INNOVATION PRIOR: LEARNING TO DISCOVER BY DISTILLING THE SCIENTIST'S MIND

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) excel at tasks that ask for known methods to be applied well, but on research tasks they recombine and retune existing methods instead of inventing new ones as human scientists do. A harness around a frozen model can only select among what the model produces, and reinforcement learning can only strengthen what it already does, so the missing ability has to come from training data. Papers are in that data, but a paper presents a finished result and never shows how its authors thought. We design a pipeline that reconstructs that thinking and use it to build INNOVATION PRIOR, a dataset of how scientists reasoned their way to their methods. Each reconstruction draws on multiple sources beyond the paper and includes the authors' exploration of routes that never reached the final method. We fine-tune Qwen3.5-4B and 9B on INNOVATION PRIOR, then run one RL recipe from the fine-tuned model and from the base model. RL from the fine-tuned model outperforms RL from the base on research benchmarks at both sizes, while none of our training data targets these benchmarks. INNOVATION PRIOR thus sets a prior for discovery that RL then amplifies. Ablations show that both the multiple sources and this exploration are necessary. The advantage carries over to test-time scaling, where the model scores higher for the same compute, and generalizes to research judgment. The model also explores more broadly, both within and across attempts, and case studies show that it reaches simple but effective methods from first principles.

1 INTRODUCTION

Large language models (LLMs) are now reliable on tasks that ask for known methods to be applied well: software engineering (Anthropic, 2025; Yang et al., 2024; OpenAI, 2025a), competitive programming (OpenAI, 2025b; OpenAI et al., 2025; Li et al., 2022) and olympiad mathematics (Luong & Lockhart, 2025; Trinh et al., 2024; Chervonyi et al., 2025; Hubert et al., 2026). This success has motivated work on **AI for research**, which builds AI systems that propose methods and test them by experiment in machine learning (Lu et al., 2024; Yamada et al., 2025), combinatorial optimization (Novikov et al., 2025; Lange et al., 2026) and the natural sciences (Gottweis et al., 2026; Ghareeb et al., 2026; Bran et al., 2024; Szymanski et al., 2023). Unlike engineering tasks, research requires the method itself to be new. However, evaluations that vary the agent framework, the base model, the field and the sampling budget all reach the same conclusion: current research agents build their methods by recombining and retuning existing ones, not by introducing new ideas (Antoniades et al., 2026; Tang & Yang, 2026; Lyu et al., 2026; Chen et al., 2026; Artiles et al., 2026; Bao et al., 2026; Li, 2026; Nathani et al., 2025; Kirgis et al., 2026). LLMs have become expert at applying methods, but not at inventing them as good human scientists do.

Current work on AI for research follows two main approaches: designing the harness, which is the agent workflow around the model (Lu et al., 2024; Gottweis et al., 2026; Novikov et al., 2025; Zhang et al., 2026b), and training the model with reinforcement learning (He et al., 2026; Si et al., 2026b; Šurina et al., 2025; Liu et al., 2026). A harness keeps the model frozen and can only select among the candidates that model generates, so it cannot fundamentally change the model's capabilities or reasoning patterns (Tang & Yang, 2026). Reinforcement learning changes the model's parameters, but these updates mainly strengthen behaviors that the base model already exhibits; a pattern the base model lacks, or produces only with very low probability, has to be taught first with supervised data (Yue et al., 2025; Wang et al., 2026; Gandhi et al., 2025; Wang et al., 2025; Zhang et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

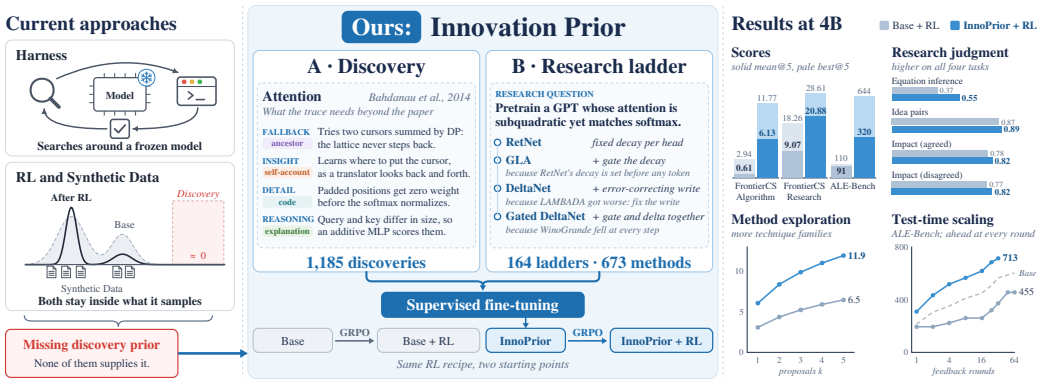


Figure 1: **Overview.** Harnesses and RL build on what a model already produces (**left**). We reconstruct discovery traces for SFT (**center**). At 4B, InnoPrior + RL ends higher on every panel (**right**).

2026a). If the reasoning patterns behind method discovery are missing from the base model, neither approach can supply them, and they have to come from the training data.

LLMs are trained on large collections of research papers (Gao et al., 2020; Touvron et al., 2023), but a paper records the result of research, not the thinking that produced it. It presents a finished method and argues for it, and leaves out the alternatives its authors weighed and dropped, the order in which the ideas came and the judgment behind each choice. Existing ways of synthesizing reasoning data do not fill this gap. Distillation and verifier filtering (Guo et al., 2025; Muennighoff et al., 2025; Zelikman et al., 2022; Yuan et al., 2023; Singh et al., 2024) need a model that already reasons this way and a reference answer to check against, and discovery offers neither (Wu et al., 2025); datasets that reconstruct scientific reasoning from a paper's text (Kumbhar et al., 2026; Bouras, 2026; Yang & Bing, 2026) inherit what the paper leaves out.

To this end, we build INNOVATION PRIOR, a dataset that teaches an LLM during mid-training how human scientists reasoned their way to important discoveries and methods, giving reinforcement learning a prior for innovation to amplify. INNOVATION PRIOR covers 1,185 discoveries, each built around one published method, and 164 research ladders, each a sequence of published methods that improve on one research question. For each, an agent collects the primary paper, the works it improved on, third-party explanations, the code and any later account by the authors, and writes a first-person trace that derives the method from the problem and the prior art of its year.

We fine-tune Qwen3.5-4B and Qwen3.5-9B on INNOVATION PRIOR and run the same RL recipe from the fine-tuned model and from the untouched base. At both sizes, RL from the fine-tuned model ends higher. Neither INNOVATION PRIOR nor the RL problems are drawn from these benchmarks, so INNOVATION PRIOR gives RL a better prior for discovery, not answers to recall. The advantage extends to test-time scaling and to research judgment, and the fine-tuned model explores both more widely and more deeply. We illustrate the problem, our approach and the main results in Figure 1. In summary, our contributions are as follows:

- We propose a pipeline that turns the record of how a method was found into a training trace, with retrieval and leakage rules that make the trace usable as training data.
- We build INNOVATION PRIOR, with 1,185 discoveries and 164 research ladders, and a set of 1,300 open-ended problems for RL on method discovery.
- We show that RL from INNOVATION PRIOR outperforms RL from the base model, and that the model uses test-time compute better, explores more widely and deeply, and judges research better.

2 INNOVATION PRIOR: RECONSTRUCTING THE SCIENTIST'S MIND

A paper seldom contains the reasoning behind its method, because it is written to present the finished method, not to record how the method was found. This shows in three ways: (1) the motivation is written after the result is known, so the paper argues for the method rather than retracing the work; (2) the paper reports the final method and seldom the alternatives that were abandoned or the

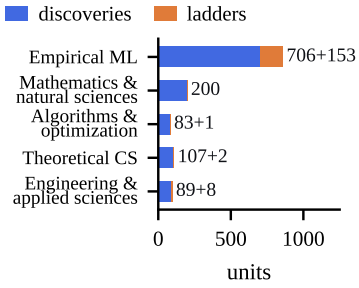


Figure 2.1: Domains.

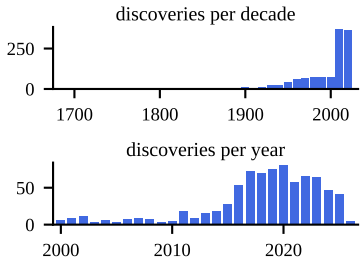


Figure 2.2: Discovery years.

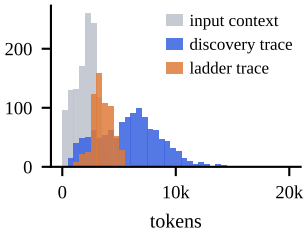


Figure 2.3: Token lengths.

Figure 2: Domains, discovery years, and token lengths of inputs and traces.

implementation details on which the method's success depends; and (3) its order breaks the causal order of the reasoning, with a derivation often split between the main text and the appendix. We therefore build INNOVATION PRIOR; Sections 2.1 and 2.2 give its units, sources and construction.

2.1 UNITS AND SOURCES

Discoveries and ladders. INNOVATION PRIOR has two kinds of unit, one for each step of research: a **discovery** reconstructs how one published method was derived from the prior art, and a **ladder** how a line of published methods improved on one another. Discoveries span five areas: (1) empirical machine learning, (2) mathematics and the natural sciences, (3) algorithms and optimization, (4) theoretical computer science, and (5) engineering and the applied sciences. Candidate methods come from four pools: the baselines of widely used machine-learning benchmarks, catalogs of classic algorithms, best papers and landmark results across the sciences, and olympiad solutions; Appendix A lists examples from each. We keep a candidate only if its idea can be derived from earlier work and the sources for that derivation can be retrieved. Each ladder follows one research problem, such as subquadratic attention for language models, through published methods that each build on the one before. INNOVATION PRIOR holds 1,185 discoveries and 164 ladders covering 673 methods, and Figures 2.1 and 2.2 give their areas and years.

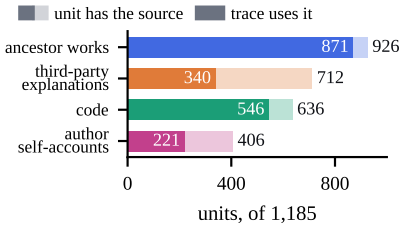


Figure 3.1: Units that have each kind of source, and units whose trace uses it.

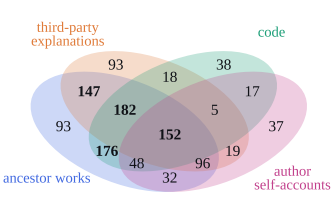


Figure 3.2: Overlap of the four kinds of source across units.

Figure 3: The four kinds of source in INNOVATION PRIOR.

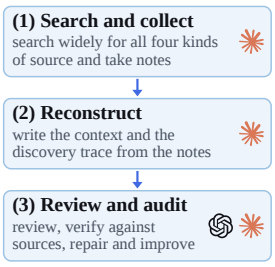


Figure 4: The construction workflow.

Sources beyond the paper. The reasoning that a paper leaves out often survives in some kinds of source outside it: (1) **Ancestor works**, the earlier works the method builds on, show the problem the authors faced and the approaches others had tried, and so rebuild the context in which the authors worked. (2) **Code**, from the authors or a widely used reimplementation, fixes the constants and steps that the paper leaves out and on which the method's success often rests, and the dated history of a repository shows which versions came first. (3) **Third-party explanations**, written to teach, say why a step is taken, often with a small example or counterexample, where the paper only takes it. (4) **Self-accounts**, the authors' own words outside the paper in lectures, interviews, memoirs, replies to reviewers and users, and drafts left in the paper's source, recall the problem that started the work, the alternatives they tried and dropped, and the order in which the ideas came. Appendix B gives examples of what each kind adds beyond the paper, and Figure 3 shows how often each kind is present and used across INNOVATION PRIOR.

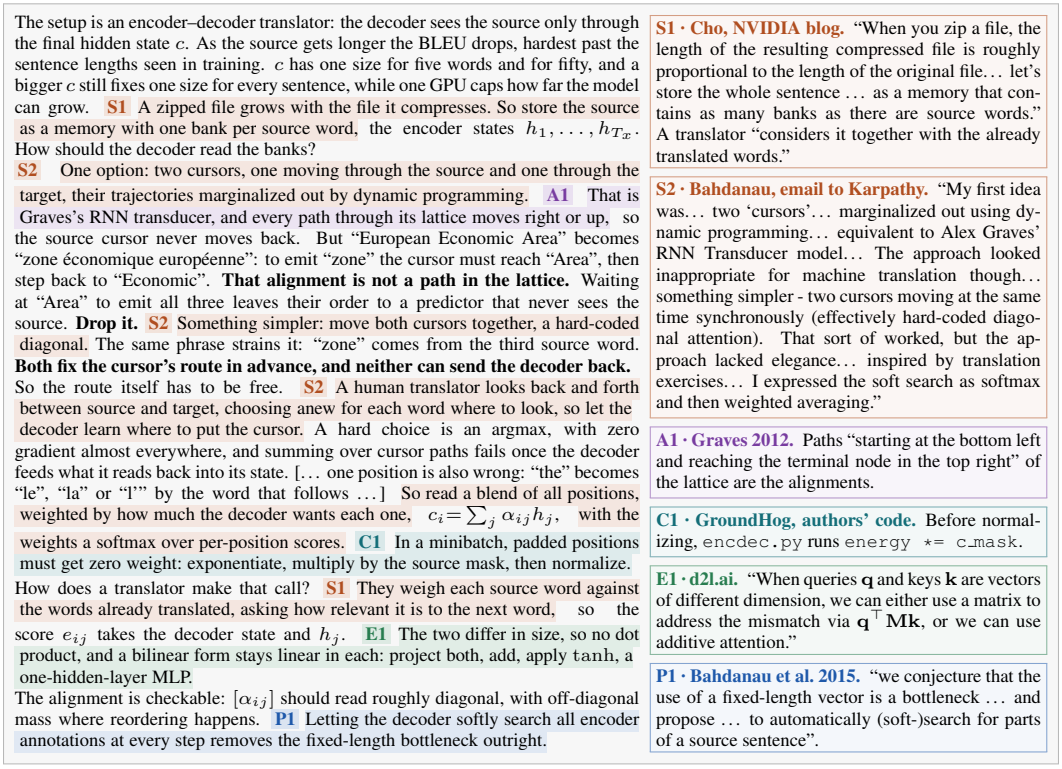


Figure 5: **A sample and its sources.** Left: part of the reconstructed discovery trace. Right: the sources. Tints: A ancestor, S self-account, E third-party explanation, C code, P original paper.

