LLM-Evolved Pattern Generators for Optimal Classical Planning

Windy Phung, Dominik Drexler, Arnaud Lequen and Jendrik Seipp

Linköping University

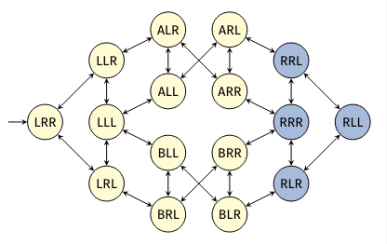
RIPL, 29.06.2026

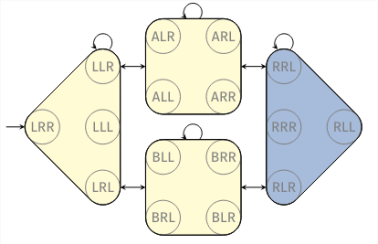
Motivation

- Optimal planning: A^* + admissible heuristic.
- Strongest admissible heuristics: domain-*independent*.
- Learned domain-dependent heuristics: only satisficing.

Idea: learn per-dosain abstractions $\Rightarrow$ **optimality by design**.

Background: Patterns





Each **pattern** $\rightarrow$ an admissible estimate $\Rightarrow$ combine a *collection*.

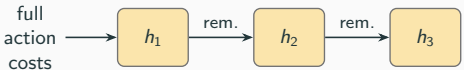
Background: Saturated Cost Partitioning

Combine several PDBs, stay admissible:

- **Maximize:** admissible, but wasteful.
- **Sum:** informative, but inadmissible.

Saturated cost partitioning (SCP): each pattern keeps the cost it needs, passes the rest on.

$\Rightarrow \sum_i h_i$ admissible



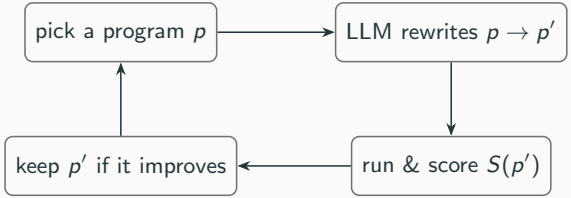
OpenEvolve: Improving Programs with an LLM

A program p generates a pattern collection.

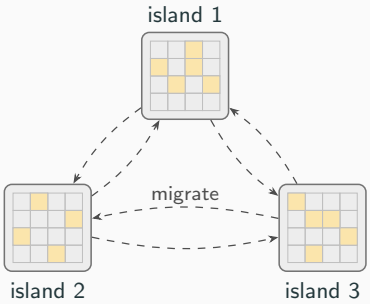
Its score $S(p)$ measures how well that collection performs on the training set.

Repeatedly improve a pool of programs.

The LLM proposes; the score decides.



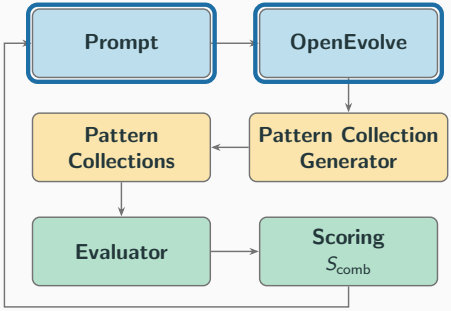
OpenEvolve: Keeping the Search Diverse



- Archive binned by **feature dimensions** $\Rightarrow$ *behaviorally different* survivors.
- Parallel **islands** + occasional **migration**.
- **Exploration** (cover the feature space) *vs.* **exploitation** (refine the best).

What Goes in the Prompt

- PDDL domain.
- Required function signature.
- Easiest / median / hardest task info.
- Naive baseline: one goal atom per pattern.

