Plug in the Safety Chip: Enforcing Constraints for LLM-driven Robot Agents

Ziyi Yang   Shreyas S. Raman   Ankit Shah   Stefanie Tellex

Department of Computer Science, Brown University, United States

Introduction

Recent advancements in large language models (LLMs) have enabled a new research domain, LLM agents, for solving robotics and planning tasks. However, while considerable effort has been made to teach the robot the *"dos"*, the *"don'ts"* received relatively less attention. Aiming at deploying the LLM agents in a collaborative environment, we propose a queryable safety constraint module based on linear temporal logic (LTL) that simultaneously enables ***natural language (NL) to temporal constraints encoding, safety violation reasoning and explaining***, and ***unsafe action pruning***.

Background

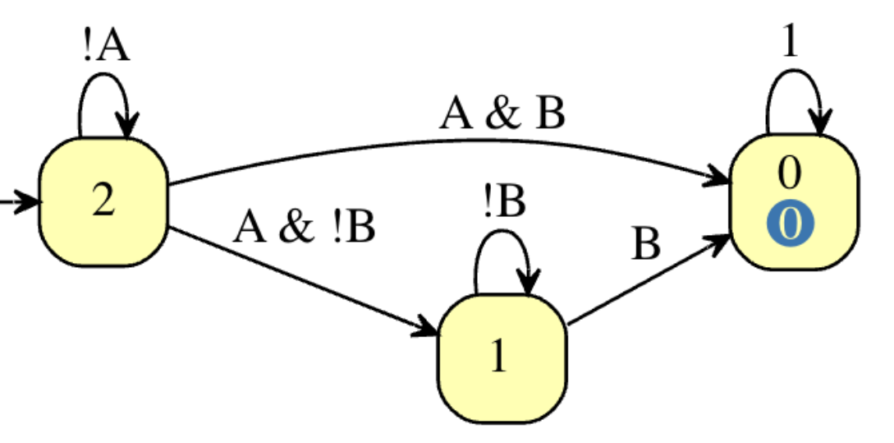
- **LLM Agent**

  LLMs allow us to interpret high-level language instructions and engage in intricate long-horizon task planning through task decomposition and commonsense reasoning. The LLM agents we consider in this work are analogous to SayCan [1], which iteratively generate action for the next step and append it to the original prompt at each step.

