

# FROM NATURAL LANGUAGE TO SAT WITH LLM AGENTS

AUTHORS

Guillaume Atencia, [gatencia@andrew.cmu.edu](mailto:gatencia@andrew.cmu.edu)  
Ruben Martins



## INTRODUCTION

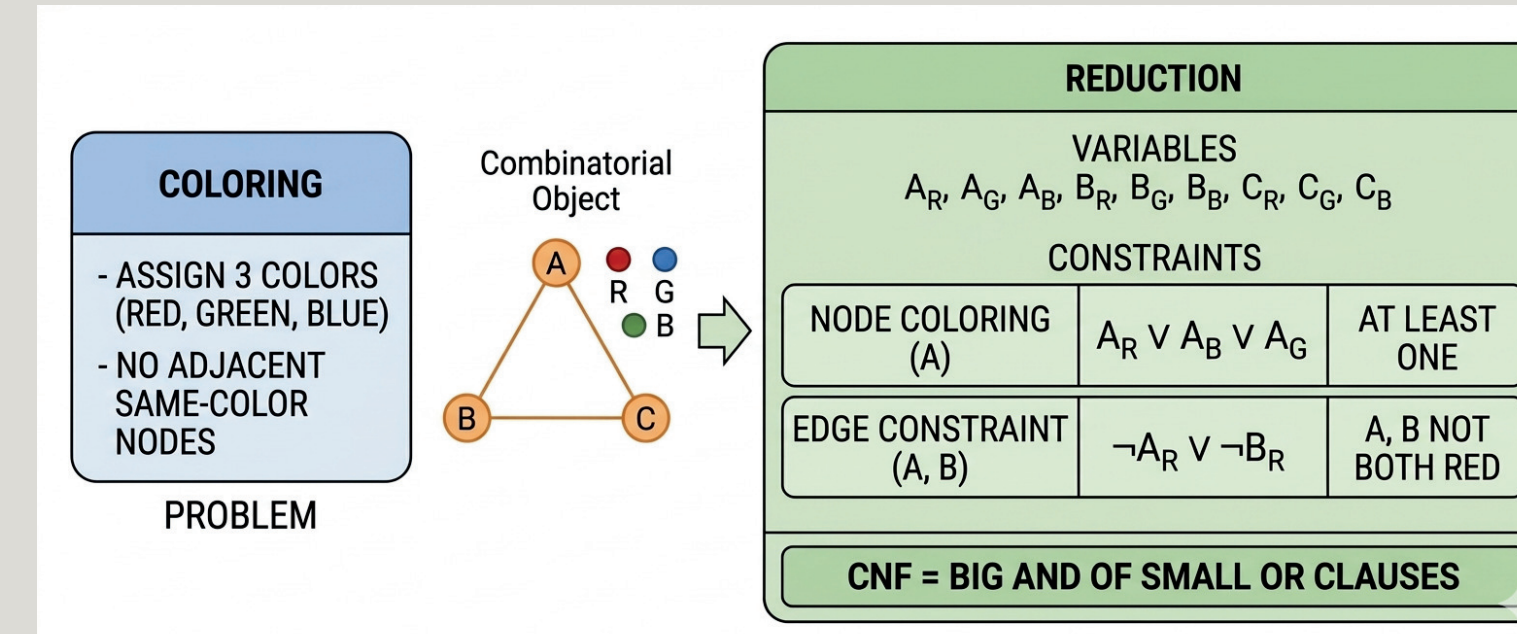
**Why this matters.** Many combinatorial problems are easy to solve once written as SAT.

**The bottleneck.** The hard part is building the encoding: choosing variables, writing constraints, and avoiding modeling errors.

**Our goal.** We study whether LLM agents can translate natural language into solver-ready SAT encodings.

### SIDE NOTE: WHAT IS A SAT ENCODING?

the process of **translating** real-world **problems** into a **logical formula** made of **boolean variables** (true/false) and **constraints** (clauses) in **Conjunctive Normal Form (CNF)**.

