

Hierarchical planning and subgoal discovery

Subgoal contracts, preconditions and completion evidence.

Reasoning agents, feedback, and scientific evidence

A plausible action can have an unmet prerequisite

Imagine a research plan schedules model fitting before a reference measurement is validated. Every individual step sounds sensible, but the sequence can use invalid evidence.

Some tasks depend on prior results, while other tasks can proceed independently. A list of plausible actions hides these differences.

This makes prerequisites and completion conditions worth representing explicitly. The direction arises from the structure of the task.

Deriving subgoals from dependencies

Work backward from a justified conclusion. Identify what would have to be established first, then express those requirements as candidate subgoals.

Connect each subgoal to the observations or actions that could establish it. Completion must mean that its evidence requirement has been met.

A failed prerequisite may alter the dependency graph. That creates a choice between a fixed plan and a plan that can change during execution.

Choosing when a plan may change

Candidate	Reason to choose it	Cost or limitation
Reactive next action	Immediate feedback is sufficient	Dependencies can be forgotten over a long task
Fixed decomposition	Prerequisites and outcomes are predictable	Unexpected evidence can invalidate later steps
Adaptive subgoals	New observations change what must be done	Replanning needs state, checks and extra compute

Use a fixed plan when dependencies are stable. Add replanning when feedback can change the requirements or feasible route.

Pedagogical reconstruction

[Fikes and Nilsson \(1971\)](#)

[Prasad et al. \(2023\)](#)

Decomposition should resolve a specific difficulty

An agent can break a failed subgoal into smaller tasks when the failure reflects insufficient operational detail.

Decomposition does not repair an impossible objective, missing resource, or false scientific assumption. Diagnose the failure before adding more steps.

Fixed decomposition, adaptive decomposition and a capable direct executor offer different trade-offs. A small reliable plan can be preferable to an elaborate but brittle hierarchy.

Classical planning checks plans within a model

A planner searches for actions whose preconditions and effects satisfy a goal in a defined domain model. An LLM may translate a natural-language problem into that representation.

A valid plan is valid relative to the translated model. Translation errors and omitted physical effects can invalidate its real-world interpretation.

Check both the formal plan and the mapping between domain predicates and observed states.

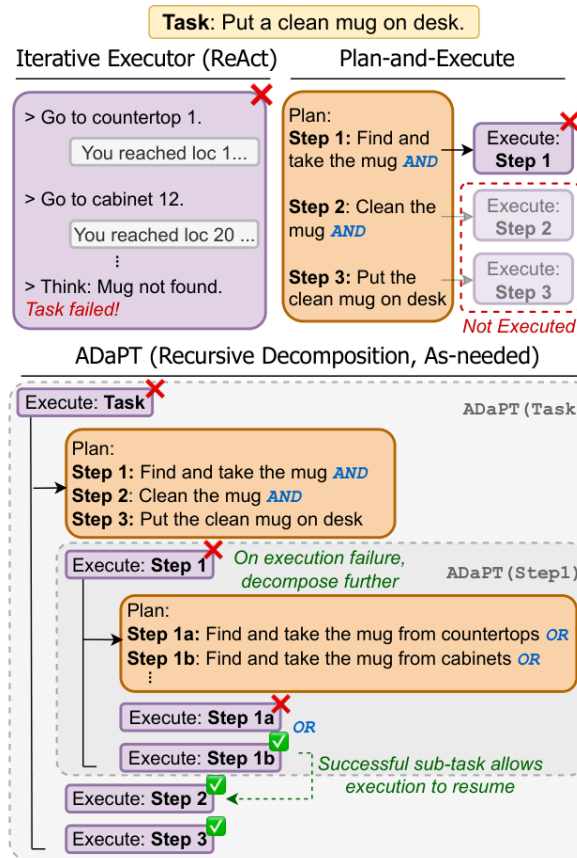
ADaPT: the method

ADaPT (As-Needed Decomposition and Planning) first tries to execute a task. When execution is insufficient, it decomposes the task into subtasks and can repeat this procedure for difficult subgoals.

The approach connects planning granularity to observed execution difficulty instead of requiring one fixed decomposition in advance.

