CS 839 · FALL 2026 · LECTURE 5

Planning and Task Decomposition

Fred Sala
University of Wisconsin–Madison

Logistics and Announcements

Office Hours: Tuesday at 3:30–5:00 pm.

Team Signup: Use this link if you haven't signed up yet.

- More in-class activities coming up!

Slack Signup: Issues should be fixed. Use this link if you haven't signed up yet: https://join.slack.com/t/cs839scientif-tav7625/shared_invite/zt-49srj0go7-fMbEgaj8RiRNr97PEENVYw

First Assignment : Coming on Thursday!

From Last Time: Different Agentic Surfaces

Examples of common agentic surfaces:

API / function: offers typed arguments and structured returns. Good for bounded operations.

Code / CLI: Flexible and composable. However, broad access means security and reproducibility issues.

GUI: Works with existing software; state may be partial or visually ambiguous.

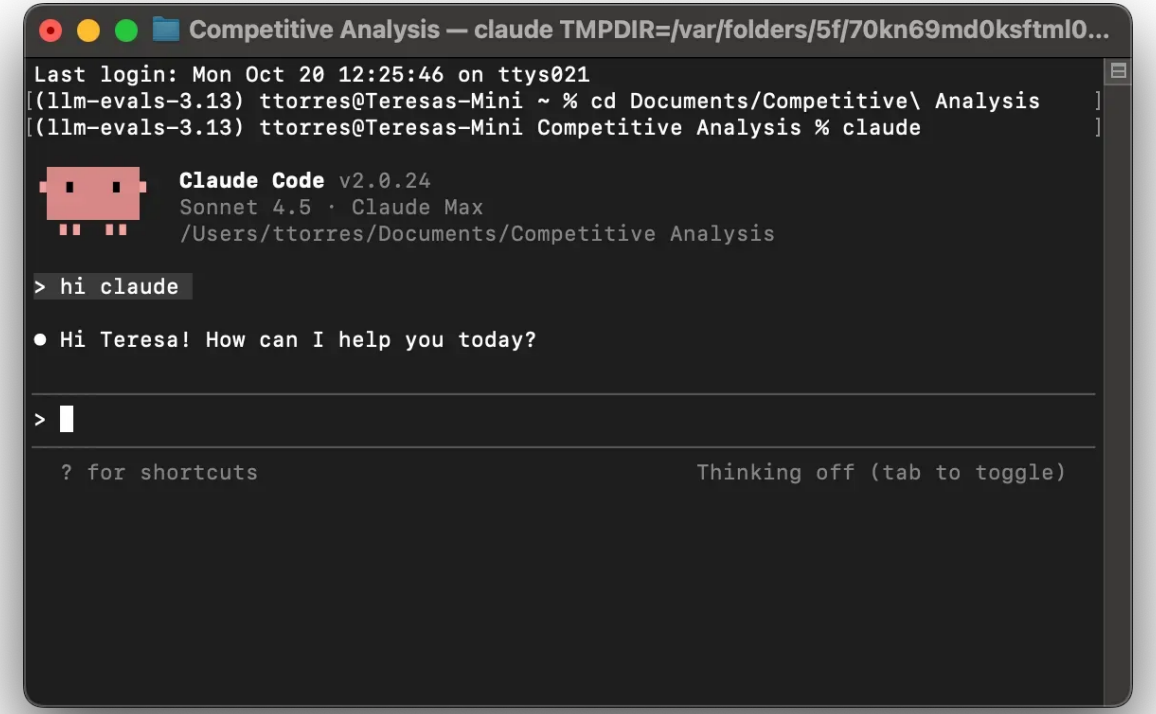
Instrument: Actions can consume samples, money, time—or create physical hazards.

From Last Time: The agent–computer interface

The interface makes a big difference!

It's not (just) the model: The available commands, observations, and feedback from the interface shape the agent's behavior.

Example: An interface designed for an agent can make work easier than a generic shell. Especially true for coding agents.



From Last Time: Protocols can help!

MCP idea: A host can connect an agent to servers exposing tools, resources, and prompts.

Why? A common interface reduces one-off integration work.

- Convenient in lots of ways!

Warning: A tool description is untrusted input; the host still decides access and approval.

From Last Time: Errors are actionable observations

We can do a lot using feedback from errors! Some examples:

Validation error: Fix an argument; do not retry unchanged.

Rate limit: Wait or switch to a cheaper path within policy.

Timeout: Check job status before assuming failure.

Domain warning: Inspect scientific assumptions before reporting a result.

From Last Time: How do we deal with timeouts?

Let's see an example:

Scenario:	You submit a simulation job, then the client times out.
Unsafe/bad move:	Submit the same job again immediately.
Safer move:	Query by job ID; retry only if no job exists.
Why:	Retries can duplicate cost, data writes, or physical actions.