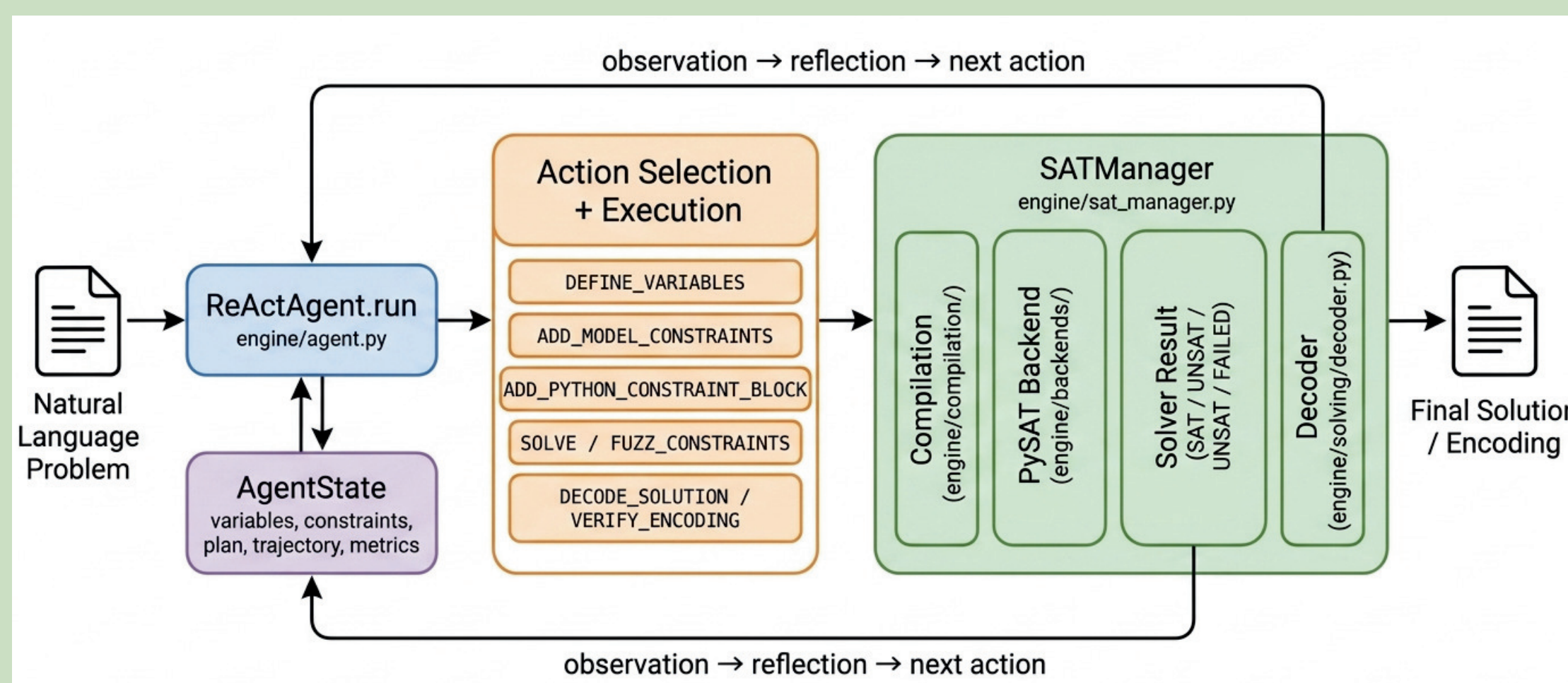


## METHODS: TWO DIRECTIONS

### 1. ARCHITECTURE

#### A. REACTE BASELINE

**ReAct.** The baseline agent builds the encoding step by step through SATManager: define variables, add constraints, solve, and decode.