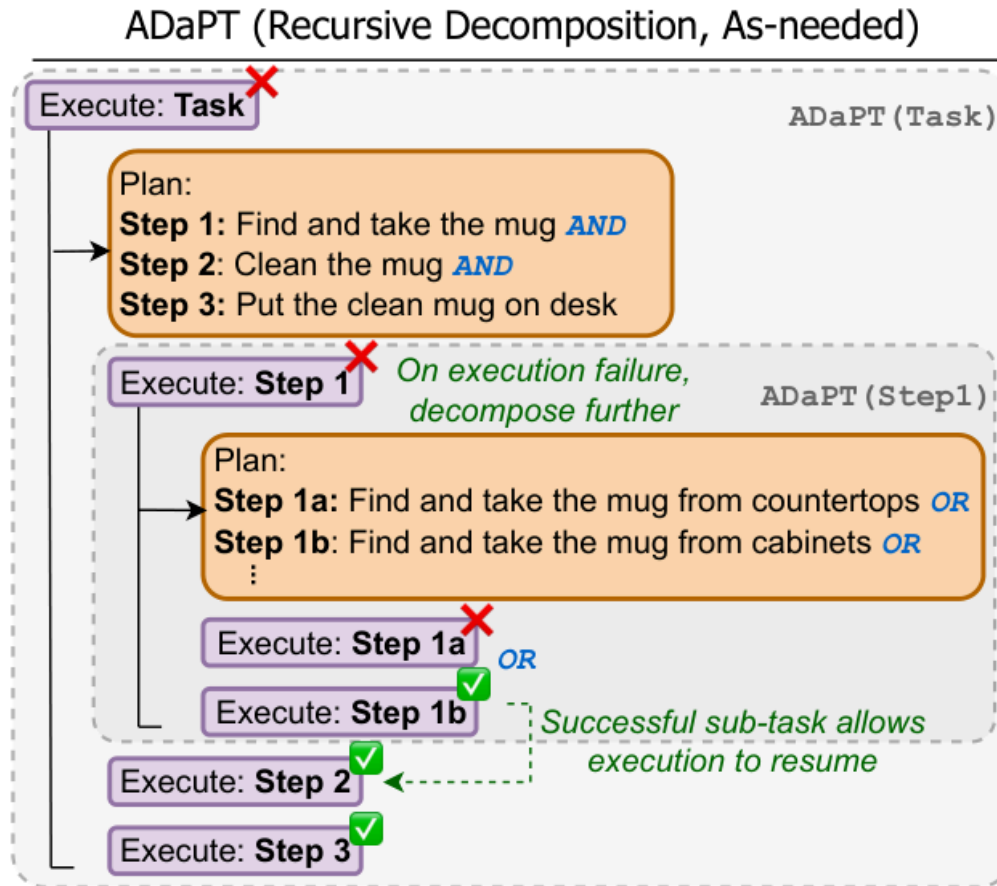
Decomposition remains useful only when the failure can be resolved by smaller tasks; it cannot create unavailable resources or make an impossible objective feasible.

ADaPT: execution and adaptive decomposition



The example contrasts iterative execution, a fixed plan and adaptive decomposition. Failure can trigger smaller subgoals whose completion allows the larger task to resume. The methods differ in when they invoke decomposition.

ADaPT: mechanism detail



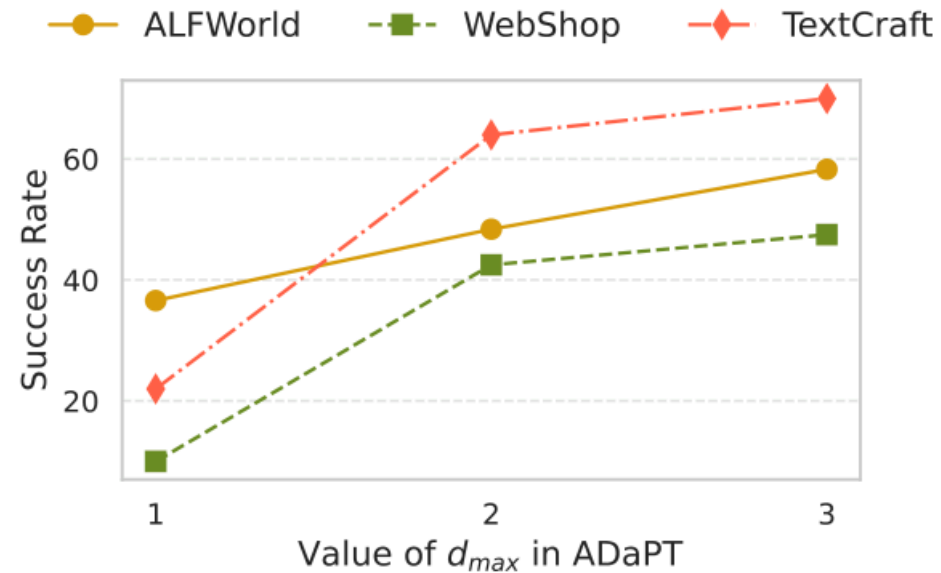
An execution failure can trigger decomposition into smaller subgoals. A failure caused by missing operational detail differs from an unavailable resource or an impossible goal; further decomposition need not resolve the latter cases.

