

From Policy Learning to Autonomous Algorithm Design: AI for Network Optimization
Universidad Carlos III de Madrid

Automatic heuristic generation II: reflection, co-evolution, AlphaEvolve and systems use cases

2026-09-22

Jose A. Ayala Romero · NEC Laboratories Europe

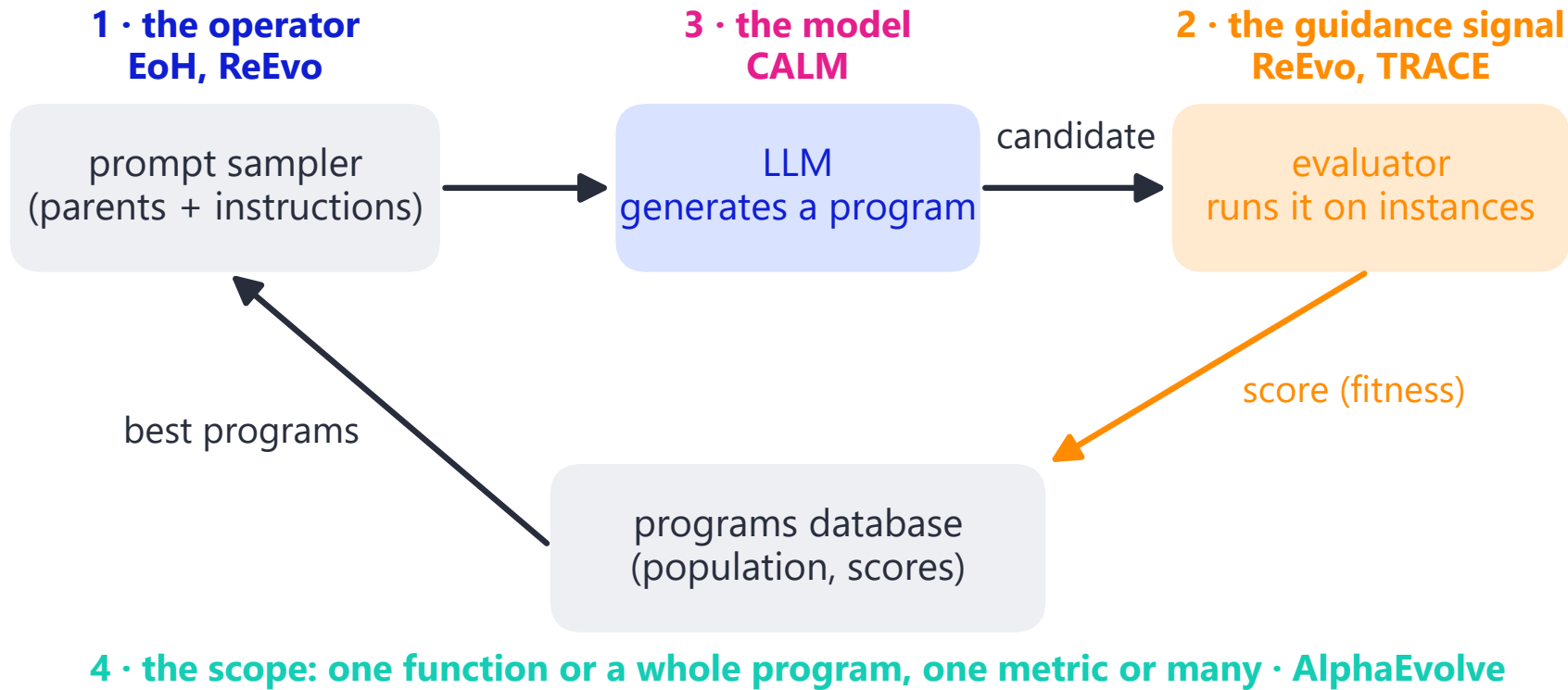
Agenda

1. Recap: the loop and its levers
2. ReEvo: reflective evolution
3. CALM: co-evolving the model
4. AlphaEvolve
5. Beyond scalar fitness: TRACE
6. Seminar wrap-up

Recap: the loop and its levers

Every method of today changes one part of the same loop

The loop, and the four places you can improve it



The methods of today at a glance

Method	What it changes	LLM	Samples	Evaluations
ReEvo	reflections as verbal gradients in the prompt	frozen API model	hundreds	about 100
CALM	fine-tunes the model from the search data	local 7B, trained	thousands	thousands
AlphaEvolve	whole files, diffs, many metrics, rich context	frontier ensemble	thousands	hours each
TRACE	logs distilled into the prompt by agents	reasoner + light model	thousands	trace-driven

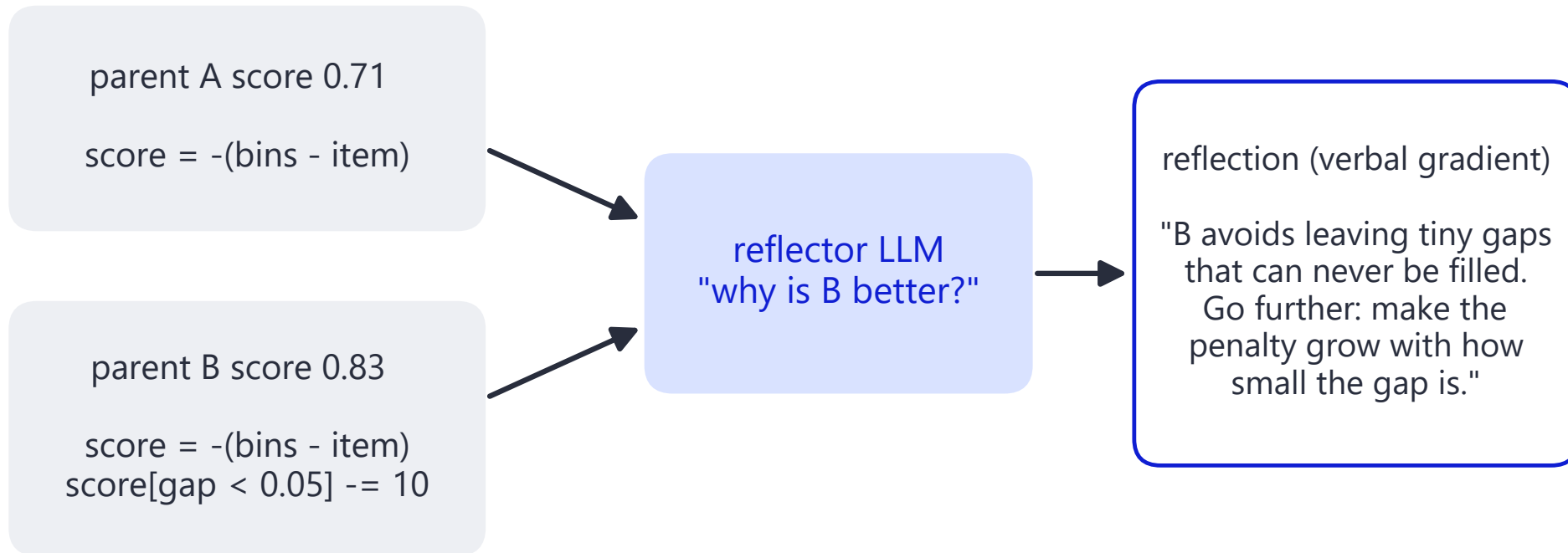
Agenda

1. Recap: the loop and its levers
2. **ReEvo: reflective evolution**
3. CALM: co-evolving the model
4. AlphaEvolve
5. Beyond scalar fitness: TRACE
6. Seminar wrap-up

