation from it. To ensure stable adaptation while preserving the behaviors learned in SFT, we duplicate the final SFT weights as both the initialization and the reference model for DPO training, and select $\beta = 0.2$ as a balanced trade-off between preference alignment and stability.

V. Experiments

We present our experimental evaluation in this section to validate the effectiveness of our proposed approach. Our investigation focuses on two distinct task scenarios that are fundamental to robotics and artificial intelligence: the Block-World problem and the path planning problem.

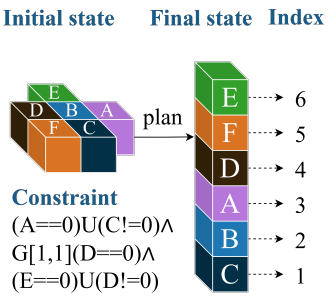


Fig. 4: Block-world.

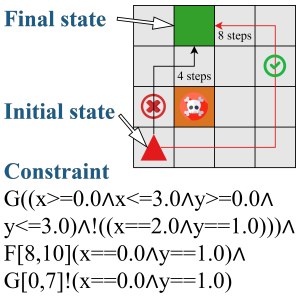


Fig. 5: Path Planning.

A. Experiment Settings

We configure the training with a batch size of 2 and 4 gradient accumulation steps for memory efficiency. Learning rates are set to 3×10^{-4} for SFT and 8×10^{-8} for DPO, with a warmup ratio of 0.05. We use Adamw 8-bit optimizer with 6 epochs for SFT and 1.2 for DPO. LoRA weights are applied to all projection layers (q_{proj}, k_{proj}, v_{proj}, o_{proj}, $gate_{proj}$, up_{proj}, $down_{proj}$) with rank dimension 16.

We evaluate 5 representative models from two families optimized for robotic applications. From the Mistral family, known for efficient attention mechanisms suitable for real-time robotic control, we use Mistral-7b-Instruct-v0.3-bnb-4bit, Mistral-Nemo-Instruct-2407-bnb-4bit, and Mistral-Small-Instruct-2409-bnb-4bit. From the Llama family, recognized for strong scaling properties in multi-modal robotic tasks, we include Llama-3.2-3B-Instruct-bnb-4bit and Llama-3.1-8B-Instruct-bnb-4bit[1]. All models use 4-bit quantization to meet the computational

[1]Parameter sizes: Mistral-7b-Instruct-v0.3-bnb-4bit: 3.87B, Mistral-Nemo-Instruct-2407-bnb-4bit: 6.97B, Mistral-Small-Instruct-2409-bnb-4bit: 11.7B, Llama-3.2-3B-Instruct-bnb-4bit: 1.85B, Llama-3.1-8B-Instruct-bnb-4bit: 4.65B

constraints and real-time requirements of robotic systems while maintaining performance for autonomous navigation and manipulation tasks.

B. Metrics

We evaluate system performance using the safe rate, which measures the fraction of tested environments/problem instances for which the system generates a valid plan that is safe and satisfies all constraints. A higher safe rate indicates greater reliability and robustness, and it is especially important in safety-critical settings where planning failures can cause damage or unsafe conditions. Formally, $R_{safe} = \frac{N_{safe}}{N}$, where N_{safe} is the number of successfully generated safe plans and N is the total number of tested environments/problem instances.

C. Block-world Scenario

The block-world problem is a classical robotic manipulation task and a convenient simulation proxy for humanoid organization and cooking behaviors. A robot must transform an initial block configuration into a desired final state via a sequence of valid manipulation moves, analogous to organizing utensils, arranging ingredients, or stacking containers during food preparation. In our setup, blocks start randomly scattered on a table. The final state is required to satisfy constraints specified by STL formulas, which capture both spatial relations and temporal requirements (similar to recipe steps or organization protocols). This formulation permits multiple valid solutions while enforcing strict satisfaction of the specified requirements, matching the flexibility needed in real robotic applications.

Figure 4 shows an example with six blocks (A–F) initially unordered on the table (like items on a kitchen counter). The robot must satisfy the STL constraint $m = (A == 0)U(C! = 0) \wedge G[1,1](D == 0) \wedge (E == 0)U(D! = 0)$, which encodes: 1) block A must remain at a higher vertical position than block C throughout manipulation, 2) block D is forbidden from occupying the bottom position, and 3) in the final state, block D must be below block E. One valid final stack is C, B, A, D, F, E (bottom to top), which satisfies all constraints. This mirrors the logical ordering requirements in cooking (e.g., layering ingredients or prioritizing tool placement).