Today's motivation: from one call to a task

So far, we have discussed our agents doing individual things

- But we want them to do entire tasks. What's the difference?

One call. Example: Find candidates below a temperature threshold.

A task. Example: Select, simulate, validate, compare, and explain a candidate.

Planning problem: Which actions, in what order?

Today: planning and task decomposition

1. Idea: Break an objective into executable subgoals.
2. ReAct: Act, observe, and revise.
3. Search, reflection, and verification when a first plan fails.
4. Keep long-horizon work within cost and safety limits.

Outline

Plan construction
Goals, dependencies, and actions

Search and verification
Replanning, alternatives, reflection, and checks

Long-horizon science
Budgets, checkpoints, and recent research systems

What is a plan?

What shall be generated in a plan? Some things to keep in mind:

Goal: A desired outcome and success criterion.

State: What is known now (including uncertainty and artifacts).

Actions: Available operations the agent can take.

Policy: A rule for what to do after the next observations.

Budget: Time or tokens or money or # of experiments.

There are multiple ways to plan!

Not every task needs same level of flexibility. For example, we might have:

- **Fixed workflow**: steps and failure cases are known in advance.
- **Decomposition**: identify subgoals and dependencies first, but then the plan is fixed.
- **Online planning**: decide again after each observation.

How much flexibility do we actually need?

Before We Plan

Let’s go back to our running example from the exercise: materials stable below 323 K.

We’ll need to answer basic questions. E.g.,

- What does “stable” mean? Which data / metric?
- What must we return: IDs, measurements, source, evidence?
- What are the limits on tools, cost, and physical actions?

Example: Candidate-Screening Plan

One possible plan for our example:

- Query the data and save the raw response.
- Normalize units; flag missing or ambiguous fields.
- Simulate a shortlist; retain job IDs and logs.
- Check the result independently before finishing.

Where could this plan fail?

Where Does Planning Fit In our Frameworks? ReAct

Reasoning and action interleaved (Yao et al. ’23).

- Reason: what is the next uncertainty?
- Act: choose and call a tool.
- Observe: use the result (or error) to choose again.

Main difference from ordinary CoT: observations come from the environment.

Example of **online planning**, as we saw earlier!

REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao*[,1], Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]

[1]Department of Computer Science, Princeton University
[2]Google Research, Brain team
[1]{shunyuy,karthikn}@princeton.edu
[2]{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com

ABSTRACT

While large language models (LLMs) have demonstrated impressive performance across tasks in language understanding and interactive decision making, their abilities for reasoning (e.g. chain-of-thought prompting) and acting (e.g. action plan generation) have primarily been studied as separate topics. In this paper, we explore the use of LLMs to generate both reasoning traces and task-specific actions in an interleaved manner, allowing for greater synergy between the two: reasoning traces help the model induce, track, and update action plans as well as handle exceptions, while actions allow it to interface with and gather additional information from external sources such as knowledge bases or environments. We apply our approach, named ReAct, to a diverse set of language and decision making tasks and demonstrate its effectiveness over state-of-the-art baselines in addition to improved human interpretability and trustworthiness. Concretely, on question answering (HotpotQA) and fact verification (Fever), ReAct overcomes prevalent issues of hallucination and error propagation in chain-of-thought reasoning by interacting with a simple Wikipedia API, and generating human-like task-solving trajectories that are more interpretable than baselines without reasoning traces. Furthermore, on two interactive decision making benchmarks (ALFWorld and WebShop), ReAct outperforms imitation and reinforcement learning methods by an absolute success rate of 34% and 10% respectively, while being prompted with only one or two in-context examples.