Evidence: aggregate and task-specific planning results

Method ($d_{\max} = 3$)	Pick	Clean	Heat	Cool	Look	Pick2	All
ReAct	33.3	<u>67.7</u>	43.5	33.3	55.6	<u>11.8</u>	43.3
Plan-and-Execute	29.2	61.3	47.8	38.1	61.1	<u>11.8</u>	43.3
Try Again with ReAct	50.0	51.6	<u>60.8</u>	47.6	61.1	5.9	47.8
Reflexion	70.8	61.3	61.0	<u>66.7</u>	61.1	5.9	<u>57.5</u>
ADAPT (Ours)	87.5	80.6	<u>60.8</u>	76.2	61.1	52.9	71.6

On the ALFWorld test split, ADaPT reports 71.6% overall success versus 43.3% for ReAct and 57.5% for Reflexion. The task columns reveal unequal effects across problem types. The paper uses a shared GPT-3.5 executor and compares model-call budgets. These controls strengthen the contrast while leaving transfer to scientific workflows untested.

ADaPT: interpretation and limits



Decomposition errors can propagate, and success depends on a capable executor and meaningful feedback. More recursion may add overhead or distract from a missing prerequisite rather than repair the task.

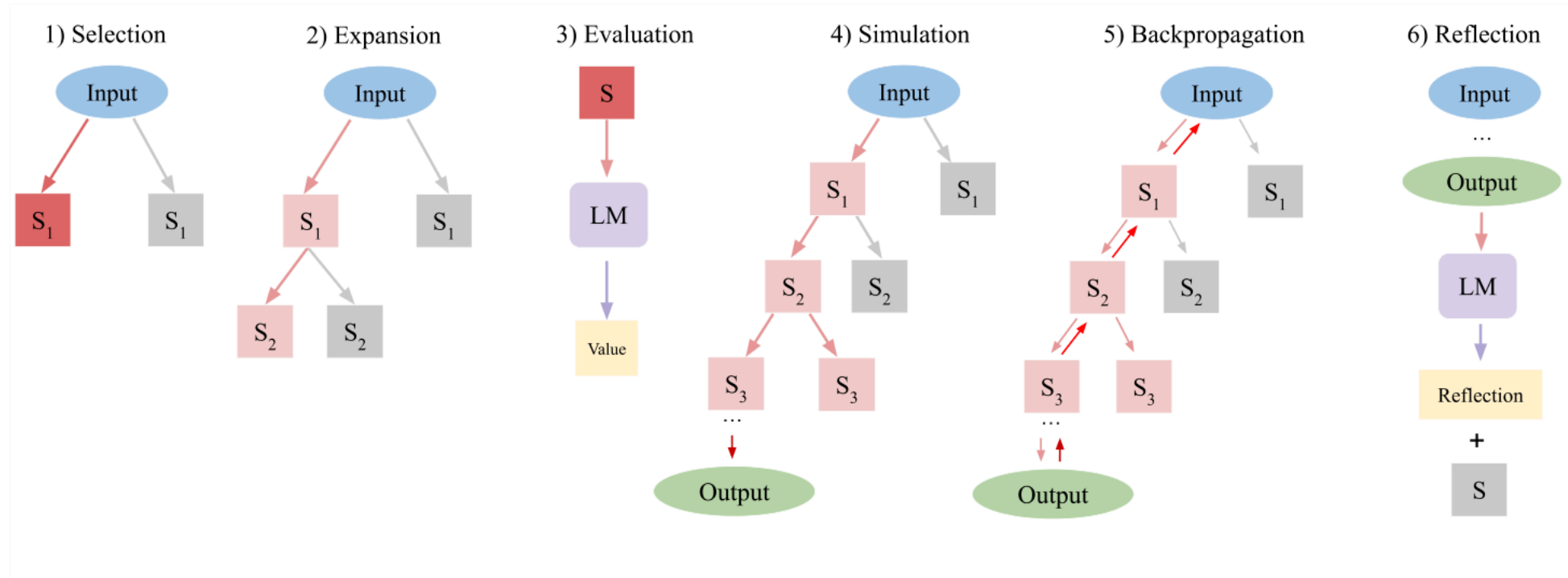
Language Agent Tree Search: the method

LATS combines language-model reasoning and action selection with Monte Carlo tree-search components and reflection.

Selection, expansion, evaluation, simulation, backup, and reflection organise candidate trajectories and their feedback. The environment can provide information that changes which branches are useful.

