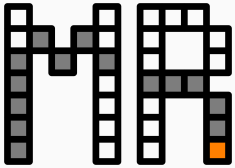
LLMs Don’t Even Need to Plan —They Can Build Planners

Jendrik Seipp

LM4Plan @ ICAPS 2026

Machine Reasoning Lab
Linköping University



[Some slides borrowed from Augusto B. Corrêa]

Overview

- **Translate**: NL $\rightarrow$ PDDL, hand it to a planner

Overview

- **Translate**: NL $\rightarrow$ PDDL, hand it to a planner
- **Solve**: the LLM emits the plan

Overview

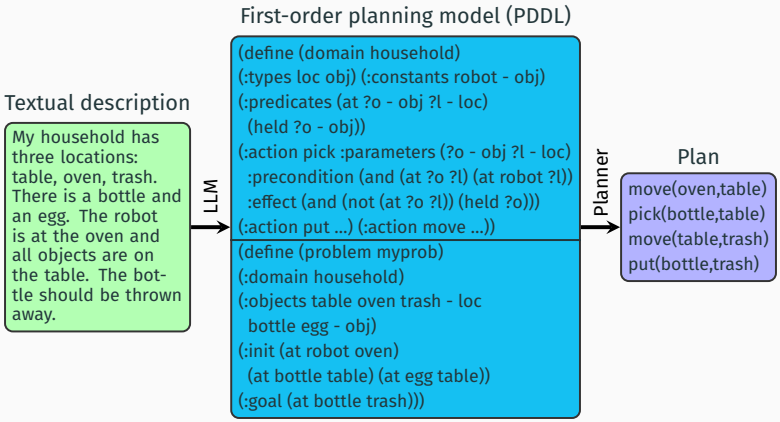
- **Translate**: NL $\rightarrow$ PDDL, hand it to a planner
- **Solve**: the LLM emits the plan
- **Build**: the LLM emits code a planner runs

Overview

- **Translate**: NL $\rightarrow$ PDDL, hand it to a planner
- **Solve**: the LLM emits the plan
- **Build**: the LLM emits code a planner runs
- **Research**: LLMs support & do the research

Translate

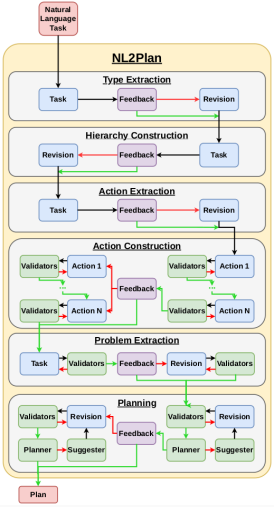
Translate: NL → PDDL, hand it to a planner



- NL2Plan [Gestrin et al. 2024], LLM+P [Liu et al. 2023], domain generators [Oswald et al. 2024]; survey [Tantakoun et al. 2025]; library `l2p`
- Inspectable PDDL, planner guarantees

NL2Plan: six careful steps to PDDL

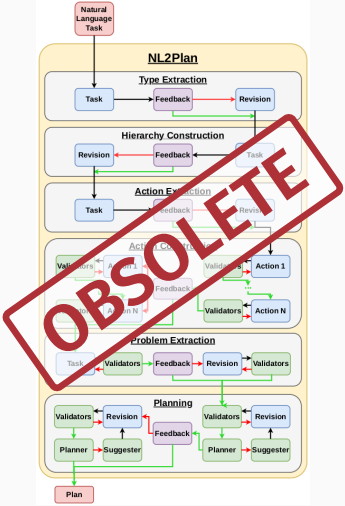
- Type Extraction
- Hierarchy Construction
- Action Extraction
- Action Construction
- Problem Extraction



NL2Plan: six careful steps to PDDL

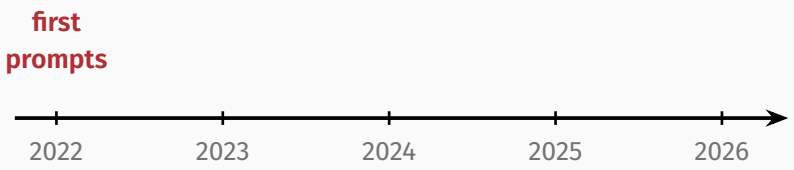
- Type Extraction
- Hierarchy Construction
- Action Extraction
- Action Construction
- Problem Extraction

⇝ now obsolete: frontier LLMs emit PDDL in one shot



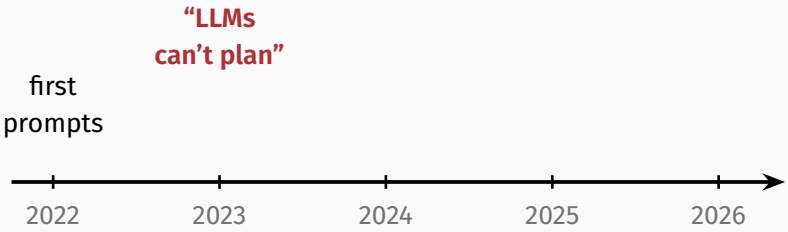
Solve

How LLM planning evolved



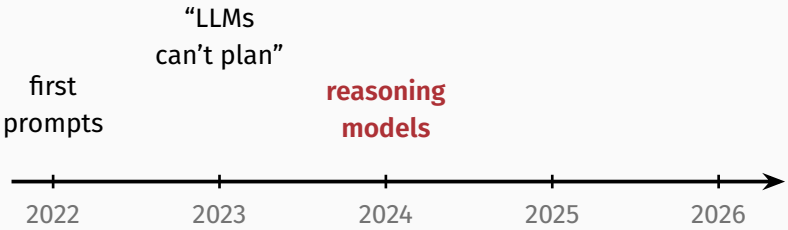
2022: can an LLM just *output* a plan?

How LLM planning evolved



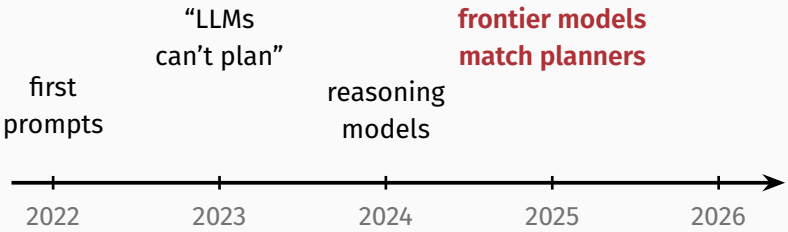
2023: “LLMs can’t plan” directly [Valmeekam et al. 2023]