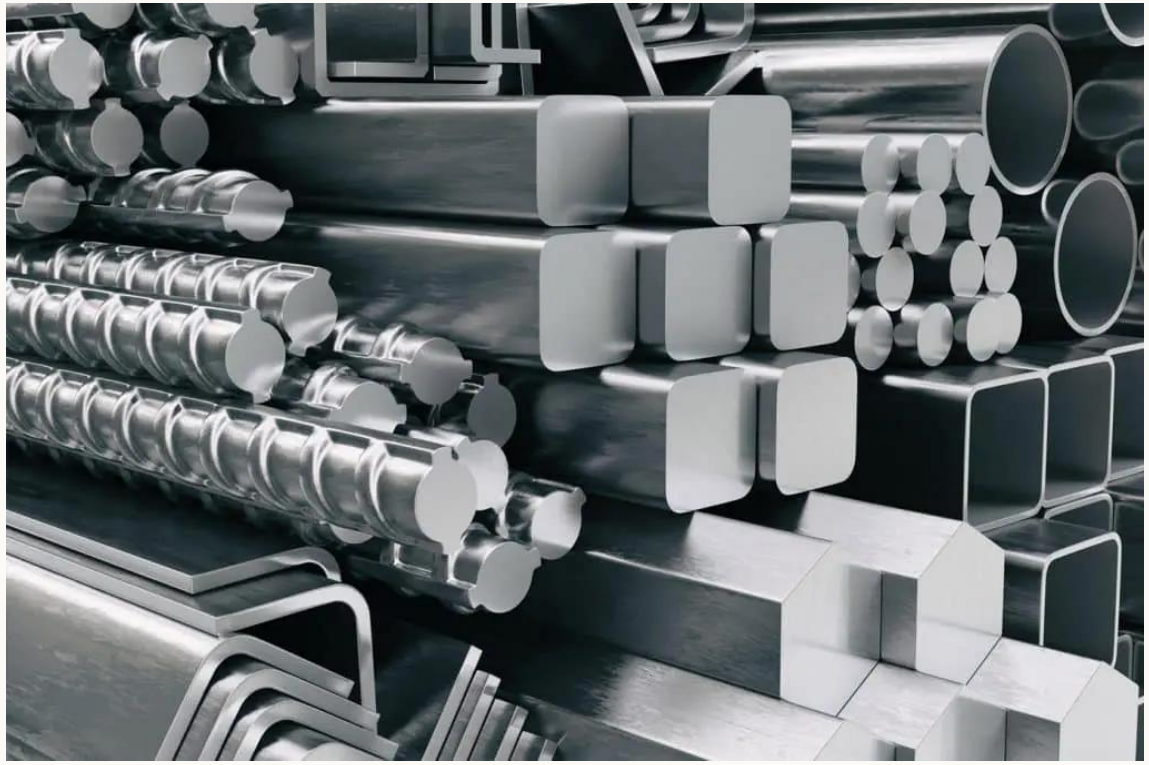
1 INTRODUCTION

A unique feature of human intelligence is the ability to seamlessly combine task-oriented actions with verbal reasoning (or inner speech, Alderson-Day & Fernyhough, 2015), which has been theorized to play an important role in human cognition for enabling self-regulation or strategization (Vygotsky, 1987; Luria, 1965; Fernyhough, 2010) and maintaining a working memory (Baddeley, 1992). Consider the example of cooking up a dish in the kitchen. Between any two specific actions, we may

ReAct: Running Example

Recall our “below 323 K” query.

- The first query gives a threshold in °C.
- 323 K is 49.85 °C; the first call used the wrong value.
- Correct the query and inspect the returned candidate.
- Then run an independent check before reporting it.



What Makes a Useful Subtask?

Just giving a label: “Search, analyze, write” does not give us much to work with. Useful to be more detailed:

- Specify the input and output of each step.
- Add a check before moving to the next step.
- Keep the raw artifact, not only the agent’s summary.