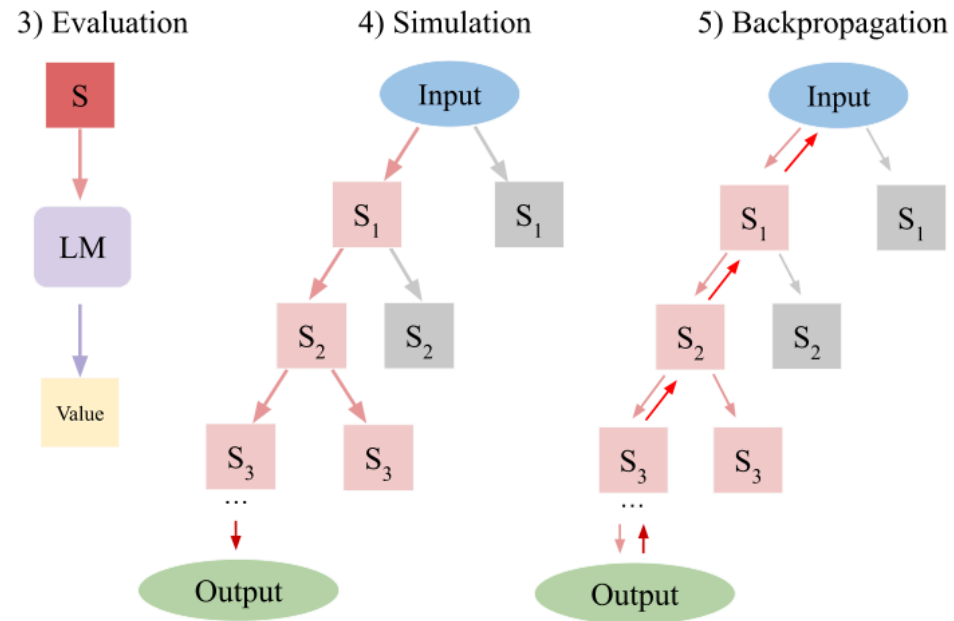
A search-value update is an estimate of future return; it should not be confused with a posterior update over a scientific mechanism.

LATS: search and feedback operations



The six operations separate model proposals from environment or evaluator feedback. Feedback changes search statistics, while reflection changes the context supplied to later proposals.

Language Agent Tree Search: mechanism detail



Evaluation, simulation and backup connect estimated values to observed returns. Their distinction matters for reproducible computation: a prediction about a branch and a validated execution result carry different evidential weight.

Evaluation scope

The study evaluates the framework on reasoning, acting, and programming tasks with task-specific success measures.

A table or architectural example must be interpreted within its model, prompt, and resource conditions.

The results motivate integrating search with environmental feedback. They do not establish that every component is necessary or that the same evaluator transfers to open scientific discovery.

Language Agent Tree Search: interpretation and limits

Prompt Method	Model	Pass@1 ↑
CoT (Wei et al., 2022)	GPT-3.5	46.9
ReAct (Yao et al., 2023b)	GPT-3.5	56.9
Reflexion (Shinn et al., 2023)	GPT-3.5	68.1
ToT (Yao et al., 2023a)	GPT-3.5	54.4
RAP (Hao et al., 2023)	GPT-3.5	63.1
LATS (ReAct)	GPT-3.5	83.8
Base LM	GPT-4	80.1
Reflexion	GPT-4	91.0
LATS (ReAct)	GPT-4	92.7

Environment feedback and prompted evaluation can have different reliability. Comparisons must account for the search budget, tool access, and reflection overhead, especially when actions have unequal cost.

Critical paths govern completion time

In a dependency graph, the earliest completion time is determined by the longest prerequisite path, assuming sufficient resources and known durations.

Parallel execution can reduce wall-clock time without reducing total work. Resource contention, shared data, and result-dependent branches complicate the schedule.

A planner should track pending jobs and revisions to duration estimates rather than assuming every tool call finishes immediately.

Replanning needs typed failure signals

Separate a syntax error, unavailable input, violated precondition, failed measurement, and contradictory scientific result.

A syntax error may justify repair and retry. A missing input may require another subgoal. A reliable contradictory result may require revising the hypothesis.

Repeatedly retrying the same action is not adaptation when its failure condition has not changed. Use the failure type to choose the repair level.

Subgoal completion requires evidence

A useful subgoal contract specifies inputs, allowed actions, expected outputs, validation checks, resource limits, and termination states.

“Analyse the data” is too broad to verify. “Estimate the predefined effect with a reproducible script and uncertainty interval” is easier to audit.

Contracts improve coordination but can still encode the wrong objective. Review their scientific relevance as well as their operational precision.