ReEvo: reflective evolution

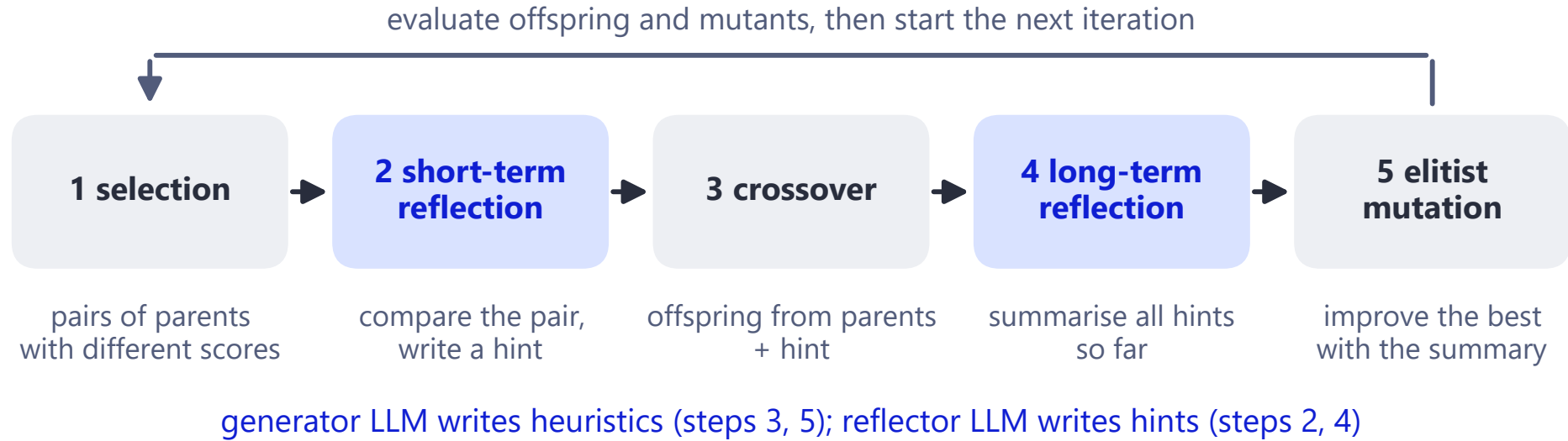
Evolution guided by verbal gradients

A verbal gradient



the hint is a direction of improvement in a space where no numerical gradient exists

The ReEvo iteration



- Two LLM roles: a **generator** writes heuristics, a **reflector** writes hints
- Individuals are code snippets constrained only by the function signature
- Both offspring and mutants are evaluated; the population is initialised from the task description, optionally with seed heuristics

Short-term versus long-term reflection

Short-term

- one pair of parents, their relative performance
- a comparative analysis: what the better one does differently
- feeds the **crossover** prompt for that pair

Long-term

- the reflector summarises all short-term reflections gathered so far
- accumulated expertise about the problem, kept short so it fits the context window
- feeds the **elitist mutation** of the current best heuristic

In agent terms: short-term reflections interpret the feedback of one interaction; long-term ones distil experience into memory.

Example: black-box TSP

Setting: the LLM is not told what the objective is, only the function signature and two heuristics with their scores.

Short-term reflection (paraphrased): "The better heuristic penalises long edges relative to the average distance from the current node; the objective seems to reward short tours. Combine the distance ratio with a spread statistic."

Result: the offspring uses the mean and standard deviation of distances to unvisited nodes and beats both parents.

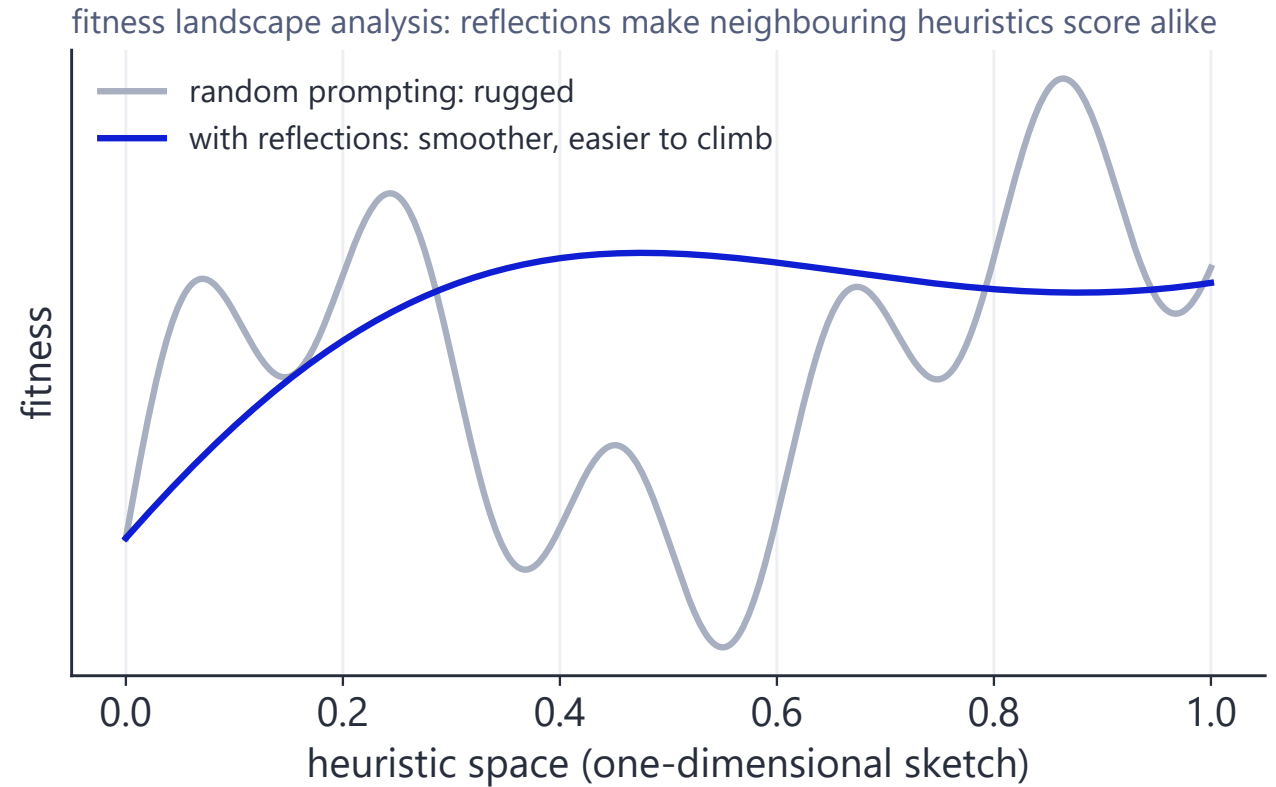
- The reflector inferred the objective from two scored programs
- With a black-box prompt, plain LLM generation could not do this ([Ye et al. 2024](#))