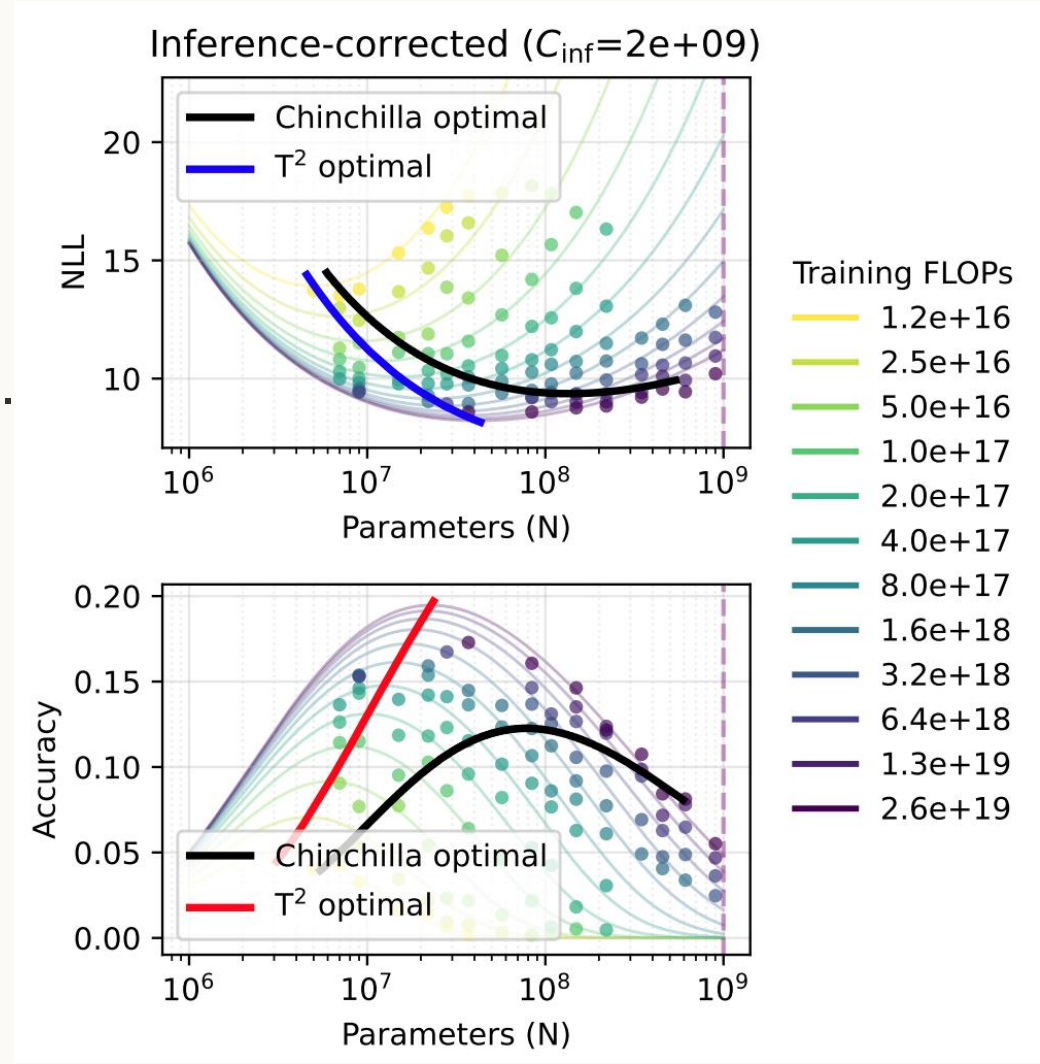
What would count as completing the “analyze” step?

Example: Reproducing a Figure From A Paper

Suppose we want to reproduce one result in a paper.

Detailed plan:

- Locate the data, code, metric, and exact target figure.
- Fix the environment and random seed.
- Run the baseline and save the raw output.
- Compare with the paper, including its tolerance.
- Explain deviations; keep failed runs in the record.



Dependencies Between Steps

Some steps can run in parallel.

- Get the data while inspecting code and dependencies.
- Wait for both before launching the experiment.
- Verification needs the raw result and run configuration.

Task list by itself does say what can run when. Example:

- Data access: 10 min. Environment setup: 20 min.
- Those two can overlap; the 30-min run waits for both.
- Verification takes another 10 min after the run.

How Big Should a Step Be?

One of our major challenges. Planning granularity matters a lot!

- “Do the experiment” is too big: it doesn’t give us any useful recovery point.
- Every keystroke is too much detail.
- Best: a step with an observable result and reasonable/bounded cost.

Example: run the baseline and save outputs plus logs.

BREAK & QUESTIONS

When should an agent revise its plan?

Next: search, reflection, and verification

Outline

Plan construction
Goals, dependencies, and actions

Search and verification
Replanning, alternatives, reflection, and checks

Long-horizon science
Budgets, checkpoints, and recent research systems

When Do We Replan?

Our original plan is based on what we knew before. Now, we might realize:

- A tool fails or the expected artifact is missing.
- New data change a key assumption.
- The next step costs too much, or a check fails.

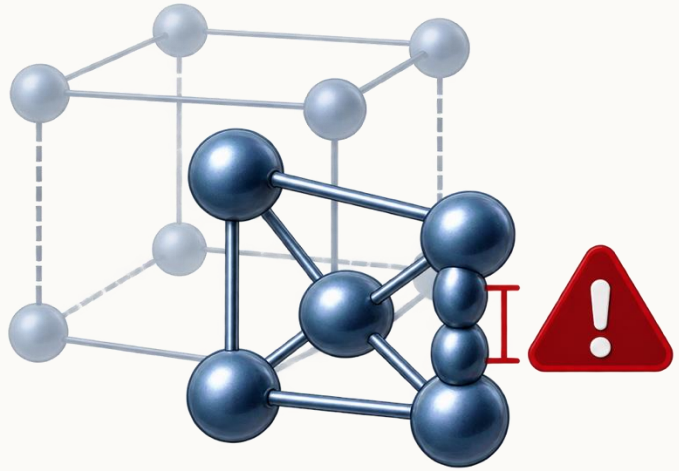
Don’t blindly continue with the old plan

Example: A Failed Simulation

The simulator rejects an invalid lattice parameter. We might:

- Check the source units and the tool input mapping.
- If the correction is clear, submit a new job.
- If the source is ambiguous, stop and ask for review.