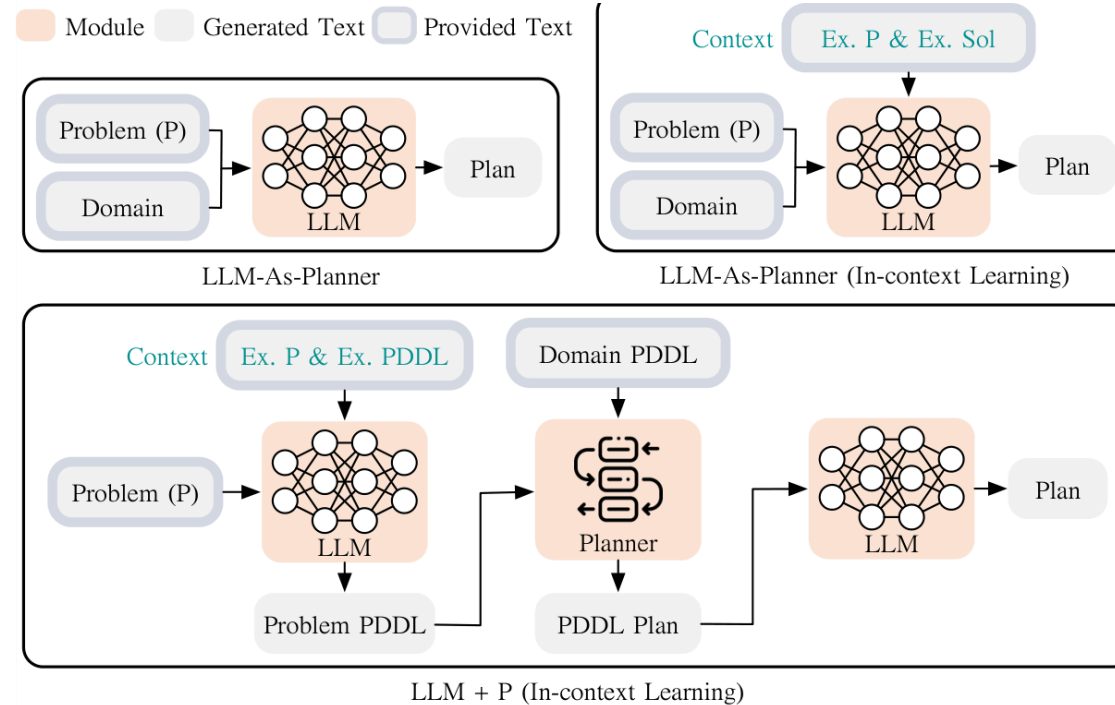
LLM+P: the method

LLM+P translates a natural-language planning problem into a formal planning representation, calls a classical planner, and translates the resulting plan back into language.

The planner can reason precisely within the supplied domain model.

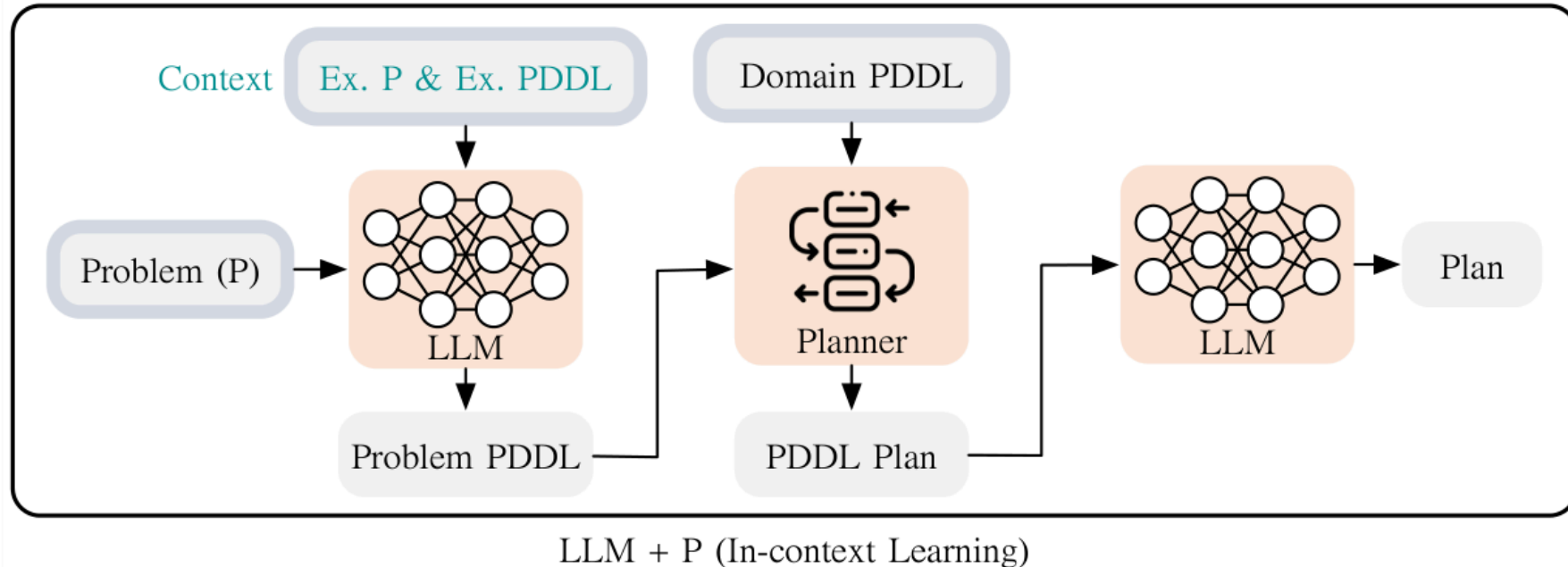
Correctness is conditional on the translation and the model's preconditions and effects. A formally valid plan does not establish that the natural-language problem or physical environment was represented accurately.