1) Settings and Dataset Preparation: Environment parameters: the number of blocks is in $[3, 9]$, with labels drawn from $[A, Z] \cup [a, z]$. We generate sub-constraints between every pair of blocks to support both relative and absolute positioning requirements. To ensure feasibility, we reverse-engineer environments: we first randomly generate a final state, then construct constraints consistent with it.

During constraint generation, each block pairs with any block (including itself) with 60% probability. Self-pairs yield `Equal` sub-constraints (the block must be at its designated position). Non-self pairs yield `Unequal` constraints with 20% probability (the first block cannot be at the second block's position) and `Less/Greater` constraints with 80% probability (spatial ordering). `Less` and `Greater` are equally split between label–number (e.g., $A < 2$) and label–label (e.g., $A < C$) forms.

We create 512, 2047, and 512 unique environments for D_{SFT}, D_{DPO}, and D_{test}, respectively. For D_{SFT}, we upper-sample to 1536 environments by diversifying state descriptions (block-collection tags: {bag, sack, container, pack}; STL tags: {stl, rules, constraints}; trajectory tags: {sequence, answer, output}; concatenation formats between blocks: {", ", ",", "-", " ", ""}). For each D_{SFT} environment, we sample one safe STL constraint and up to 3 additional safe final states, producing $N_{SFT} = 5102$ data points. For D_{DPO}, we sample up to 3 safe, 4 unsafe, and 4 invalid final states, yielding 21354 data points; after forming positive/negative pairs, the final size is $N_{DPO} = 85439$.

2) Main Result: We use the Mistral-7B model as the pre-trained model in this scenario. We defined invalid trajectories through two conditions: 1) trajectories that fail to conform to the required format specifications, and 2) trajectories that either omit necessary block placements or contain redundant block placements.

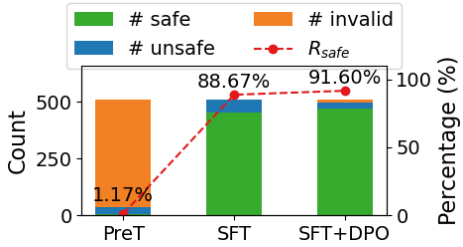


Fig. 6: Block-world Results.

The initial results from the pre-trained model revealed significant limitations in safe trajectory planning: out of 512 test cases, only 6 trajectories were safe, while 31 were unsafe, and a substantial 475 trajectories were invalid. This baseline performance highlighted the challenges in utilizing raw pre-trained models for safety-critical planning tasks. The SFT-ed model demonstrated remarkable improvement, generating 454 safe trajectories, 56 unsafe trajectories, and only 2 invalid trajectories. Further optimization through DPO yielded even more refined results, with 469 safe trajectories, 29 unsafe trajectories, and 14 invalid trajectories. These improvements represent a significant improvement in the model's ability to generate safe trajectories. The safe rate (R_{safe}) is improved from 1.2% in the pre-trained model to 91.6% after the complete fine-tuning process, which demonstrates the effectiveness of our approach for a reliable trajectory planner. The results are shown in Fig. 6.

D. Path Planning Scenario

This experiment evaluates autonomous robot navigation: generating a trajectory from a random start to a goal while avoiding obstacles and satisfying temporal (STL) constraints, measuring safety and efficiency in structured, time-constrained environments.

Figure 5 illustrates the setup: the robot (red triangle) starts at a random location in a bounded workspace and must reach the goal (green marker). The robot is governed by the STL

specification $m = G\Big((x \geq 0.0 \wedge x \leq 3.0 \wedge y \geq 0.0 \wedge y \leq 3.0) \wedge \neg(x = 2.0 \wedge y = 1.0)\Big) \wedge F_{[8,10]}(x = 0.0 \wedge y = 1.0) \wedge G_{[0,7]} \neg(x = 0.0 \wedge y = 1.0)$, which enforces: 1) remain within $x \in [0.0, 3.0]$, $y \in [0.0, 3.0]$; 2) avoid the obstacle at $(2.0, 1.0)$; 3) reach $(0.0, 1.0)$ within steps $[8, 10]$; and 4) do not reach the goal before step 8.