Where ReEvo-generated heuristics were plugged in

Host algorithm	What was evolved	Outcome
Guided local search for TSP	the penalty function	beats learned and expert local-search variants
Ant colony optimisation, five problems	the heuristic measure	beats expert and neural heuristics, robust to size
Genetic algorithm for chip design	crossover and mutation operators	beats the expert GA and a neural solver
Constructive TSP	the next-node rule	beats a classic genetic-programming hyper-heuristic
Neural solvers (POMO, LEHD)	attention reshaping	better generalisation, no retraining

Sample efficiency and the fitness landscape

- Designed for a budget of about **100 heuristic evaluations**: the regime of real systems, where one evaluation is a simulation run
- Fitness-landscape analysis: reflections make neighbouring heuristics score alike, so the search climbs instead of jumping
- LLM calls cost a fraction of a cent and a second; evaluations cost minutes, so count evaluations, not queries



Generated heuristics versus neural solvers

Generated heuristics

- rule-based, interpretable
- robust to distribution shift, limited data, adversarial inputs
- CPU only at deployment
- may underfit very complex environments

Neural combinatorial optimisation

- expressive black-box policies
- millions of samples, days of training, GPUs at inference
- brittle out of distribution and across problem sizes
- architectures and training need expert tuning

Reflection buys sample efficiency, which is what matters when every evaluation is a simulation run

Exercise 1: be the reflector

Two scheduling heuristics for the task-to-node problem, with their costs on the same traces:

- **A** (cost 812): send each task to the node with the shortest queue
- **B** (cost 640): send each task to the shortest queue, except tasks above 80% of the maximum size, which go to node 1

Write the short-term reflection: why is B better, and what should the offspring try?

- B is better because node 1 handles heavy tasks faster; queue length alone ignores node speed
- Hint: replace the fixed 80% threshold by a rule that estimates the expected finishing time per node from task size and node speed, and keep the queue term as a tie-breaker

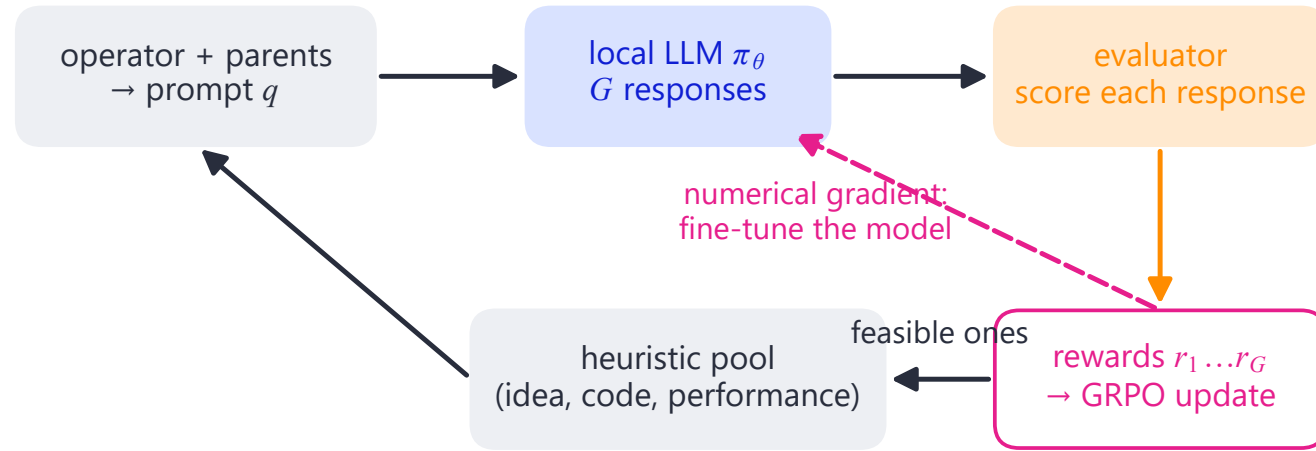
Agenda

1. Recap: the loop and its levers
2. ReEvo: reflective evolution
3. **CALM: co-evolving the model**
4. AlphaEvolve
5. Beyond scalar fitness: TRACE
6. Seminar wrap-up

CALM: co-evolving the model

Verbal and numerical gradients from the same evaluations

Verbal and numerical gradients



verbal guidance through prompts, numerical guidance through the weights, from the same evaluations

- Every method so far keeps the LLM frozen and steers it through prompts (verbal guidance)
- Yet every evaluated heuristic is a **(prompt, response, performance)** triple: training data
- CALM uses those triples to fine-tune the model with reinforcement learning while the search runs (numerical guidance) ([Huang et al. 2025](#))

GRPO in one slide

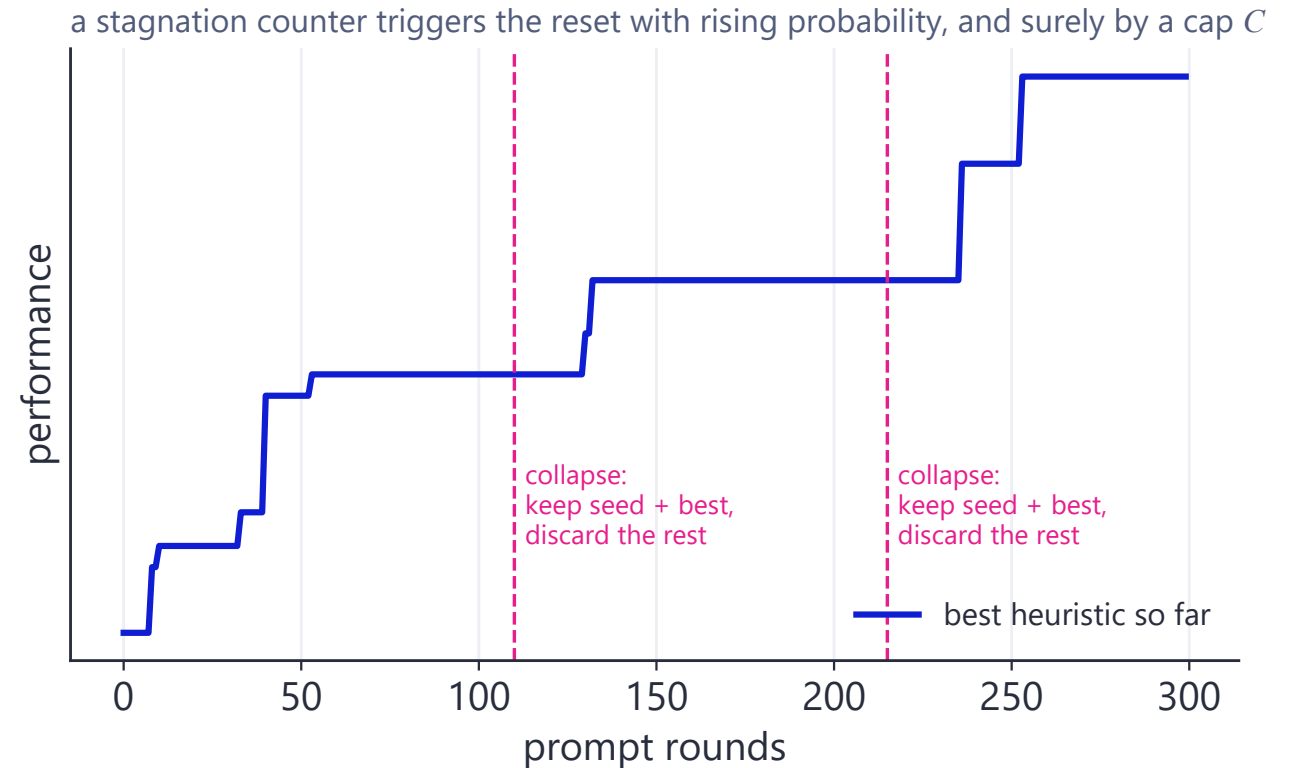
- For each prompt q , sample a **group** of G responses from the current model and score them: $r_1, \dots, r_G$
- Advantage of response i : its reward standardised within the group, $(r_i - \bar{r})/\text{std}(r)$; no value network is needed
- Update the model to raise the probability of above-average responses, with a clipped ratio and a KL penalty towards a reference model for stability
- The same algorithm that trained DeepSeek-R1; it fits on one 24 GB GPU with a 7B model in 4-bit precision ([Shao et al. 2024](#))