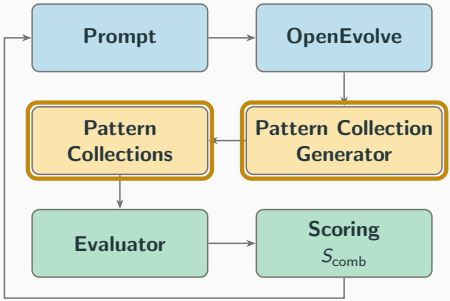


The Generator: Input & Output

Input: structured view of the task

- static & fluent atoms
- initial-state & goal atoms

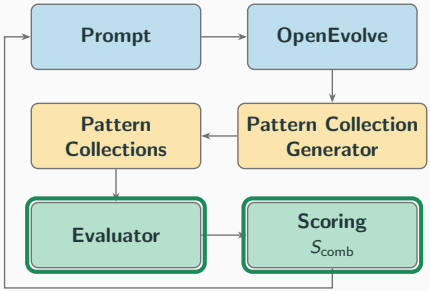
Output: list of `Patterns` (sets of atoms)



Evaluation: Scoring

Run $A^* + \text{SCP}$ per task. Per-task components:

- $S_{\text{valid}} \in \{0, 1\}$: validity check.
- S_{exp}: expansions until last jump.
- S_{time}: search time.



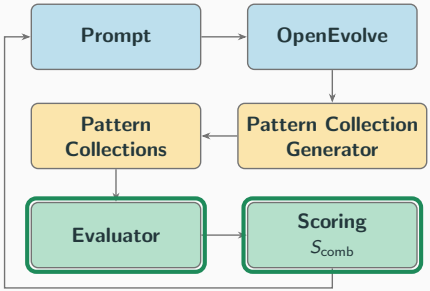
Evaluation: Scoring

Run $\mathrm{A}^* + \mathrm{SCP}$ per task. Per-task components:

- $S_{\text{valid}} \in \{0, 1\}$: validity check.
- S_{exp}: expansions until last jump.
- S_{time}: search time.

Per-task score:

$$S_{\text{problem}} = S_{\text{valid}} \left(1 + w_{\text{exp}} S_{\text{exp}} + w_{\text{time}} S_{\text{time}}\right)$$



Evaluation: Scoring

Run $A^* + \mathrm{SCP}$ per task. Per-task components:

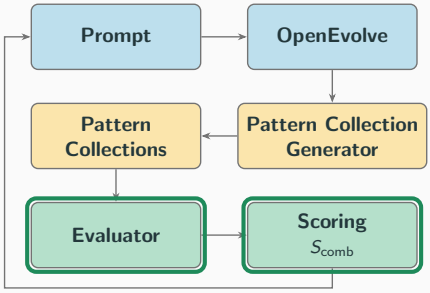
- $S_{\text{valid}} \in \{0, 1\}$: validity check.
- S_{exp}: expansions until last jump.
- S_{time}: search time.

Per-task score:

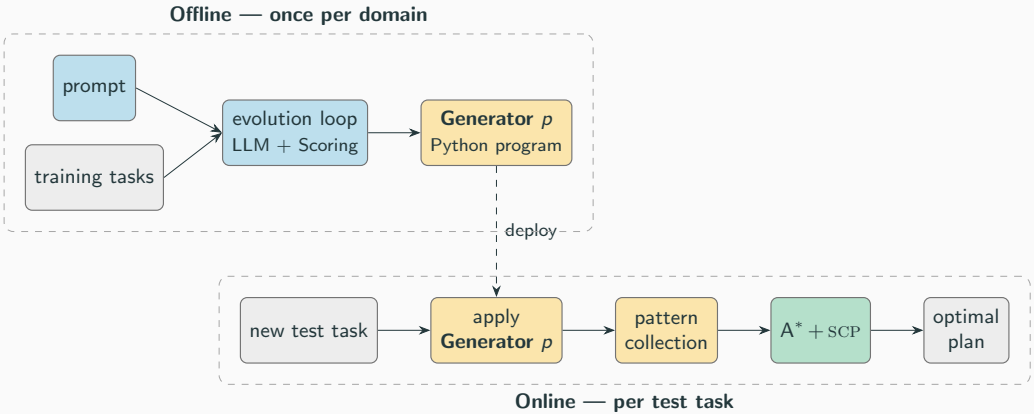
$$S_{\text{problem}} = S_{\text{valid}} \left(1 + w_{\text{exp}} S_{\text{exp}} + w_{\text{time}} S_{\text{time}}\right)$$

Average over the domain:

$$S_{\text{comb}} = \text{average}(S_{\text{problem}}(t))$$



Evolve Once, Solve Any Size



- A **program**, not a fixed collection → handles every task size.
- Effort to find pattern collection paid **offline**; test-time generation in milliseconds.