LLM+P: translation and classical planning



Direct language-model planning contrasts with translation into PDDL, classical planning and translation back to action. Both translation boundaries can introduce errors independently of the planner's search algorithm.

LLM+P: mechanism detail



The formal planner uses domain predicates, action preconditions and effects, an initial state and a goal. Each translated fact must correspond to the environment; a valid formal plan cannot repair an incorrect translation.

Evaluation scope

The source evaluates planning performance and includes a demonstrated execution sequence. The original robot-action figure shows ordered actions in a particular environment.

A successful demonstration supports that execution example; a broader planning claim needs systematic task comparisons.

Separate formal plan validity, translation fidelity, and successful environmental execution when reporting results.

LLM+P: interpretation and limits



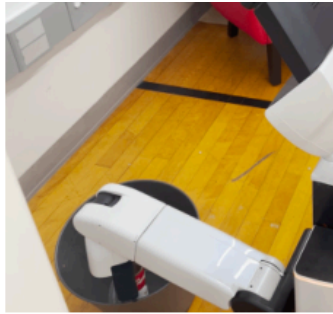
(a) grasp bottle



(b) free gripper



(c) grasp soup can



(d) place soup can



(e) re-grasp bottle



(f) place bottle

The domain model may omit relevant effects or uncertainty. A plan can be optimal under an incomplete model yet fail in the actual environment. Natural-language translation should be checked as a substantive component.

Formal reasoning defines a target statement

A theorem-proving agent searches for a proof of a specified proposition. A proof assistant checks the submitted derivation under its formal rules and dependencies.

LeanDojo studies retrieval-augmented theorem proving with access to a Lean environment. Retrieving useful premises and producing an accepted proof are distinct subproblems.

Three levels of checking

Check	Object validated	Remaining obligation
Unit test	Selected executions	Untested cases
Formal proof	Specified proposition	Adequate specification
Scientific validation	Model-to-world claim	Measurement and scope

A stronger check at one level does not remove assumptions at another.

The induction step needs an actual algebraic bridge

Step	Expression	Justification
Inductive hypothesis	$S(n) = n(n + 1)(2n + 1)/6$	Assume the claim at n
Add the next square	$S(n + 1) = S(n) + (n + 1)^2$	Definition of the sum
Factor the numerator	$(n + 1)[2n^2 + 7n + 6]/6$	Common denominator and expansion
Factor once more	$(n + 1)(n + 2)(2n + 3)/6$	The claimed form with n replaced by $n + 1$

$S(n) = 1^2 + \dots + n^2$; base case $S(0) = 0$. A proof plan becomes a proof when each algebraic bridge is justified.

A failed precondition changes the plan

Plan state	Feedback	Next justified step
Compare sensor with reference	Sensor: °C; reference: kelvin	Convert units before comparing values
Attempt the comparison	Reference calibration date missing	Retrieve the certificate; do not invent it
Certificate shows expired calibration	Reference is not a reliable control	Obtain an independent calibrated reference
Repeat under valid conditions	Residual discrepancy remains	Only now compare drift explanations

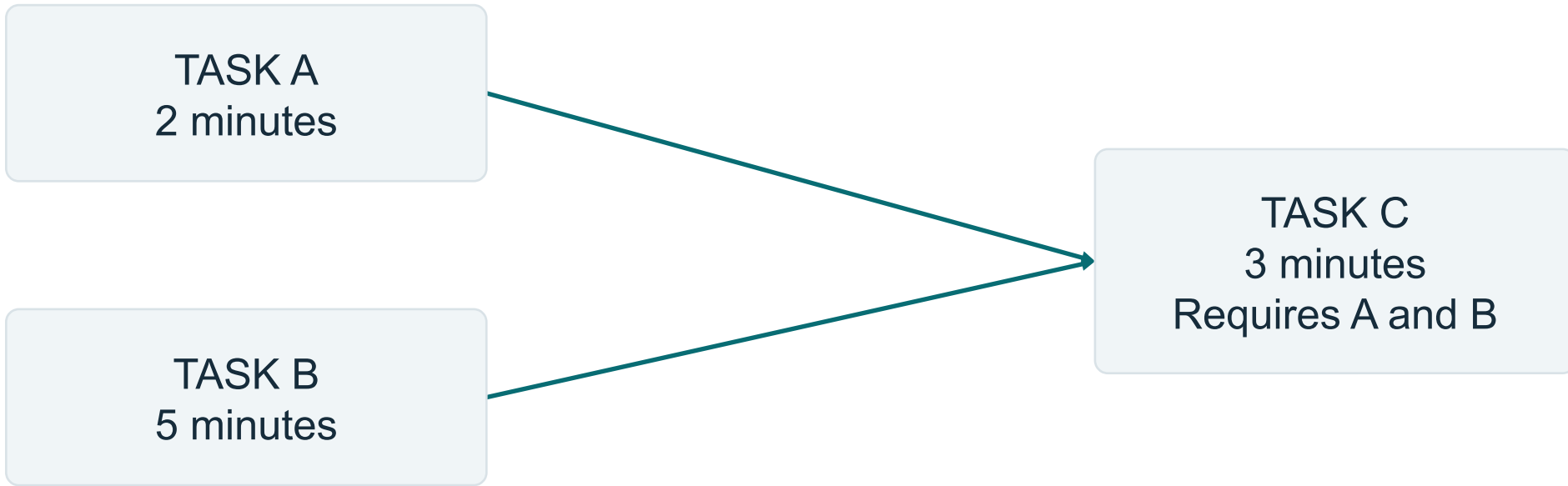
Synthetic tool-and-planning trace. Local execution failure can require replanning, not merely retrying the same action.