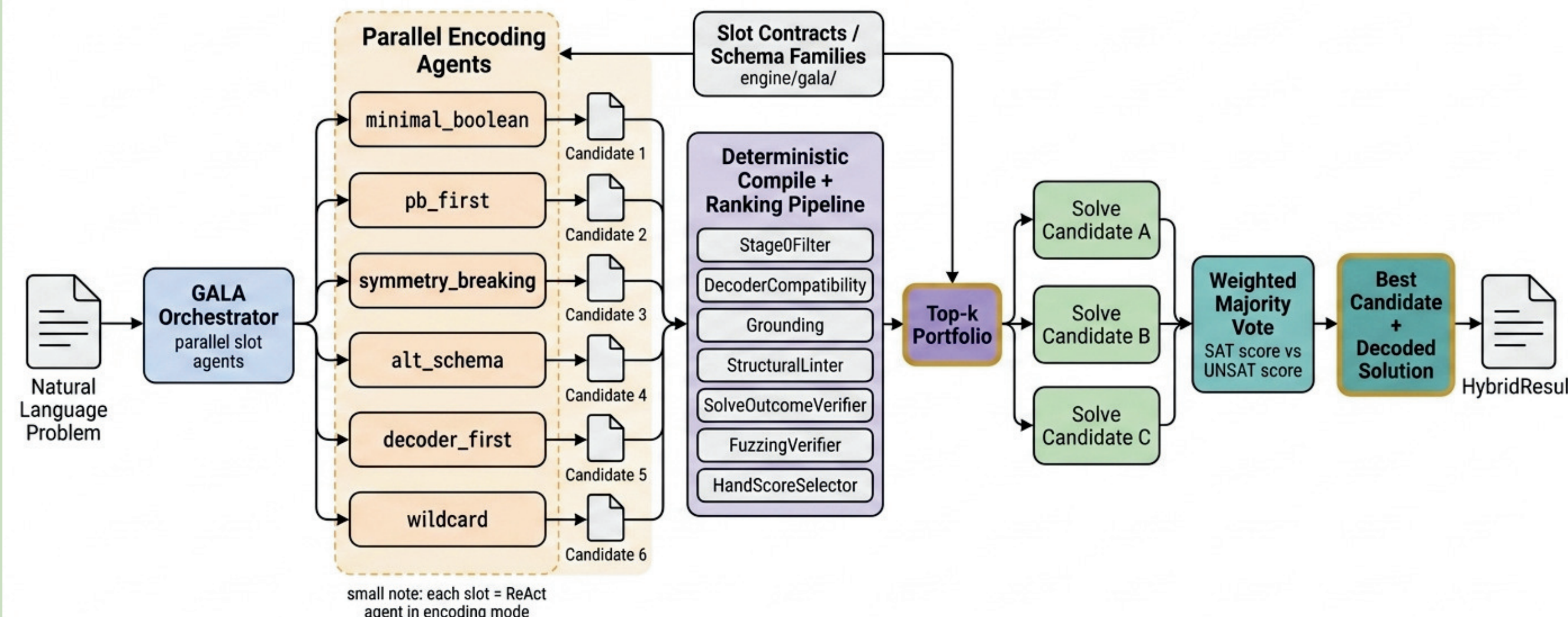


**Key idea.** The model can reflect and revise instead of generating one full encoding in a single pass.

#### B. GALA-INSPIRED