Setup

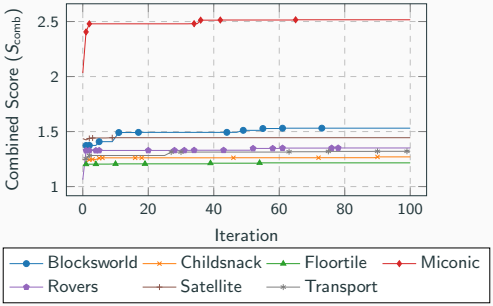
- **Train:** IPC 2023 Learning Benchmark.
- **Test:** Autoscale Benchmark.
- 7 domains; evolution: 3 islands, 100 iter., Kimi K2.5.

Baselines:

- $\text{SYS}_1/\text{SYS}_2/\text{SYS}_3$: patterns up to size n;
- SYS-SCP: kept if useful to SCP.
- RAND: our patterns, non-goal atoms randomized.

The Evolutionary Process

- Most gains: **first five iterations**.
- Refinement continues to iter. 100.
- Late breakthroughs: Childsnack, Transport.
- Gains: +0.009 to +0.484.
- Cost per domain: avg. 14h, 20$, 12% failed.



An Evolved Generator Is Interpretable (Transport)

One large pattern per package:

- $\texttt{at}(p, \ell_0)$ — start location
- $\texttt{at}(p, \ell_g)$ — goal location
- $\texttt{in}(p, \texttt{truck}_i)$ — each in-vehicle state

$2 + \#\textit{vehicles atoms (up to 20)}$

Plus three more pattern families:

- **per vehicle:** $\texttt{at}(v, \ell) + \texttt{capacity}(v)$ — *truck's location & load*
- **vehicle cargo:** all $\texttt{in}(p, v) + \texttt{capacity}(v)$
- **critical locations:** every $\texttt{at}(\cdot, \ell)$ at a package start/goal — *one pattern per hot-spot, any object parked there*

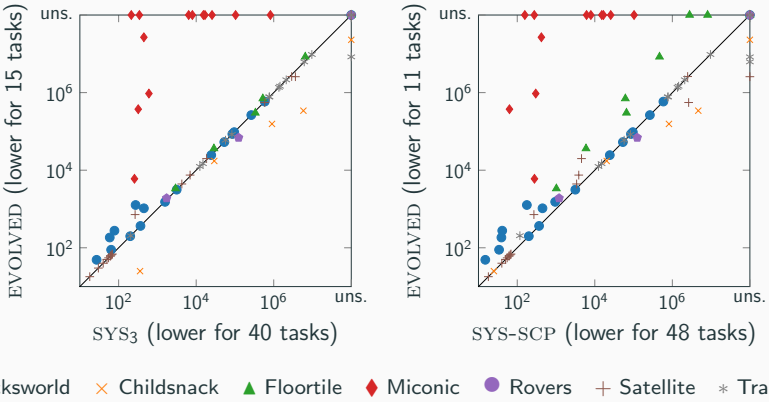
```
def generate_pattern_collection(
        task_info):
    patterns = []
    # one pattern per package
    pkgs = {a.binding[0] for a in
            task_info.fluent_goal_atoms
            if a.predicate.name == 'at'
    }
    for pkg in pkgs:
        # start/goal/in-vehicle
        atoms  = init_at(pkg)
        atoms += goal_at(pkg)
        atoms += in_vehicle_atoms(pkg)
        if 0 < len(atoms) <= 20:
            patterns.append(Pattern(
    atoms))
    # ... cargo, location clusters ...
    return patterns
```