A dependency graph determines the schedule

Operation	Calculation or state	Consequence
Independent tasks	A takes 3; B takes 4	Start both at time 0
Join task C	Needs A and B; takes 2	Start at 4, finish at 6
Validation D	Needs C; takes 1	Finish at 7
Serial comparison	$3+4+2+1=10$	Parallel makespan is 7

Synthetic deterministic schedule with enough resources for A and B to run together. A dependency edge represents a required completion condition.

Worked schedule: three dependent computations



A and B run concurrently, while C waits for both valid outputs. Under sufficient processors, fixed durations and no shared contention, total work differs from elapsed time. Failure of A's integrity check delays C even if B completes.

Compute work and the critical path

$$T = \max(2, 5) + 3 = 8 \text{ minutes}$$

Total work is two plus five plus three, or ten processor-minutes. Running A and B concurrently allows C to start after five minutes and finish after eight.

The critical path is B followed by C. Speedup relative to the ten-minute sequential schedule is 1.25.

This limited gain is determined by the dependency structure. Adding more workers cannot shorten B or C under the stated assumptions.

An observation can invalidate the schedule

Suppose A returns a failed data-integrity check after two minutes. C cannot legitimately use its output. B can continue if it does not depend on the corrupted data.

The planner should create a repair or reacquisition branch and revise the earliest completion estimate.

Do not label C complete merely because its process ran on invalid input. The distinction between operational completion and evidential validity is part of the subgoal contract.

Planning performance at matched resources

Controlled missing inputs and delayed results reveal differences between sequential execution, static directed acyclic graph (DAG) scheduling and adaptive planning.

With available tools and processors matched, completion quality, wall-clock time, total work, repeated actions and recovery describe the resulting system trade-offs.

Tasks without useful parallelism distinguish dependency recognition from gains caused simply by launching additional jobs.

Preconditions and effects make a plan executable

$$\begin{aligned} \text{applicable}(a, s) &\iff \text{pre}(a) \subseteq s \\ s' &= (s \setminus \text{del}(a)) \cup \text{add}(a) \end{aligned}$$

This deterministic set-of-facts model uses preconditions, delete effects, and add effects. It is a deliberately simple planning formalism.

An action label such as “analyse the data” is incomplete until the required inputs and resulting artifacts are specified.

For uncertain outcomes, replace the deterministic transition with an observation-aware model and replan after execution. A dependency DAG alone does not encode every possible failure.

From critical path to a robust schedule

The worked DAG assumes independent tasks A and B can execute concurrently. A shared instrument or software lock invalidates that assumption and can increase the makespan.

Add resource constraints explicitly and compare the new schedule with the eight-unit lower bound from the unconstrained critical path.

Track planned duration, actual duration, and failed prerequisites. This separates a planning error from an inaccurate duration model.