How LLM planning evolved



2024: reasoning models (o1, R1) — a big jump

How LLM planning evolved



2025: frontier models match classical planners

A harder benchmark

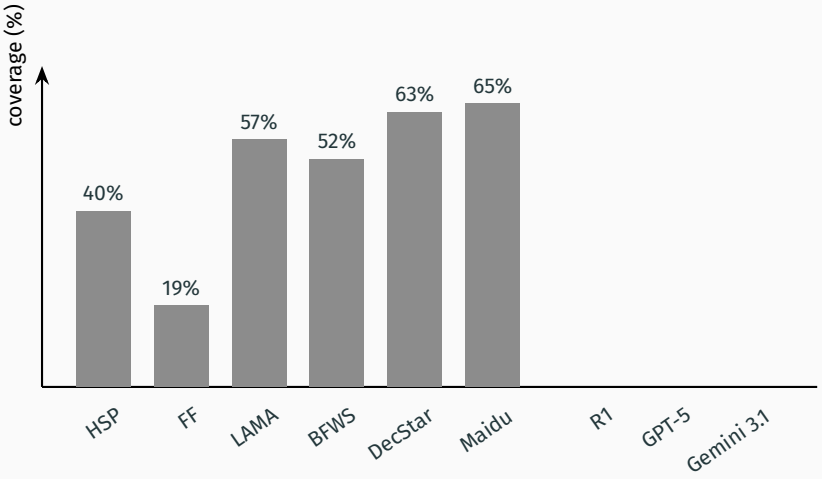
- 8 domains from the IPC 2023 Learning Track, with *freshly generated* tasks [Corrêa et al. 2025a]
  - fresh instances, no contamination
- Baseline planners from previous IPCs
- Every plan checked with VAL; planners get one CPU core, 8 GiB

A harder benchmark

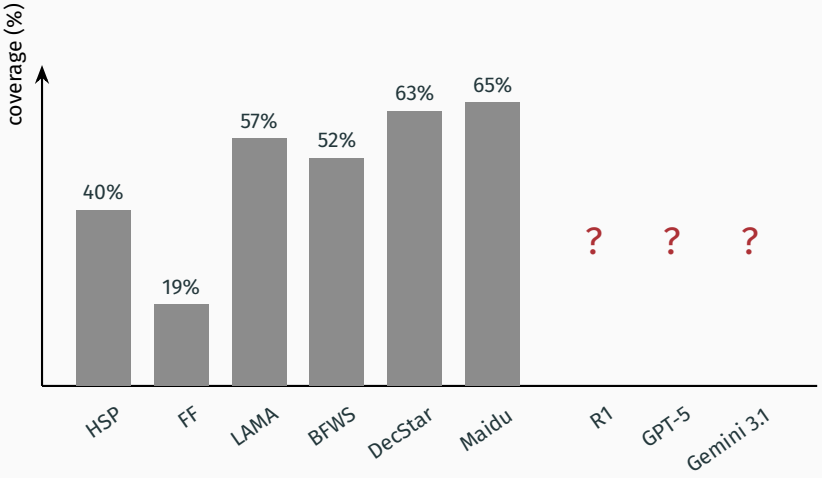
- 8 domains from the IPC 2023 Learning Track, with *freshly generated* tasks [Corrêa et al. 2025a]
  - fresh instances, no contamination
- Baseline planners from previous IPCs
- Every plan checked with VAL; planners get one CPU core, 8 GiB

	PlanBench	this benchmark
blocks	≤ 5	up to 477
plan length	≤ 16	> 1000 steps

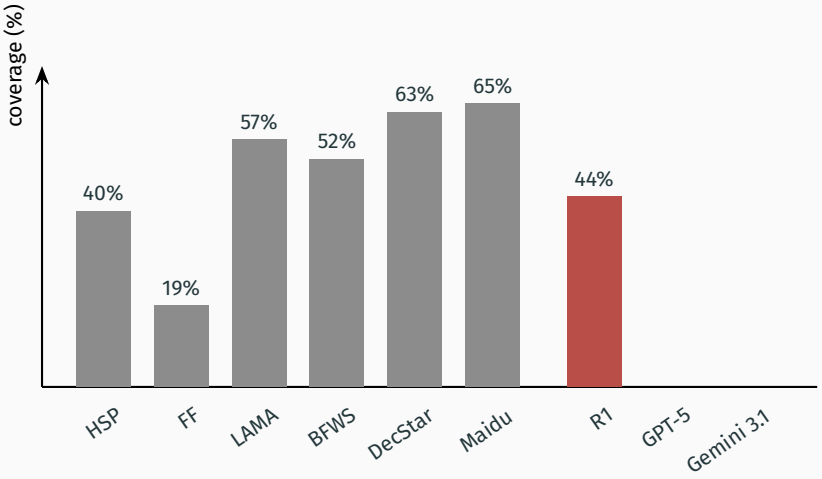
Coverage


coverage (%)
40%
19%
57%
52%
63%
65%
HSP
FF
LAMA
BFWS
DecStar
Maidu
R1
GPT-5
Gemini 3.1

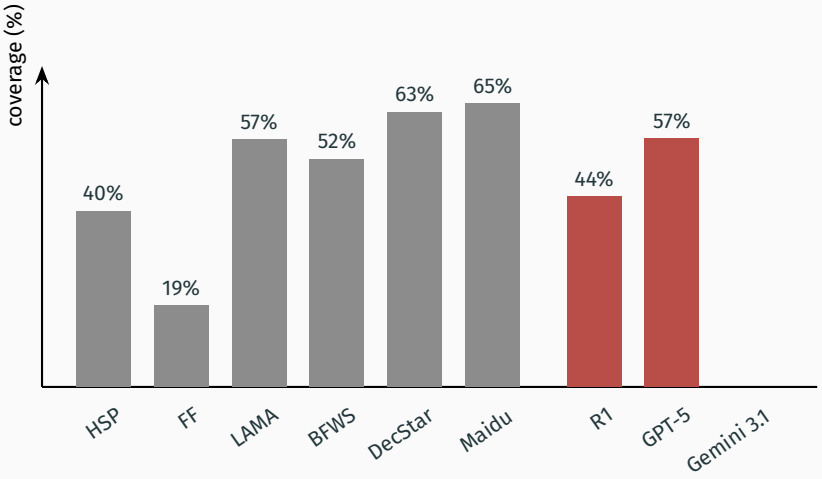
Coverage


coverage (%)
40%
19%
57%
52%
63%
65%
?
?
?
HSP
FF
LAMA
BFWS
DecStar
Maidu
R1
GPT-5
Gemini 3.1

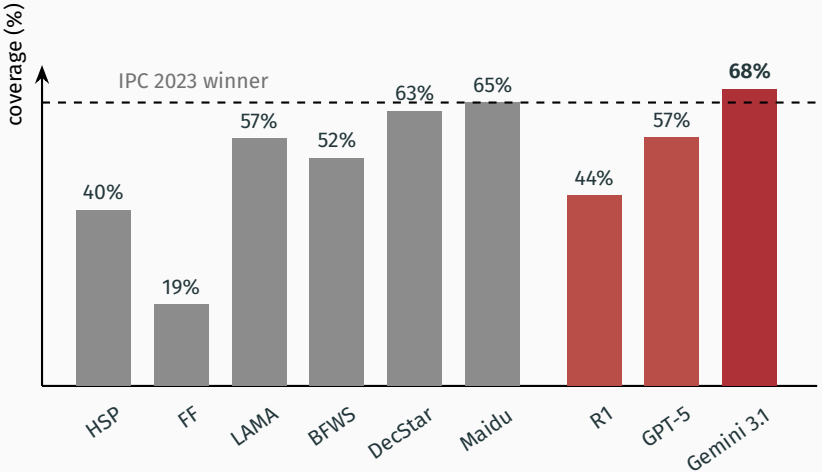
Coverage


coverage (%)
40%
19%
57%
52%
63%
65%
44%
HSP
FF
LAMA
BFWS
DecStar
Maidu
R1
GPT-5
Gemini 3.1

Coverage


coverage (%)
40%
19%
57%
52%
63%
65%
44%
57%
HSP
FF
LAMA
BFWS
DecStar
Maidu
R1
GPT-5
Gemini 3.1

Coverage



An LLM, writing a plan token by token, beats the IPC winner [Corrêa et al. 2025a].