We present one unit of INNOVATION PRIOR and the sources it is built from: Figure 5 gives part of the trace for attention in neural machine translation (Bahdanau et al., 2015), with the sources behind some critical steps. That paper states only the conclusion; the self-accounts supply the two cursor designs the authors tried and dropped before it and the analogy to how a translator reads that led to a learned cursor, the code supplies the mask over padded positions, and no version of the paper mentions any of them. Appendix C shows more examples.

2.2 CONSTRUCTION

Coding agents build each unit in three steps, shown in Figure 4: **(1) Search and collect.** A Claude Code agent with Claude Opus 4.8 searches widely for the four kinds of source and, from these sources alone, records each design choice with its reason and the alternatives it rules out. **(2) Reconstruct.** From these records the agent writes a unit as a triple $(\mathcal{C}, \mathcal{R}, \mathcal{A})$. The **context** $\mathcal{C}$ gives the research question and the background of the field up to the year t of the discovery, with a code framework to fill in when the answer is code; nothing after t appears in $\mathcal{C}$ to prevent any hint or leak. The discovery trace, i.e. **reasoning** $\mathcal{R}$, reconstructs how the method was reached from the collected records, in the first person and the order of discovery, and includes **fallbacks**, routes that the records show the authors considered or tried but that never reached the final method. The **answer** $\mathcal{A}$ sums up the method and its reasoning and ends with complete code if feasible. **(3) Review and audit.** Codex with GPT-5.5 corrects reasoning errors and hallucinated facts, the Claude Code agent rechecks each correction, and an adversarial verifier checks the key step of $\mathcal{R}$ against the sources, with the agent repairing what it flags. All agents run at `xhigh` reasoning effort. Appendix D quotes their rules. Note that INNOVATION PRIOR is not a distillation of these agents: it recombines what the web records, and we neither keep nor train on the agents' reasoning.

Data format and training. Each discovery gives one training example $(t, \mathcal{C}) \to (\mathcal{R}, \mathcal{A})$, with $\mathcal{C}$ as the user prompt and t stated in the system prompt, so that the model takes t as its present. We call this **frontier conditioning**. Each example is one step beyond the frontier of year t. Given the base model has read much of what came after t, with t the trace is how a model standing at t reasons, so

Table 1: Main results of our models and the baselines on FrontierCS Algorithm, FrontierCS Research, ALE-Bench and MLS-Bench. The best result for each model size is in **bold**.

Model	FrontierCS Algorithm		FrontierCS Research		ALE-Bench		MLS-Bench
	mean@5	best@5	mean@5	best@5	mean@5	best@5	
Qwen3.5-27B	11.41	23.51	22.58	44.73	356	774	23.26
Qwen3.5-35B-A3B	8.81	19.11	16.89	40.79	363	791	10.88
Qwen3.5-4B series							
Base	4.89	**12.40**	6.52	16.73	200	413	2.98
InnoPrior	3.97	9.60	6.99	15.97	196	388	6.63
Base + RL	0.61	2.94	9.07	18.26	91	110	19.28
InnoPrior + RL	**6.13**	11.77	**20.88**	**28.61**	**320**	**644**	**19.95**
Qwen3.5-9B series							
Base	4.47	12.07	10.37	27.66	243	534	11.77
InnoPrior	4.57	12.27	9.90	27.76	232	524	7.17
Base + RL	7.56	13.26	16.33	30.23	316	634	16.59
InnoPrior + RL	**8.25**	**14.82**	**18.41**	**31.32**	**317**	**660**	**19.52**

training teaches the step beyond the frontier rather than the recall of a known result. At evaluation t is set to the present without changing any benchmark prompt, so the model takes that step from today's frontier. Figure 2.3 gives the lengths of inputs and traces. In a ladder every method is trained on and sees only the context and methods before it. Each ladder is also converted automatically into a tool-using agent episode in our training.

3 Experiments

We test whether the prior changes what reinforcement learning achieves. At two model sizes, we run one RL recipe from the base model and from the same model fine-tuned on Innovation Prior.

3.1 Setup

Models and Training. We use Qwen3.5-4B and Qwen3.5-9B (Yang et al., 2025a) as base models. For **SFT**, the model companies would inject prior patterns during the mid-training stage (Team OLMo et al., 2024; Wang et al., 2025; Akter et al., 2026), the academia labs can neither get to checkpoints before mid-training nor run mid-training and inject the prior there. We therefore approximate it on the instruct checkpoints with a rank-32 LoRA adapter (Hu et al., 2022) trained for one epoch and merged into the base at a tenth of its magnitude (Wortsman et al., 2022). Since the code in Innovation Prior is mostly Python, the fine-tuning data also holds 338 ordinary competitive-programming problems solved in C++, added only to keep the model's ability to write single-file C++ programs. For **RL**, we run the same GRPO (Shao et al., 2024) training in verl (Sheng et al., 2025) from the fine-tuned model and the untouched base, so that each size has four arms: Base, the released instruct model; InnoPrior, the model fine-tuned on Innovation Prior; and Base + RL and InnoPrior + RL, the two after RL. We evaluate all RL models at step 20. Since no open RL data exists for this kind of task, we build our own with a pipeline that follows FrontierSmith (He et al., 2026). Appendices E and F give the training configuration and the RL problems.

Benchmarks. We evaluate on four benchmarks of frontier-level problems in computer science research and optimization: (1) **FrontierCS Algorithm** (Mang et al., 2026), 172 algorithm optimization problems; (2) **FrontierCS Research**, 64 of its 68 open research problems, excluding four security tasks whose exploit generation would put our machines at risk; (3) **ALE-Bench** (Imajuku et al., 2025), 40 AtCoder Heuristic Contest problems, scored by the performance rating a submission would have earned among human contestants; and (4) **MLS-Bench** (Lyu et al., 2026), machine learning research tasks that ask an agent to improve the algorithm for one research question inside a fixed scaffold, of which we use the 20 CPU tasks. We follow the original evaluation of each benchmark: FrontierCS and ALE-Bench are evaluated with five samples per problem and MLS-Bench runs its own agent loop once per task. Appendix G gives further evaluation details.

Table 2: Ablations at 4B after RL, under the protocol of Table 1.

Data	FrontierCS Algorithm		FrontierCS Research		ALE-Bench		MLS-Bench
	mean@5	best@5	mean@5	best@5	mean@5	best@5	
Base + RL	0.61	2.94	9.07	18.26	91	110	19.28
InnoPrior + RL	**6.13**	**11.77**	**20.88**	**28.61**	**320**	**644**	**19.95**
w/o multiple sources	5.22	10.47	20.47	28.19	242	494	13.71
w/o fallback patterns	6.09	11.17	19.68	26.18	250	497	14.31

Data contamination. We find no contamination. An 8-gram overlap check shows that FrontierCS and ALE-Bench problems share only generic wording with the fine-tuning data and the RL problems.

3.2 Main results

Table 1 supports our central claim: **Innovation Prior sets a prior for discovery that RL then amplifies.** Most scores are taken over five samples per problem, which already removes much of the randomness. Before RL, InnoPrior is not consistently stronger than Base. This is expected: the traces in Innovation Prior are written by other models and thus off-policy, and enter only as a small update to a released checkpoint rather than during mid-training. Under the same RL recipe, however, RL from the prior ends clearly above RL from the base at both sizes. RL from the base is itself effective: at 4B it gains substantially on FrontierCS Research and MLS-Bench. At 9B, RL from the prior pulls further ahead as test-time compute grows (Section 3.4). The gain appears only after RL and from data with no solution to any benchmark problem (Section 3.1), so **Innovation Prior does not teach answers to recall but gives RL a better prior for discovery**.

3.3 Ablations

We ablate the two key components of Innovation Prior to test whether each matters: (1) **multiple sources**, records beyond the paper that recover the reasoning it omits, and (2) **fallback patterns**, the authors' exploration of routes that never reached the final discovery. We build Innovation Prior once without each component and keep everything else the same, including the SFT and RL recipes and the data for maintaining coding ability, and run both ablations on the 4B model.

Removing either component lowers the scores in Table 2, most on ALE-Bench and MLS-Bench, so **both the reasoning from sources beyond the paper and the exploration in the fallback patterns are necessary for the prior that RL amplifies**. Moreover, since the data for maintaining coding ability is unchanged, the drop also shows that the gains come from the discovery traces.

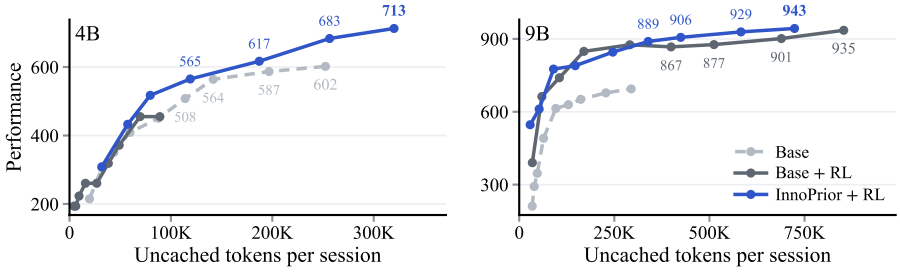


Figure 6: Performance over feedback rounds, by uncached tokens per session.

3.4 Test-time scaling

Discovery is iterative; we therefore test whether the prior also helps a model improve from feedback with more test time computing. On ALE-Bench Lite, the benchmark's official subset of 10 problems, each model refines one program over 64 rounds, with three sessions per problem: in each round it reads the official feedback on its last submission over the public test cases and writes a new program. Following ALE-Bench's own evaluation, the submission with the best public score so far is scored

on the private test cases and converted to a performance rating, so a curve can dip when a later choice fails a private case. We measure compute in uncached tokens, which are roughly proportional to the FLOPs used (Kaplan et al., 2020; Zheng et al., 2024a).

Figure 6 shows that **the advantage of the prior carries over to test-time scaling**. Neither the fine-tuning data nor RL trains the model to refine over this many rounds of feedback, yet RL from the prior keeps improving as the rounds grow. At 9B, both RL models improve far beyond the base model, and RL from the prior pulls ahead in the later rounds and ends higher while spending fewer tokens. At 4B, RL from the prior leads from the first round on. **With the prior, RL uses test-time compute more efficiently: it scores higher for the same compute and reaches the same score with less compute.**

4 Analysis

We further analyze what the prior changes by comparing RL from the prior with RL from the base. Section 4.1 tests whether the gains generalize to research judgment, Section 4.2 measures how broadly each model explores, and Section 4.3 asks what methods earn the higher scores.

4.1 Generalization to research judgment

If the prior teaches how scientists think, it should also help the model judge research. We use four tasks from three third-party sources: (1) **Equation inference** from AAAR-1.0 (Lou et al., 2025): given the LaTeX source of a paper around a masked equation, the model picks the one the authors wrote from four candidates. (2) **Idea pair** from LiveIdeaBench (Ruan et al., 2024): of two research ideas for the same scientific keyword, the model picks the one the critic models rated more original. (3) **Impact** from the OpenReview dataset of Höpner et al. (2025): of two papers accepted at the same venue, the model picks the one later cited more. We split these pairs into **Impact (agreed)** and **Impact (disagreed)** by whether the reviewers also preferred the more cited paper, so copying the reviewers is always right on the first and always wrong on the second.

Figure 7 shows that **the scientist's mind learned from Innovation Prior generalizes to research judgment**, with RL from the prior ahead at 4B. Innovation Prior was not built for judgment, yet the model learns to tell good methods and ideas from weak ones, so **what it learns is not how to narrate a discovery but how a scientist thinks**.

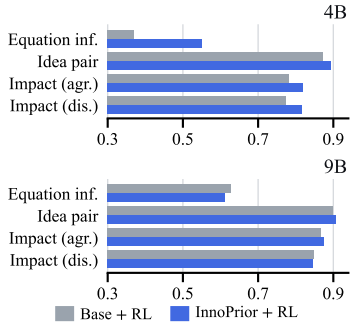


Figure 7: Accuracy on research judgment tasks.

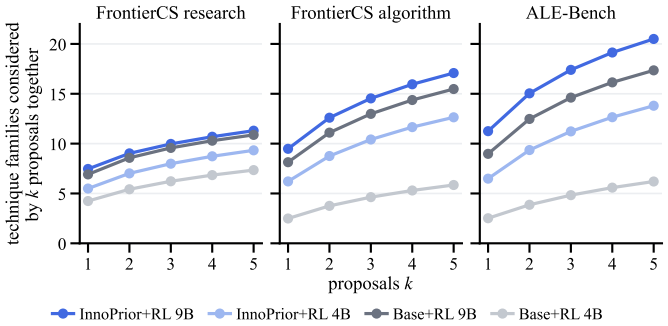


Figure 8: Technique families that k proposals consider together, averaged over all choices of k of the five proposals for a problem.

4.2 Breadth of exploration

A scientist weighs several methods before committing to one. We therefore count the methods a model considers, both within one proposal and across the five proposals it makes for each FrontierCS and ALE-Bench problem. We identify methods by 84 technique families from the solutions that frontier models submitted to the FrontierCS leaderboard and released in its GitHub repository. Each family is a set of keywords, the names of the technique and the identifiers that implement it in C++ or Python (more details in Appendix H), and a proposal considers a family if its reasoning before

the final answer contains any of them. For k proposals of a problem, we count the families that any of them considers and average this count over all choices of k of the five proposals, then over problems. The count at $k = 1$ measures how widely one proposal explores, and its rise with k shows whether further proposals bring new methods or repeat the ones already considered.

Figure 8 shows that **RL from the prior explores more methods in each proposal, and further proposals keep adding new ones**. At $k = 1$, RL from the prior weighs more technique families than RL from the base at both sizes. Most of the added families never reach the final code: the model considers them and sets them aside. This pays off, since among the proposals for the same problem, those that weigh more families for each one they put into code score higher. As k grows, the count keeps rising and RL from the base does not catch up at any k: each further proposal brings in methods the earlier ones did not consider, so the model is not confined to the methods of any single attempt. **The prior teaches the model to weigh many methods before committing to one, as a scientist does, and widens the range of methods it draws on.**

4.3 Case Study

While RL from the prior scores higher, a score alone does not show how a method was found. Current research agents build their methods by recombining and retuning existing ones, while a scientist often reaches a simple but effective method by reasoning from first principles. We therefore ask which of the two explains the higher scores. Human experts examined the reasoning and the programs of both models after RL. Figure 9 demonstrates one ALE-Bench problem through the reasoning and the proposal of the two 9B models, and Appendix I gives more cases.