Useful: keep the failed call in the trace.

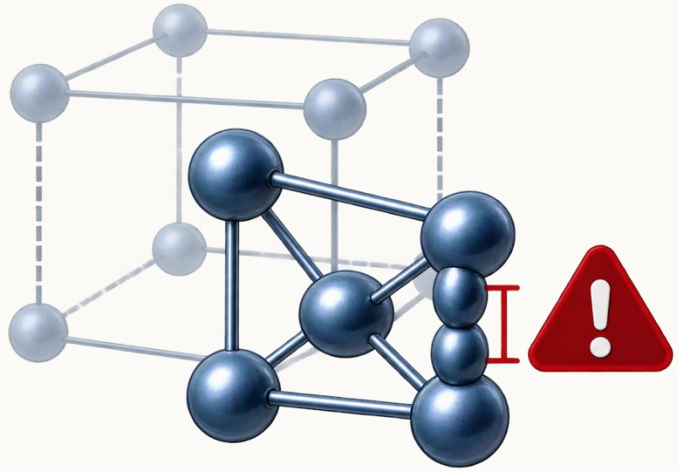


Why Not Just Take the Next Easy Action?

I.e., a “greedy” loop. A greedy loop can look productive while going nowhere.

- The easiest call may not reduce the main uncertainty.
- An early guess can trap the rest of the trajectory.
- Compare a few plausible next steps before spending.

Especially important when experiments are expensive.



Search Over Plans

Search Over Actions (Koh et al. ’25)

- The agent proposes several possible next actions.
- Execute them in a resettable environment, score the resulting observations for progress toward the goal.
- Expand the most promising branch first, and backtrack when a path fails.

Tree Search for Language Model Agents

Jing Yu Koh *jingyuk@cs.cmu.edu*
Carnegie Mellon University

Stephen McAleer *smcaleer@cs.cmu.edu*
Carnegie Mellon University

Daniel Fried *dfried@cs.cmu.edu*
Carnegie Mellon University

Ruslan Salakhutdinov *rsalakhu@cs.cmu.edu*
Carnegie Mellon University

Reviewed on OpenReview: https://openreview.net/forum?id=QF0N3x2XVm

Abstract

Autonomous agents powered by language models (LMs) have demonstrated promise in their ability to perform decision-making tasks such as web automation. However, a key limitation remains: LMs, primarily optimized for natural language understanding and generation, struggle with multi-step reasoning, planning, and using kenvironmental feedback when attempting to solve realistic computer tasks. Towards addressing this, we propose an inference-time search algorithm for LM agents to explicitly perform exploration and multi-step planning in interactive web environments. Our approach is a form of best-first tree search that operates within the actual environment space, and is complementary with most existing state-of-the-art agents. It is the first tree search algorithm for LM agents that shows effectiveness on realistic web tasks. On the challenging VisualWebArena benchmark, applying our search algorithm on top of a GPT-4o agent yields a 39.7% relative increase in success rate compared to the same baseline without search, setting a state-of-the-art success rate of 26.4%. On WebArena, search also yields a 28.0% relative improvement over a baseline agent, setting a competitive success rate of 19.2%. Our experiments showcase the effectiveness of search for web agents, and we demonstrate that performance scales with increased test-time compute.

1 Introduction

Building agents that can perceive, plan, and act autonomously has been a long-standing goal of artificial

Example: Three Possible Next Calls

The agent returned a candidate material. What should the agent do next?

- Run the same query again: cheap, little new information.
- Check units and provenance: cheap and diagnostic.
- Independent simulation: stronger evidence, higher cost.

Check the source before paying for the simulation.

Does Repeating the Model Fix a Unit Error?

A: No. The same pipeline can repeat a systematic mistake.

- Inspect the original source and metadata.
- Check the unit conversion separately.
- Compare with a method that does not share the same error.