Fine-grained operators

Operator	What it asks the LLM	Why fine-grained
Injection	add one new component to a heuristic; describe it; avoid components already registered	small edits make credit assignment by GRPO easier
Replacement	rewrite one component: its parameters, an instance-independent rule into an adaptive one, or equal credit into differentiated credit	targeted, keeps the structure
Diversity-aware crossover	one parent by performance, the other by performance or by novelty of its idea	avoids inbreeding
Simplification	a more concise version	counterbalances growth; its removal hurt most in ablations

The collapse mechanism

- Good parents produce good children: a self-reinforcing loop that also causes **inbreeding** and premature convergence
- A stagnation counter c_n counts rounds without a new global best
- Collapse with probability growing in c_n , and surely at a cap C : keep only the seed and the current best, restart the population from them
- Too eager a collapse hurts; the paper gives a formula for the expected rounds before it fires



The reward: correctness first, then improvement over the parent

feasible, better than the best parent	≥ 1 , growing with the gain
feasible, same as a parent	small positive
feasible, worse than the best parent	scaled negative
uses randomness	-0.75
runtime error or time-out	-0.85
wrong function format	-0.9
no code block	-0.95
no idea given	-1.0



correctness first:
every feasible response
beats every infeasible one

then credit relative to
the parents in the prompt,
not absolute performance

Results

- A local 7B model with 4-bit weights on a single 24 GB GPU discovers heuristics that beat API-based state-of-the-art methods on bin packing, routing and orienteering problems
- The operators alone, without fine-tuning and with an API model, already match the best prompt-only methods
- In the ablations, removing the reinforcement-learning update causes the largest loss; performance-based rewards and eager collapses both hurt
- The training curve lags early, then overtakes as the model adapts

What co-evolution costs and buys

Costs

- a trainable local model and a GPU
- reward engineering and RL hyperparameters
- longer wall-clock per round than an API call

Buys

- independence from commercial APIs
- a model that internalises what works *for this problem*
- data that would otherwise be thrown away after evaluation

The evolutionary loop is also a data generator

Agenda

1. Recap: the loop and its levers
2. ReEvo: reflective evolution
3. CALM: co-evolving the model
4. **AlphaEvolve**
5. Beyond scalar fitness: TRACE
6. Seminar wrap-up

AlphaEvolve

Evolving whole programs with frontier models and several metrics

From one function to a whole program

FunSearch	AlphaEvolve
evolves a single function	evolves an entire code file
10 to 20 lines	up to hundreds of lines
Python only	any language
needs fast evaluation (minutes on one CPU)	can evaluate for hours, in parallel, on accelerators
millions of LLM samples	thousands suffice
small models; larger ones did not help	benefits from state-of-the-art models
minimal context: previous solutions only	rich context and feedback in the prompt
one metric	several metrics at once

Task specification: mark what may change

```
1 # EVOLVE-BLOCK-START
2 def schedule_score(job, machine):
3     """Priority of placing `job` on `machine`; higher wins."""
4     return 0.0 # trivial initial version
5 # EVOLVE-BLOCK-END
6
7 def evaluate(trace) -> dict[str, float]:
8     result = simulate(trace, schedule_score)
9     return {"utilisation": result.util,
10            "stranded_cpu": -result.stranded,
11            "runtime": -result.seconds}
```

- Blocks between the markers are evolved; everything else is the fixed skeleton that calls them
- The initial content must be complete but can be trivial
- `evaluate` returns a **dictionary of metrics**, all to be maximised

Rich prompts

system instructions	act as an expert developer; propose diffs; keep changes consistent
explicit context	problem description, equations, relevant papers, code of the evaluator
prior programs with scores	several previously found programs sampled from the database
current program + rendered results	the program to improve, its outputs and all its metrics
stochastic formatting	template variants sampled per prompt for diversity
meta-prompt evolution	instructions the LLM itself proposed, evolved in their own database

Output as diffs

```
<<<<<< SEARCH
    return -1.0 * (cpu_residual + mem_residual)
=====
    return -1.0 * (cpu_residual + mem_residual +
                    mem_residual / cpu_residual +
                    cpu_residual / mem_residual)
>>>>>> REPLACE
```