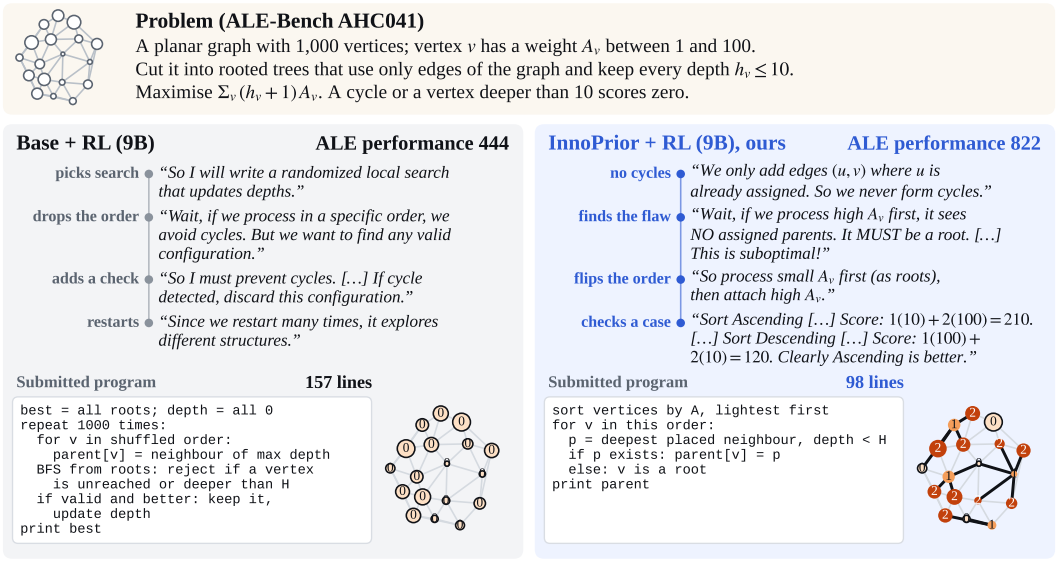


Figure 9: An ALE-Bench problem, excerpts of each 9B model's reasoning after RL, and its program. Small graphs: each program's output on a toy input with depth bound 2, labelled with depths.

Figure 9 shows that **RL from the prior reaches a simple but effective method from first principles, while RL from the base turns to a generic search**. Both models see that deeper vertices score more, that parent pointers can close a cycle, and that building the forest in a fixed order would rule cycles out. Base+RL sets the order aside. It reassigns every parent pointer in a random order, discards any forest that fails a cycle check and repeats. Each attempt gives every vertex a parent, so no forest passes, and it submits the initial one, where every vertex is a root. InnoPrior+RL builds the forest in one pass and asks which order to use. It sees that the first vertex processed must become a root, so it processes light vertices first, confirms this on a two-vertex example, and submits a single pass in which each vertex hangs from the deepest neighbour already placed. The pass cannot create a cycle and leaves heavy vertices deep, and reversing only its order loses much of its score. The stronger program is also the shorter one: **its higher score comes from one insight reached from first principles as a scientist would, not from tuning a known method**.

5 Related Work

The novelty deficit of research agents. Proposals from research agents often read as novel, but turn out to be trivial once executed or compared with the literature. Reviewers rate model ideas above expert ideas when both are written proposals (Si et al., 2025), yet this advantage disappears once the ideas are carried out (Si et al., 2026a). When reviewers also see the agent's workspace, agent research ranks below rejected human papers (Zhang et al., 2026c), and its ideas fall in lower-impact areas of the literature (Tang & Yang, 2026; Jiang, 2026). Judges who read only the written text are easily swayed by its surface quality (Sinhahajari et al., 2026; Zheng et al., 2025; Wataoka et al., 2024). Whatever the framework or the model, the proposals stay close to the papers the agent started from (Tang & Yang, 2026; Chen et al., 2026; Artiles et al., 2026; Bao et al., 2026), and benchmarks built to test invention find the same gap (Antoniades et al., 2026; Zhang et al., 2025; Nathani et al., 2025; Kirgis et al., 2026; Li, 2026).

Current approaches to improving research agents. To improve these agents, current work either designs a harness around a frozen model or trains the model itself with reinforcement learning. Harnesses include research pipelines (Lu et al., 2024; Yamada et al., 2025; Schmidgall et al., 2025; Yuan et al., 2025; Jansen et al., 2025), multi-agent laboratories (Gottweis et al., 2026; Swanson et al., 2025; Ghareeb et al., 2026; Mitchener et al., 2025), evolutionary program search (Romera-Paredes et al., 2024; Novikov et al., 2025; Lange et al., 2026; Ye et al., 2024), self-modifying scaffolds (Hu et al., 2025; Yin et al., 2025; Zhang et al., 2026b) and code search for machine-learning engineering (Jiang et al., 2025; Liu et al., 2025; Nam et al., 2025); reinforcement learning has been applied to algorithm and heuristic design, GPU kernel generation and research agents (Šurina et al., 2025; Huang et al., 2026; Baronio et al., 2026; Li et al., 2025; Liu et al., 2026; Yang et al., 2026; Weng et al., 2025; He et al., 2026; Si et al., 2026b). A harness can only select among the candidates the frozen model generates (Tang & Yang, 2026; Chan et al., 2025; Ishibashi et al., 2026), and reinforcement learning tends to strengthen what the model can already do rather than teach a reasoning pattern the model lacks (Yue et al., 2025; Wang et al., 2026; Gandhi et al., 2025).

Synthesizing reasoning data for supervised fine-tuning. Teaching such a pattern takes new training data. Reasoning traces for fine-tuning are now mostly written by models: distilled from a stronger reasoner (Guo et al., 2025; Mukherjee et al., 2023; Muennighoff et al., 2025), sampled and kept when a verifier accepts them (Zelikman et al., 2022; Yuan et al., 2023; Singh et al., 2024), or serialized from a search (Gandhi et al., 2024; Zhang et al., 2024). Each needs a generator that already reasons well and, for filtering, a reference answer, and discovery has neither. Studies of data quality find that trace content matters more than trace count (Zhou et al., 2023; Ye et al., 2025; Gandhi et al., 2025; Wang et al., 2025). Existing syntheses of scientific reasoning reconstruct it from the finished paper (Kumbhar et al., 2026; Bouras, 2026; Yang et al., 2025b; Yang & Bing, 2026); Innovation Prior builds it from the record around the paper.

6 Conclusion and Discussion

LLMs apply known methods well but rarely invent new ones, and harnesses and RL can only build on what the model already produces. What is missing is the reasoning by which scientists arrive at new methods, and it can be taught as a prior before RL. This prior gives RL a better starting point rather than answers to recall. With it, RL ends higher at both model sizes, and the model learns how a scientist thinks: (1) it weighs many methods before committing to one; (2) it reaches simple but effective methods from first principles; (3) it uses test-time compute better; and (4) it judges research better. The reasoning behind human discoveries thus transfers to AI research.

Limitations and future work. A model developer would add a prior of this kind during mid-training of a large model, at a cost infeasible for academic labs, so we approximate it with a small update to released 4B and 9B checkpoints, which remain behind large open models on absolute scores. Our evaluation is bounded by compute in the same way: the most telling tests, such as whether the model can propose a better pre-training algorithm, need training runs of their own that academic labs cannot afford. Both limits are limits of scale, so the next steps are to mid-train a larger model with the prior and to test it on research problems that only large experiments can check.

AI USE STATEMENT

We used generative AI tools under human supervision to assist in writing code, to polish the text, and to check its grammar. Coding agents are part of our method: Section 2.2 and Appendix D describe how Claude Code and Codex build and review each unit of INNOVATION PRIOR, and Appendix F describes how agents write and check the RL problems.

ETHICS STATEMENT

All authors have read and adhere to the ICLR Code of Ethics. This paper presents work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

REPRODUCIBILITY STATEMENT

Section 2.2 gives the workflow that builds INNOVATION PRIOR, Appendix D quotes the rules of the skills and prompts that the agents run with, and Appendices A and B give the candidate pools and the four kinds of source; Appendix C shows further traces together with the sources they draw on. Section 3.1 gives the models, the four arms, the benchmarks, the evaluation protocol of each and the contamination check, and Section 3.4 gives the protocol of test-time scaling. Appendix E lists every hyperparameter of fine-tuning, reinforcement learning and evaluation, together with the software frameworks and the hardware: all training runs on NVIDIA H200 GPUs, and evaluation runs on H200 or A100 GPUs. Appendix F describes the RL problem set: the problem formats and their deterministic checkers, the seeds, and the steps that turn a seed into a problem, with the gates every problem must pass. Appendix G lists the MLS-Bench tasks and the context and time limits of each run. For the analyses, Section 4.1 names the public sources of the research judgment tasks, Appendix H lists the keywords of every technique family and the rules that match them, and Appendix I recomputes every score in the case studies with the official tools of each benchmark and quotes model outputs verbatim. The supplementary material contains the training data of INNOVATION PRIOR, the RL problems with their checkers and reference solutions, and the code and model outputs of the analyses in Sections 4.1 and 4.2, and we will release them publicly.

REFERENCES

Syeda Nahida Akter, Shrimai Prabhumoye, Eric Nyberg, Mostofa Patwary, Mohammad Shoeybi, Yejin Choi, and Bryan Catanzaro. Front-loading reasoning: The synergy between pretraining and post-training data. In *The Fourteenth International Conference on Learning Representations*, 2026.

Anthropic. System card: Claude Opus 4 & Claude Sonnet 4. https://www.anthropic.com/claude-4-system-card, 2025.

Antonis Antoniades, Deepak Nathani, Ritam Saha, Alfonso Amayuelas, Ivan Bercovich, Zhaotian Weng, Vignesh Baskaran, Kunal Bhatia, and William Yang Wang. Heuresis: Search strategies for autonomous AI research agents across quality, diversity and novelty. *arXiv preprint arXiv:2606.25198*, 2026.

Martín Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. In *International Conference on Learning Representations*, 2017.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 214–223. PMLR, 2017.

Alejandro H. Artiles, Martin Weiss, Levin Brinkmann, Iyad Rahwan, Bernhard Schölkopf, Christopher Pal, Hugo Larochelle, Anirudh Goyal, and Nasim Rahaman. The alien space of science: Sampling coherent but cognitively unavailable research directions. *arXiv preprint arXiv:2603.01092*, 2026.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256, 2002.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *The Third International Conference on Learning Representations*, 2015.

Honglin Bao, Siyang Wu, Xiao Liu, Sida Li, Shiyun Cao, and James A. Evans. Contemporary AI lacks the imagination to diverge or negate in science. *arXiv preprint arXiv:2606.08251*, 2026.

Carlo Baronio, Pietro Marsella, Ben Pan, Simon Guo, and Silas Alberti. Kevin: Multi-turn RL for generating CUDA kernels. In *The Fourteenth International Conference on Learning Representations*, 2026.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signSGD: Compressed optimisation for non-convex problems. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 560–569. PMLR, 2018.

Andrew Bouras. CrossTrace: A cross-domain dataset of grounded scientific reasoning traces for hypothesis generation. *arXiv preprint arXiv:2603.28924*, 2026.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D. White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6(5):525–535, 2024.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pp. 213–229. Springer International Publishing, 2020.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *The Thirteenth International Conference on Learning Representations*, 2025.

Ziyu Chen, Yilun Zhao, and Arman Cohan. Measuring the gap between human and LLM research ideas. *arXiv preprint arXiv:2607.01233*, 2026.

Yuri Chervonyi, Trieu H. Trinh, Miroslav Olšák, Xiaomeng Yang, Hoang Nguyen, Marcelo Menegali, Junehyuk Jung, Junsu Kim, Vikas Verma, Quoc V. Le, and Thang Luong. Gold-medalist performance in solving olympiad geometry with AlphaGeometry2. *Journal of Machine Learning Research*, 26(241):1–39, 2025.

George B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In Tjalling C. Koopmans (ed.), *Activity Analysis of Production and Allocation*, pp. 339–347. John Wiley & Sons, 1951.

George B. Dantzig. Origins of the simplex method. Technical Report SOL 87-5, Systems Optimization Laboratory, Department of Operations Research, Stanford University, 1987.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, pp. 10088–10115, 2023.

Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

Arnaud Doucet and Adam M. Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. In Dan Crisan and Boris Rozovskii (eds.), *The Oxford Handbook of Nonlinear Filtering*, pp. 656–704. Oxford University Press, 2011.

Robert M. Fano. The transmission of information. Technical Report 65, Research Laboratory of Electronics, Massachusetts Institute of Technology, 1949.

Alex Gammerman, Volodya Vovk, and Vladimir Vapnik. Learning by transduction. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pp. 148–155, 1998.

Kanishk Gandhi, Denise H. J. Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (SoS): Learning to search in language. In *First Conference on Language Modeling*, 2024.

Kanishk Gandhi, Ayush K. Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective STaRs. In *Second Conference on Language Modeling*, 2025.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Ali E. Ghareeb, Benjamin Chang, Ludovico Mitchener, Angela Yiu, Caralyn J. Szostkiewicz, Dmytro Shved, Gavin J. Gyimesi, Jon M. Laurent, Samantha M. Wright, Muhammed T. Razzak, Andrew D. White, Silvia C. Finnemann, Michaela M. Hinks, and Samuel G. Rodriques. A multi-agent system for automating scientific discovery. *Nature*, 655(8122):497–505, 2026.

N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F Radar and Signal Processing*, 140(2):107–113, 1993.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, Anil Palepu, Keran Rong, Ryutaro Tanno, Khaled Saab, Fan Zhang, Jacob Blum, Andrew Carroll, Kavita Kulkarni, Nenad Tomašev, Dina Zverinski, Ivor Rendulic, Elahe Vedadi, Florian Hasler, Luka Rimanic, Marina Boia, Ivan Budiselic, Ben Feinstein, Mathias Bellaiche, Tom Sheffer, Jan Freyberg, Jeremy Ratcliff, Ottavia Bertolli, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Yuan Guan, Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R. D. Costa, José R. Penadés, Gary Peltz, Yossi Matias, James Manyika, Demis Hassabis, Yunhan Xu, Pushmeet Kohli, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Accelerating scientific discovery with Co-Scientist. *Nature*, 655(8122):487–496, 2026.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng

Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2555–2565. PMLR, 2019.