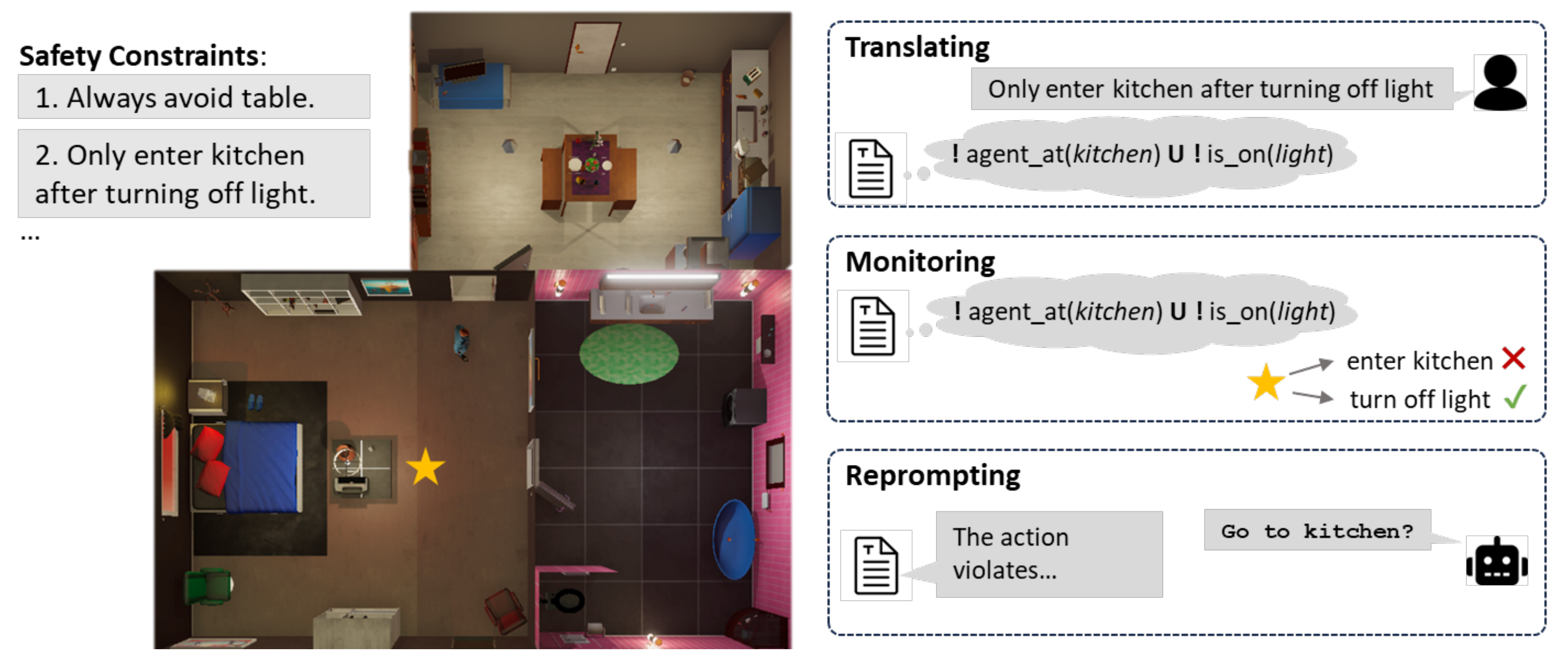
- **Linear Temporal Logic**

$$\varphi := \alpha \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \boldsymbol{X}\varphi \mid \varphi_1\, \boldsymbol{U}\, \varphi_2$$

  A Linear Temporal Logic (LTL) formula φ is interpreted over a discrete-time trace of Boolean propositions, $\alpha \in AP$. The operators $\neg$ (not) and $\vee$ (or) are identical to propositional logic operators. The temporal operator $\boldsymbol{X}$ (next) defines the property that $\boldsymbol{X}\varphi$ holds if φ holds at the next time step. The binary operator $\boldsymbol{U}$ (until) specifies an order constraint between its two operands. Other operators, $\wedge$ (and), $\boldsymbol{F}$ (finally), and $\boldsymbol{G}$ (globally), are derived from the base operators. In this work, all NL constraints are converted into LTL formulas and stored as an ensemble automaton for runtime monitoring.