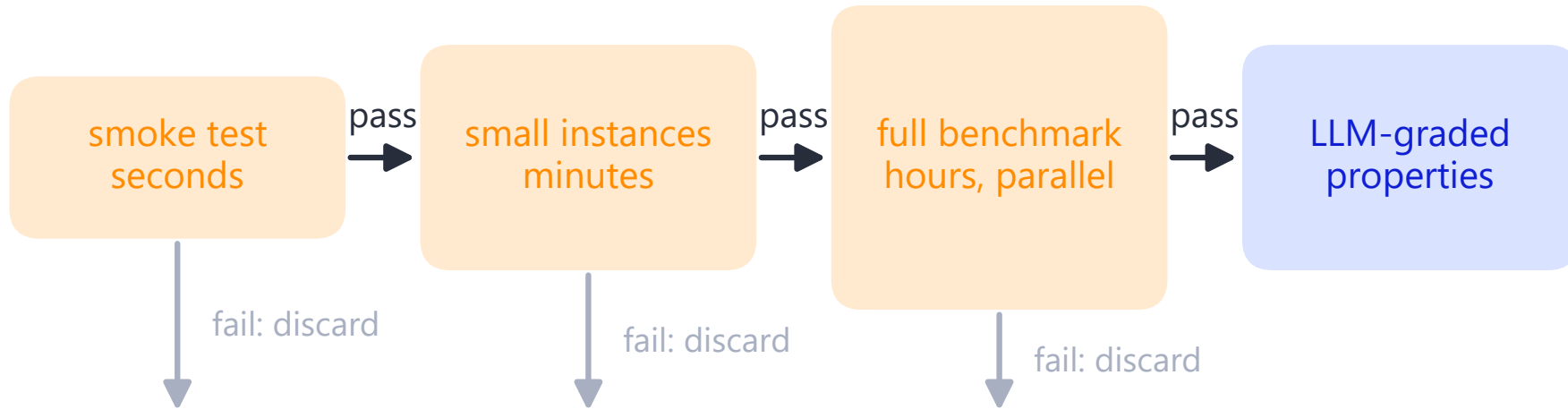
- The model proposes **search/replace** blocks; the exact original segment is located and replaced
- Targeted edits in large files, several per response, required to be mutually consistent
- For short code, or when a rewrite is better, the whole block can be emitted instead

An ensemble of models

- A **fast** model generates many candidates per unit of time: breadth of ideas
- A **strong** model occasionally proposes higher-quality changes: breakthroughs
- The mix maximises the number of evaluated ideas while keeping the chance of a large step
- Unlike FunSearch, the whole system improves as the base models improve

Evaluation: cascades, LLM feedback, parallelism

most candidates die cheaply; only promising ones reach the expensive evaluation

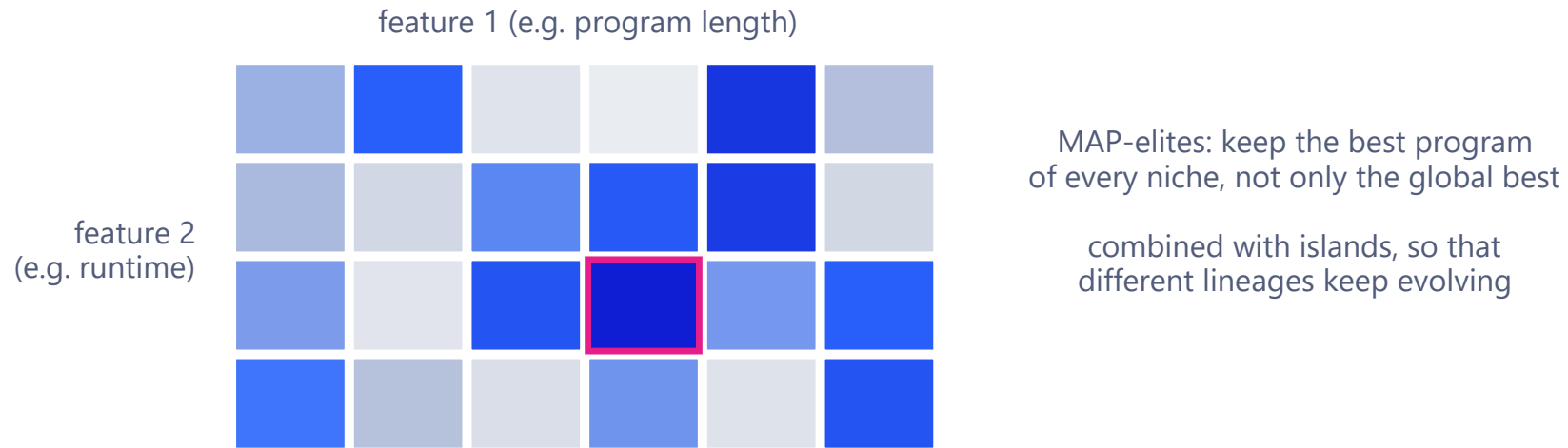


- **Cascade:** test sets of increasing difficulty; a candidate reaches the next stage only if it did well on all previous ones
- **LLM-graded properties:** qualities hard to score in code, such as simplicity, graded by a separate LLM call and added to the metrics
- **Parallel evaluation:** a single candidate may consume 100 compute-hours, spread across an evaluation cluster

Multiple scores help even a single target

- Several metrics can be optimised at once; solutions strong on all of them are the goal in many applications
- Even when one metric matters, adding others **improves that metric**
- Programs that excel on different criteria have different structures; showing them together in the prompt makes the model's proposals more varied
- The same effect that islands and diversity-aware crossover pursue, obtained through the metrics

The database: MAP-elites meets islands



- The evolutionary database must resurface earlier ideas at the right time: exploit the best while keeping diversity
- **MAP-elites**: keep the best program in every niche of a feature grid, not only the global best
- Combined with the **island** model from FunSearch, so separate lineages keep evolving

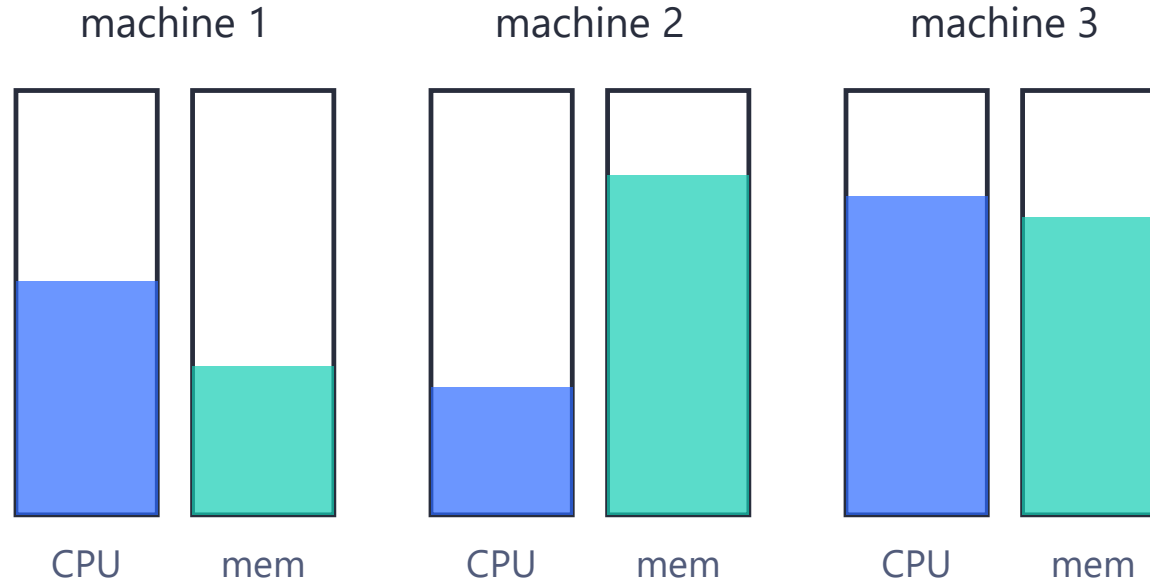
Results in mathematics and algorithms

Problem	Result
Multiplying two 4×4 complex matrices	48 scalar multiplications, improving on Strassen's 49 from 1969
14 matrix-multiplication sizes	new best-known upper bounds on the tensor rank
Over 50 open problems in analysis, geometry and combinatorics	best known matched on about 75%, improved on about 20% (e.g. kissing number in dimension 11)