1) Settings and Dataset Preparation: Training maps have sizes in $[7, 15]$ with $[0, 6]$ obstacles; robot, goal, and obstacles are sampled uniformly at random with non-overlap constraints. For each environment, STL timing is sampled by first computing the minimum steps to reach the goal $t_{\min}$, then sampling $t^G_{\max} \sim [t_{\min} \cdot 1.2, 40]$, the goal deadline $\sim [t_{\min}, t^G_{\max}]$, and the minimum-time requirement $\sim [0, t_{\min}]$. Three test sets increase difficulty: small (size $[7, 9]$, obstacles $[0, 2]$), medium (size $[10, 12]$, obstacles $[3, 4]$), and large (size $[13, 15]$, obstacles $[5, 6]$).

We sample 2048, 8192, and 1536 unique environments for D_{SFT}, D_{DPO}, and D_{test}. Claude 3.5 Sonnet generates candidate paths for D_{SFT}; after filtering for constraint satisfaction, $N_{SFT} = 752$ remain. To diversify prompts, we vary agent names ({robot, drone, car, bot, vehicle, rover}), start descriptions ({starting point, kickoff point, onset, base, birthplace}), coordinate axes ({(x, y), (a, b), (i, j), (u, v)}), and bracket styles ({("[", "]"), ("{", "}"), ("(", ")")}). For D_{DPO}, multiple models (Claude 3.5 Sonnet, Claude 3.5 Haiku, pretrained Mistral 7B, Mistral Nemo) generate paths; all outputs are kept (valid and invalid) to provide both positive and negative training signal. Random path modifications produce 284,203 total paths, and after constructing preference pairs the final dataset size is $N_{DPO} = 753558$.

2) Main Results: We use Mistral-7B, Mistral-Nemo, Mistral-Small, Llama-3.2-3B, and Llama-3.1-8B as the pretrained models. Invalid trajectories are defined as 1) trajectories that fail to conform to the required format specifications, or 2) trajectories that contain illegal movements, e.g., moving in diagonal directions or moving two steps per time.

Experimental results for the small level are shown in Table I. The results demonstrate significant improvements in path planning safety across all tested models. Pre-trained models showed poor performance, with safe rates (R_{safe}) ranging from 0.78% to 7.23%. The SFT substantially improved performance across all models, with the Mistral-Small model achieving the highest safety rate of 72.07%. DPO further enhanced performance, with all models showing additional gains in safety rates, with the Mistral-Small model having the highest R_{safe} of 90.63%. Notably, while larger models generally performed well, the smaller models achieved competitive performance after fine-tuning, highlighting the effectiveness of the proposed approach.

We conducted experiments using the Mistral-7B foundation model across three levels. The results are shown in Fig. 8a. As the complexity increased, performance declined notably in the medium level with 334 safe paths (65.2%), 140 unsafe paths, and 38 wrong paths. The large level showed the most challenging results with only 286 safe paths (55.9%), accompanied by 188 unsafe paths and 38 wrong paths. These

TABLE I: Path planning small level results.

Model	Stage	#safe	#unsafe	#invalid	#total	R_{safe}
M 7B	pre-trained	6	267	239	512	1.17%
	SFT	345	88	79	512	67.38%
	SFT+DPO	439	37	36	512	85.74%
Nemo	pre-trained	11	388	113	512	2.15%
	SFT	350	101	61	512	68.36%
	SFT+DPO	449	30	33	512	87.70%
Small	pre-trained	37	376	99	512	7.23%
	SFT	369	87	56	512	72.07%
	SFT+DPO	464	22	26	512	90.63%
L 3B	pre-trained	4	352	156	512	0.78%
	SFT	225	207	80	512	43.95%
	SFT+DPO	243	194	75	512	47.46%
L 8B	pre-trained	7	402	103	512	1.37%
	SFT	364	83	65	512	71.09%
	SFT+DPO	451	29	32	512	88.09%

results demonstrate a clear correlation between environmental complexity and navigation performance, though it's worth noting that using a full-precision model instead of the 4-bit quantized version could potentially yield better results.