Runyuan He, Qiuyang Mang, Shang Zhou, Kaiyuan Liu, Hanchen Li, Huanzhi Mao, Qizheng Zhang, Zerui Li, Bo Peng, Lufeng Cheng, Tianfu Fu, Yichuan Wang, Wenhao Chai, Jingbo Shang, Alex Dimakis, Joseph E. Gonzalez, and Alvin Cheung. FrontierSmith: Synthesizing open-ended coding problems at scale. *arXiv preprint arXiv:2605.14445*, 2026.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural Networks for Machine Learning, Lecture 6e: rmsprop: Divide the gradient by a running average of its recent magnitude. Coursera lecture slides, https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, 2012.

C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

C. A. R. Hoare. Oral history of Sir Antony Hoare. Interviewed by Jonathan P. Bowen, Computer History Museum, reference X3698.2007, 2006.

Niklas Höpner, Leon Eshuijs, Dimitrios Alivanistos, Giacomo Zamprogno, and Ilaria Tiddi. Automatic evaluation metrics for artificially generated scientific research, 2025.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025.

Ziyao Huang, Weiwei Wu, Kui Wu, Wei-Bin Lee, and Jianping Wang. CALM: Co-evolution of algorithms and language model for automatic heuristic design. In *The Fourteenth International Conference on Learning Representations*, 2026.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, Ottavia Bertolli, Tom Zahavy, Amol Mandhane, Jessica Yung, Iuliya Beloshapka, Borja Ibarz, Vivek Veeriah, Lei Yu, Oliver Nash, Paul Lezeau, Salvatore Mercuri, Calle Sönne, Bhavik Mehta, Alex Davies, Daniel Zheng, Fabian Pedregosa, Yin Li, Ingrid von Glehn, Mark Rowland, Samuel Albanie, Ameya Velingker, Simon Schmitt, Edward Lockhart, Edward Hughes, Henryk Michalewski, Nicolas Sonnerat, Demis Hassabis, Pushmeet Kohli, and David Silver. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651(8106):607–613, 2026.

David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. ALE-Bench: A benchmark for long-horizon objective-driven algorithm engineering. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

Yoichi Ishibashi, Taro Yano, and Masafumi Oyamada. Effective harness engineering for algorithm discovery with coding agents. *arXiv preprint arXiv:2605.15221*, 2026.

Peter Jansen, Oyvind Tafjord, Marissa Radensky, Pao Siangliulue, Tom Hope, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Daniel S. Weld, and Peter Clark. CodeScientist: End-to-end semi-automated scientific discovery with code-based experimentation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 13370–13467, 2025.

Bo Jiang. HindSight: Evaluating LLM-generated research ideas via future impact. *arXiv preprint arXiv:2603.15164*, 2026.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. AIDE: AI-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Amirhossein Kazemnejad. Transformer architecture: The positional encoding. Blog post, https://kazemnejad.com/blog/transformer_architecture_positional_encoding/, 2019.

Peter Kirgis, Sayash Kapoor, Andrew Schwartz, Stephan Rabanser, David Africa, Konstantinos Voudouris, Viet Nguyen, Toby Pilditch, Magda Dubois, Harry Coppock, Cozmin Ududec, Nitya Nadgir, Matilda Orona, Tilman Bayer, Derrick Chan-Sew, Yue Ling, Abhishek Shetty, Helen Toner, Gillian Hadfield, Seth Lazar, Steve Newman, Shoshannah Tekofsky, Rishi Bommasani, and Arvind Narayanan. Can AI agents conduct open-ended AI research? Early evidence from two case studies. *arXiv preprint arXiv:2607.27191*, 2026.

Shrinidhi Kumbhar, Santosh Mashetty, Divij Handa, Kevin Coutinho, Siddharth Sambhaji Ghule, and Chitta Baral. COGTRL: Training LLMs for scientific discovery assistance using cognitive traces via reinforcement learning. In *The 2026 Conference on Empirical Methods in Natural Language Processing*, 2026.

Jeremy Kun. Optimism in the face of uncertainty: the UCB1 algorithm. *Math ∩ Programming* (blog), https://www.jeremykun.com/2013/10/28/optimism-in-the-face-of-uncertainty-the-ucb1-algorithm/, 2013.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. In *The Fourteenth International Conference on Learning Representations*, 2026.

Ning Li. The ideation bottleneck: Decomposing the quality gap between AI-generated and human economics research. *arXiv preprint arXiv:2604.03338*, 2026.

Shangzhan Li, Zefan Wang, Ye He, Yuxuan Li, Qi Shi, Jianling Li, Yonggang Hu, Wanxiang Che, Xu Han, Zhiyuan Liu, and Maosong Sun. AutoTriton: Automatic Triton programming with reinforcement learning in LLMs. *arXiv preprint arXiv:2507.05687*, 2025.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pp. 21–37. Springer International Publishing, 2016.

Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Weinan E, and Siheng Chen. ML-Master: Towards AI-for-AI via integration of exploration and reasoning. *arXiv preprint arXiv:2506.16499*, 2025.

Zexi Liu, Jingyi Chai, Xinyu Zhu, Shuo Tang, Rui Ye, Weiyu Ma, Bo Zhang, Lei Bai, and Siheng Chen. ML-Agent: Reinforcing LLM agents for autonomous machine learning engineering. In *Forty-third International Conference on Machine Learning*, 2026.

Renze Lou, Hanzi Xu, Sijia Wang, Jiangshu Du, Ryo Kamoi, Xiaoxin Lu, Jian Xie, Yuxuan Sun, Yusen Zhang, Jihyun Janice Ahn, Hongchao Fang, Zhuoyang Zou, Wenchao Ma, Xi Li, Kai Zhang, Congying Xia, Lifu Huang, and Wenpeng Yin. AAAR-1.0: Assessing AI's potential to assist research. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 40361–40383. PMLR, 2025.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Thang Luong and Edward Lockhart. Advanced version of Gemini with Deep Think officially achieves gold-medal standard at the International Mathematical Olympiad. https://deepmind.google/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/, 2025.

Bohan Lyu, Yucheng Yang, Siqiao Huang, Jiaru Zhang, Qixin Xu, Xinghan Li, Xinyang Han, Yicheng Zhang, Huaqing Zhang, Runhan Huang, Kaicheng Yang, Zitao Chen, Wentao Guo, Junlin Yang, Xinyue Ai, Wenhao Chai, Yadi Cao, Ziran Yang, Kun Wang, Dapeng Jiang, Huan ang Gao, Shange Tang, Chengshuai Shi, Simon S. Du, Max Simchowitz, Jiantao Jiao, Dawn Song, and Chi Jin. MLS-Bench: A holistic and rigorous assessment of AI systems on building better AI. *arXiv preprint arXiv:2605.08678*, 2026.

Qiuyang Mang, Wenhao Chai, Zhifei Li, Huanzhi Mao, Shang Zhou, Alexander Du, Hanchen Li, Shu Liu, Edwin Chen, Yichuan Wang, Xieting Chu, Zerui Cheng, Yuan Xu, Tian Xia, Zirui Wang, Tianneng Shi, Jianzhu Yao, Yilong Zhao, Qizheng Zhang, Charlie F. Ruan, Zeyu Shen, Kaiyuan Liu, Zhaoyang Hong, Alex Gu, Ziyi Zhang, Runyuan He, Dong Xing, Zerui Li, Zirong Zeng, Yige Jiang, Lufeng Cheng, Ziyi Zhao, Youran Sun, Suyang Zhong, Junpeng Wang, Donglin Li, Wenyuan Huang, Jialiang Gu, Wesley Kai Zheng, Wangmeiyu Zhang, Ruyi Ji, Xuechang Tu, Zihan Zheng, Zhaozi Wang, Zexing Chen, Jingbang Chen, Jialu Zhang, Aleksandra Korolova, Peter Henderson, Pramod Viswanath, Vijay Ganesh, Saining Xie, Zhuang Liu, Dawn Song, Sewon Min, Ion Stoica, Joseph E. Gonzalez, Jingbo Shang, and Alvin Cheung. FrontierCS: Evolving challenges for evolving intelligence. In *Forty-third International Conference on Machine Learning*, 2026.

Ludovico Mitchener, Angela Yiu, Benjamin Chang, Mathieu Bourdenx, Tyler Nadolski, Arvis Sulovari, Eric C. Landsness, Daniel L. Barabasi, Siddharth Narayanan, Nicky Evans, Shriya Reddy, Martha Foiani, Aizad Kamal, Leah P. Shriver, Fang Cao, Asmamaw T. Wassie, Jon M. Laurent, Edwin Melville-Green, Mayk Caldas, Albert Bou, Kaleigh F. Roberts, Sladjana Zagorac, Timothy C. Orr, Miranda E. Orr, Kevin J. Zwezdaryk, Ali E. Ghareeb, Laurie McCoy, Bruna Gomes, Euan A. Ashley, Karen E. Duff, Tonio Buonassisi, Tom Rainforth, Randall J. Bateman, Michael Skarlinski, Samuel G. Rodriques, Michaela M. Hinks, and Andrew D. White. Kosmos: An AI scientist for autonomous discovery. *arXiv preprint arXiv:2511.02824*, 2025.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, 2025.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of GPT-4. *arXiv preprint arXiv:2306.02707*, 2023.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan Ö. Arık, and Tomas Pfister. MLE-STAR: Machine learning engineering agent via search and targeted refinement. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Mikhail Plekhanov, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob Nicolaus Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. MLGym: A new framework and benchmark for advancing AI research agents. In *Second Conference on Language Modeling*, 2025.

Jerzy Neyman and Egon Sharpe Pearson. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 231(694–706):289–337, 1933.

Alexander Novikov, 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. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

OpenAI. GPT-5 system card. https://cdn.openai.com/gpt-5-system-card.pdf, 2025a.

OpenAI. OpenAI o3 and o4-mini system card. https://openai.com/index/o3-o4-mini-system-card/, 2025b.

OpenAI, Ahmed El-Kishky, Alexander Wei, Andre Saraiva, Borys Minaiev, Daniel Selsam, David Dohan, Francis Song, Hunter Lightman, Ignasi Clavera, Jakub Pachocki, Jerry Tworek, Lorenz Kuhn, Lukasz Kaiser, Mark Chen, Max Schwarzer, Mostafa Rohaninejad, Nat McAleese, o3 contributors, Oleg Mürk, Rhythm Garg, Rui Shu, Szymon Sidor, Vineet Kosaraju, and Wenda Zhou. Competitive programming with large reasoning models. *arXiv preprint arXiv:2502.06807*, 2025.

R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

Bernardino Romera-Paredes, 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. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Kai Ruan, Xuan Wang, Jixiang Hong, Peng Wang, Yang Liu, and Hao Sun. Evaluating LLMs' divergent thinking capabilities for scientific idea generation with minimal context. *arXiv preprint arXiv:2412.17596*, 2024.

Louise Ryan. A conversation with Nan Laird. *Statistical Science*, 30(4):582–596, 2015.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent Laboratory: Using LLM agents as research assistants. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 5977–6043, 2025.

Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(12):371–421, 2008.

Cosma Shalizi. The shadow price of power. *Three-Toed Sloth* (blog), http://bactra.org/weblog/630.html, 2009.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52(4):R2493–R2496, 1995.

Peter W. Shor. The early days of quantum computation. arXiv:2208.09964, 2022.

Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. Can LLMs generate novel research ideas? A large-scale human study with 100+ NLP researchers. In *The Thirteenth International Conference on Learning Representations*, 2025.

Chenglei Si, Tatsunori Hashimoto, and Diyi Yang. The ideation-execution gap: Execution outcomes of LLM-generated versus human research ideas. In *The Fourteenth International Conference on Learning Representations*, 2026a.

Chenglei Si, Zitong Yang, Yejin Choi, Emmanuel Candès, Diyi Yang, and Tatsunori Hashimoto. Towards execution-grounded automated AI research. In *Forty-third International Conference on Machine Learning*, 2026b.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron T Parisi, Abhishek Kumar, Alexander A Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Fathy Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura A Culp, Lechao Xiao, Maxwell Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024.

Soumitra Sinhahajari, Navonil Majumder, and Soujanya Poria. On the limits of LLM-as-Judge for scientific novelty assessment. *arXiv preprint arXiv:2606.12071*, 2026.

Anja Šurina, Amin Mansouri, Lars C. P. M. Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. Algorithm discovery with LLMs: Evolutionary search meets reinforcement learning. In *Second Conference on Language Modeling*, 2025.

Kyle Swanson, Wesley Wu, Nash L. Bulaong, John E. Pak, and James Zou. The Virtual Lab of AI agents designs new SARS-CoV-2 nanobodies. *Nature*, 646(8085):716–723, 2025.

Nathan J. Szymanski, Bernardus Rendy, Yuxing Fei, Rishi E. Kumar, Tanjin He, David Milsted, Matthew J. McDermott, Max Gallant, Ekin Dogus Cubuk, Amil Merchant, Haegyeom Kim, Anubhav Jain, Christopher J. Bartel, Kristin Persson, Yan Zeng, and Gerbrand Ceder. An autonomous laboratory for the accelerated synthesis of inorganic materials. *Nature*, 624(7990): 86–91, 2023.

Yixuan Tang and Yi Yang. AI research agents narrow scientific exploration. *arXiv preprint arXiv:2605.27905*, 2026.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Allyson Ettinger, Michal Guerquin, David Heineman, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Jake Poznanski, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 furious, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Zengzhi Wang, Fan Zhou, Xuefeng Li, and Pengfei Liu. OctoThinker: Mid-training incentivizes reinforcement learning scaling. *arXiv preprint arXiv:2506.20512*, 2025.

Zhilin Wang, Yafu Li, Shunkai Zhang, Zhi Wang, Haoran Zhang, Xiaoye Qu, and Yu Cheng. New skills or sharper primitives? A probabilistic perspective on the emergence of reasoning in RLVR. *arXiv preprint arXiv:2602.08281*, 2026.

Koki Wataoka, Tsubasa Takahashi, and Ryokan Ri. Self-preference bias in LLM-as-a-Judge. In *NeurIPS Safe Generative AI Workshop 2024*, 2024.

Yixuan Weng, Minjun Zhu, Guangsheng Bao, Hongbo Zhang, Jindong Wang, Yue Zhang, and Linyi Yang. CycleResearcher: Improving automated research via automated review. In *The Thirteenth International Conference on Learning Representations*, 2025.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 23965–23998. PMLR, 2022.