Same tasks; three model generations

GPT-3.5	GPT-4.1	GPT-5
0%		
Nov 2022		

Same tasks; three model generations

GPT-3.5		GPT-4.1	GPT-5
0%	→	6%	
Nov 2022		Apr 2025	

Same tasks; three model generations

GPT-3.5		GPT-4.1		GPT-5
0%	→	6%	→	57%
Nov 2022		Apr 2025		Aug 2025

Same tasks; three model generations

GPT-3.5		GPT-4.1		GPT-5
0%	→	6%	→	57%
Nov 2022		Apr 2025		Aug 2025

Gemini 2.5		Gemini 3.0		Gemini 3.1
45%	→	65%	→	68%
Mar 2025		Nov 2025		2026

Same tasks; three model generations

GPT-3.5		GPT-4.1		GPT-5
0%	→	6%	→	57%
Nov 2022		Apr 2025		Aug 2025

Gemini 2.5		Gemini 3.0		Gemini 3.1
45%	→	65%	→	68%
Mar 2025		Nov 2025		2026

The 2023–2024 critiques measured those models correctly. The models improved.

“It just memorized Blocksworld”

```
(:action pick-up
 :parameters (?b)
 :precondition (and
  (clear ?b) (on-table ?b)
  (arm-empty))
 …)
```

“It just memorized Blocksworld”

```
(:action pick-up
 :parameters (?b)
 :precondition (and
  (clear ?b) (on-table ?b)
  (arm-empty))
 …)
```

```
(:action gxuralqhfigujlqy
 :parameters (?x)
 :precondition (and
  (jmklkrgqahidtiuy ?x)
  (hbzvungehlbssavm ?x)
  (xmulpmadgkjorkin))
 …)
```

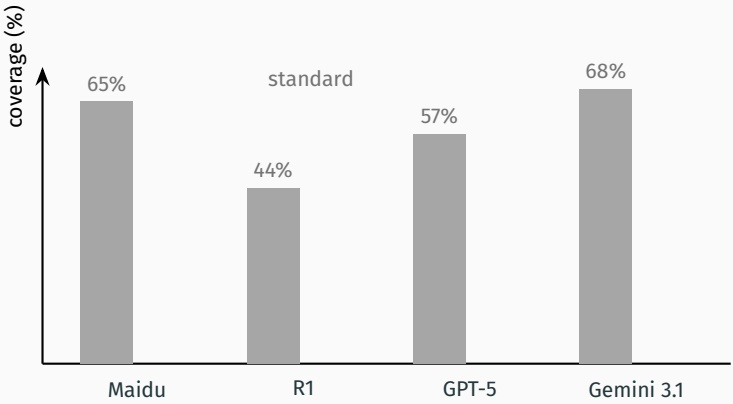
“It just memorized Blocksworld”

```
(:action pick-up
 :parameters (?b)
 :precondition (and
  (clear ?b) (on-table ?b)
  (arm-empty))
 …)
```

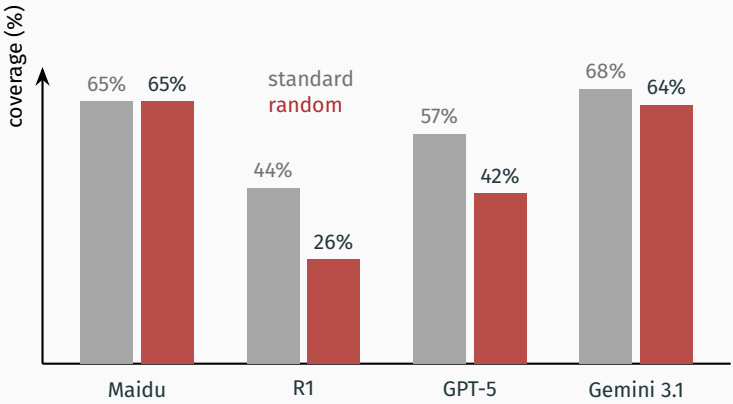
```
(:action gxuralqhfigujlqy
 :parameters (?x)
 :precondition (and
  (jmklkrgqahidtiuy ?x)
  (hbzvungehlbssavm ?x)
  (xmulpmadgkjorkin))
 …)
```

- Randomize the vocabulary: if the models only pattern-match the words, performance should collapse again

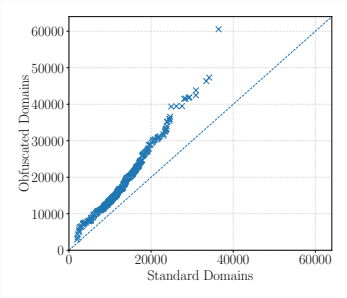
Results on obfuscated tasks



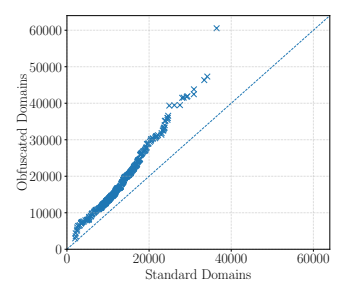
Results on obfuscated tasks



Obfuscation costs tokens



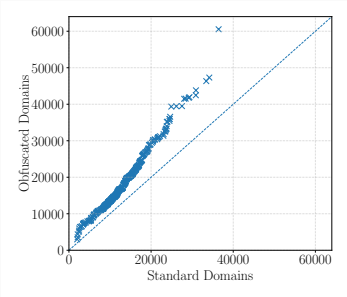
Obfuscation costs tokens



`(on -table ? b )` **vs.** `(j m kl kr g q ahid ti uy ? x )`

- 16 random characters per name: one predicate goes from 5 to 12 tokens — before any reasoning

Obfuscation costs tokens

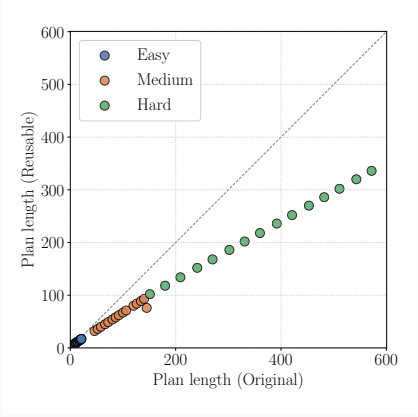


`(on -table ? b )` **vs.** `(j m kl kr g q ahid ti uy ? x )`

- 16 random characters per name: one predicate goes from 5 to 12 tokens — before any reasoning
- And the model reasons much longer, sometimes hitting its thinking-token limit