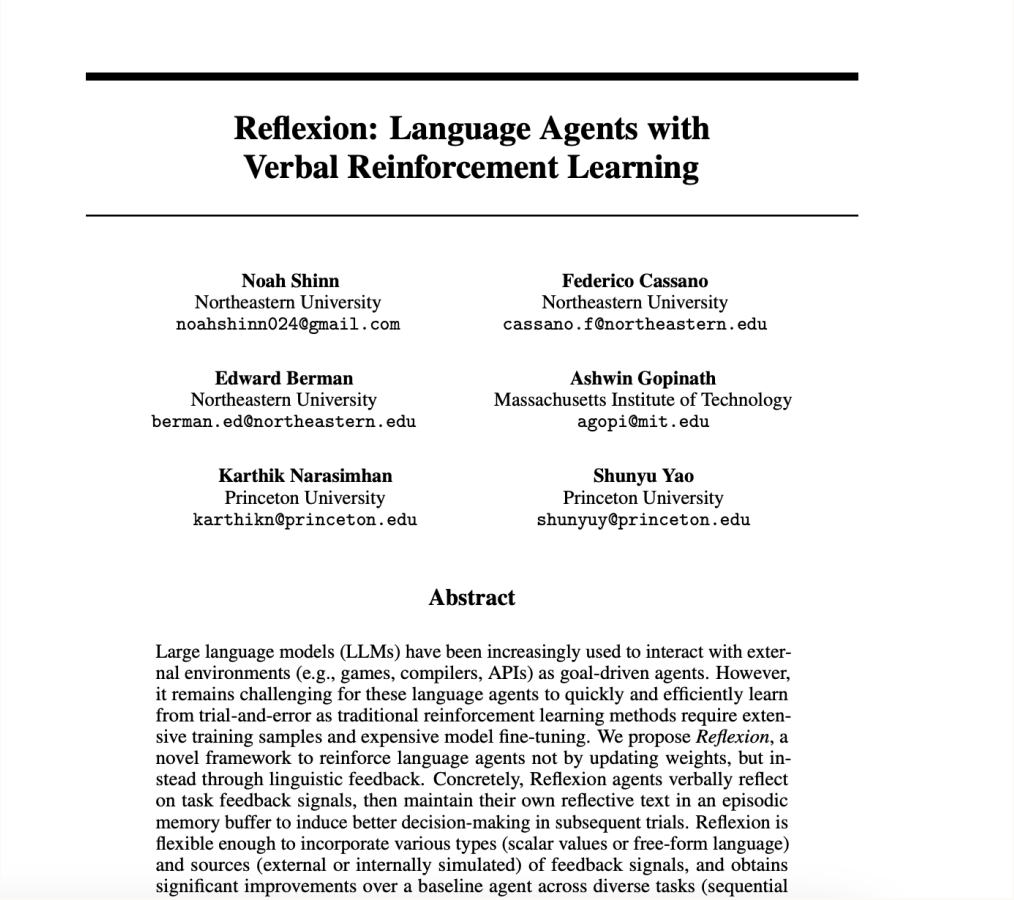
Which evidence would change your mind about A-03?

Reflection Versus Verification

Self-reflection can help when we have feedback (recall our evaluator workflow). Idea:

- Use a failed attempt and a concrete signal to revise.
- Without that signal, the model may rationalize the error.
- Verification needs a check outside the same reasoning trace (from verifier, evaluator, etc.)

E.g., Reflexion (Shinn et al. ’23); Huang et al. ’23.



Reflexion: Language Agents with Verbal Reinforcement Learning

Noah Shinn
Northeastern University
noahshinn024@gmail.com

Federico Cassano
Northeastern University
cassano.f@northeastern.edu

Edward Berman
Northeastern University
berman.ed@northeastern.edu

Ashwin Gopinath
Massachusetts Institute of Technology
agopi@mit.edu

Karthik Narasimhan
Princeton University
karthikn@princeton.edu

Shunyu Yao
Princeton University
shunyuy@princeton.edu

Abstract

Large language models (LLMs) have been increasingly used to interact with external environments (e.g., games, compilers, APIs) as goal-driven agents. However, it remains challenging for these language agents to quickly and efficiently learn from trial-and-error as traditional reinforcement learning methods require extensive training samples and expensive model fine-tuning. We propose *Reflexion*, a novel framework to reinforce language agents not by updating weights, but instead through linguistic feedback. Concretely, Reflexion agents verbally reflect on task feedback signals, then maintain their own reflective text in an episodic memory buffer to induce better decision-making in subsequent trials. Reflexion is flexible enough to incorporate various types (scalar values or free-form language) and sources (external or internally simulated) of feedback signals, and obtains significant improvements over a baseline agent across diverse tasks (sequential

Example: The Metric Does Not Match

Our run gives 0.84; the paper reports 0.90.

- Same data split? Same metric definition?
- Same code version and preprocessing?
- Fix a specific discrepancy, rerun, keep both attempts.

One thing to avoid: don’t just ask the model whether 0.84 is “close enough.”

How Do We Check a Scientific Result?

A useful sequence of checks:

- **Syntax**: did the tool accept the inputs?
- **Semantics**: do the units and definitions make sense?
- **Reproduction**: can the procedure recover the result?
- **Independence**: does a different method agree?
- **Claim**: are we saying more than the evidence supports?

Are We Evaluating the Plan or the Execution?

Important distinction! Starting to get attention. Recent example: Agent Planning Benchmark (Sun et al. '26).

- Tests planning with extra tools, broken tools, and impossible tasks.
- Separates a bad plan from a failed execution.