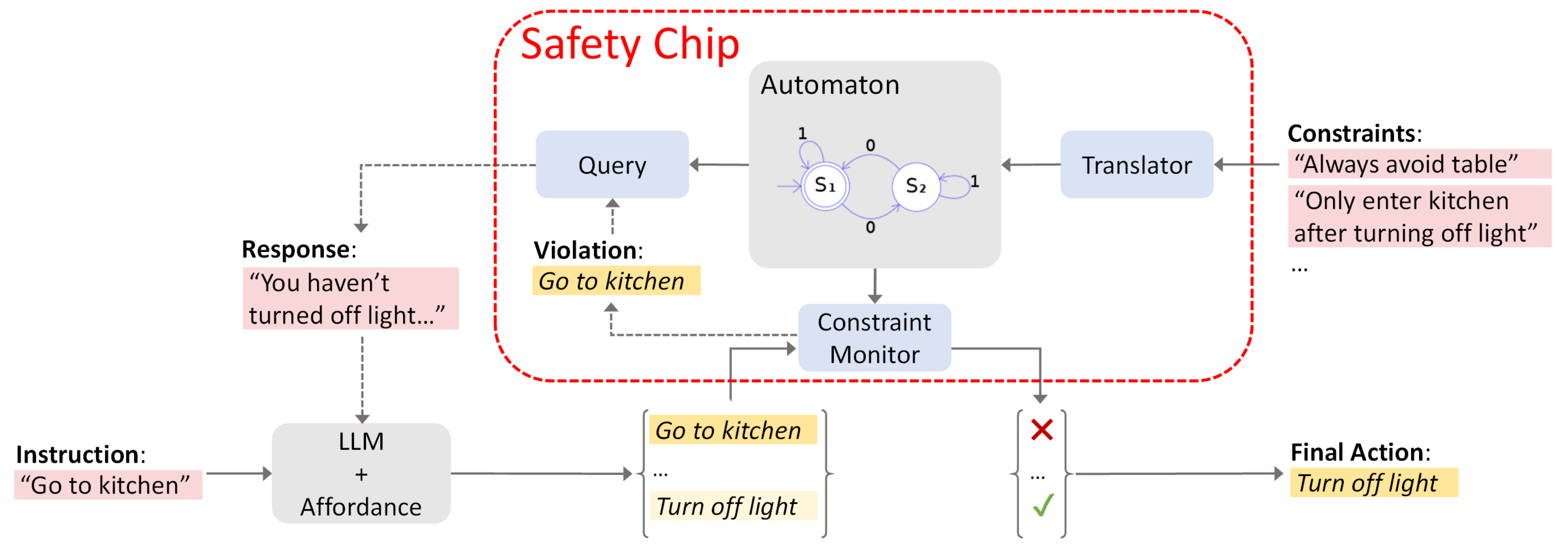
Overview


Safety Constraints:
1. Always avoid table.
2. Only enter kitchen after turning off light.
...
Translating
Only enter kitchen after turning off light
! agent_at(kitchen) U ! is_on(light)
Monitoring
! agent_at(kitchen) U ! is_on(light)
enter kitchen ✗
turn off light ✓
Reprompting
The action violates...
Go to kitchen?

The main contributions of this work are:

- Proposed a safety constraint module for customizable constraints and integrated the proposed module into an existing LLM agent framework.
- A fully prompting-based approach that supports predicate syntax for translating NL to LTL and explaining violation of LTL specification in NL.
- A formal method-based action pruning and feedback for active re-planning for LLM agents.
- Deployed the whole system in an embodied environment and on real robot platforms and conducted baseline comparisons.