- The evolved objects are search algorithms and constructors, verified by the evaluator, so the results are provably correct
- The same system, with no problem-specific architecture

Systems use case: scheduling jobs in Google's data centres

stranded resources: a machine full on memory but idle on CPU can take no more jobs



pending job
needs CPU 0.2, mem 0.5

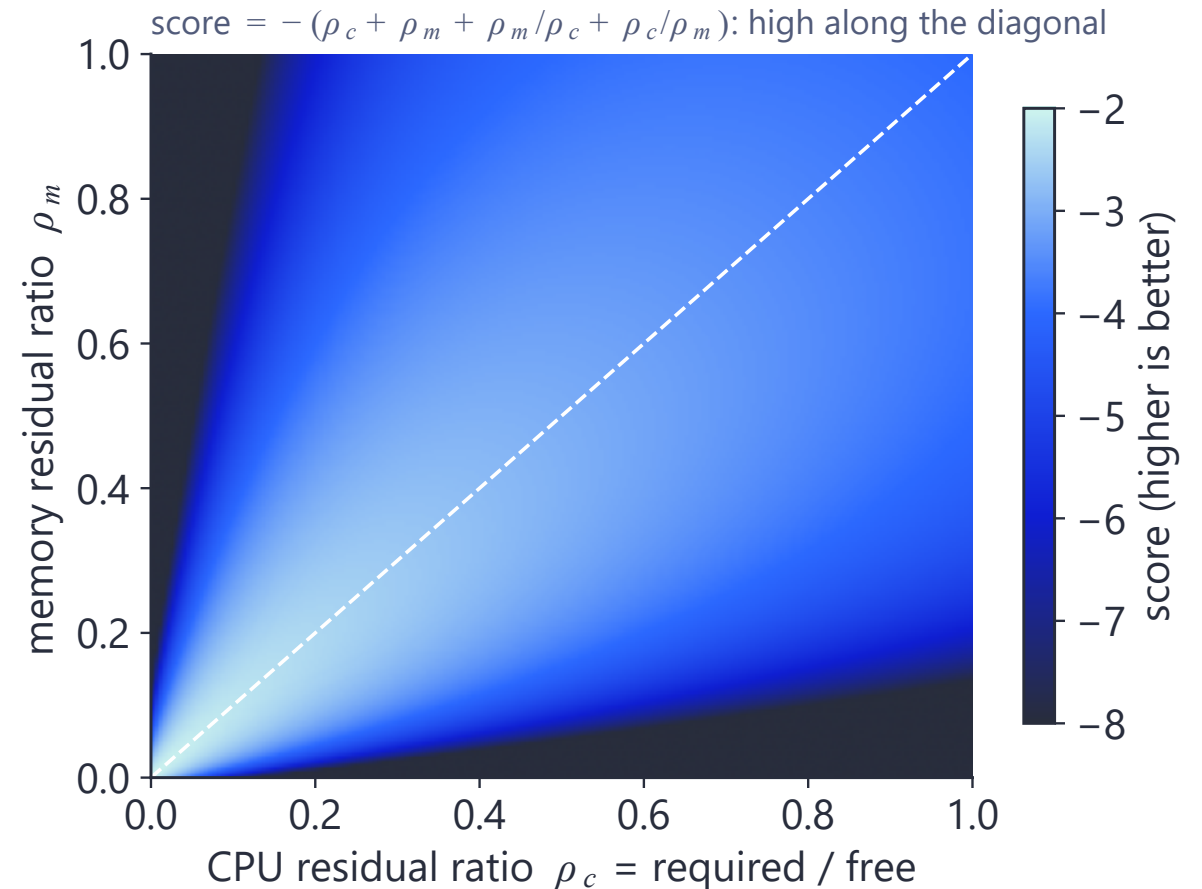
the heuristic scores each
feasible machine; Borg picks
the highest score

- Online job placement as **vector bin packing** in two dimensions: CPU and memory
- **Stranded resources**: a machine full on one resource cannot take jobs although the other is free
- The heuristic only ranks machines that Borg already found feasible, so its decisions are correct by construction

The discovered scoring function

```
def alpha_evolve_score(required, free):  
    cpu_residual = required.cpu / free.cpu  
    mem_residual = required.mem / free.mem  
    return -1.0 * (cpu_residual + mem_residual  
                  + mem_residual / cpu_residual  
                  + cpu_residual / mem_residual)
```

- Evolved from the production heuristic on a simulator fed with historical snapshots of the fleet
- Validated on unseen recent workloads, then rolled out fleet-wide
- Recovers on average 0.7% of Google's fleet-wide compute that would otherwise be stranded



Why it was chosen over deep RL

The decision

- A deep RL approach was on the table
- The code solution won on performance **and** on interpretability, debuggability, predictability and ease of deployment
- Essential qualities for a mission-critical system

Other engineering uses

- tiling heuristics for matrix-multiplication kernels used to train Gemini
- a functionally equivalent simplification in a TPU arithmetic circuit
- directly optimised compiler-generated code for attention

The Day 2 thesis in production: a five-line auditable rule beat a learned policy where it mattered

Exercise 2: set up AlphaEvolve for the vRAN scheduler

You want to evolve the task-to-node heuristic of a 5G virtual base station with AlphaEvolve.

1. What goes inside the evolve block, and what stays outside?
2. Which metrics should evaluate return?
3. What does the evaluation cascade look like?

- Inside: the scoring function and its helper features. Outside: the simulator, the deadline checks, the parser of the trace, the deployment interface
- Energy per task, fraction of deadline misses (as a negative), a fairness or tail-latency term, and the heuristic's own runtime per decision
- Unit tests on a handful of tasks → a one-minute trace → the full multi-hour trace on many nodes → an LLM-graded readability check

Agenda

1. Recap: the loop and its levers
2. ReEvo: reflective evolution
3. CALM: co-evolving the model
4. AlphaEvolve
5. **Beyond scalar fitness: TRACE**
6. Seminar wrap-up