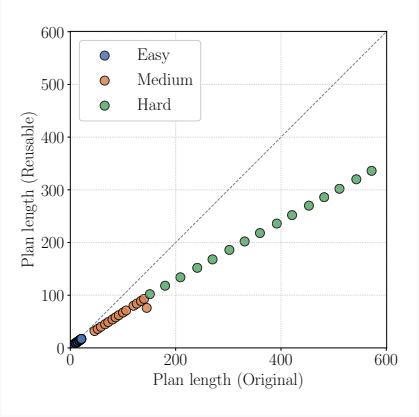
Robustness to modifications

- Simplify a domain so a simpler strategy exists



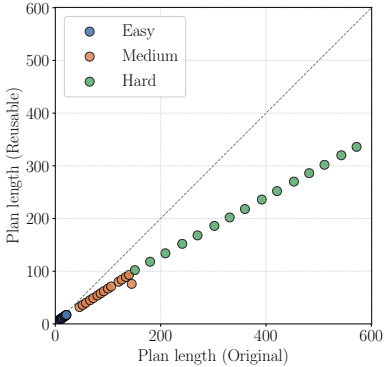
Robustness to modifications

- Simplify a domain so a simpler strategy exists
- The canonical solution stays valid: memorizing it would still solve every task



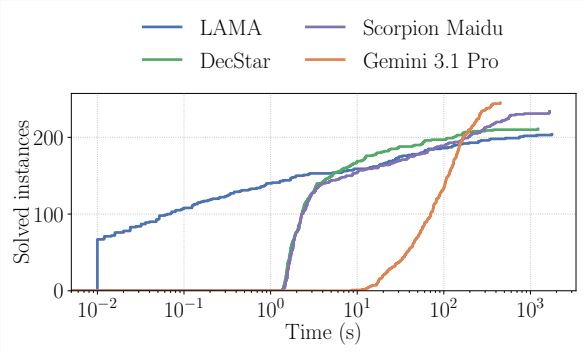
Robustness to modifications

- Simplify a domain so a simpler strategy exists
- The canonical solution stays valid: memorizing it would still solve every task
- The frontier model adapts: shorter plans on 42 of 45 tasks, longer on none
  - mean length 157 $\rightarrow$ 98 actions



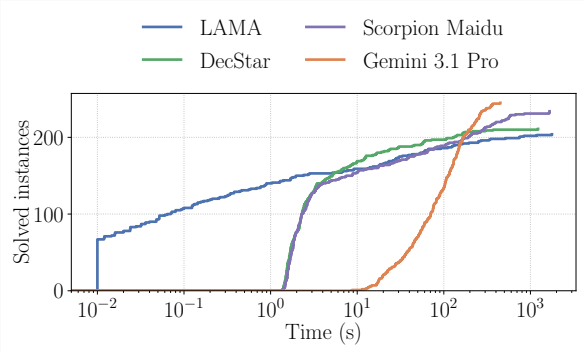
It exploits the simplification instead of replaying the known solution.

The computational cost



- Planners: seconds, one CPU core, 8 GiB
- DeepSeek-R1: 671B parameters, $>$1 000 GiB of GPU memory just to run
- Orders of magnitude more energy and wall-clock time

The computational cost



- Planners: seconds, one CPU core, 8 GiB
- DeepSeek-R1: 671B parameters, $>$1 000 GiB of GPU memory just to run
- Orders of magnitude more energy and wall-clock time

Whether the trade-off is worth it depends on the application. But the capability is new.

Can LLMs plan? At the frontier: yes.

Can LLMs plan? At the frontier: yes.

Closed models, frontier scale,
no guarantees, expensive compute.

What actually changed?

1. **Scale**
   - GPT-4 was already huge and solved $\sim$12%; scale alone was not it
2. **RL on verifiable rewards**
   - models trained to “reason”, not just imitate text

End-to-end planning: still open

- Optimal planning — everything here is satisficing

End-to-end planning: still open

- Optimal planning — everything here is satisficing
- Guarantees — a planner can prove unsolvability; an LLM cannot even promise an answer

End-to-end planning: still open

- Optimal planning — everything here is satisficing
- Guarantees — a planner can prove unsolvability; an LLM cannot even promise an answer
- Cost — seconds on one core vs. GPU clusters

End-to-end planning: still open

- Optimal planning — everything here is satisficing
- Guarantees — a planner can prove unsolvability; an LLM cannot even promise an answer
- Cost — seconds on one core vs. GPU clusters
- Hard domains remain hard
  - Rovers, Sokoban, Floortile still favor planners

They can plan.

They can plan.

So: use them as the planner — or to build one?

Build

Build: generate the planner, not the plan

The LLM can write efficient code.

Build: generate the planner, not the plan

The LLM can write efficient code.

Run it once at build time. The artifact solves every task,
with the planner's guarantees and no LLM at inference.

What can an LLM build?

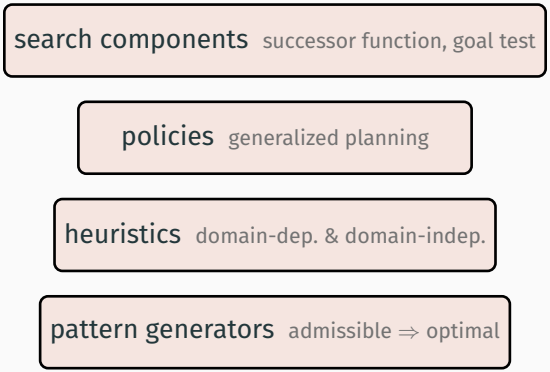
search components successor function, goal test

policies generalized planning

heuristics domain-dep. & domain-indep.

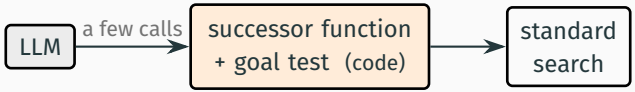
pattern generators admissible $\Rightarrow$ optimal

What can an LLM build?



All plug into a classical planner. All are verifiable code.

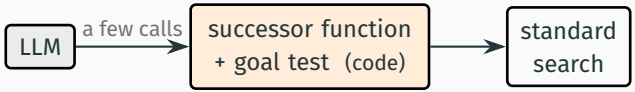
Search components



Thought of Search [Katz et al. 2024]

100% accuracy with a handful of LLM calls — LLM-as-planner needed hundreds of thousands.

Search components

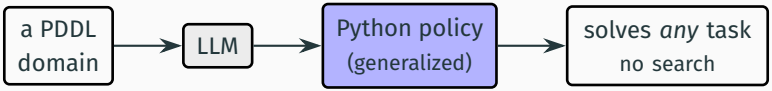


Thought of Search [Katz et al. 2024]

100% accuracy with a handful of LLM calls — LLM-as-planner needed hundreds of thousands.