Would the agent notice an infeasible step before spending credits?

Agent Planning Benchmark: A Diagnostic Framework for Planning Capabilities in LLM Agents

Haoyu Sun[1,2,*] Wenxuan Wang[3,2,*] Mingyang Song[4] Jujie He[5]
Weinan Zhang[3] Yang Liu[6] Yang Yang[7,†] Yu Cheng[8,2,†]
[1]Tongji University [2]Shanghai AI Laboratory [3]Harbin Institute of Technology
[4]Fudan University [5]Skywork AI [6]University of California, Santa Cruz
[7]Shanghai Jiao Tong University [8]The Chinese University of Hong Kong

Abstract

Planning is central to LLM agents: before acting, an agent must decompose goals, select tools, reason over constraints, and decide when a task is infeasible. Yet existing agent evaluations often report only end-to-end success, making it difficult to determine whether failures stem from planning or execution. We introduce **Agent Planning Benchmark (APB)**, a planning-specific diagnostic benchmark with 4,209 multimodal cases across 22 domains and five settings, covering holistic planning, feedback-conditioned step-wise planning, and robustness under extraneous tools, broken tools, and unsolvable tasks. Across 12 MLLMs, APB reveals systematic weaknesses in long-horizon planning, tool-noise robustness, calibrated refusal, and inference-time refinement. We further validate APB on 200 ToolSandbox tasks and 200 τ^2-bench tasks, where APB-guided refinement consistently improves plan correctness, plan grade, and downstream execution metrics across three representative models. APB thus serves as an upstream diagnostic complement to execution benchmarks. The APB benchmark and code are available in this URL.

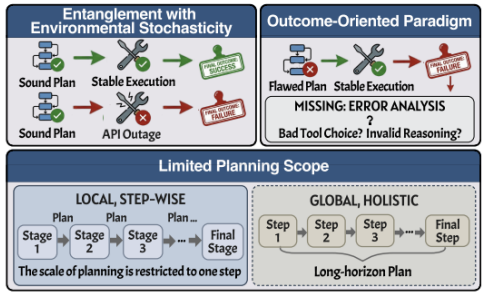


Figure 1: Systematic limitations in existing agent planning benchmarks.

et al., 2023); in open-ended agents such as Voyager (Wang et al., 2023a); and in workflow-level systems such as MetaGPT, LLMCompiler, and SWE-agent (Hong et al., 2023; Kim et al., 2024; Yang et al., 2024a). These systems demonstrate that planning is a foundational capability for reliable agents, shaping both high-level task decomposition and downstream tool-use behavior.

Despite this importance, current evaluations do not yet provide a sufficiently diagnostic view of planning. End-to-end benchmarks are indispens-

Outline

Plan construction
Goals, dependencies, and feedback-driven action

Search and verification
Replanning, alternatives, reflection, and checks

Long-horizon science
Budgets, checkpoints, and recent research systems

Planning With a Small Budget

Even lots of money in API credits can be a significant constraint!

How to handle it?

- Track calls and compute as the task runs.
- Use cheap checks to eliminate bad branches early.
- Keep enough for a repeat or independent verification.

Stop when another call cannot change the decision.

Long Tasks Need Checkpoints

A research task may outlive one context window.

- Save the goal, current plan, and open questions.
- Keep artifacts and the exact state needed to resume.
- Inspect that state before launching more work.