Beyond scalar fitness: TRACE

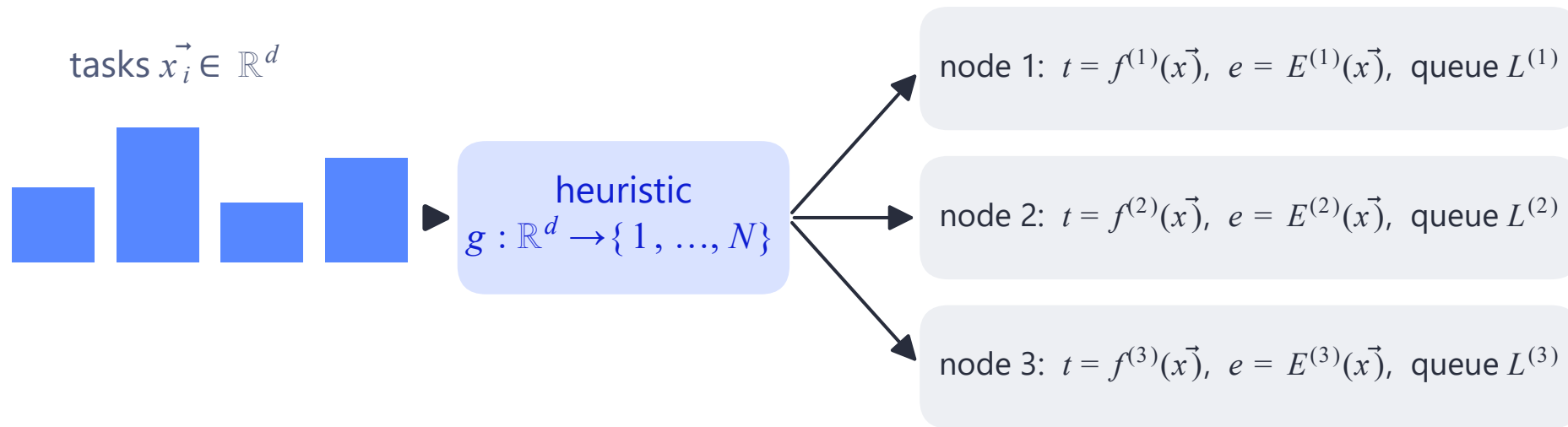
Heuristic discovery for partially observable resource-assignment systems

AHD benchmarks are static and fully observable

	Classical benchmarks (TSP, bin packing, flow shop)	Real resource assignment
Inputs	every parameter given a priori	processing-time functions, energy profiles, queue capacities unknown
Dynamics	static, deterministic	streaming tasks, noisy behaviour, drops
Feedback	a scalar fitness is enough	a scalar says what works, not why
Deployment	offline solution quality	industrial: auditability, latency, reliability

- FunSearch, EoH, ReEvo, CALM and most AHD work are evaluated on the left column
- The right column is where operators live ([Ayala-Romero et al. 2026](#))

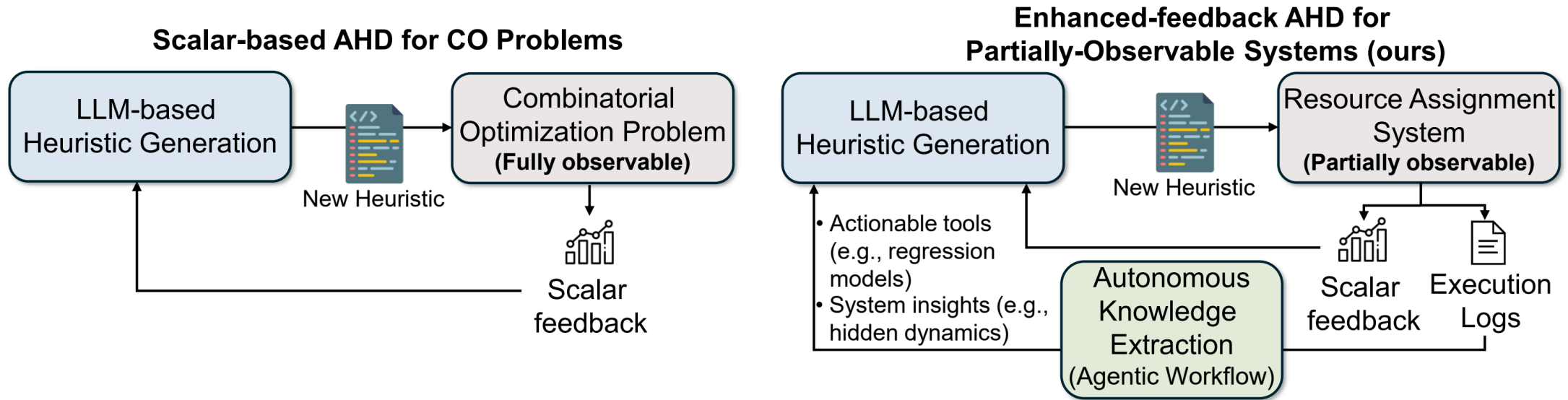
The problem: a stream of tasks onto heterogeneous nodes



all of $f^{(n)}, E^{(n)}, L^{(n)}$ are unknown a priori; cost $C_i = P^{(n)}(\vec{x}_i)$ if processed, Γ if dropped

- Two instantiations: **cloud computing**, cost = queueing plus processing time, drops on queue overflow; **5G vRAN**, cost = energy per task, drops on deadline violation
- Goal: a heuristic g that maps task features to a node, minimising the total cost over a period

A scalar score says what works, not why

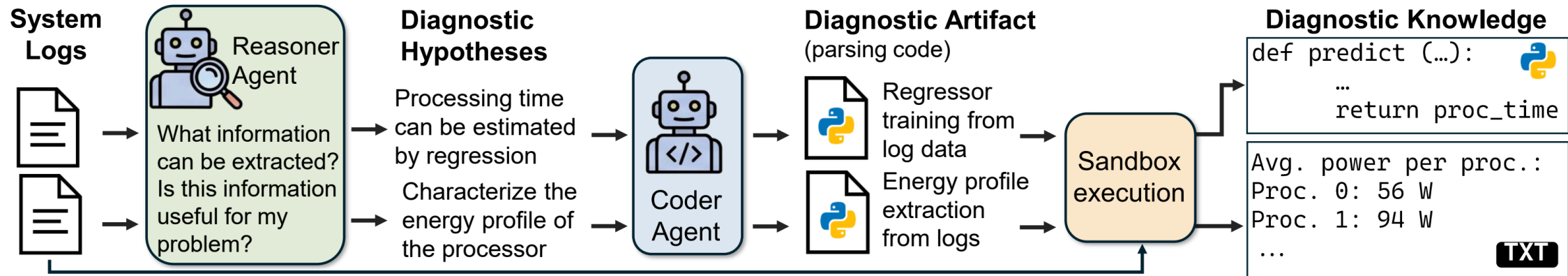


- Example: node 1 is fast but power-hungry, so an energy-aware heuristic should use it only for heavy tasks or under saturation
- With a scalar score alone (left), the loop must discover that by trial and error; as dimensionality grows, it does not
- The explanation sits in the execution logs; TRACE (right) turns them into actionable tools and system insights