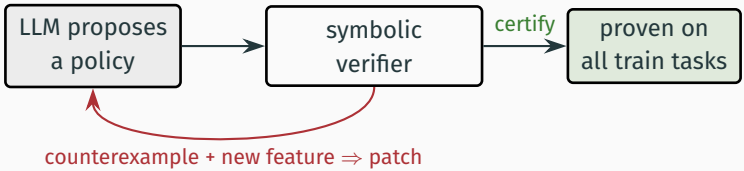
- AutoToS removes the human via unit-test feedback [Cao et al. 2024]

Policies as Python programs



- Sound by construction w.r.t. the PDDL domain — no external verifier [Dillon Ze Chen et al. 2025]
- Beats PDDL planners on more competition tasks; scales to hundreds of objects
- Sometimes plans *better* on obfuscated tasks — structure, not surface names

Policies: prompt, prove, patch

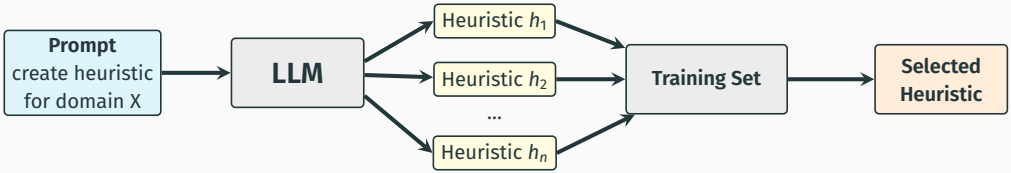


- Policies over a description-logic feature grammar (formal semantics) [Drexler et al. 2026]
- Verifier certifies, or returns structural counterexamples: unhandled open states, infinite loops
- A feature generator discovers missing relational features $\rightsquigarrow$ patch until formally proven
- Related policy synthesis [Silver et al. 2024; Stein et al. 2026]

Prompt, prove and patch: results

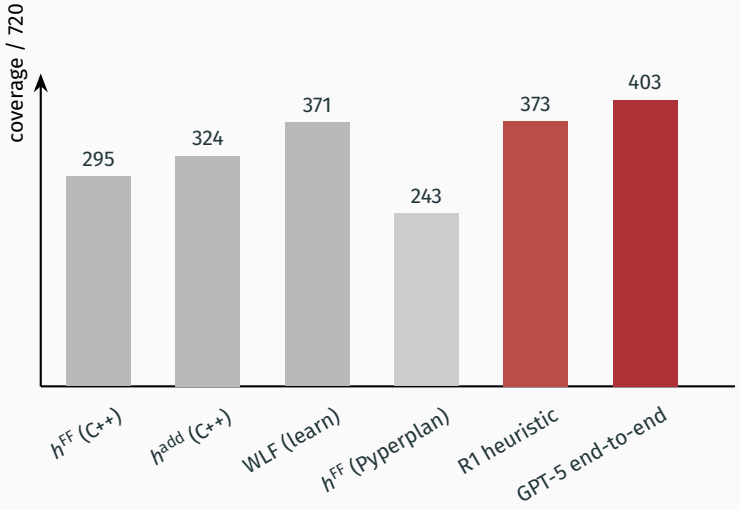
- 15 classical planning domains — several beyond prior program synthesis
- 14 of 15: a policy proven on training tasks generalizes perfectly to held-out tasks (greedy execution)
- The one miss: a feature-language expressivity limit — with a concrete fix identified
- Sound programmatic policies even plan *better* on obfuscated tasks [Dillon Ze Chen et al. 2025]

Heuristics: one per domain, in Python



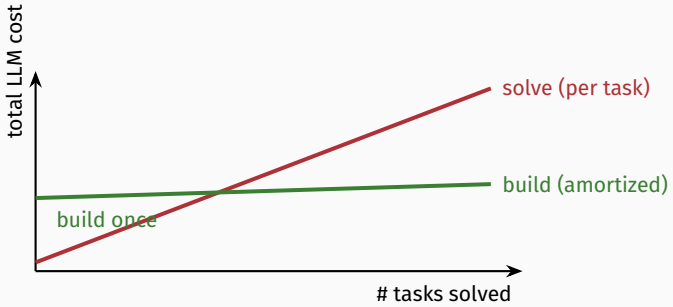
- Runs in Pyperplan, an unoptimized Python planner [Corrêa et al. 2025b]
- The heuristic is the artifact: interpretable, no LLM at solve time
- Also for successor-generator / numeric tasks [Tuisov et al. 2026]

An LLM heuristic beats decades of C++



- R1 heuristic in slow Pyperplan outscores every C++ baseline [Corrêa et al. 2025b]

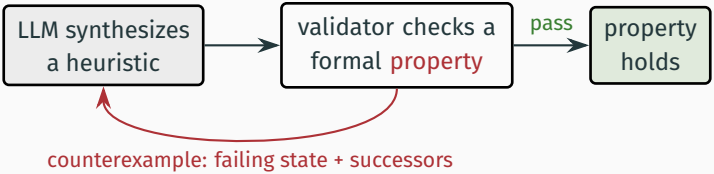
Why build instead of solve? Cost



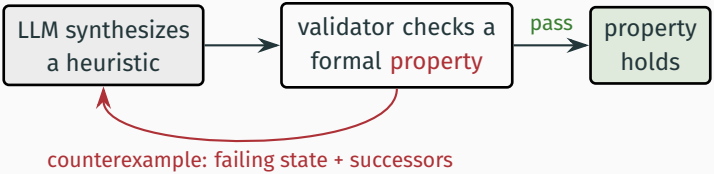
- Solve: each task pays for reasoning tokens, forever
- Build: pay the LLM once; the artifact runs for free
- ...and the artifact is sound and reusable

What about heuristics with certain properties?

Heuristics: counterexamples, not scores

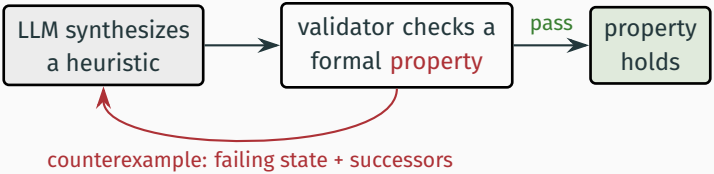


Heuristics: counterexamples, not scores



- Numeric score gives no *why* $\Rightarrow$ generate many candidates

Heuristics: counterexamples, not scores



- Numeric score gives no *why* ⇒ generate many candidates
- A property violation yields a concrete counterexample — stop early, repair the exact failure [Pereira et al. 2026]

Direct heuristics: hill-climbing, no search

- Target the direct property: every state reachable by strictly improving moves has a strictly improving successor

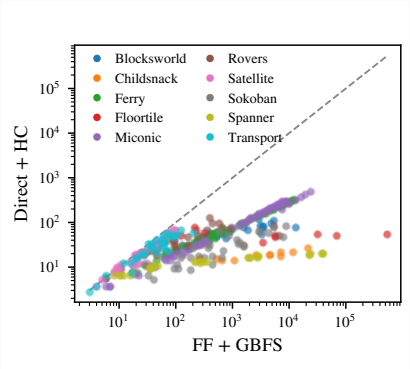
Direct heuristics: hill-climbing, no search

- Target the direct property: every state reachable by strictly improving moves has a strictly improving successor
- $\Rightarrow$ hill-climbing walks to a goal with no search

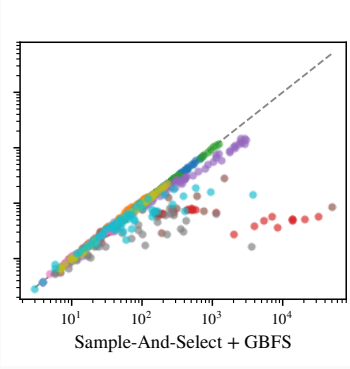
IPC 2023 Learning Track, out-of-distribution test

vs. the best prior generator [Corrêa et al. 2025b]: 7$\times$ fewer programs per domain, orders of magnitude less evaluation compute, solves more tasks without search, and stays direct on virtually all test tasks.