Ting Wu, Xuefeng Li, and Pengfei Liu. Progress or regress? Self-improvement reversal in post-training. In *The Thirteenth International Conference on Learning Representations*, 2025.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652, 2024.

Sherry Yang, Joy He-Yueya, and Percy Liang. Reinforcement learning for machine learning engineering agents. In *The Fourteenth International Conference on Learning Representations*, 2026.

Zonglin Yang and Lidong Bing. MOOSE-Star: Unlocking tractable training for scientific discovery by breaking the complexity barrier. In *Forty-third International Conference on Machine Learning*, 2026.

Zonglin Yang, Wanhao Liu, Ben Gao, Tong Xie, Yuqiang Li, Wanli Ouyang, Soujanya Poria, Erik Cambria, and Dongzhan Zhou. MOOSE-Chem: Large language models for rediscovering unseen chemistry scientific hypotheses. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems*, volume 37, pp. 43571–43608, 2024.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. LIMO: Less is more for reasoning. In *Second Conference on Language Modeling*, 2025.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel Agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913, 2025.

Jiakang Yuan, Xiangchao Yan, Bo Zhang, Tao Chen, Botian Shi, Wanli Ouyang, Yu Qiao, Lei Bai, and Bowen Zhou. Dolphin: Moving towards closed-loop auto-research through thinking, practice, and feedback. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21768–21789, 2025.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488, 2022.

Charlie Zhang, Graham Neubig, and Xiang Yue. On the interplay of pre-training, mid-training, and RL on reasoning language models. In *Forty-third International Conference on Machine Learning*, 2026a.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS*: LLM self-training via process reward guided tree search. In *Advances in Neural Information Processing Systems*, volume 37, pp. 64735–64772, 2024.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin Gödel Machine: Open-ended evolution of self-improving agents. In *The Fourteenth International Conference on Learning Representations*, 2026b.

Yunxiang Zhang, Muhammad Khalifa, Shitanshu Bhushan, Grant D Murphy, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. MLRC-Bench: Can language agents solve machine learning research challenges? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

Zhengxin Zhang, Ning Wang, Sainyam Galhotra, and Claire Cardie. How far are we from true auto-research? *arXiv preprint arXiv:2605.19156*, 2026c.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583, 2024a.

Xiaosen Zheng, Tianyu Pang, Chao Du, Qian Liu, Jing Jiang, and Min Lin. Cheating automatic LLM benchmarks: Null models achieve high win rates. In *The Thirteenth International Conference on Learning Representations*, 2025.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 400–410, 2024b.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. LIMA: Less is more for alignment. In *Advances in Neural Information Processing Systems*, volume 36, pp. 55006–55021, 2023.

A EXAMPLES FROM THE CANDIDATE POOLS

Section 2.1 draws the candidate methods for discoveries from four pools, and Table 3 lists typical examples from each.

Table 3: Typical discoveries in INNOVATION PRIOR from each candidate pool.

Pool	**Examples**
Baselines of widely used machine-learning benchmarks	Adam, dropout, batch normalization, layer normalization, residual networks, U-Net, word2vec, BERT, DQN, PPO, generative adversarial nets, variational autoencoders, LoRA
Catalogs of classic algorithms	The simplex method, quicksort, Dijkstra's shortest paths, the fast Fourier transform, Huffman coding, the Viterbi algorithm, max-flow min-cut, the Miller–Rabin primality test, PageRank
Best papers and landmark results across the sciences	The Dirac equation, Noether's theorem, BCS superconductivity, Kohn–Sham density functional theory, the renormalization group, the Metropolis–Hastings algorithm, the expectation–maximization algorithm, Shannon's channel coding theorem, RSA, Shor's factoring algorithm
Olympiad solutions	Vieta jumping for IMO 1988 Problem 6, the roots-of-unity filter for IMO 1995 Problem 6, Boreico's sum-of-squares solution for IMO 2005 Problem 3, the windmill invariant for IMO 2011 Problem 2, centroid decomposition for IOI 2011 Race, Lagrangian relaxation for IOI 2016 Aliens

B EXAMPLES OF THE FOUR KINDS OF SOURCE

Section 2.1 names four kinds of source that keep what a paper omits. This appendix gives examples of each kind, listed in Table 4, from methods in INNOVATION PRIOR whose traces use that source. Each example gives a step of the method and what its paper says about the step, quotes the source, and states what the paper alone does not tell, as tagged beside its heading: the reasoning or insight behind the step, an abandoned attempt, a hidden detail on which the method depends, or the order in which the ideas arose. Quotations are verbatim, with [. . .] marking omitted text.

Table 4: Examples of each kind of source and the content it supplies that the method's paper lacks.

Kind	Method	Source	What the paper alone lacks
Ancestor	Huffman coding	Fano's 1949 report	why top-down splits fall short
	RSA	Diffie and Hellman 1976	the failed matrix scheme
	Wasserstein GAN	Arjovsky and Bottou 2017	how fast the gradient vanishes
	Conformal prediction	Gammerman et al. 1998	the special case it generalizes
Code	QLoRA	bitsandbytes commits	the cutoff of the NF4 grid
	SSD	the authors' Caffe branch	the target scaling and its date
	PlaNet	the released planner	how the planner refits its spread
Explanation	Bootstrap particle filter	Doucet and Johansen 2011	why resampling is needed
	Neyman–Pearson lemma	Shalizi 2009	why a ratio, not a difference
	signSGD	Hinton's 2012 lecture	why signs fail on minibatches
	Sinusoidal positions	Kazemnejad 2019	why the frequencies are geometric
	UCB1	Kun 2013	why act on the upper bound
Author	Simplex method	Dantzig 1987	the edge walk first rejected
	Quicksort	Hoare's oral history	the motivating problem, bubble sort
	EM algorithm	Laird's interview	the special case that came first
	DETR	Carion's issue reply	fixed target orders, dropped
	Shor's nine-qubit code	Shor 2022	the failed first code

B.1 ANCESTOR WORKS

Ancestor works show the problem the authors faced and the approaches others had already tried, and so rebuild the context in which the method was found.

Huffman coding (Huffman, 1952), and Fano's report (Fano, 1949) reasoning

Huffman's paper dismisses the earlier codes of Shannon and Fano in one sentence: "Their coding procedures are not optimum, but approach the optimum behavior when N approaches infinity." Fano's own report builds a code by splitting the messages top-down into groups of nearly equal probability, and states where that procedure gets stuck.

> **Fano 1949, p. 18.** "Of course, we wish to perform all divisions in such a way as to obtain at the end the most efficient code. Unfortunately, no general rule could be found for determining at once how the divisions should be made in doubtful cases in order to obtain maximum code efficiency."

Not in the paper. The paper says that the earlier codes are not optimal but not why. Fano's report locates the difficulty in how to divide the messages in close cases, the top-down decision that Huffman's bottom-up merging of the two least probable messages never has to make.

RSA (Rivest et al., 1978), and Diffie and Hellman (Diffie & Hellman, 1976) reasoning

RSA credits Diffie and Hellman with the concept alone: they "introduced the concept of trap-door one-way functions but did not present any examples." Their paper had in fact tried a concrete public-key scheme, encryption by an invertible matrix, and worked out why it fails.

> **Diffie and Hellman 1976, p. 648.** "Unfortunately, matrix inversion takes only about n^3 operations. The ratio of 'cryptanalytic' time (i.e., computing D from E) to enciphering or deciphering time is thus at most n, and enormous block sizes would be required to obtain ratios of 10^6 or greater."

Not in the paper. The paper tells a reader only that no example existed. The ancestor turns the search into a quantitative requirement: the attacker's work must exceed the user's by far more than a polynomial factor, which is what the difficulty of factoring is meant to supply.

Wasserstein GAN (Arjovsky et al., 2017), and Arjovsky & Bottou (2017) insight

WGAN cites the vanishing gradient of the original GAN objective in one clause, pointing to "Theorem 2.4 of (Arjovsky & Bottou, 2017)." That earlier paper, by two of the same authors, proves the bound, measures the decay on a real model and points to an alternative.

> **Arjovsky and Bottou 2017, pp. 7 and 12.** "We see the gradient norms decay quickly, in the best case 5 orders of magnitude after 4000 discriminator iterations." [...] "We could argue to use the JSD of the noisy variants to measure a similarity between the original distributions, but this would depend on the amount of noise, and is not an intrinsic measure [...]. Luckilly [sic], there are alternatives."

Not in the paper. The paper treats the vanishing gradient as a known fact. The ancestor shows how fast the gradient vanishes on a real model and that a distance independent of the added noise was already being sought, which is what the Wasserstein distance provides.

Conformal prediction (Shafer & Vovk, 2008), and Gammerman et al. (1998) insight

The conformal prediction tutorial treats support vectors as one of many possible nonconformity scores, noting that the Lagrange multipliers of a support-vector machine "can serve as nonconformity scores." It does not cite the 1998 transductive method, by one of its own authors, in which the same construction appears with a single fixed test of strangeness.

> **Gammerman et al. 1998, p. 149.** "Our method works well (gives predictions of small incertitude or, in other words, confident predictions) when the number of support vectors is small in both pictures."

Not in the paper. The 1998 paper shows the order in which the idea arose: confidence was first read off the count of support vectors, which works only when they are few, and the general method replaces that one test with any symmetric score.

B.2 Code and its development history

Released code fixes the constants and steps that the paper leaves out, and the dated history of a repository keeps the versions that came before.

QLoRA (Dettmers et al., 2023), and the first author's library hidden detail

QLoRA's 4-bit NormalFloat type places its levels at quantiles of a standard normal distribution, $q_i = \frac{1}{2}\big(Q_X(\frac{i}{2^k+1}) + Q_X(\frac{i+1}{2^k+1})\big)$ in Eq. 4 of the paper. At $i = 0$ the formula needs $Q_X(0) = -\infty$, so the published table depends on where the grid is cut off, and the paper gives no value. The first author's library, bitsandbytes, sets it.[1]

bitsandbytes, commit 4ad999d, 2 April 2023.

```
-def create_normal_map(offset=0.966666, use_extra_value=True):
+def create_normal_map(offset=0.9677083, use_extra_value=True):
```

Not in the paper. The cutoff is a tail probability of 0.9677083, and the history shows that this value replaced 0.966666 half an hour after it was first written. Neither value appears in the paper.

SSD (Liu et al., 2016), and the authors' Caffe branch hidden detail

SSD regresses box offsets relative to default boxes: "we regress to offsets for the center (cx, cy) of the default bounding box (d) and for its width (w) and height (h)." The first author's official implementation rescales these targets before the loss, and its history dates the rescaling.[2]

weiliu89/caffe, commit b5ddd78, 20 March 2016.

```
code_type = P.PriorBox.CENTER_SIZE
# variance used to encode/decode prior bboxes.
if code_type == P.PriorBox.CENTER_SIZE:
prior_variance = [0.1, 0.1, 0.2, 0.2]
```

[1] Commit `4ad999d` of https://github.com/bitsandbytes-foundation/bitsandbytes.
[2] Commit `b5ddd78` on the `ssd` branch of https://github.com/weiliu89/caffe.

Not in the paper. The code divides the center offsets by 0.1 and the log-size offsets by 0.2, a rescaling that Eq. 2 of the paper never mentions. The same commit replaced an encoding of box corners with the center-size encoding. The first arXiv version of the paper still regressed corners, and the center-size form appears in the second, ten days after this commit.

PlaNet (Hafner et al., 2019), and the released planner

hidden detail

PlaNet plans with the cross-entropy method, refitting a distribution over action sequences to the best candidates at every iteration. Algorithm 2 of the paper refits the spread as a mean absolute deviation, $\sigma_{t:t+H} = \frac{1}{K-1} \sum_{k \in \mathcal{K}} |a^{(k)}_{t:t+H} - \mu_{t:t+H}|$.[3]

google-research/planet, planning.py.

```
mean, variance = tf.nn.moments(best_actions, 1)
stddev = tf.sqrt(variance + 1e-6)
```

Not in the paper. Instead, the released planner takes the standard deviation of the best action sequences, with a floor on their variance, and clips the sampled actions to their bounds, a step absent from the paper's algorithm.

B.3 Third-party explanations

Explanations are written to teach, so they often say why a step is taken where the paper only takes it.

Bootstrap particle filter (Gordon et al., 1993), and Doucet & Johansen (2011)

reasoning

The bootstrap filter resamples its particles at every step. The paper justifies the step by an asymptotic result, "They show that Bayes theorem can be implemented as a weighted bootstrap," and does not consider the alternative of keeping the particles and carrying their weights forward.

Doucet and Johansen 2011, Sec. 3.3. "the variance increases exponentially with n even in this simple case. For example, if we select $\sigma^2 = 1.2$ then we have a reasonably good importance distribution [...] We would need to use $N \approx 2 \times 10^{23}$ particles to obtain a relative variance [...] $= 0.01$."

Not in the paper. The paper alone backs resampling only by a limit theorem. The tutorial shows what happens without it: even a nearly correct proposal makes the variance of the weights grow exponentially in time, so resampling is necessary, not optional.

Neyman–Pearson lemma (Neyman & Pearson, 1933), and Shalizi (2009)

insight

The lemma chooses the rejection region by the likelihood ratio. The paper obtains it from the calculus of variations, as a condition on the boundary of the best region: "we must have $p_0(x_1, x_2, \dots x_n) = kp_1(x_1, x_2, \dots x_n)$."

Shalizi 2009. "What is the marginal benefit of expanding R to include a small neighborhood around the point x? Just, by the definition of 'probability density', $q(x)$. The marginal cost is likewise $tp(x)$. We should include x in R if $q(x) > tp(x)$, or $q(x)/p(x) > t$. [...] that is why we need the likelihood *ratio* and not the likelihood *difference*"

Not in the paper. The variational argument yields the boundary but no reason for a ratio. The explanation reads the size of the test as a budget: each point costs its probability under the null and buys its probability under the alternative, so points enter the region by benefit per unit cost, and the threshold is the price of power.

signSGD (Bernstein et al., 2018), and Hinton's lecture (Hinton et al., 2012)

reasoning

signSGD updates each weight by the sign of its stochastic gradient. The paper cites Rprop as a successful precedent and names the difficulty only in general terms: "The sign of the stochastic gradient is a biased approximation to the true gradient."

Hinton et al. 2012, Lecture 6e. "Consider a weight that gets a gradient of +0.1 on nine mini-batches and a gradient of -0.9 on the tenth mini-batch. We want this weight to stay roughly where it is. rprop would increment the weight nine times and decrement it once by about the same amount [...]. So the weight would grow a lot."