ARCHITECTURE

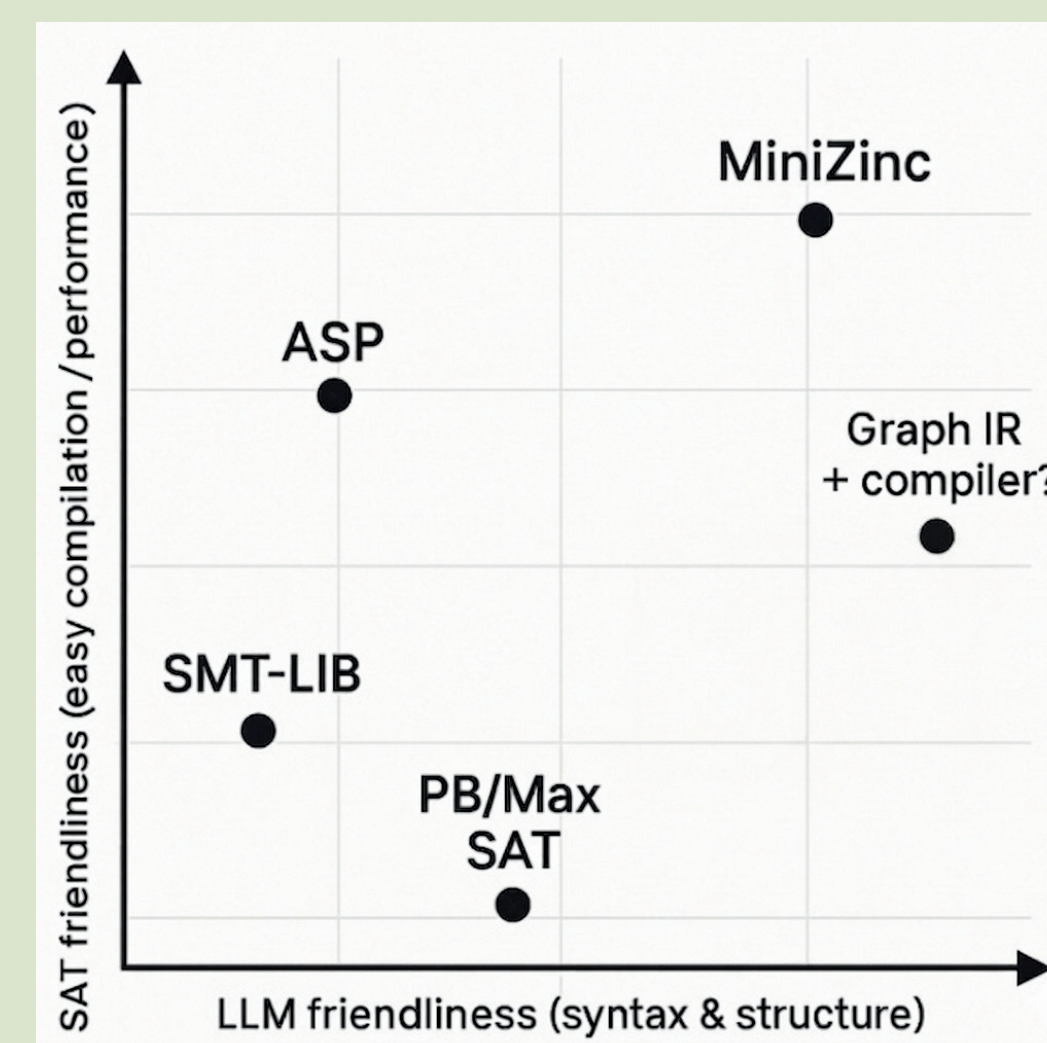
**GALA-inspired.** The second architecture generates multiple candidate encodings in parallel and ranks them before solving.