Direct + hill-climbing: fewer expansions



vs. FF



vs. a sampling baseline

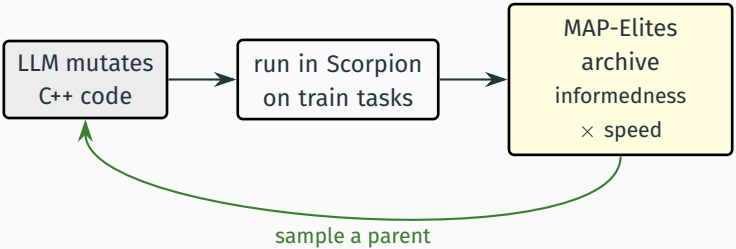
State expansions per task — below the diagonal favors direct + HC.

One heuristic per domain is nice.

One heuristic per domain is nice.

Can an LLM build one for every domain?

Domain-*independent* heuristics by evolution

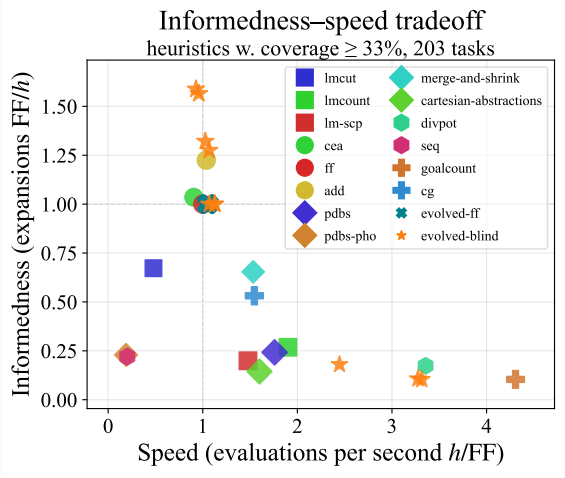


LLM-Evolved Heuristics [Gestrin and Seipp 2026]

First LLM-generated domain-independent heuristics that beat hand-engineered SOTA — as drop-in C++.

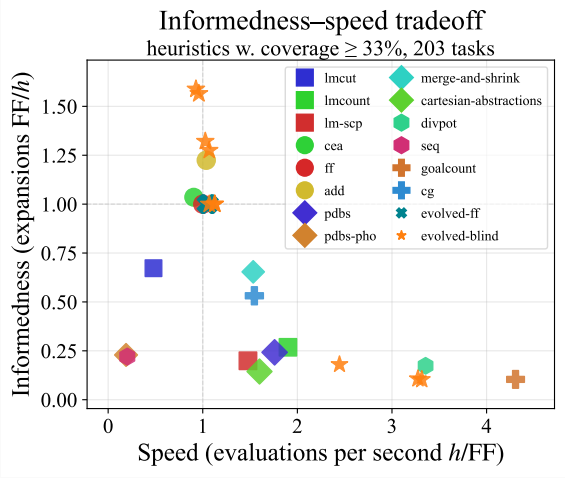
- Program-synthesis lineage: FunSearch, AlphaEvolve [Novikov et al. 2025; Romera-Paredes et al. 2024]

Spanning the informedness–speed frontier



- 19 hand-engineered baselines, one planner, one plane
- Evolved heuristics (stars) span the Pareto frontier
- Best evolved: 368/720 vs. 352 for the strongest baseline

Spanning the informedness–speed frontier



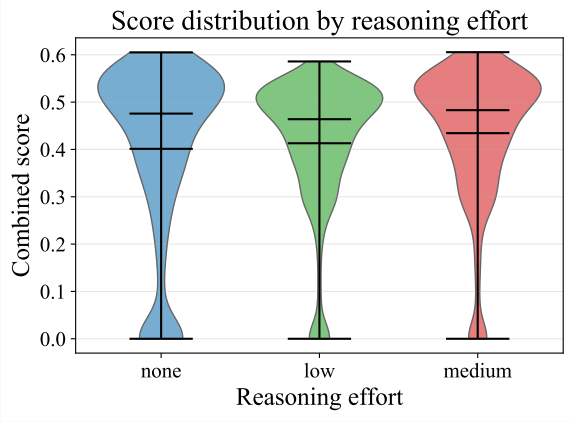
- 19 hand-engineered baselines, one planner, one plane
- Evolved heuristics (stars) span the Pareto frontier
- Best evolved: 368/720 vs. 352 for the strongest baseline

Drop-in C++ on CPU — not a black-box GPU neural heuristic [Dillon Z. Chen et al. 2024; Ståhlberg et al. 2022].

Surprise 1: start from *nothing*

- Seeding from blind beats seeding from FF — diversity beats a strong start
- …even when evolution re-discovers FF from the blind seed

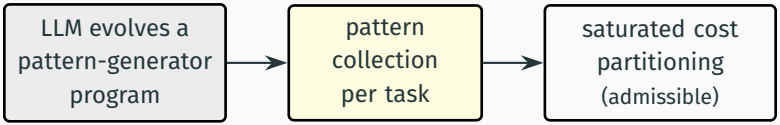
Surprise 2: reasoning effort buys *compilation*



- $\sim$79% of candidates compile on the first attempt
- More reasoning $\Rightarrow$ fewer broken (zero-scoring) heuristics
- But the quality of working heuristics is essentially unchanged

What about optimality?

Optimal planning: admissible by design



LLM-Evolved Pattern Generators [Phung et al. 2026]

First *learned* domain-dependent heuristics admissible by design.

- Don't learn a state $\rightarrow$ value map; learn to construct abstractions
- Per domain: a program that builds a pattern collection for any task
- Combine admissibly via saturated cost partitioning [Seipp et al. 2020] $\rightsquigarrow$ preserves A^* optimality

The thread tying it together: verification

- Hallucination is real — so don't trust, verify

The thread tying it together: verification

- Hallucination is real — so don't trust, verify
- Property-guided synthesis: check a formal property; on violation, return a concrete counterexample to the LLM [Pereira et al. 2026]

The community's edge

Our verifiers and counterexamples are what generic agentic LLM work lacks.

The thread tying it together: verification

- Hallucination is real — so don't trust, verify
- Property-guided synthesis: check a formal property; on violation, return a concrete counterexample to the LLM [Pereira et al. 2026]
- Same idea throughout: Thought of Search, prove-and-patch, property checks

The community's edge

Our verifiers and counterexamples are what generic agentic LLM work lacks.

The LLM builds the planner.

The LLM builds the planner.

The planner keeps the guarantees.

Research

Who is doing the research?