[3] File `planet/control/planning.py` of https://github.com/google-research/planet.

Not in the paper. The lecture's example makes the difficulty concrete: the mean gradient is zero, yet a sign rule moves the weight eight steps in one direction, which is the failure a sign method has to overcome on minibatches.

Sinusoidal position encoding (Vaswani et al., 2017), and Kazemnejad (2019) insight

The Transformer encodes position with sinusoids whose "wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$." The paper justifies sinusoids by relative positions, a shift being a linear function of the encoding, and says nothing about the spacing.

> **Kazemnejad 2019.** "The LSB bit is alternating on every number, the second-lowest bit is rotating on every two numbers, and so on. But using binary values would be a waste of space in the world of floats. So instead, we can use their float continous [sic] counterparts - Sinusoidal functions."

Not in the paper. The explanation supplies the spacing: each dimension acts as one digit of a binary counter that changes at half the rate of the digit below, and sinusoids are the counter's continuous version.

UCB1 (Auer et al., 2002), and Kun (2013) insight

UCB1 plays the arm with the largest index, which the paper defines as the average reward plus a term "related to the size [...] of the one-sided confidence interval for the average reward."

> **Kun 2013.** "[...] the principle of optimism in the face of uncertainty. [...] If our guess is wrong, then our optimistic guess will quickly decrease and we'll be compelled to switch to a different action. [...] we'll need to ensure that we don't rule out good actions that fare poorly early on."

Not in the paper. The paper defines the index but not why the upper end of the interval is the one to act on. The explanation gives the reason: an optimistic estimate corrects itself once it is acted on, whereas a greedy rule can abandon a good arm that starts unluckily.

B.4 Self-accounts

The authors' later accounts recall where the work started, what they tried first, and why they dropped it.

Simplex method (Dantzig, 1951), and Dantzig (1987) abandoned attempt

Dantzig's 1951 chapter presents the simplex method in the geometry of the columns: "Each column of coefficients in (3) may be viewed as representing the coordinates of a point in Euclidean R_m space."

> **Dantzig 1987, pp. 1 and 9.** "the idea of step by step descent [...] along edges of the convex polyhedral set from one vertex to an adjacent one. I rejected this algorithm outright on intuitive grounds — it had to be inefficient because it proposed to solve the problem by wandering along some path of outside edges until the optimal vertex was reached." [...] "Of course, I soon observed that what appeared in the column geometry to be a new algorithm was, in the row geometry, the vertex descending algorithm that I had rejected earlier."

Not in the paper. The paper gives no hint that the method was first conceived as a walk along the edges of the feasible polyhedron and rejected as obviously inefficient, nor that it was accepted only when the same procedure reappeared in the column geometry.

Quicksort (Hoare, 1962), and Hoare's oral history (Hoare, 2006) abandoned attempt

Hoare's paper presents Quicksort as a general method: "A description is given of a new method of sorting in the random-access store of a computer."

> **Hoare 2006, p. 11.** "it paid to sort the words of the sentence into the same alphabetical order before consulting the dictionary so that you could look up all the words in the sentence on a single pass of the magnetic tape." [...] "After a few moments, I thought of the obvious algorithm, which is now called bubble sort, and rejected that, because that was obviously rather slow, and thought of Quicksort as the second thing I thought of."

Not in the paper. The paper names no application and no rejected alternative. The interview gives both: the method was built to sort the words of a sentence for a Russian–English dictionary stored on magnetic tape, and bubble sort was considered first and dropped as too slow.

EM algorithm (Dempster et al., 1977), and Laird's interview (Ryan, 2015)

order of steps

The EM paper opens in full generality, as "a general approach to iterative computation of maximum-likelihood estimates when the observations can be viewed as incomplete data," and treats variance components as one example among several. Laird, its second author, recalls how the work began with Dempster, its first.

> **Ryan 2015.** "I was working with Art on a random effects approach to smooth two-way contingency tables. [...] I asked Art about estimating the variance component; he said, 'Why don't you just use that substitution algorithm?' [...] Art said, 'Well, this derivation will work for any exponential family likelihood with any kind of missing data.' "

Not in the paper. The paper runs from the general definition to the examples. The interview gives the order in which the method was found: a single variance component in a model for smoothing tables, the observation that its maximum-likelihood estimate is a fixed point of a substitution algorithm, and only then the generalization to any exponential family with missing data.

DETR (Carion et al., 2020), and the first author's issue reply

abandoned attempt

DETR matches its predictions to the ground truth with the Hungarian algorithm, and the paper states the need as a requirement: "the loss function should be invariant by a permutation of the predictions." A user of the official repository asked whether the matching is necessary, and the first author answered.[4]

> **Carion, GitHub issue 73, 2020.** "We did try a few fixed orderings, including arbitrary fixed order (eg the dataset order), as well as a few lexicographical orders (sorting by position, size, class, ...), but all these attempts performed worse." [...] "In coco for ex, there is no image with more than ~75 objects. With a fixed ordering, it means that the object queries in [75-100] will never be used."

Not in the paper. The reply shows that the matching was chosen after the cheaper alternative failed: fixed orders of the targets were tried and all did worse, for reasons such as queries late in a fixed order never receiving a target.

Shor's nine-qubit code (Shor, 1995), and Shor's memoir (Shor, 2022)

abandoned attempt

Shor's paper describes the finished nine-qubit code as a repetition code used twice: "once in an outer layer (repeating the triplet of qubits three times) and once in an inner layer (using $|000\rangle \pm |111\rangle$ for each triplet)."

> **Shor 2022, p. 6.** "This code corrects bit errors, but it makes phase errors three times as likely. What I realized is that there's a transform on quantum codes, which we now call the Hadamard transformation, that takes bit errors to phase errors and vice versa."

Not in the paper. The paper does not say why the inner layer takes this form, and neither phase errors nor the Hadamard transform is named in it. The memoir gives the route: the three-qubit repetition code fails on phase errors, the Hadamard transform exchanges the two kinds of error, and concatenating the two codes gives the nine-qubit code.

[4] https://github.com/facebookresearch/detr/issues/73

C More Traces and Their Sources

Figure 5 shows one trace with the sources behind it, and this appendix shows four more. In each subsection, the left column condenses part of a trace from Innovation Prior and the right column quotes its sources. A highlighted passage rests on a source and is tagged by the kind of source: A an ancestor work, C code, E a third-party explanation, S a self-account, and P the method's own paper. Unhighlighted text works out the reasoning between the sourced steps, and gray brackets summarize omitted parts of the trace. Quotations from sources are verbatim, with [. . .] marking omitted text.

C.1 Kohn–Sham density functional theory and Kohn's Nobel lecture

In a single sentence, the 1965 paper splits what remains of the universal functional after the Hartree term into $T_s[n]+E_{xc}[n]$, defining E_{xc} as the remainder. Only after the derivation does it remark that a local approximation to T_s would bring back two defects of Thomas–Fermi theory. It says neither that this lesson about the kinetic energy came first nor that the split answered a task Kohn had set for himself and Sham: to extract the Hartree equations from the Hohenberg–Kohn variational principle. Both appear in Kohn's Nobel lecture thirty-four years later.

Tag	Type	Source and what it supplies
S1	self-account	Kohn (1999), Nobel Lecture: the kinetic-energy lesson from Thomas–Fermi and Hartree; the extraction task in his own words; the route through non-interacting electrons; the exchange-correlation hole and its sum rule.
P1	primary	Hohenberg & Kohn (1964); Kohn & Sham (1965): theorems and equations, fourteen pages combined.
E1	explainer	Argaman & Makov (2000), Am. J. Phys. 68:69–79: the regime in which LDA is justified, and the argon atom as a case far outside it.

[… the trace first trades the wavefunction for the density, locates the failure of Thomas–Fermi theory in its local kinetic energy, and proves on the page, by reductio, that the density fixes the external potential. …]

S1 $F[n]$ is defined but unknown, and unpacking the definition sends me back to $3N$-dimensional wavefunctions. So the game is: find a computable approximation to $F[n]$.

S1 If I'm careless I rebuild Thomas–Fermi: approximate $T[n]$ by the local gas value and I am back to no binding. The lesson is loud: do not model the kinetic energy by a local density functional. That single term is what killed the cheap theory.

So what do I do with the kinetic energy instead? In the orbital-based single-particle theory it is computed exactly, as the sum of orbital kinetic energies. S1 A fictitious system of non-interacting electrons, tuned to have the same density n, has an easy and exact kinetic energy $T_s[n]$. Split off $T_s[n]$ and the classical Hartree term; shovel the rest into $E_{xc}[n]$.

[… the trace then derives the self-consistent equations on the page, by matching the Euler–Lagrange equation of the split functional to that of an auxiliary non-interacting system. …]

P1 What falls out is a set of self-consistent single-particle equations: orbitals from an effective potential v_{eff}, the density from the N occupied orbitals, iterate to convergence. S1 Drop E_{xc} and v_{xc} and I recover the orbital-based self-consistent equations exactly.

[… the trace derives the total energy and takes E_{xc} locally from the uniform electron gas, the LDA. …]

E1 Atoms are about as far from slowly varying as an electronic system gets; by the stated logic LDA has no business working for an argon atom. […] S1 E_{xc} is, formally exactly, the interaction of each electron with its own coupling-averaged exchange-correlation hole, which integrates to -1. The LDA hole, the hole of the uniform gas, has exactly that normalization. Even with the wrong shape it has the right charge.

S1 · Kohn 1999. "I knew that the Hartree equations described atomic groundstates much better than TF theory. The difference between them lay in the different treatments of the kinetic energy T… I set ourselves the task of extracting the Hartree equations from the HK variational principle… which I knew to be formally exact and which therefore had to have the Hartree equations and improvements 'in them'."

E1 · Argaman & Makov 2000. "The LDA has been shown to give very good results for many atomic, molecular and crystalline interacting electron systems, even though in these systems the density of electrons is not slowly varying. As an example, we show in Fig. 4 the solution for an atom of Argon."

P1 · what the primaries give. The HK uniqueness theorem proved by reductio; the KS equations, with T_s and E_{xc} introduced as definitions and LDA as "our sole approximation." Only after the derivation does KS note that a local T_s would bring back two Thomas–Fermi defects. The extraction task appears in neither paper.

C.2 The primal–dual Steiner forest algorithm and a later survey by one of its authors

The 2-approximation grows the dual variables of all active components simultaneously and uniformly. The 1995 paper states this rule as a definition, and Williamson's survey, written by one of its authors, motivates it with a star instance on which growing one set at a time gives only a k-approximation.

Tag	Type	Source and what it supplies
S1	co-author survey	Williamson (2002): the reverse-delete step, the star counterexample, the $2k/(k+1)$ averaging and the uniform-increase idea, presented as successive modifications of the basic primal-dual algorithm.
P1	primary	Goemans & Williamson (1995): the algorithm and the theorem, definition-first.

[… the trace first recasts feasibility as hitting every cut that separates a requested pair, writes the cut LP and its dual as a packing of moats, and notes that any feasible dual lower-bounds the optimum, so the dual can be grown instead of solved. …]

The textbook move is: pick one violated set, grow its moat until an edge goes tight, buy it, repeat. Let me test that move on a small instance before trusting it.

S1 Take one common source s and k sinks, each reachable by one edge of cost 1. Pick $\{s\}$ and grow; its moat crosses all k star edges at once, all go tight together, and I buy all k, charged to a single moat of width 1. Cost k, dual 1: a k-approximation. Wall.

But look again at the same picture. Those edges look expensive only because I charge them to the one moat I happened to grow. S1 Every star edge crosses exactly two of the $k{+}1$ violated sets, so total crossings $= 2k$ over $k{+}1$ moats: an average of $2k/(k{+}1) < 2$. So don't pick one violated set: grow every active moat at once, uniformly, and each shares the cost.

The algorithm follows: P1 grow every active component by the same ε, buy the tight edge, merge, recompute who is active; S1 then reverse-delete so the kept edges form a minimal augmentation.

S1 · Williamson 2002. "…$|D \cap T| = k$. Thus the algorithm and analysis we have developed so far would only give a k-approximation… averaged over the number of minimal violated sets… leading to an average of $2k/(k{+}1) \approx 2$. **This leads to the following idea: suppose we choose multiple violated sets and increase their dual variables simultaneously and uniformly.**"

P1 · what the primary gives. "Define an active component to be any component C of F for which $f(C) = 1$. In each iteration the algorithm tries to increase y_C uniformly…" Stated as a definition. The star and its $2k/(k{+}1)$ count appear nowhere; averaging enters only in the proof, as the average degree of a forest. The final cleanup drops every unneeded edge at once, not in reverse order.

C.3 DeBERTa and a paragraph commented out of its source

DeBERTa's enhanced mask decoder adds absolute positions only at the top of the network, after all Transformer layers and before the softmax that predicts the masked tokens, and the paper motivates this placement by contrast with BERT, which adds them at the input. The paper's LaTeX source on arXiv also holds a paragraph that the authors commented out: to predict a masked but unchanged token from its position without its own content, it names XLNet's two-stream attention as a feasible solution and sets it aside for its memory and compute cost. The paragraph is in the source of all six arXiv versions but in the rendered text of none of them, and not in the ICLR 2021 paper. Our trace weighs the same option when it builds the decoder and sets it aside for the same reason.

Tag	Type	Source and what it supplies
S1	authors' draft	Commented-out paragraph in the paper's own LaTeX source, `arxiv.org/e-print/2006.03654`, present in all six versions: XLNet's two-stream attention as a feasible solution, set aside for its memory and compute cost.
P1	primary	DeBERTa, ICLR 2021: the enhanced mask decoder, set against a single alternative, absolute positions at the input as in BERT.

S1 So I need the decoder to see **where** the masked token sits without seeing **what** it is. The move that is already on the table for exactly this problem is XLNet's two-stream attention: run a content stream and a query stream side by side, the query stream carrying position but not content. S1 It would work, but it costs memory and compute: it roughly doubles what the encoder has to hold and compute at every layer.

[... the trace grants that XLNet needs the second stream at every depth, because its autoregressive factorization requires a content-blind representation at each layer, whereas here the encoder output already carries the full context. ...]

I only need that mechanism once, at the very end of the stack; running it through every layer is a whole-model tax for a decoder-shaped problem. So: keep one stream. P1 Inject the absolute positions once, late, as one extra step on top of the encoder: the relative positions have already done their work in the layers below, and absolute position enters only where the masked tokens are decoded.