Logs are large, heterogeneous and problem dependent

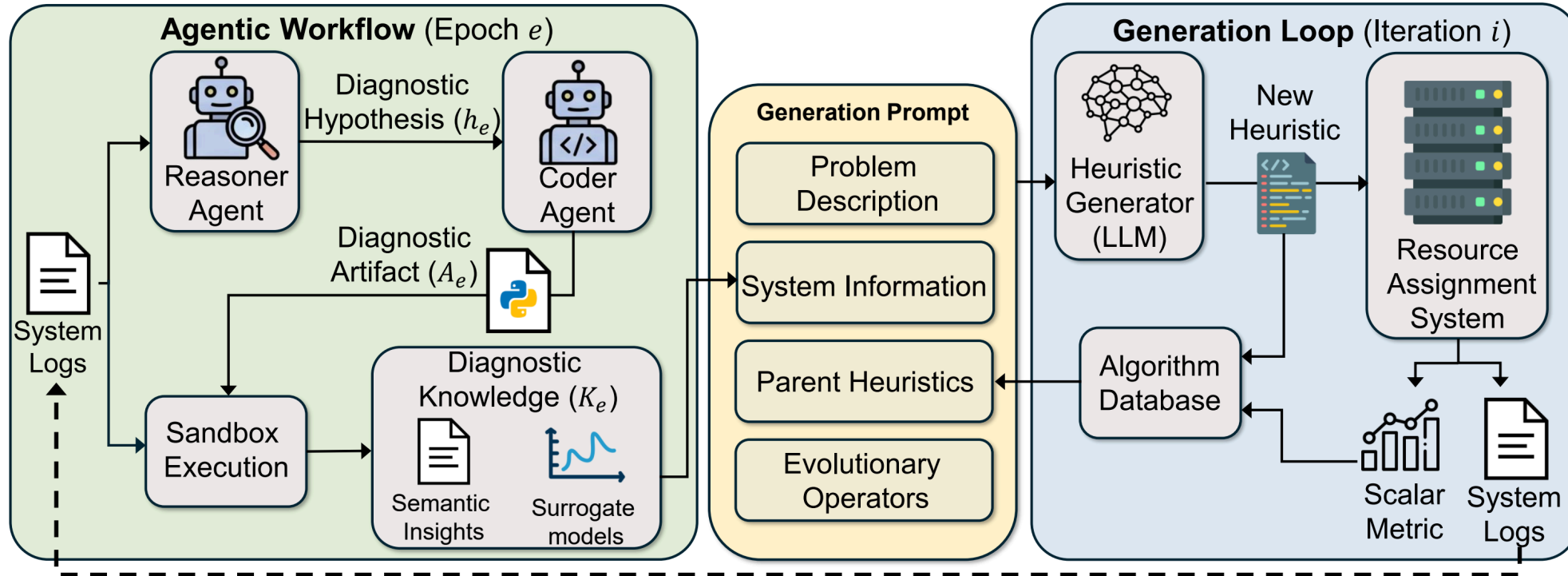
- **Size:** gigabytes, mostly noise and irrelevant fields; they cannot go into a prompt as they are
- **Problem dependence:** which metrics matter depends on the objective (latency vs energy) and is not always obvious
- **Heterogeneity:** structured or unstructured, different fields per hardware, different formats per system; no universal parser exists
- Turning logs into knowledge therefore needs intelligence, not a fixed pipeline

The reasoner–coder agentic workflow



- **Reasoner** (a reasoning model): reads the log schema and the problem, proposes diagnostic hypotheses
- **Coder** (a light model): writes the diagnostic artefact, a parsing script run in a sandbox with retries
- **Diagnostic knowledge**: a surrogate model callable from heuristics (top), or a behavioural insight in words (bottom); both enter the system-information part of the generation prompt

TRACE's dual loop



- **Epochs** trigger the agentic workflow, at most $E_{\max}$ times; after that the system information is frozen
- **Iterations** run the standard AHD loop with a light model, and every evaluation produces new logs

TRACE's use cases

Cloud computing benchmark

- synthetic, to study complexity and scale
- latency objective, drops on queue overflow
- less than 2% overhead over scalar-fitness AHD

5G vRAN with industrial data

- energy objective, drops on deadline violation
- heuristics meet the reliability targets with far less energy than industry rules and than state-of-the-art AHD
- judged more auditable in an LLM-as-a-judge study

Give the loop the system's own evidence, not only a number

Agenda

1. Recap: the loop and its levers
2. ReEvo: reflective evolution
3. CALM: co-evolving the model
4. AlphaEvolve
5. Beyond scalar fitness: TRACE
6. **Seminar wrap-up**

Seminar wrap-up

Two days, two kinds of artefact, one question: what will you deploy

The AHD methods compared

Method	Evolves	Guidance	Model	Multi-metric	Typical budget
FunSearch	one function	scores of two parents	small, frozen	no	10^6 samples
EoH	thought + code	five prompt strategies	any API, frozen	no	10^3 queries
ReEvo	code	short- and long-term reflections	API, frozen	no	10^2 evaluations
CALM	idea + code	fine-grained operators + RL fine-tuning	local 7B, trained	no	10^3 queries
AlphaEvolve	whole files	rich context, diffs, meta-prompts	frontier ensemble	yes	10^3 samples, hours per evaluation
TRACE	code + system information	scalar score + log-derived knowledge	reasoner + light model	no	10^3 steps, few agentic epochs

Learned policy or generated heuristic

Day 1: learn a policy online

the response is unknown and must be
learned on the system itself
fast adaptation to context and drift
per-step constraints with risk control
the artefact is a model (network or GP)

bandits · contextual bandits
safe and risk-aware learning · RL

Day 2: generate a heuristic offline

a simulator or trace-driven evaluator exists
auditability, latency and portability
are required
the artefact is a short program
logs can inform the search (TRACE)

FunSearch · EoH · ReEvo · CALM
AlphaEvolve · TRACE

Open questions for discussion

- **Evaluators for noisy systems:** how many repetitions, and what to do when the simulator and the system disagree?
- **Safety of generated code:** who reviews thousands of candidates, and what may a heuristic touch?
- **Non-stationarity:** when should a deployed heuristic be regenerated, and can a bandit tell us?
- **Auditing the audit:** an LLM judging the interpretability of an LLM's program
- **Cost:** where do the LLM calls, the GPUs and the simulation hours sit in an operator's budget?

References

Ayala-Romero, Jose A. et al. 2026. "TRACE: Tackling Real-World Resource Assignment Problems via Agentic Heuristic Discovery." Under review. *arXiv Preprint arXiv:2610.01887*.

Huang, Ziyao, Weiwei Wu, Kui Wu, Jianping Wang, and Wei-Bin Lee. 2025. "CALM: Co-Evolution of Algorithms and Language Model for Automatic Heuristic Design." *arXiv Preprint arXiv:2505.12285*.

Novikov, Alexander, Ngân Vũ, Marvin Eisenberger, et al. 2025. *AlphaEvolve: A Coding Agent for Scientific and Algorithmic Discovery*. White paper. Google DeepMind.

Shao, Zhihong, Peiyi Wang, Qihao Zhu, et al. 2024. "DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models." *arXiv Preprint arXiv:2402.03300*.

Ye, Haoran, Jiarui Wang, Zhiguang Cao, et al. 2024. "ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution." *Advances in Neural Information Processing Systems (NeurIPS)*.

NEC

\Orchestrating a brighter world