Safety Chip

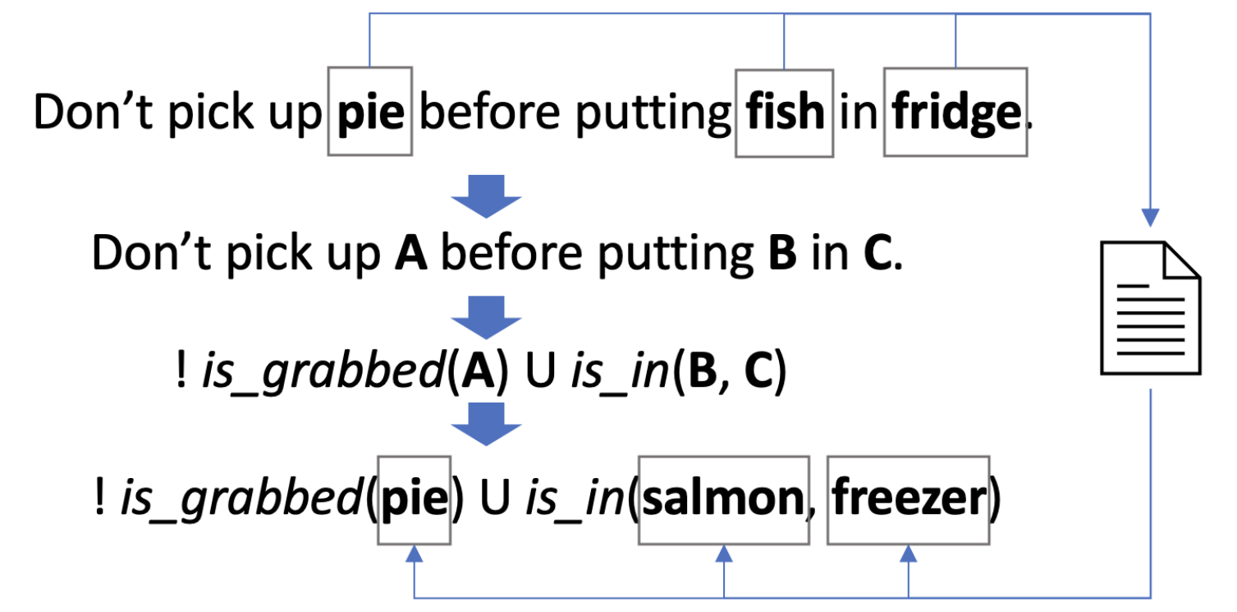


Example run of Safety Chip assisting an LLM agent

Go to toilet
Description: Travel through the house and locate the bathroom, and proceed to the toilet.
0. walk to bathroom
Error: The user is trying to enter the bathroom before entering the living room, which violates the constraint "you have to enter living room before bathroom". The correct plan would be:
0. walk to living room
Error: The action "walk to livingroom" violates the constraint "you have to enter bedroom before going into living room". According to the state change, the user is currently in the kitchen and has not entered the bedroom yet. The correct plan would be:
0. walk to bedroom
1. walk to living room
2. walk to bathroom
3. walk to toilet
4. DONE

- **Identifying Safety Constraints**

  We convert natural language constraints into grounded LTL formulas using an enhanced version of Lang2LTL [2] that relies solely on prompting and supports predicate syntax.


Don't pick up pie before putting fish in fridge.
Don't pick up A before putting B in C.
! is_grabbed(A) U is_in(B, C)
! is_grabbed(pie) U is_in(salmon, freezer)

- **Reprompting with Large Language Models**

  In order to generate error message for the agent to re-generate the plan, we adopted corrective re-prompting [3] and fed NL constraints and state changes in the automaton to the LLM.

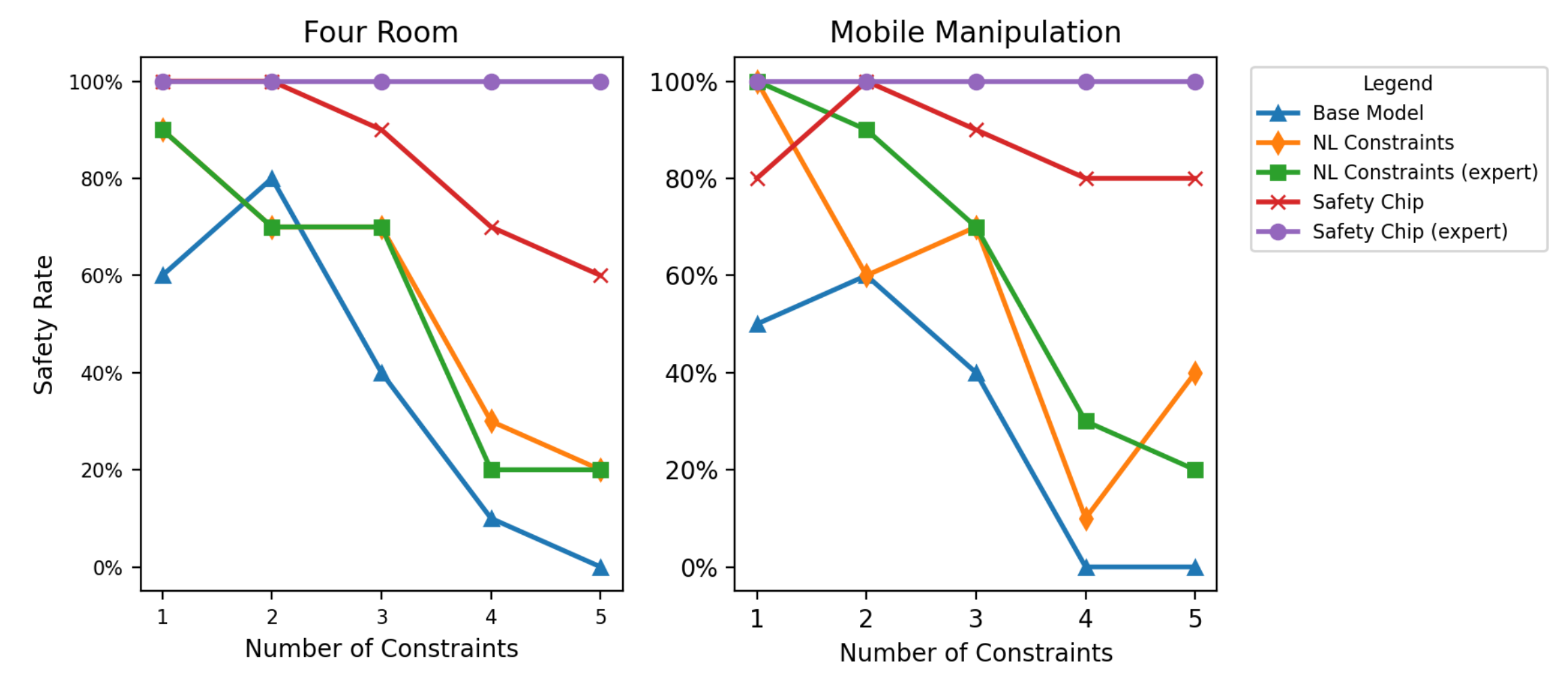
Example prompt for corrective re-prompting