**Key idea.** This reduces the risk of getting stuck in one early but incorrect formulation.



### 2. INTERMEDIATE REPRESENTATIONS

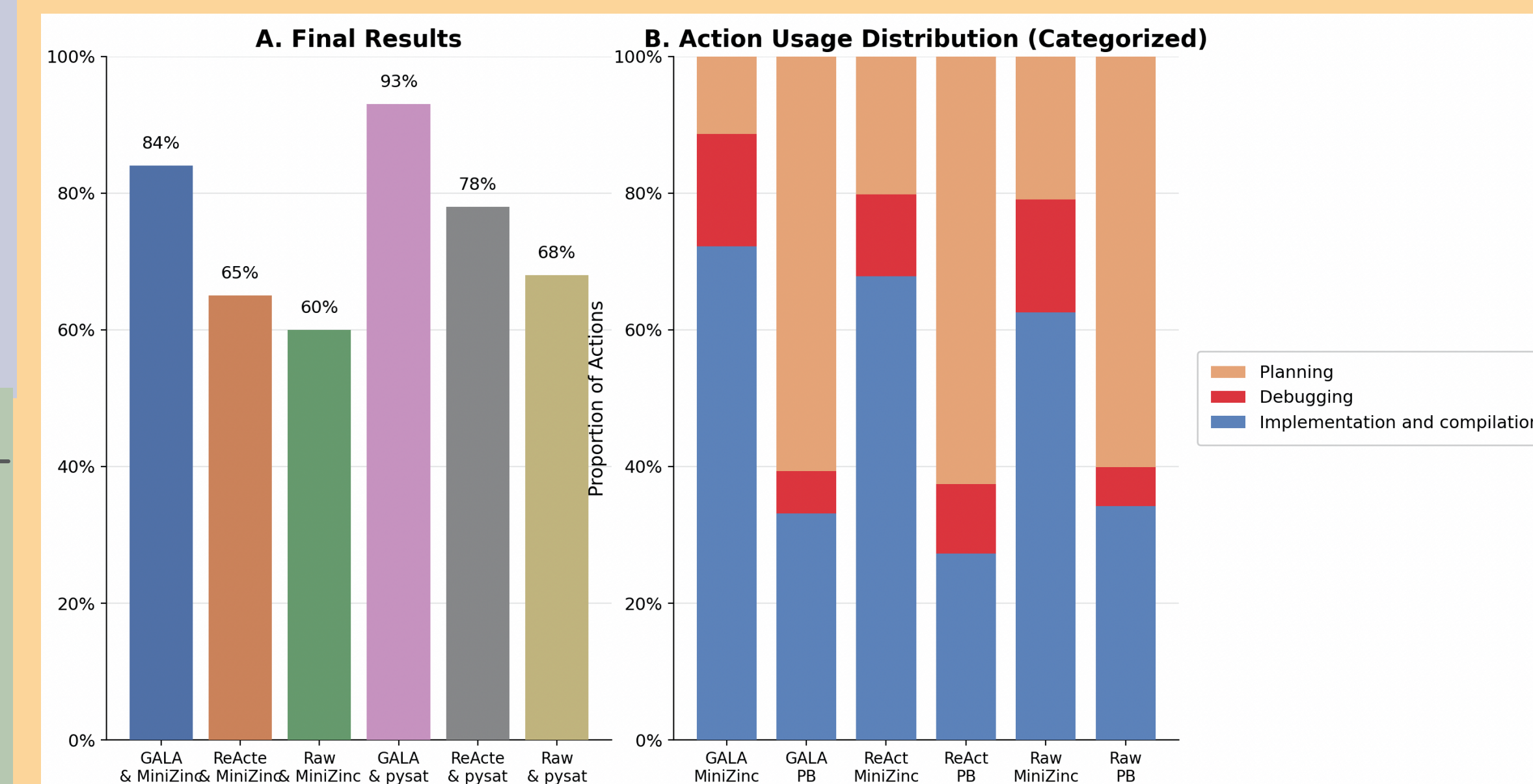
The **IR** choice matters because it changes how the problem is **expressed to the solver**. Different IRs shift the balance between **planning, debugging, and solver efficiency**.



**Focus.** Our main comparison is between MiniZinc and PySAT / pseudo-Boolean representations.

## RESULTS

**EVALUATION** We benchmarked the system on natural-language combinatorial problems such as **Slitherink, graph coloring, Poly Pentomino, and multi-robot path planning**. We compared architectures and IRs using **success rate, runtime, and step count**.



### IR EFFICIENCY

PySAT / pseudo-Boolean performs best across both architectures, reaching about **93% with GALA** and **78% with ReAct**, above the corresponding MiniZinc results. The **action distributions** also show a cleaner fit with the agent pipeline: **MiniZinc is more code-writing heavy**, while PySAT / PB **relies more on structured constraint construction**.

