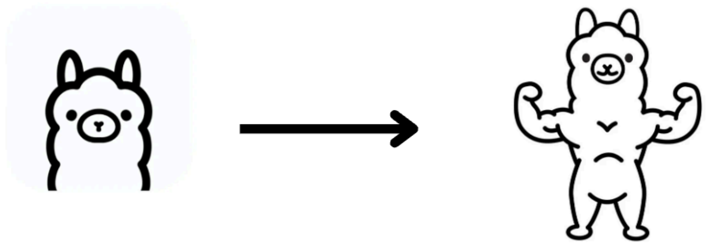


Finetune Founding Engineer Project

**subject to change as new research is released*

Finetune Founding Engineer Project

Project Context

Finetune is the easiest way to fine-tune AI agents. Currently, agents are unreliable; if they are, they're highly hand-crafted, meaning workflows are very static, and all steps of the workflow are pre-defined. Finetune is building a future where organizations and developers can reliably spin up a workflow on the fly given an objective and a set of tools/resources.

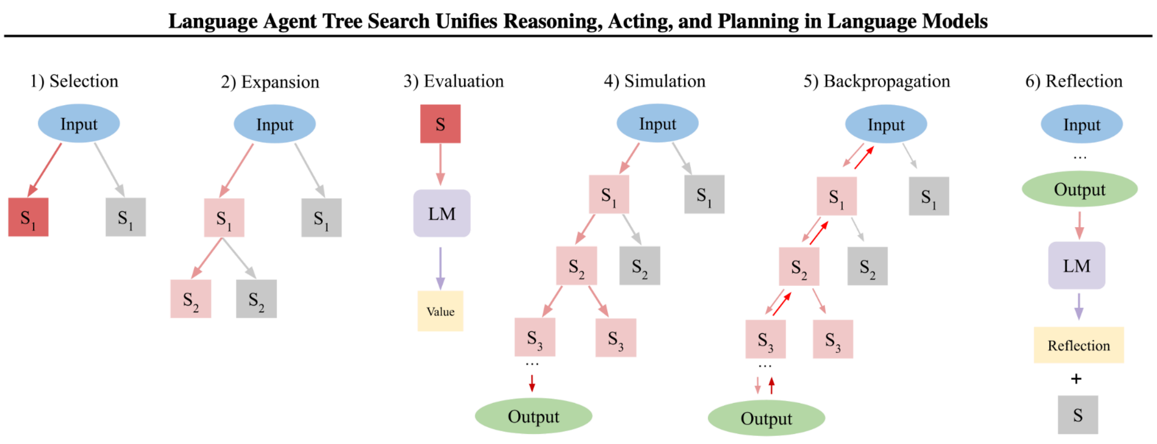
We tackle this by simulating executions in a sandbox environment before agent deployment to understand which workflows/paths are most likely to succeed, given an objective meaning that when agent builders deploy their agent into production, they can reference the successful workflow, leading to fewer hallucinations which translates to lower processing times and potentially lower costs as well.

One of the core components of this approach is a highly efficient search algorithm that evaluates all possible actions and can infer which action path to take. Monte Carlo Tree Search (MCTS) is a heuristic search algorithm that has proved successful in many decision-making environments. MCTS builds a decision tree where every node is a state, and the edge is an action. MCTS runs for k episodes; each episode starts from the root (i.e., initial state) and iteratively expands the tree to explore potential successful routes. More information and a working example of how MCTS works can be found here.

Researchers at the University of Indiana Urbana-Champaign recently released Language Agent Tree Search (LATS). LATS adapts MCTS to language agents to enable agents to learn from experience by constructing the best trajectory from sampled actions, allowing more flexible and adaptive problem-solving. Their main contributions can be summarized as follows:

- Proposing a novel value function that guides the search process and incorporates successful heuristics (a self-generated LM score & a self-consistency score),
- Incorporating external feedback and self-reflection in cases where terminal state nodes are partially successful or fail

The LATS paper can be found here, the code is here and the overall search algorithm architecture is as follows:



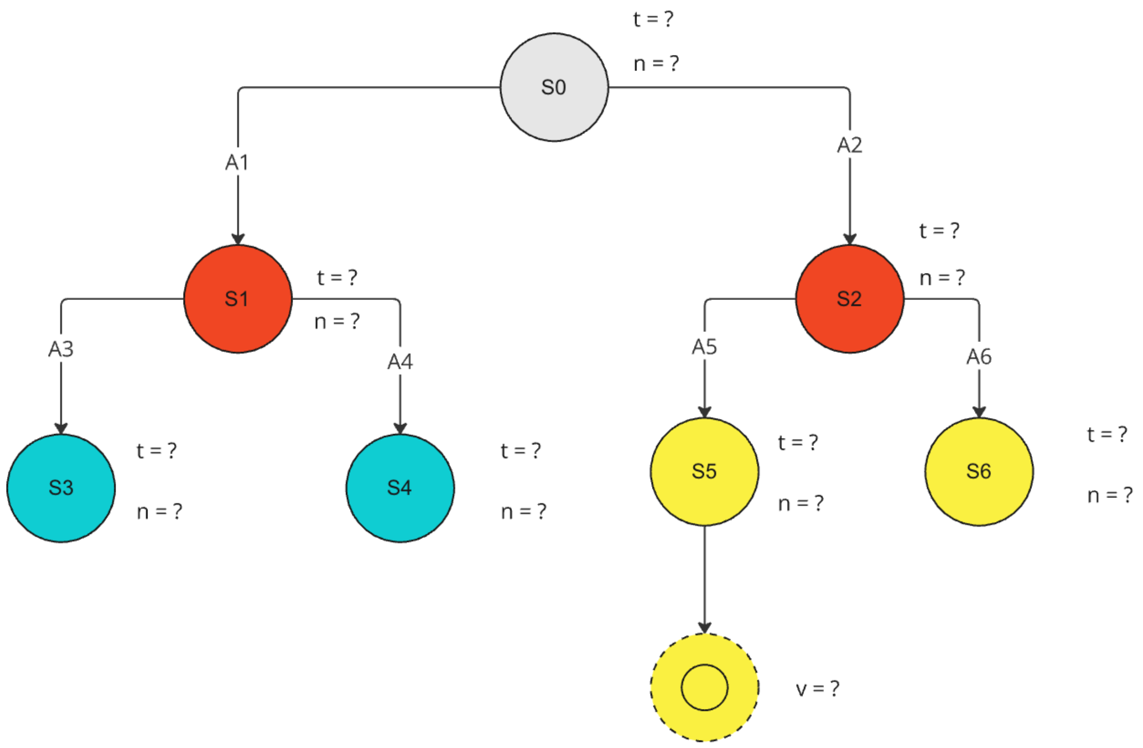
LATS achieved SOTA results across several benchmarks: 71% exact match accuracy on HotPotQA, 92.7% Pass@1 rate on HumanEval with GPT-4, and a 75.9 average score on WebShop, demonstrating its effectiveness across various domains.

LATS is currently limited to single-agent systems that are relatively simple environments. Future directions include scaling LATS to more complex environments or multi-agent frameworks.

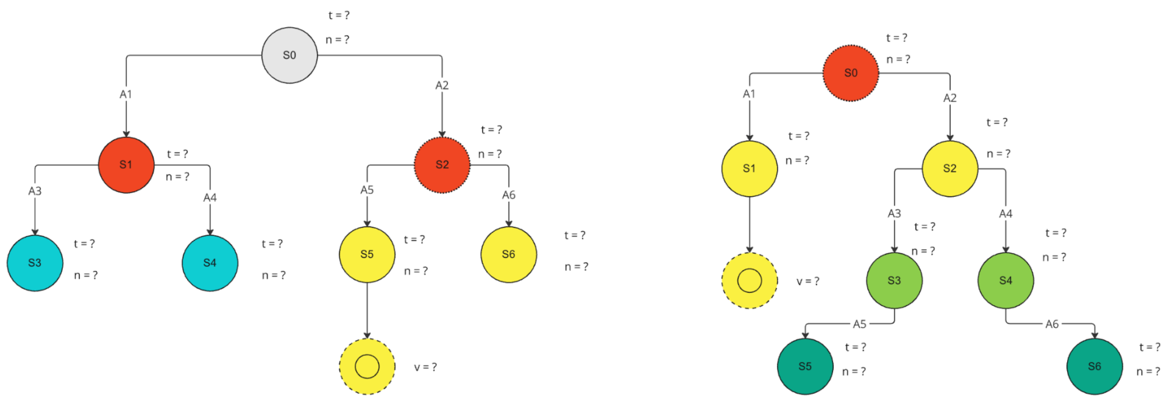
Project Task

The main project task is to extend LATS to multi-agent systems, meaning that instead of deciding which action path and tool to use, the edges are agents themselves, which have a nested LATS configuration within each node. This setup may introduce two current states - one for the agent-agent state connection and one for the agent-tool connection. This expansion can be summarized by the following designs:

Single agent LATS



Multi agent LATS



Note: A fork at state S2 exists, and there are now two current states. The agent tree on the right (single-agent LATS) is fully explored before moving down the tree on the left (multi-agent LATS).

Good luck 🚢🫡