Results: Coverage

	Baselines				Ours	
Domain	SYS_1	SYS_2	SYS_3	SYS-SCP	RAND	EVOLVED
Blocksworld (30)	12	**16**	**16**	**16**	12	**16**
Childsnack (30)	**4**	**4**	**4**	**4**	**4**	**4**
Floortile (30)	4	5	5	**7**	5	5
Miconic (30)	3	6	**20**	19	3	6
Rovers (30)	**2**	**2**	**2**	**2**	**2**	**2**
Satellite (30)	5	15	**16**	15	5	**16**
Transport (30)	10	10	12	11	10	**14**
Sum (210)	40	58	**75**	74	41	63

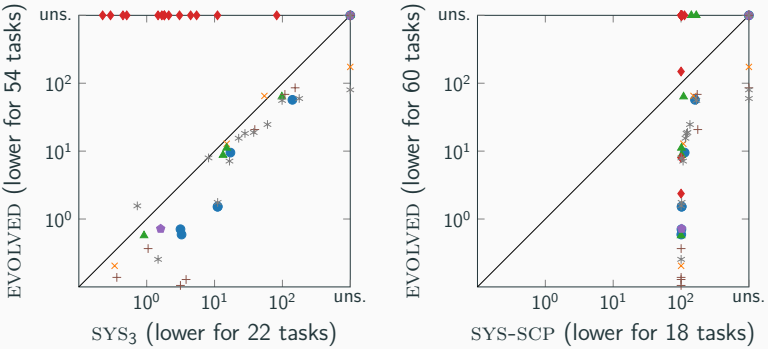
- Matches/beats best baseline in **5 of 7**; uniquely best on Transport.
- Aggregate gap is entirely Miconic.
- RAND never beats EVOLVED $\Rightarrow$ structure matters.

Weaker Per-State Guidance — the Price



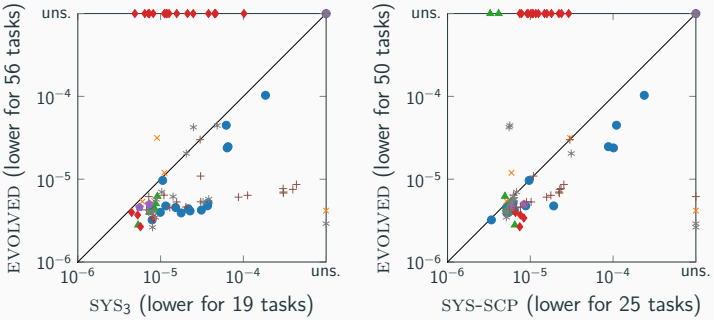
Expansions before the last f-layer. Above the diagonal means EVOLVED expands more, so its per-state guidance is weaker.

. . . Yet Faster Overall



Total search time. Now most points fall below the diagonal: EVOLVED is faster overall, clearing the time limit on more tasks.

Why: Fewer, Larger Patterns → Cheaper per State



Time per state evaluation. EVOLVED is cheaper per state because its collection is far smaller — across the 40 shared tasks, 1 230 patterns vs. 5 998 SYS-SCP and 49 157 SYS_3.

Limitations & Future Work

Limitations

- Weaker per-state guidance; loses where small systematic patterns dominate.
- Training score vs. coverage mismatch.
- May lean on the LLM's prior exposure to these benchmarks.

Future work

- Richer domain info: causal graph, DTG, SAS^+/FDR task.
- Larger training sets.
- Refine the scoring function.
- Obfuscate domain.

Takeaways

- **Admissible-by-construction**, learned, domain-dependent heuristics via **LLM program synthesis**.
- Pattern generators evolved with **OpenEvolve** (evolutionary search over programs).
- Interpretable **Python** pattern-database generators.
- Match/beat best baseline in 5 of 7 domains.

Thank you!