S1 · authors' commented-out draft. "%To make prediction of p_θ with the consideration of position P_i but without the content itself, one feasible solution is to adopt a two-stream attention as XLNet\cite{yang2019xlnet}. **However, this will increase memory consumption and computational complexity.** Instead, we use a different approach to solve this problem without introducing eligible cost."

`3_approach.tex`, line 322 in v6, repeated in `appendix.tex`, line 23; present in the source of v1 to v6.

P1 · what the primary gives. "There are two methods of incorporating absolute positions. The BERT model incorporates absolute positions in the input layer. In DeBERTa, we incorporate them right after all the Transformer layers but before the *softmax* layer for masked token prediction..."

Two-stream attention and its cost appear in the rendered text of no arXiv version, v1 to v6, and of neither ICLR 2021 PDF.

C.4 SNIP AND THE AUTHORS' REPLY TO A REVIEWER

SNIP scores every connection once at initialization, keeps the top-κ connections as a fixed mask c, and then trains the surviving weights w. The paper writes down the joint problem over c and w and sets it aside in one sentence: it doubles the parameters and is "even more difficult" to optimize directly. Reviewer 3 asked how far the relaxation is from solving that problem. The authors replied that they had tried alternating between c and w, which sometimes matched the accuracy of the reference network but was in general quite unstable. Neither the attempt nor the instability appears in the paper. Our trace considers the same alternating scheme and, as the authors did, sets it aside as a direction that needs separate stability work.

Tag	Type	Source and what it supplies
S1	authors' rebuttal	OpenReview reply to Reviewer 3, ICLR 2019, note `HkeeUt2G0X` in forum `B1VZqjAcYX`: the alternating (c, w) optimization they tried, its success in some cases, and its general instability.
P1	primary	SNIP, ICLR 2019: the joint problem, set aside as more difficult, and the connection-sensitivity criterion stated as the design.

The clean formulation is joint: the mask c and the weights w are both free, minimize the loss over the pair subject to $\|c\|_0 \leq \kappa$. S1 I could alternate the two blocks: freeze c, take a few optimizer steps on w; freeze w, take a few steps on c; back and forth. If the two blocks converge together, I'd get a sparse network straight out of the training objective. But each block is chasing a target the other block is simultaneously moving. Without something extra bolted on, I have no argument that the pair settles rather than cycles. So I park the joint optimization: a real direction, but not one I've earned the right to trust without separate stability work. P1 What introducing c does buy me: c_j is the dial that turns connection j on and off, decoupled from its weight w_j, and the importance of connection j is how much the loss reacts when I flip that dial.

[... the trace writes the exact loss change from knocking out one connection, relaxes c so that one forward–backward pass replaces $m{+}1$ forward passes and yields every $\partial L/\partial c_j$, and checks the result on a toy network. ...]

P1 What I care about is the magnitude of that effect, regardless of sign, so I keep the top-κ connections by $|\partial L/\partial c_j|$, fix the binary mask, and train the sparse network the standard way.

S1 · authors' OpenReview reply. "We have attempted to optimize c and w together in an alternating optimization paradigm... We were able to achieve sparse networks with comparable accuracies to the reference network in some cases, **however, in general the optimization was quite unstable.** We believe that this is a promising direction to pursue, and yet further investigation will be required."

Note `HkeeUt2G0X`, "Response to R3," signed by the Paper512 authors, 21 November 2018.

P1 · what the primary gives. "Compared to Equation 1, we have doubled the number of learnable parameters in the network and directly optimizing the above problem is even more difficult."

Section 4.1, after Eq. 3. The paper then turns to measuring each connection's effect on the loss. Alternating optimization of c and w and its instability appear in neither arXiv version, v1 or the camera-ready v2.

D AGENT SKILLS AND PROMPTS

The coding agents that build INNOVATION PRIOR run with two skills, `paper-to-reasoning` and `paper-to-reasoning-audit`, and three prompts, one each for cleaning the input context, for repair and for verification. This appendix quotes the rules that govern each of the three steps in Section 2.2. Each box names the file it comes from and quotes it verbatim, with [...] marking omitted text, and section numbers are those of the skill files.

Search and collect: grounding rules

From the skill `paper-to-reasoning`.

Grounding: every claim must be sourced, never written from memory

This is non-negotiable. Everything in the three deliverables — every equation, constant, hyperparameter, exponent, sign, loss coefficient, the exact form of each update/architecture, every baseline figure, every historical/background claim, and every design rationale — must be grounded in material you actually **retrieved and read this run**: the primary paper text/source, its cited references, the canonical implementation, and reputable explainers. It must **not** be produced from memory.

Memory is a *hypothesis*, not a source. If you cannot point to a specific retrieved artifact that confirms a fact, look it up before you write it — `curl` the arXiv source, fetch the actual repo file, `WebSearch`/`WebFetch` a reputable page — and read the relevant passage. Recalled-but-unverified specifics (e.g. "β_2=0.999", a particular exponent, a normalization constant, which prior work was the ancestor, the precise clip/case form) are the single most common source of subtle, confident errors; **verify each against a retrieved source**. When you genuinely cannot find a source for a claim, do not assert it — omit it, or flag the uncertainty explicitly in `notes/`. This rule binds the audit/verification passes as well: a reviewer confirms facts against retrieved sources, not against memory.

The three-source bottom line (non-negotiable — applies even when there is no arXiv version). For every method you must retrieve, actually read, and capture into `refs//notes/` ALL THREE of: **(1) the primary source in full** — the original paper/text that introduced the method, read end to end; for a pre-arXiv classic, the original publication itself, or — only if it is genuinely unobtainable (paywalled + OCR-hostile after a real search) — the closest faithful reproduction of its *actual* equations, with the gap flagged in `notes/`. **(2) The background sources** it builds on — the load-bearing ancestors (§1.2), read for their core idea and the limitation that motivated this method. **(3) At least one third-party explainer/interpretation** of it (§1.3) — a blog, lecture notes, course page, annotated implementation, or retrospective analysis. None of the three is optional. If any one genuinely cannot be obtained, say so explicitly in `notes/` **and** in your final report — never paper over a missing source with memory.

Search and collect: the lineage

From the skill `paper-to-reasoning`.

1.2 Research the key cited references (the method's lineage)

• From the bibliography, pick the **load-bearing ancestors** — the works the method directly builds on or reacts against, not every citation. (For TRPO that was: the conservative-policy-iteration / surrogate-bound ancestor, natural gradient, the natural-actor-critic line, the KL-constrained-update line, the vanilla policy-gradient baseline, plus the relevant optimization framing.)

• For **each** key reference, find: its one-line contribution, its core idea/math, and **the specific limitation that motivated the next step toward this paper**. This is what lets you "elaborate" related work instead of name-dropping it. Save downloaded refs into `<method>/refs/` and the structured report into `<method>/notes/`.

• Dispatch a subagent (`Agent`, subagent_type `general-purpose`) to research these in parallel; ask for a structured ~800–1200 word report. Also ask it for the **big-picture state of the field at the paper's time** (what was the prevailing wisdom / the pain points).

Search and collect: the authors' own account

From the skill `paper-to-reasoning`.

1.2b Hunt for the author's OWN account of the discovery (highest-value source)

• **This is the single most valuable kind of source for this skill, when it exists.** A finished paper hides the discovery path; but authors often record the *actual* path elsewhere — and that first-person record is exactly what `reasoning.md` is trying to reconstruct. Actively search for it and, if found, **download it to `<method>/refs/` and use the author's own narrative as the backbone of the trace** (the walls they hit, the wrong turn they took, the moment it clicked — in their words).

• **What you are really after is the *reasoning the paper omits*, NOT a dramatic anecdote.** The famous-anecdote kind (Mullis's night drive, Dijkstra's 20-minute café, Hoare in Moscow) is welcome when real, but it is *not* the point and you must not over-weight it or stretch for it. The far more valuable content — and the thing to prioritize — is the author's **technical** account of how they actually thought: *why* they chose this route over an alternative, the approaches they tried and abandoned that never made the paper, what a particular definition / constant / normalization / design choice was really for, the intuition or half-formed picture that guided them, "the way I was thinking about it at the time." Many of the best self-accounts contain **no story at all** — they are the author, years later, finally writing down the motivating reasoning that the theorem-first paper edited out (e.g. a Nobel lecture's technical middle section, 't Hooft's *Glorious Days of Physics*, Lamport's per-paper notes in *My Writings*, Pearl's *A Personal Journey into Bayesian Networks*). A self-account with rich omitted-reasoning and zero anecdote is MORE useful than one that is all anecdote and no reasoning. Mine it for the reasoning, and let any anecdote stay incidental.

Search and collect: explainers and the reason for each design choice

From the skill `paper-to-reasoning`.

1.3 Find third-party explainers — and the *why* behind every design choice

• Dispatch a second subagent to find the best blogs, lecture notes, course pages, and analyses that explain the **derivation, intuition, and design rationale** (not just the final algorithm) — e.g. canonical lecture slides, OpenAI Spinning Up, well-known author blogs, Distill/DepthFirstLearning-type walkthroughs, annotated implementations, and deeper retrospective analyses.

• **This is the step that makes the reasoning *deep* instead of thin.** Papers routinely state a design decision without justifying it (the scaling factor, the number of heads, the projection head, the FFN width, layer-norm placement, the warmup schedule, a particular reparameterization, the choice of one estimator over another, . . .). For **every non-obvious design choice**, you must come away understanding *why it was made and why the alternatives are worse*. When the primary text doesn't explain a choice, go find the analysis that does — dig until you can reconstruct the rationale, the alternatives that were on the table, and the failure mode each alternative hits.

• Capture, into `<method>/notes/`: the intuitive "why" behind each step, the common ways people explain the key bound/identity, where theory was relaxed for practice, and a running **list of design decisions → the reason for each** (and the rejected alternatives).

• **Hindsight discipline.** These analyses are *your* research input to understand the design; many are written after the fact. In `reasoning.md` you will re-derive each rationale **in-frame, as the narrator's own derivation-time reasoning** — never cite the later analysis, never import its posterior framing or its results. Use the explainer to *understand the why*, then reconstruct that why from what was knowable at the time.

Search and collect: synthesis before writing

From the skill `paper-to-reasoning`.

1.5 Synthesize understanding (main agent)

• Read the subagent reports and the code yourself. Before writing, make sure you can answer: What pain point existed? What tools existed and exactly where did each fall short? What is the precise first-principles object being optimized, and what is the central difficulty? What chain of approximations bridges theory → practice? Which step makes a well-known prior method fall out as a special case?

• **Build the design-decision → why table** from 1.3. Walk the method end to end and list every concrete choice (every constant, every architectural knob, every "we use X" sentence). For each, can you state *why this and not the obvious alternative*? Any choice you can only describe but not justify is a hole — go back to 1.3 and fill it before writing. The depth of `reasoning.md` is bounded by the completeness of this table.

• **Write the synthesis to notes *before* Phase 2.** Persist your understanding — the answers to the questions above, the load-bearing-ancestor write-ups, and the full design-decision→why table (with rejected alternatives and their failure modes) — into `<method>/notes/synthesis.md` first. Composing the deliverables *from this notes file* (rather than from memory) is what makes the output reliable and deep: the notes force every gap to surface while you can still fix it, and Phase 2 becomes transcription-with-voice rather than figuring-it-out-while-writing. Don't open Phase 2 until the notes are written and the table has no holes.

Reconstruct: the input context

From the skill `paper-to-reasoning`; context-cleaning prompt.

No conceptual leakage — the prompt-side discipline (this is what makes the file usable as a training PROMPT). `context.md` is the *prompt* whose target is the discovery in `reasoning.md`; the sample only teaches *insight* if the prompt poses an open problem rather than handing over the answer. So `context.md` may state the **problem and the tools that already exist**, but must NOT state the **move / insight / object that `reasoning.md` is supposed to discover** — and this ban is *conceptual, not lexical*: it is violated even when the method's name never appears. The failure is describing "the very thing to be found" so specifically that a reader could name the key idea from `context.md` alone. **Operational test — apply it before delivering:** *could someone who has NOT read `reasoning.md` reconstruct the method's central move from `context.md`?* If yes, it leaks — soften until the answer is no. Two recurring leak channels (policed again in the Baselines and Code-framework rules below): (a) a baseline's gap written as a *prescription of the fix* rather than an *observed limitation*; (b) a scaffold slot *pre-shaped around the answer*. The discovery must happen in `reasoning.md`, not be pre-given in the prompt.

This file is the INPUT shown to the model: a plain, contemporaneous snapshot of the research situation at the time, BEFORE the method-to-be-invented exists. It must NOT do the reasoning's job of finding problems.

(A) Contemporaneous viewpoint ONLY — describe what exists, is known, or is practiced at that era, as a scientist standing in that moment would see it. Remove ALL hindsight: "would later", "this anticipates", "turns out", "in retrospect", "the key insight that...", "this paved the way", or any reference to how things developed after this point.

(B) **Do NOT point out problems, gaps, weaknesses, or what is missing. This is the single most important rule.** Delete EVERY "Gap:", "Gap it leaves:", "Weakness:", "Limitation:", and every clause of the form "but it fails / stalls / cannot / is fatal / is too expensive / doesn't scale / leaves open / the problem is...". Describe each existing method ONLY by what it IS and what it does. Also strip any "pain points" / "the problem with X is..." editorializing from the Background and Research-question sections. State the setting plainly and naively — finding the shortcomings is the job of the separate reasoning trace, not this file.

(C) Never name or telegraph the invented method (you learned it from the answer). The Research-question section should pose the broad question / setting only — NOT an answer-shaped wishlist of desiderata that the new method happens to satisfy, and NOT "none of the existing methods achieves all of this".

Reconstruct: the trace

From the skill `paper-to-reasoning`.

2.2 `reasoning.md` — the LLM-style reasoning trace (the heart)

Write **one continuous first-person, present-tense, stream-of-consciousness monologue** that re-derives the method. This is the original output of this skill — get it right.

What it MUST be (the voice)

• **First person, present tense, "thinking out loud."** e.g. "OK, let me think this through from scratch…" / "wait — stare at this for a second…" / "huh." **Write the deliverables in English by default** (all three files), even when the conversation is in another language — switch only if the user explicitly asks for another language.

• **Problem-driven order, not theorem-first.** Start from the pain points and the goal, *then* survey the tools, *then* derive. Never open with the final method.

• **Genuinely elaborate related work** as part of the thinking: for each ancestor, what it did, why it's not enough, what gap it leaves. This is reasoning about prior art, not a citation list.