- If LLMs build planners, they also build the research around them
- PhD education board: allowed to use LLMs? $\rightarrow$ should use LLMs
- One idea $\rightarrow$ three paper stories in $\sim$10 prompts
  - code, paper, talk — one prompt each

Who is doing the research?

- If LLMs build planners, they also build the research around them
- PhD education board: allowed to use LLMs? $\rightarrow$ should use LLMs
- One idea $\rightarrow$ three paper stories in $\sim$10 prompts
  - code, paper, talk — one prompt each
- LLMs as critics: of ideas, code, papers
- LLMs as visual explainers of our results [live example]

Who is doing the research?

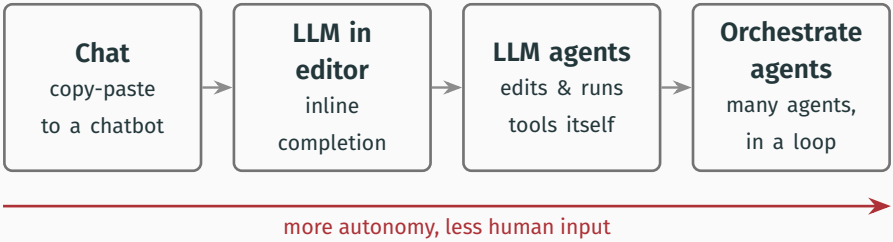
- If LLMs build planners, they also build the research around them
- PhD education board: allowed to use LLMs? $\rightarrow$ should use LLMs
- One idea $\rightarrow$ three paper stories in $\sim$10 prompts
  - code, paper, talk — one prompt each
- LLMs as critics: of ideas, code, papers
- LLMs as visual explainers of our results [live example]
- LLMs as researchers

Who is doing the research?

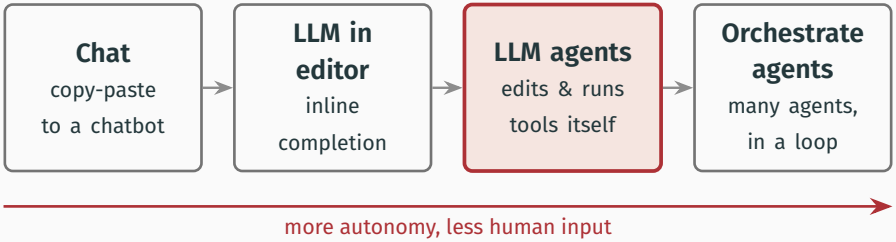
- If LLMs build planners, they also build the research around them
- PhD education board: allowed to use LLMs? $\rightarrow$ should use LLMs
- One idea $\rightarrow$ three paper stories in $\sim$10 prompts
  - code, paper, talk — one prompt each
- LLMs as critics: of ideas, code, papers
- LLMs as visual explainers of our results [live example]
- LLMs as researchers

We are in the ideas business.
Prototyping can be delegated.

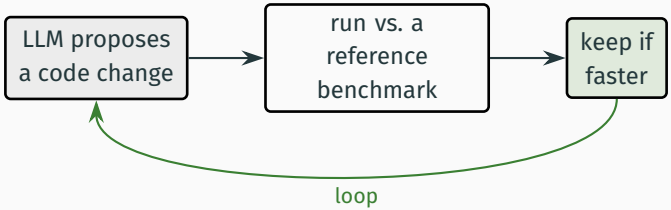
From chat to orchestrating agents



From chat to orchestrating agents



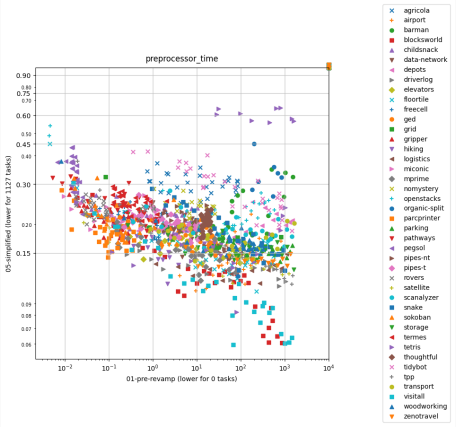
Autoresearch



- Needed a few months ago: a whole skill — prompts, scaffolding, harness
- Now: a single short prompt is enough
- The LLM figures out what to run, what to profile, what to test
- verifiable output is crucial: a test, a metric, a reference

Case study: a faster h^2 preprocessor

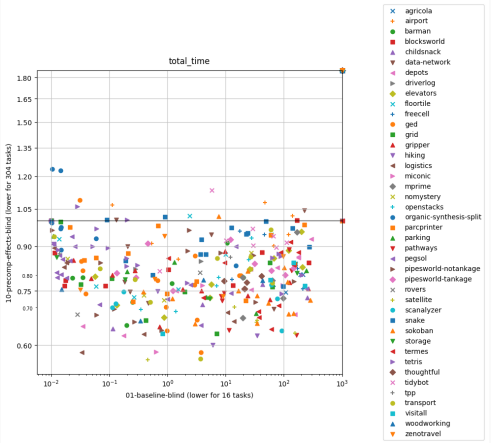
- The h^2 preprocessor [Alcázar and Torralba 2015], part of Scorpion
- Autoresearch with Claude Opus 4.8
- Faster on 1127 tasks, slower on none



Below the line $\Rightarrow$ the revamped preprocessor is faster

Case study: speeding up a *mature* planner

- Blind A* in Fast Downward [Helmert 2006]
- hand-tuned for 20 years by many people
- The agent still cut search time by −17.7%
- Example changes:
  - MurmurHash
  - flat-vector caches
  - skip conditional-effect tests on STRIPS



Per-task total time, optimized vs. baseline — almost all below 1

Case study: an agent modernizes Pyperplan

Clarity (same behavior)

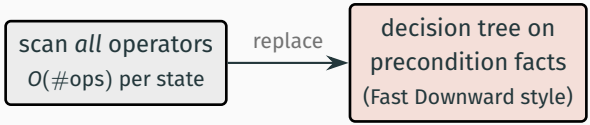
- f-strings, `isinstance`, snake_case
- fixed long-standing bugs: broken `lm_cut`, crash on fractional costs
- tests that pin the behavior

Speed (same results)

- O(1) statics & init-state lookups in grounding
- CNF writing $O(n^2) \rightarrow O(n)$
- lazy A* heuristic eval $\Rightarrow \sim$50% fewer evaluations

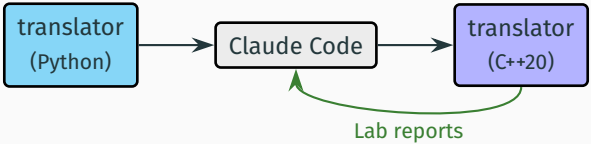
Cleaner *and* faster — verified by tests

Case study: an agent adds a successor generator to Pyperplan



- `get_applicable_operators`: 11 s → 3 s (BrFS, elevators/task01)

Case study: an agent rewrites a planner component

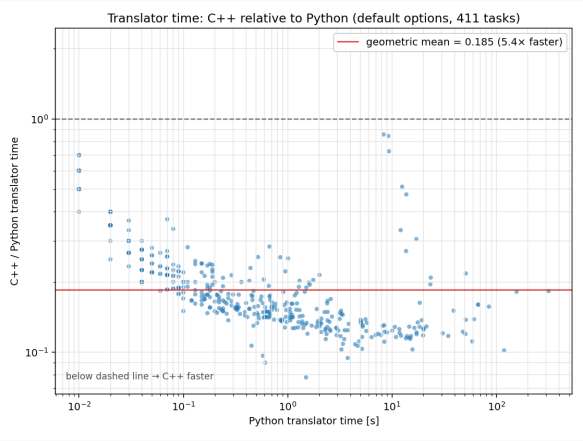


Porting Fast Downward's translator [Corrêa 2026]

PDDL $\rightarrow$ SAS$^+$ grounding & invariant synthesis, reimplemented from scratch.