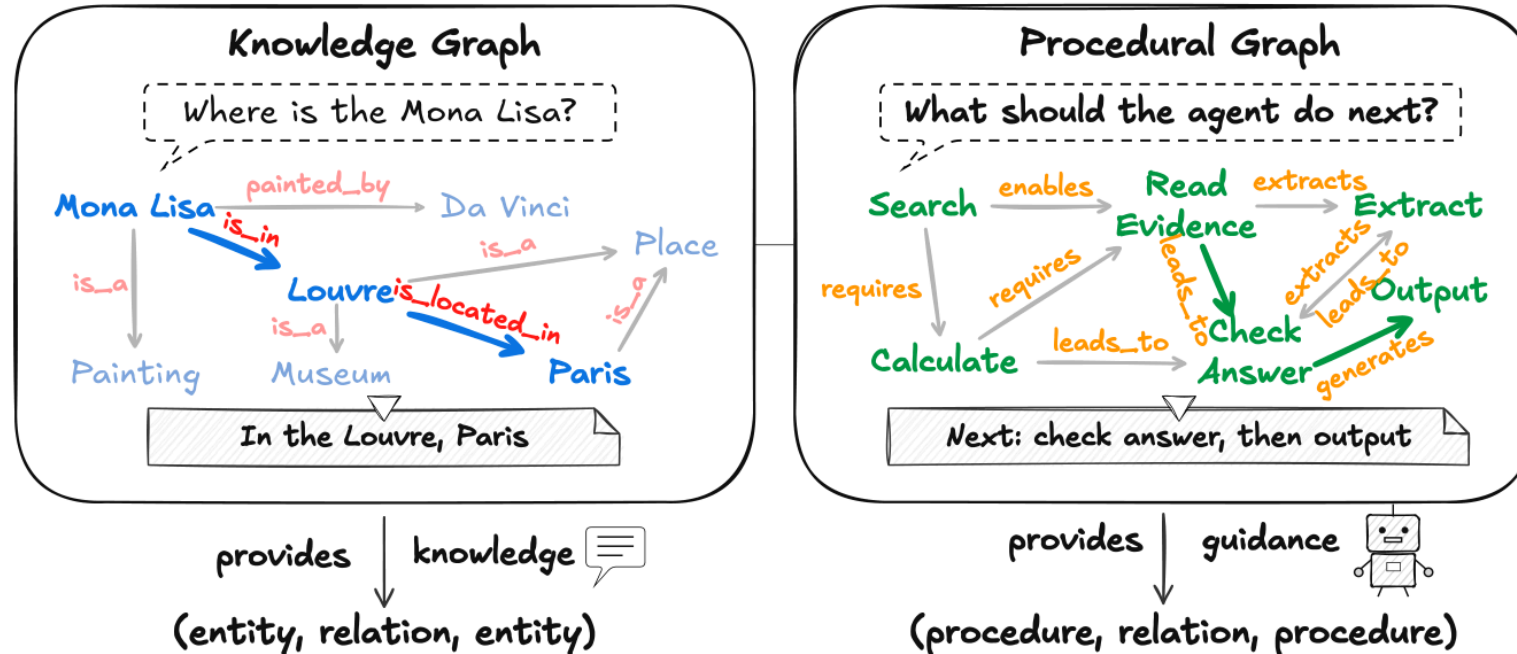
Recent research: Procedural Graphs

Procedural Graphs represent relationships between procedures and use the active subgraph to guide an agent's next action.

A refinement loop compares trajectories and proposes graph edits, retaining changes according to held-out validation performance.

The literature snapshot is dated 11 September 2026. Findings remain conditional on the source version and evaluation conditions. Recent preprints contribute reported results with distinct validation requirements.

Procedural Graphs: factual and procedural relations



Factual knowledge triplets describe what is; procedural triplets describe what to do. The illustration motivates that representational distinction, while the framework and experiments provide evidence about execution benefits.

Recent methods in context

AlphaProof (Nature 2025/26). Uses reinforcement learning in a formal proof environment, including problem-specific test-time training. Adaptation cost and human formalization affect the result; a verified proposition alone does not establish broader mathematical novelty.

Hilbert (ICLR 2026). Combines informal decomposition, a formal prover, retrieval and verifier feedback to construct proofs recursively. Formal verification validates a formal statement under its assumptions; correspondence to the intended mathematical question remains a separate issue.

Aletheia (preprint; revised 2026). Uses generation, verification and revision in natural language for research-level mathematics. The strength of each discovery claim depends on human contributions, priority searches and independent mathematical checking.

References (1/2)

[Archiki Prasad, Alexander Koller, Mareike Hartmann et al. \(2023\). ADaPT: As-Needed Decomposition and Planning with Language Models](#)

[Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman et al. \(2023\). Language Agent Tree Search Unifies Reasoning Acting and Planning in Language Models](#)

[Bo Liu, Yuqian Jiang, Xiaohan Zhang et al. \(2023\). LLM+P: Empowering Large Language Models with Optimal Planning Proficiency](#)

[Emil Biju, Shayan Talaei, Zhemin Huang et al. \(2025\). SPRINT: Enabling Interleaved Planning and Parallelized Execution in Reasoning Models](#)

[Richard E. Fikes, Nils J. Nilsson \(1971\). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving](#)

References (2/2)

[Yuxing Lu, Yicheng Chen, Shanchan Wu et al. \(2026\). Procedural Graphs: Self-Evolving Execution Structures for LLM Agents](#)

[Kaiyu Yang, Aidan M. Swope, Alex Gu et al. \(2023\). LeanDojo: Theorem Proving with Retrieval-Augmented Language Models](#)

[Thomas Hubert, Rishi Mehta, Laurent Sartran et al. \(2025\). Olympiad-level formal mathematical reasoning with reinforcement learning](#)

[Sumanth Varambally, Thomas Voice, Yanchao Sun et al. \(2025\). Hilbert: Recursively Building Formal Proofs with Informal Reasoning](#)

[Tony Feng, Trieu H. Trinh, Garrett Bingham et al. \(2026\). Towards Autonomous Mathematics Research](#)