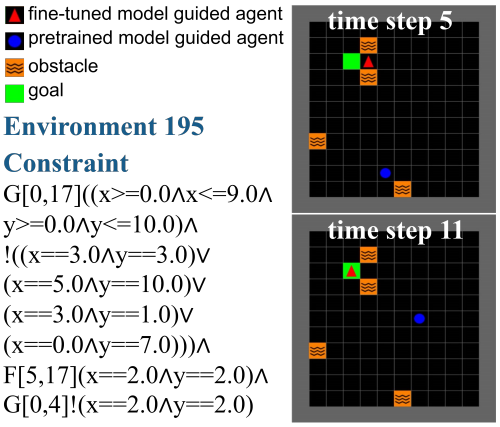


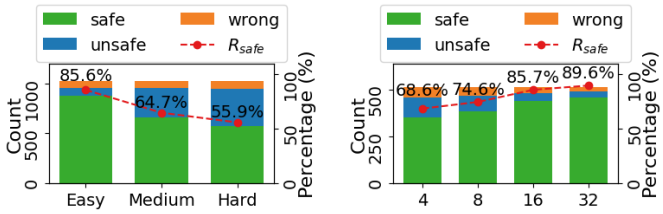
Fig. 7: Path Planning Experiment Visualization Example.

3) Example Result Visualization: A visual example is shown in Figure 7. Two robotic agents follow two trajectories that are generated by the pretrained model and fine-tuned model. The red circle represents the robot following the fine-tuned model's trajectory and the blue dot represents the robot following the pretrained model's trajectory. The starting position is close to the target location, but the temporal constraint specified that the robot should reach the target between time step 5 and 17. Therefore, the robot executing the fine-tuned model's trajectory stops at the last grid cell that is one step away from the target until time step 5 and finally reaches the target within the required time span. However, the robot following the pretrained trajectory cannot accomplish the task because the pretrained model cannot understand and follow the STL constraints for robotic path planning.

E. Sensitivity Analysis

We present a sensitivity analysis on the small level using the Mistral-7B model as the pre-trained foundation model. We investigate the impact of varying a critical hyperparameter, the rank value used in LoRA. The rank parameter

determines the dimension of the low-rank matrices used for fine-tuning, directly affecting both the model's capacity and computational efficiency. We conduct experiments with rank values of 4, 8, 16, and 32, and present our findings in Fig. 8b.



(a) Path planning across levels. (b) Sensitivity analysis.

Fig. 8: Path Planning Experimental Results.

Results show a clear correlation between higher rank values and better model performance, with consistent gains across all metrics. The safe rate (R_{safe}) increases from 45.61% at rank 4 to 58.40% at rank 32. However, gains diminish as rank grows: rank 4 to 8 yields a 6.25% increase, while rank 16 to 32 yields only 2.74%, suggesting an upper bound that rank increases alone cannot overcome. Rank increases can help substantially but are not a complete solution; further improvements likely require complementary approaches such as better data sampling/preparation and enhanced training methods.

VI. Conclusion

This paper explores the vital importance of safe planning in robotic systems and presents an innovative neural-symbolic network architecture that combines formal method-based reward machines with Large Language Model (LLM) fine-tuning. The proposed approach addresses the growing need for safety in increasingly integrated robotic systems by implementing a proactive safety strategy that anticipates and prevents potential hazards. The architecture uniquely merges the interpretability of formal methods with neural networks' learning capabilities, demonstrating promising results.

Acknowledgment

This work was supported in part by NSF CNS-2333980 and CNS 2442914.

References

[1] N. R. S. Muda, M. F. Fadilah, B. Faranadila, and D. Safanabila, "Design and construction of a remotely controlled multy-tasking chain-wheel combat robot," *Eduvest-Journal of Universal Studies*, vol. 4, no. 3, pp. 723–740, 2024.

[2] L. Zhang, Z. Wang, and F. Kong, "Optimal checkpointing strategy for real-time systems with both logical and timing correctness," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 4, pp. 1–21, 2023.

[3] Z. Yu, Z. Kaplan, Q. Yan, and N. Zhang, "Security and privacy in the emerging cyber-physical world: A survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1879–1919, 2021.

[4] Z. Wang, L. Zhang, Q. Qiu, and F. Kong, "Catch you if pay attention: Temporal sensor attack diagnosis using attention mechanisms for cyber-physical systems," in *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2023, pp. 64–77.

[5] A. H. El-Kady, S. Halim, M. M. El-Halwagi, and F. Khan, "Analysis of safety and security challenges and opportunities related to cyber-physical systems," *Process Safety and Environmental Protection*, vol. 173, pp. 384–413, 2023.