- long-standing issue in the Fast Downward tracker
- +12% coverage on hard-to-ground tasks; $\sim 26 + 10$ h of agentic coding

The C++ translator is **5.4**× faster



- Below the dashed line ⇒ C++ wins — on every task
- Available in Scorpion

Wrapping up

Hot takes

- Researchers should let LLMs criticize their work.

Hot takes

- Researchers should let LLMs criticize their work.
- Researchers should let LLMs fix errors in their papers and code.

Hot takes

- Researchers should let LLMs criticize their work.
- Researchers should let LLMs fix errors in their papers and code.
- Researchers should let LLMs optimize their code.

Future work

- ... planning with guarantees, learned by LLMs
- ... synthesize other components: the search algorithms themselves
- ... harder formalisms: numeric, temporal, HTN
- ... in general: let LLMs output algorithms, not solutions

Summary

- LLMs can plan now, but as *the planner* they stay costly and unsound
- So have them build the planner: heuristics, search algorithms, policies
- Plain code with the planner's guarantees and no LLM at inference

References i

- Alcázar, Vidal and Álvaro Torralba (2015). “A Reminder about the Importance of Computing and Exploiting Invariants in Planning”. In: *Proc. ICAPS 2015*, pp. 2–6.
- Cao, Daniel, Michael Katz, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi (2024). *AutoToS: Automating Thought of Search*. arXiv:2408.11326 [cs.AI].
- Chen, Dillon Z., Sylvie Thiébaux, and Felipe Trevizan (2024). “Learning Domain-Independent Heuristics for Grounded and Lifted Planning”. In: *Proc. AAAI 2024*, pp. 20078–20086.
- Chen, Dillon Ze, Johannes Zenn, Tristan Cinquin, and Sheila A. McIlraith (2025). “Language Models for Generalised PDDL Planning: Synthesising Sound and Programmatic Policies”. In: *ICAPS Workshop on Planning in the Era of LLMs*.
- Corrêa, Augusto B., André G. Pereira, and Jendrik Seipp (2025a). *Frontier Large Language Models Rival State-of-the-Art Planners*. arXiv:2511.09378 [cs.AI].
- Corrêa, Augusto B., André Grahl Pereira, and Jendrik Seipp (2025b). “Classical Planning with LLM-Generated Heuristics: Challenging the State of the Art with Python Code”. In: *Proc. NeurIPS 2025*, pp. 42070–42108.

References ii

- Drexler, Dominik, Markus Fritzsche, Farid Musayev, and Simon Ståhlberg (2026). “Prompt, Prove, Patch: The Neuro-Symbolic Loop for General Policy Synthesis”. In: *ICAPS Workshop on Reliability In Planning and Learning (RIPL)*.
- Gestrin, Elliot, Marco Kuhlmann, and Jendrik Seipp (2024). “NL2Plan: Robust LLM-Driven Planning from Minimal Text Descriptions”. In: *ICAPS Workshop on Human-Aware and Explainable Planning*.
- Gestrin, Elliot and Jendrik Seipp (2026). “LLM-Evolved Domain-Independent Heuristics for Symbolic AI Planning”. In: *ICAPS Workshop on Planning in the Era of LLMs*.
- Helmert, Malte (2006). “The Fast Downward Planning System”. In: *JAIR* 26, pp. 191–246.
- Katz, Michael, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi (2024). “Thought of Search: Planning with Language Models Through The Lens of Efficiency”. In: *Proc. NeurIPS 2024*, pp. 138491–138568.
- Liu, Bo, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone (2023). *LLM+P: Empowering Large Language Models with Optimal Planning Proficiency*. arXiv:2304.11477.
- Novikov, Alexander, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog (2025). *AlphaEvolve: A coding agent for scientific and algorithmic discovery*. arXiv:2506.13131.

References iii

- Oswald, James, Kavitha Srinivas, Harsha Kokel, Junkyu Lee, Michael Katz, and Shirin Sohrabi (2024). "Large Language Models as Planning Domain Generators". In: *Proc. ICAPS 2024*, pp. 423–431.
- Pereira, André G., Augusto B. Corrêa, and Jendrik Seipp (2026). *Property-Guided LLM Program Synthesis for Planning*. arXiv:2605.16142 [cs.AI].
- Phung, Windy, Dominik Drexler, Arnaud Lequen, and Jendrik Seipp (2026). "LLM-Evolved Pattern Generators for Optimal Classical Planning". In: *ICAPS Workshop on Reliability In Planning and Learning (RIPL)*.
- Romera-Paredes, Bernardino, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi (2024). "Mathematical discoveries from program search with large language models". In: *Nature* 625, pp. 468–475.
- Seipp, Jendrik, Thomas Keller, and Malte Helmert (2020). "Saturated Cost Partitioning for Optimal Classical Planning". In: *JAIR* 67, pp. 129–167.
- Silver, Tom, Soham Dan, Kavitha Srinivas, Josh Tenenbaum, Leslie Pack Kaelbling, and Michael Katz (2024). "Generalized Planning in PDDL Domains with Pretrained Large Language Models". In: *Proc. AAAI 2024*, pp. 20256–20264.

References iv

- Ståhlberg, Simon, Blai Bonet, and Hector Geffner (2022). “Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits”. In: *Proc. ICAPS 2022*, pp. 629–637.
- Stein, Katharina, Nils Hodel, Daniel Fišer, Jörg Hoffmann, Michael Katz, and Alexander Koller (2026). “Improved Generalized Planning with LLMs through Strategy Refinement and Reflection”. In: *Proc. ICAPS 2026*.
- Tantakoun, Marcus, Christian Muise, and Xiaodan Zhu (2025). “LLMs as Planning Formalizers: A Survey for Leveraging Large Language Models to Construct Automated Planning Models”. In: *Findings of the Association for Computational Linguistics: ACL 2025*. Association for Computational Linguistics, pp. 25167–25188.
- Tuisov, Alexander, Yonatan Vernik, and Alexander Shleyfman (2026). “Successor-Generator Planning with LLM-Generated Heuristics”. In: *Proc. ICAPS 2026*.
- Valmeekam, Karthik, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati (2023). “PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change”. In: *Proc. NeurIPS 2023*, pp. 38975–38987.

Attributions

- Truck icon created by Freepik - Flaticon
- Package icon created by Hilmy Abiyyu A. - Flaticon