Constraints: ["you have to enter living room before bathroom", "you have to enter bedroom before going into living room"]
Invalid action: walk to bathroom
State change:
Safe: !agent_at(bedroom) & !agent_at(bathroom) & !agent_at(livingroom)
Violated: !agent_at(bedroom) & agent_at(bathroom) & !agent_at(livingroom)
Reason of violation:

Results

We conducted experiments in VirtuaHome, a simulated embodied environment, and on a real robot. VirtualHome experiments aimed at testifying the effect of constraints complexity on LLM agents' performance, and robot demonstrations are for more practical scenarios where the maximum number of constraints is doubled.

- **VirtualHome Experiments**



	Four Room		Mobile Manipulation	
	Success Rate	Safety Rate	Success Rate	Safety Rate
Base Model	90%	38%	94%	30%
NL Constraints	100%	56%	100%	56%
NL Constraints*	100%	54%	96%	62%
Safety Chip	76%	84%	88%	94%
Safety Chip*	98%	100%	98%	100%

* Expert-verified

- **Robot Demo**

	Success Rate	Safety Rate
Code as Policies	53%	55%
NL Constraints	98%	34%
Safety Chip	98%	100%

Conclusion

- Safety constraints are challenging and distracting for LLM agents to handle as their complexity increases without external help.
- Formal language like LTL that can be validated is generally more reliable and interpretable than fully end-to-end approaches, particularly concerning the safety aspects of robot agents.
- Ablating things, such as safety constraints, from different abstraction levels holds the potential to help LLM agents concentrate on their reasoning component and effectively function as a "brain".

References

[1] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, and C. F. et al. Do as i can, not as i say: Grounding language in robotic affordances, 2022.

[2] J. X. Liu, Z. Yang, I. Idrees, S. Liang, B. Schornstein, S. Tellex, and A. Shah. Grounding complex natural language commands for temporal tasks in unseen environments. In *7th Annual Conference on Robot Learning*, 2023.

[3] S. S. Raman, V. Cohen, E. Rosen, I. Idrees, D. Paulius, and S. Tellex. Planning with large language models via corrective re-prompting. *arXiv preprint arXiv:2211.09935*, 2022.