[6] J. M. Bradley and E. M. Atkins, "Optimization and control of cyber-physical vehicle systems," *Sensors*, vol. 15, no. 9, pp. 23 020–23 049, 2015.

[7] Y. Huang, L. Chen, P. Chen, R. R. Negenborn, and P. Van Gelder, "Ship collision avoidance methods: State-of-the-art," *Safety science*, vol. 121, pp. 451–473, 2020.

[8] Y. Cheng and W. Zhang, "Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels," *Neurocomputing*, vol. 272, pp. 63–73, 2018.

[9] Y. Wu, Z. Xiong, Y. Hu, S. S. Iyengar, N. Jiang, A. Bera, L. Tan, and S. Jagannathan, "Selp: Generating safe and efficient task plans for robot agents with large language models," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 2599–2605.

[10] W. Xu, M. Liu, O. Sokolsky, I. Lee, and F. Kong, "Llm-enabled cyber-physical systems: Survey, research opportunities, and challenges," in *2024 IEEE International Workshop on Foundation Models for Cyber-Physical Systems & Internet of Things (FMSys)*. IEEE, 2024, pp. 50–55.

[11] A. Cimatti, A. Micheli, and M. Roveri, "Validating domains and plans for temporal planning via encoding into infinite-state linear temporal logic," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[12] K. Cho, J. Suh, C. J. Tomlin, and S. Oh, "Cost-aware path planning under co-safe temporal logic specifications," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2308–2315, 2017.

[13] R. Zuo, Z. Wang, C. E. C. Bastidas, M. C. Gursoy, A. Solomon, and Q. Qiu, "A predictive control framework for uas trajectory planning considering 4g/5g communication link quality," in *2023 Integrated Communication, Navigation and Surveillance Conference (ICNS)*. IEEE, 2023, pp. 1–10.

[14] S. Alqahtani, I. Riley, S. Taylor, R. Gamble, and R. Mailler, "Mtl robustness for path planning with a," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018, pp. 247–255.

[15] X. Li, Z. Serlin, G. Yang, and C. Belta, "A formal methods approach to interpretable reinforcement learning for robotic planning," *Science Robotics*, vol. 4, no. 37, p. eaay6276, 2019.

[16] Y. Yang, J.-R. Gaglione, C. Neary, and U. Topcu, "Automaton-based representations of task knowledge from generative language models," *arXiv preprint arXiv:2212.01944*, 2022.

[17] T. van de Laar, Z. Zhang, S. Qi, S. Haesaert, and Z. Sun, "Vernacopter: Disambiguated natural-language-driven robot via formal specifications," *arXiv preprint arXiv:2409.09536*, 2024.

[18] I. Obi, V. L. Venkatesh, W. Wang, R. Wang, D. Suh, T. I. Amosa, W. Jo, and B.-C. Min, "Safeplan: Leveraging formal logic and chain-of-thought reasoning for enhanced safety in llm-based robotic task planning," *arXiv preprint arXiv:2503.06892*, 2025.

[19] J. Pan, G. Chou, and D. Berenson, "Data-efficient learning of natural language to linear temporal logic translators for robot task specification," *arXiv preprint arXiv:2303.08006*, 2023.

[20] Y. Yang, N. P. Bhatt, T. Ingebrand, W. Ward, S. Carr, Z. Wang, and U. Topcu, "Fine-tuning language models using formal methods feedback: A use case in autonomous systems," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 339–350, 2024.

[21] Z. Wan, C.-K. Liu, H. Yang, R. Raj, C. Li, H. You, Y. Fu, C. Wan, S. Li, Y. Kim *et al.*, "Towards efficient neuro-symbolic ai: From workload characterization to hardware architecture," *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, 2024.

[22] J. Fan and F. Kong, "Survey of signal temporal logic specification mining: techniques, applications, and future directions," in *Disruptive Technologies in Information Sciences IX*, vol. 13480. SPIE, 2025, pp. 34–41.

[23] I. Haghighi, N. Mehdipour, E. Bartocci, and C. Belta, "Control from signal temporal logic specifications with smooth cumulative quantitative semantics," in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 4361–4366.

[24] B. Anand, "Argus," 2024. [Online]. Available: https://anandb.dev/argus/dev/index.html

[25] Z. Wang and C. Ormerod, "Generative language models with retrieval augmented generation for automated short answer scoring," *arXiv preprint arXiv:2408.03811*, 2024.