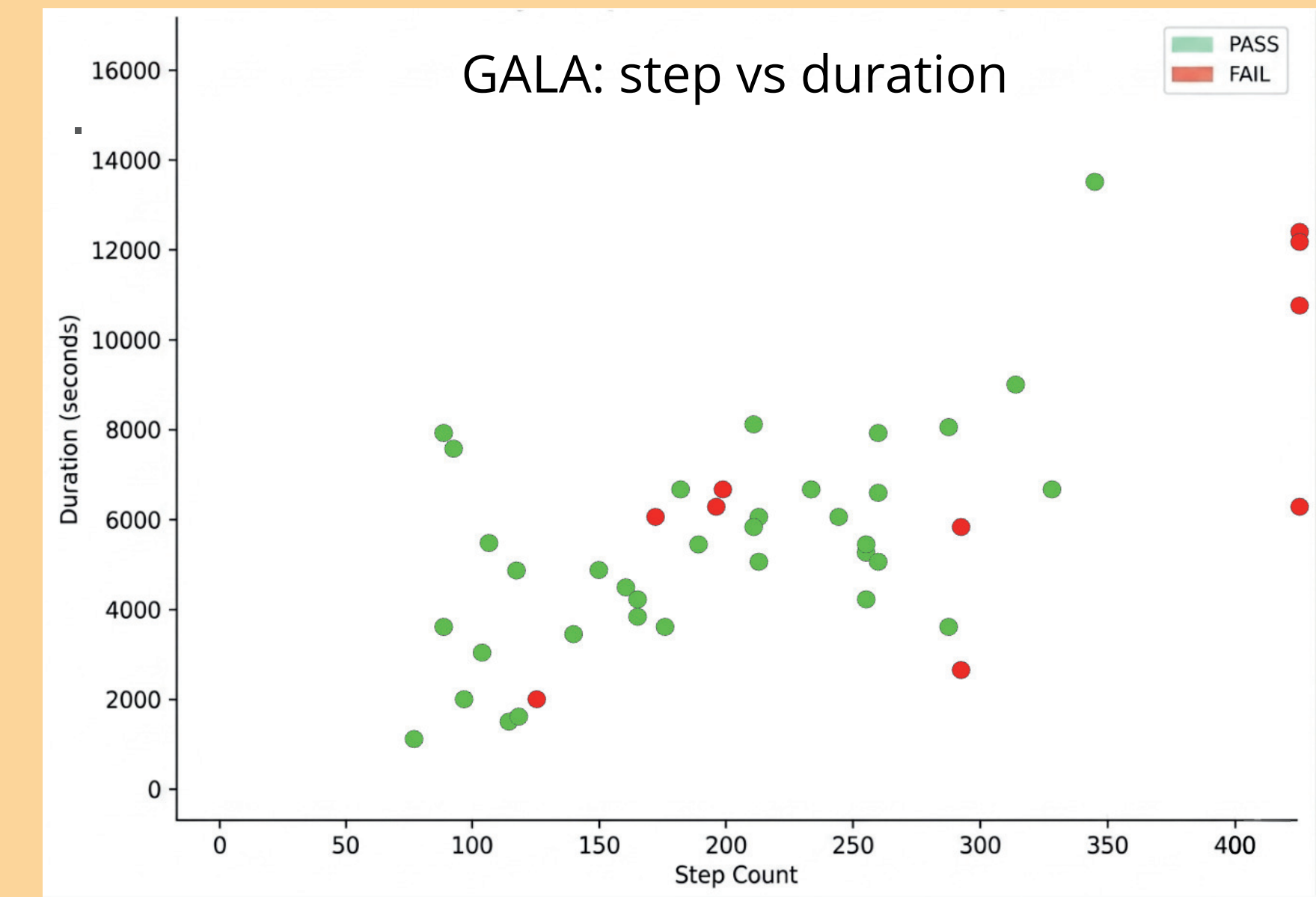
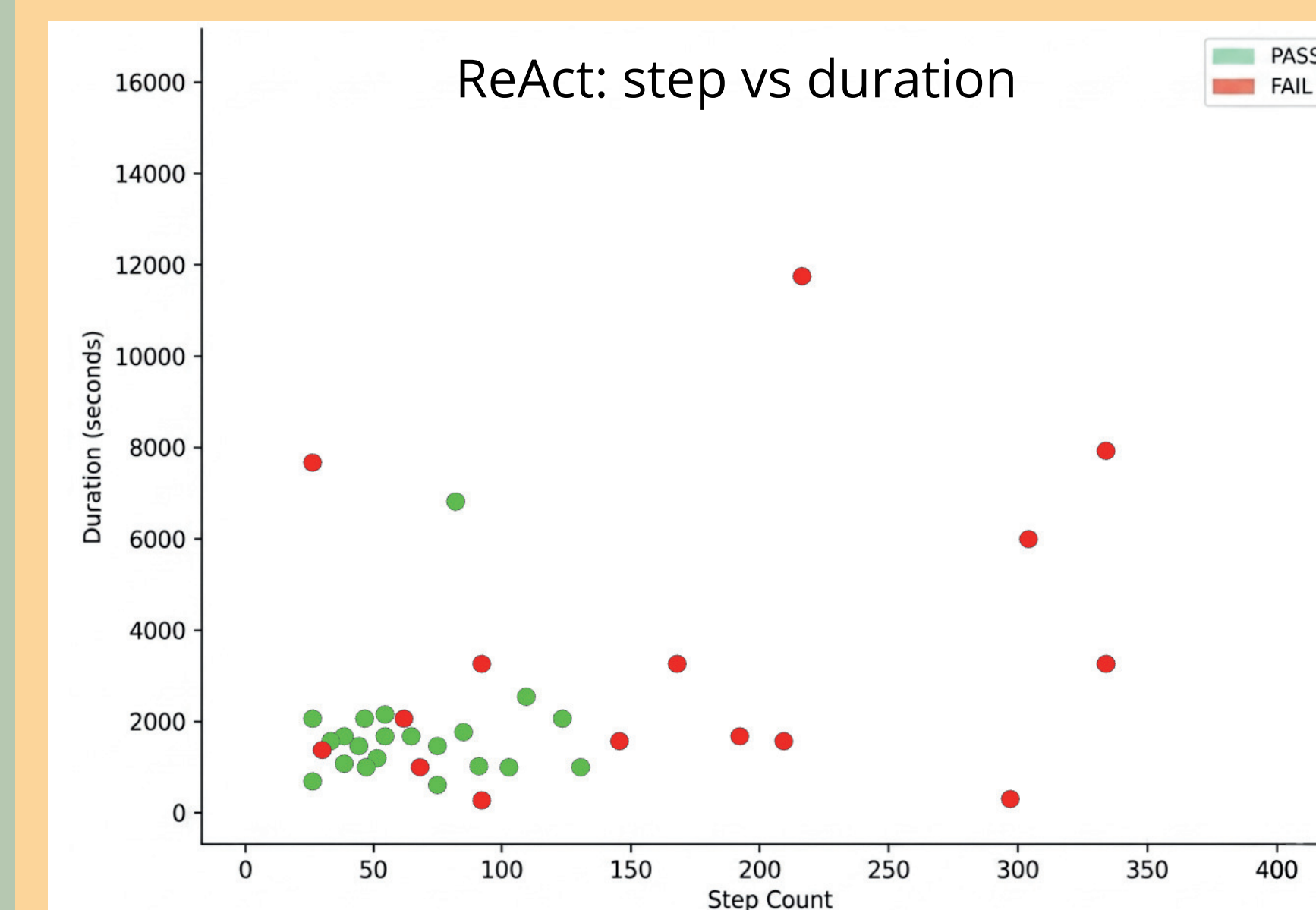
### ARCHITECTURE

**Trade-off.**

- GALA is less efficient** because it evaluates **3 candidates per encoding**.
- It trades **more compute** for **better robustness**.

**Architecture result.**

- GALA** shows **successful runs** across a wider range of **step counts** and **durations**.
- This suggests it explores a **larger state space** than the ReAct baseline.



## DISCUSSION

So far, PySAT / pseudo-Boolean looks like the most practical representation for this system. It fits the agent pipeline better than MiniZinc and has given the most reliable end-to-end behavior. We also find that the GALA-inspired architecture helps by exploring multiple candidate encodings instead of committing too early to one path.

## FUTURE WORK

A key next step is to keep developing this neuro-symbolic approach as language models improve. We also want to build a reusable memory of correct SAT reductions and expand it with more high-quality examples, so the agent can reuse past successful constructions instead of starting from scratch each time.