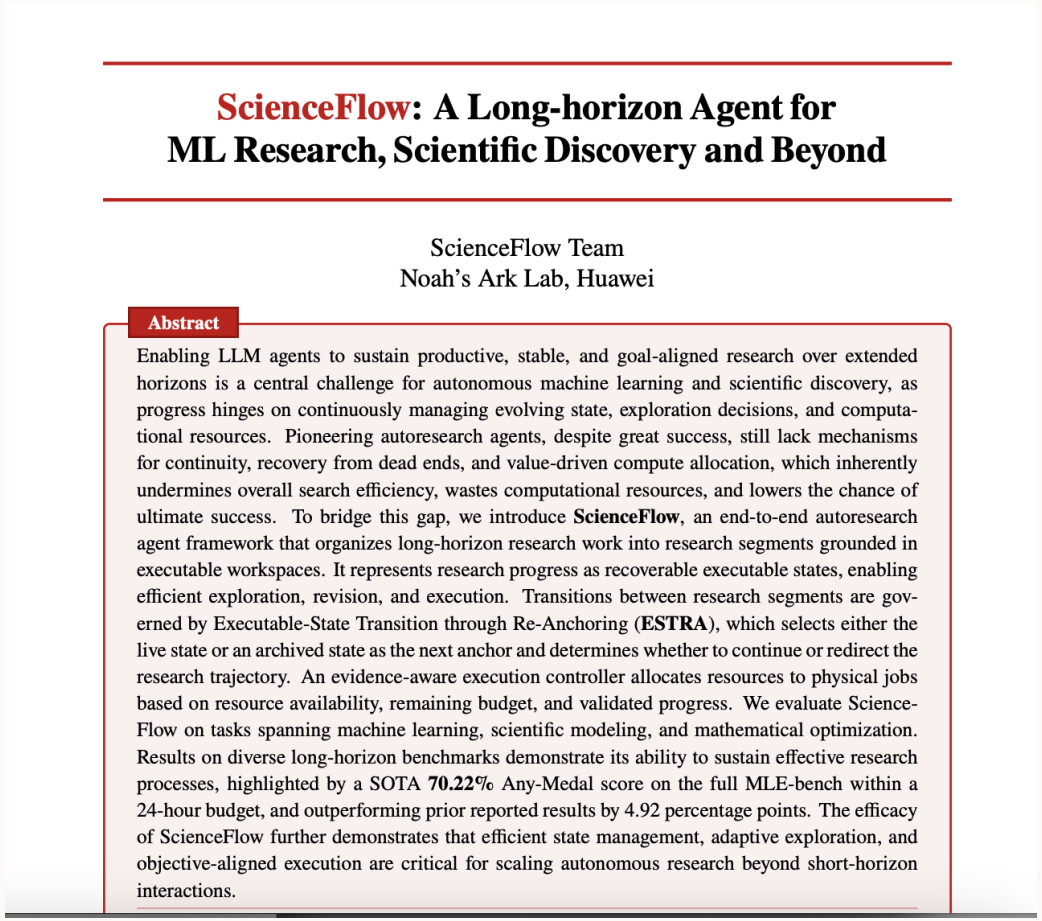
Next lecture: memory systems for agents.

Example: ScienceFlow

Zhao et al. ’26 study long-running research agents.

- Store executable states, not only prose summaries.
- Return to a live or archived state after a dead end.
- Allocate compute based on budget and evidence of progress.

Good area for a class project ☺



ScienceFlow: A Long-horizon Agent for ML Research, Scientific Discovery and Beyond

ScienceFlow Team
Noah’s Ark Lab, Huawei

Abstract

Enabling LLM agents to sustain productive, stable, and goal-aligned research over extended horizons is a central challenge for autonomous machine learning and scientific discovery, as progress hinges on continuously managing evolving state, exploration decisions, and computational resources. Pioneering autoresearch agents, despite great success, still lack mechanisms for continuity, recovery from dead ends, and value-driven compute allocation, which inherently undermines overall search efficiency, wastes computational resources, and lowers the chance of ultimate success. To bridge this gap, we introduce **ScienceFlow**, an end-to-end autoresearch agent framework that organizes long-horizon research work into research segments grounded in executable workspaces. It represents research progress as recoverable executable states, enabling efficient exploration, revision, and execution. Transitions between research segments are governed by Executable-State Transition through Re-Anchoring (**ESTRA**), which selects either the live state or an archived state as the next anchor and determines whether to continue or redirect the research trajectory. An evidence-aware execution controller allocates resources to physical jobs based on resource availability, remaining budget, and validated progress. We evaluate ScienceFlow on tasks spanning machine learning, scientific modeling, and mathematical optimization. Results on diverse long-horizon benchmarks demonstrate its ability to sustain effective research processes, highlighted by a SOTA **70.22%** Any-Medal score on the full MLE-bench within a 24-hour budget, and outperforming prior reported results by 4.92 percentage points. The efficacy of ScienceFlow further demonstrates that efficient state management, adaptive exploration, and objective-aligned execution are critical for scaling autonomous research beyond short-horizon interactions.

Putting the Pieces Together

One reasonable planning loop:

- Set a success criterion and constraints.
- Decompose the task into checkable steps.
- Take one action and record what happened.
- Verify, revise the plan if needed, then stop or continue.

Takeaways

Planning is key!

- **Decomposition** tells us what each step must produce.
- **ReAct** gives us online planning via an action-observation loop.
- **Search** and **replanning** help us deal with uncertain paths.
- V**erification** is separate from model confidence.

Readings:

ReAct: Yao et al. (2023), arXiv:2210.03629.

Tree of Thoughts: Yao et al. (2023), arXiv:2305.10601.

Reflexion: Shinn et al. (2023), arXiv:2303.11366.

Self-correction limits: Huang et al. (2023), arXiv:2310.01798.

Agent Planning Benchmark: Sun et al. (2026), arXiv:2606.04874.

ScienceFlow: Zhao et al. (2026), arXiv:2608.14354.