• **Be complete: cover ALL of the background/motivation/method thinking and EVERY derivation, from both the main text and the appendix.** This is a hard completeness bar — nothing load-bearing from those parts may be left out or merely summarized. Work inline as the narrator does them — telescoping tricks, every lemma, the full bound, alternative proofs, the matrix-vector-product / efficiency derivations, the parameterization details, and every motivating argument. Re-derive, don't gesture ("the proof skeleton is as follows" is banned — actually do the steps). If the appendix proves something the main text only states, the proof is lived out here. (Only the experiments/case-studies are out of scope — see Phase 1.1.) A thin reasoning trace that skips appendix derivations is a failure even if the voice is right.

• **Justify every design choice — reach for the *why*, not just the *what*.** This is where thin traces fail. Don't write "I'll scale by $1/\sqrt{}$d_k" / "I'll use 8 heads" / "I'll add a projection head" / "the FFN is $4\times$ wide" as bare decisions. For each, reconstruct *why* — what breaks without it, what the alternatives are, and why they're worse — at the depth of the analysis you gathered in 1.3.

• **Insight before method — derive, don't state-then-justify.** This ordering is local, not just global: at *every* step, walk the motivating reasoning *first* and let the formula / architecture / objective drop out as its conclusion. The pain, the constraint, the failed alternative come first; then the thing that resolves them appears — and the reader should feel it become inevitable a beat *before* it's written down. Never present the result and then back-explain it ("the update is $\theta \leftarrow \theta$-$\alpha \cdot \hat{m}/\sqrt{\hat{v}}$, and the reason is…", "we use multi-head attention; this helps because…"). Flip every such moment so the reasoning leads and the method follows.

• **Let the "aha" moments emerge from the math**, not announced upfront. The unification ("wait, that direction... isn't this just the natural gradient?!") should land *when the derivation produces it*, as a surprise.

• **End by landing on the method's final concrete form.** For a *computational* method that is **real code**, grounded in the standard implementation from 1.4, with short comments tying each block back to a step in the reasoning. For a *non-computational* discovery (pure-math/physics/chemistry/biology) it is instead the clean final artifact the derivation has been driving toward — the theorem stated precisely and its proof completed, the final formula, the mechanism, the experimental protocol — **do not bolt on code that the field wouldn't use.** Close with a one-paragraph causal-chain recap either way.

What it must NOT be (anti-patterns — these were all explicitly corrected)

• **Not a structured essay.** No section headers, tables, boxes, or flow diagrams describing the reasoning. That is writing *about* the reasoning; we want the reasoning itself. Keep markdown structure near-zero — continuous prose (code block at the end is fine).

• **No name-dropping** related work (e.g. just "REPS, Peters 2010" with no substance). Elaborate it.

• **No empty contrastive filler.** The "…, not from X" tic where X is vague ("let me start from the pain, not from any clever model", "…, not from a method") is meaningless padding — it reads like it means something but doesn't. If you draw a contrast, X must be concrete (e.g. "not the way it's written up, theorem-first"); otherwise just state the thing and cut the clause.

• **Don't pre-announce or label the rhetorical move.** Sentences that narrate the *shape* of the upcoming argument — "Here's the argument that turns 'huh, weird' into 'this is provably absurd, and that absurdity is the clue'", "now for the key insight", "let me make the move that unlocks everything" — are writing *about* the reasoning, not the reasoning. Cut the framing sentence and walk straight into the actual argument; if it's good, it announces itself.

• **Don't overclaim the stakes.** Don't inflate a *surprise* into a *paradox*: "provably absurd", "an outright contradiction", "impossible" when the situation is merely "an at-least-as-good solution provably exists yet the optimizer can't find it." Name precisely what's surprising; the precision is what makes it a clue.

• **No state-then-justify (expository order).** Presenting a formula/architecture/objective and *then* giving its rationale is textbook order, not discovery order — it reads as explaining a known result. If a paragraph names the method's piece in its first sentence and spends the rest justifying it, flip it: the motivating reasoning leads and the piece emerges at the end (see "Insight before method" above).

• **No fabricated results, and no *proposed-method* evaluation results.** The trace ends before the proposed method is benchmarked — don't invent numbers or claim "it beat the baseline"/"it learned X"; "what I'd want to validate" as forward-looking intent is fine. **But the *motivating/diagnostic* empirical findings the paper starts from are part of the setup** — the narrator may (and for an empirical-first paper *should*) open from the observed phenomenon and reason about it (staring at the degradation curve; noticing the outlier dimensions; running the diagnostic ablation in their head). Use the real reported observations; don't invent new ones.

• **No hindsight / posterior content.** No "later, PPO did X", no "honest tally of compromises", no comparisons to follow-up work. The narrator only knows what was knowable at derivation time.

• **No meta footer / disclaimer note** at the end. End on the method and code.

• **No made-up code.** Mirror the canonical implementation's structure.

Reconstruct: the decisive step and the attempts that fail

From the repair prompt.

`results/reasoning.md` = a first-person, present-tense reconstruction of how a landmark method was actually discovered: the pain of the problem, the obvious first move and WHY it fails (a checkable reason), the real obstacle, the move that resolves it, landing on the method + code. [. . .]

[. . .] But finished papers edit the struggle out. What we want is the documented route: what the author actually tried first and why it failed, which counterexample/obstacle forced the design — evidence that lives OUTSIDE the primary paper: author self-accounts (Nobel/Turing/award lectures, PhD theses of first authors, retrospectives, interviews/podcasts/talk transcripts, blogs, OpenReview replies, GitHub issue replies, competition reports), co-author surveys, authoritative third-party re-derivations.

3. VERIFY THE MATERIAL SAYS IT. Quote the load-bearing passage ($\leq$3 sentences) into notes/sources.md with type (self-account / explainer / ancestor / primary), URL, and the local file path. Only content you can point to in a saved file may shape the trace. Never invent an anecdote.

• The passage now runs: obvious move $\rightarrow$ fails for the REAL, checkable reason from the source (a counterexample worked on the page, a quantity that doesn't close, a sign that comes out wrong, a training run that diverged with the stated symptom) $\rightarrow$ the documented resolution. Mechanism, not story: "I spent months on X and it was intractable to train because Y" is good; "over coffee I suddenly saw it" is useless.

• Present as the scientist's own live thinking: first person, present tense. No "the paper", "the authors", "arXiv", "et al.", "later work showed", no hindsight ("as we now know").

• Do NOT: [. . .] add "Wait"/"Alternatively"/"Hmm" filler (a genuine dead end is the actual computation that failed, not a hedge word); [. . .]

Reconstruct: experimental outcomes

From the skill `paper-to-reasoning`; repair prompt.

Empirical-content discipline — recall and reason, never measure

Many methods — **especially engineering / simulation / optimization ones, where a simulator is literally at hand** — are *born from an observed phenomenon*: you cannot derive the fix without knowing the failure mode (checkerboard patterns in density-based FE, Gerchberg–Saxton stagnating at a local minimum, deep fades dominating BER). This empirical content is load-bearing, but it is also the single biggest temptation to fabricate. Split it three ways so the trace never invents an experiment:

• **Established phenomena / prior-art failure modes $\rightarrow$ `context.md` (Background).** Anything observed about *existing* systems or the design space, knowable before the method, goes into context as a sourced pre-method fact. In `reasoning.md` the narrator **invokes it as known** ("it's well documented that low-order elements make a checkerboard artificially stiff") and reasons *about* it — never stages a fresh run to "discover" it.

• **Analytically-anticipatable observations $\rightarrow$ derive them on the page in `reasoning.md`.** Whenever an observation can be *predicted* by math rather than measured, convert "we saw X works" into "here's why X must work" and actually do the derivation (e.g. read off the $E(\rho)=\rho^p$ curve that intermediate density is uneconomical, so 0/1 designs win on their own). This is the safest and the deepest mode — prefer it wherever possible.

• **The proposed method's own results $\rightarrow$ excluded** (already the rule): no benchmark/ablation numbers, no "it converged to a clean truss with compliance C". "What I'd want to validate" as forward-looking intent is fine.

The line the narrator must not cross: a **thought experiment** reasons from established premises to a conclusion; a **fabricated experiment** asserts a *new measured outcome*. The narrator may reason ("picture a solid/void checkerboard — Q4 elements overstate its stiffness, a known FE pathology, so the optimizer will exploit it") but may **never** write "let me run it. . . I see the densities settle at 0.5" or report any number that wasn't in a retrieved source. **Recall and reason; do not measure.** Every empirical premise must trace back to `context.md`, which traces back to a real source.

EMPIRICAL decisive steps (most modern ML methods): a mathematical counterexample can be genuinely computed on the page, but an experimental outcome (an ablation result, a training curve, a benchmark delta) can only be OBSERVED, never derived — and in a single-turn reasoning.md there is NO legitimate way for the think-voice to obtain an observation. HARD RULE: reasoning.md must NEVER state the outcome of an experiment the narrator claims to run mid-reasoning ("I train X ... the numbers come back 78.9") — real numbers or not, that trains the model to fabricate evidence inside its own chain of thought. In reasoning.md the empirical step goes only this far: (1) competing hypotheses and why first principles cannot decide; (2) the discriminating experiment's DESIGN (controls, matched budgets) and what each hypothesis PREDICTS; (3) the decision rule ("whichever survives at both scales ships"). The method's OWN experimental results may not appear in ANY channel of a single-turn method unit — not reasoning.md, and NOT answer.md / train_answer.md either (those are trained model voice too; a write-up reporting numbers for experiments that were never run in the loop teaches fabrication in the answer channel). [. . .] context.md may carry empirical facts that PRE-DATE the method (prior work's published numbers, known at the time, stated as given). If you find the method's own numbers already sitting in answer/train_answer/reasoning, that is a defect to remove, not a pattern to imitate. A trace that "derives" an empirical outcome with logical-necessity tone is FAKE DERIVATION; a trace whose think-voice reports self-run experiment results is SELF-SUPPLIED OBSERVATION; both are defects. [. . .]

Review and audit: an independent second model

From the skill `paper-to-reasoning`.

2.6 Independent Codex review-and-fix pass (per method) — required

Your own self-check misses things — especially **math errors** (flipped inequalities, dropped factors, reversed case-analyses) and residual meta-commentary. So after the three files for a method are done, **run an independent, write-enabled reviewer over them yourself** — and **if you are a subagent, call it yourself** (you have Bash; don't defer it to a parent). Do this **per method**, once that method's deliverables exist.

Review and audit: reading as the inventor, grounded in sources

From the skill `paper-to-reasoning-audit`.

PRIMARY LENS (the single most important check — apply before all others). Does `reasoning.md` read as a *natural first-person derivation by the person who is inventing the method*, discovering it for the first time? It must never betray that a finished paper exists. Concretely, the trace fails this lens if it:

• **references or cites the source paper as an external artifact** — "this paper", "the authors", "as shown in §4", "the original work", a venue/arXiv/year, or any phrasing that treats the method as already-published rather than being figured out right now;

• **quotes or paraphrases the paper's own framing** — importing its theorem-first / result-first narration, its section structure, its named contributions, or its abstract's summary sentences, instead of arriving at those things as the *output* of the reasoning;

• **reads as exposition of a known result** rather than discovery — i.e. it explains the method as established fact ("the method works by…") instead of reasoning *toward* it from the problem ("I need X; the obvious move fails because…; so what if I…").

The narrator IS the inventor. If a sentence could only have been written by someone who already read the finished paper, it is a violation — rewrite it into the discovery voice (motivation → attempt → wall → fix), or cut it.

This lens applies to `answer.md` too. `answer.md` *may* name the method and reference its canonical implementation, but it must still not present the method as a cited external paper (no citation line, no authors/venue/arXiv, no "as the paper shows"). It is the inventor writing up *their own* result, not a literature summary of someone else's.

WEB-GROUNDING (verify facts against retrieved sources, never against memory). Every factual specific in all three files — equations, constants, exponents, signs, hyperparameters, loss coefficients, the exact form of each update/architecture, baseline numbers, historical/lineage claims — must be checked against an **authoritative source the reviewer actually retrieves this run** (the primary paper / arXiv source, the canonical implementation, reputable explainers via `WebSearch`/`WebFetch`). Do not "confirm" a constant or formula from memory: look it up and read the passage. A confident-looking but unsourced specific is a defect; correct it to the sourced value, or if no source can be found, flag it. This is the same grounding rule the generator must follow (`~/.claude/skills/paper-to-reasoning/SKILL.md` → "Grounding").

Review and audit: the adversarial verifier

From the verifier prompt.

[…] (a) does the decisive step now DEPEND on the source content (delete the sourced sentences → does the step lose its justification)? If it's a name-drop/atmosphere → FAIL(decorative). (b) Does the failed first attempt fail for a concrete checkable reason (a worked counterexample, a quantity that doesn't close, an explicit symptom)? If it's "this felt wrong" → FAIL(vague_failure). […]

[…] (d) Provenance leak: "the paper", "the authors", "arXiv", "et al.", "later work", "as we now know" in added text → FAIL(leak). […]

[…] But check the epistemic type: if the decisive step is an EMPIRICAL outcome (ablation/benchmark/training behavior), the think-voice may go only as far as hypotheses → test DESIGN → predictions → decision rule. If reasoning.md claims to run an experiment and states its result inline ("I train ... the numbers are ..."), fail with "self_supplied_observation" — even when the numbers match the paper; the method's OWN experiment outcomes belong ONLY in a trajectory's observation turns — never in reasoning.md AND never in answer.md/train_answer.md (both trained model voice); finding them in any of those channels → fail with "self_supplied_observation". context.md may carry only empirical facts that pre-date the method (prior work's published numbers, as given); spot-check such numbers exist in the record (invented → fail). If the trace presents an empirical outcome as logically necessary, fail with "fake_derivation".

E TRAINING AND INFERENCE CONFIGURATION

Table 5 gives the settings of the supervised fine-tuning that produces the prior, Table 6 those of reinforcement learning, which are the same from both starting points, and Table 7 the sampling with which every model is evaluated under the same system prompt.

Table 5: Supervised fine-tuning.

Parameter	**Value**
Base model	Qwen3.5-4B / Qwen3.5-9B
Fine-tuning	LoRA (Hu et al., 2022), bf16
LoRA rank	32
LoRA α	64
LoRA dropout	0
Data	INNOVATION PRIOR
Template	`qwen3_5`
Thinking channel	on
Loss masking	per turn
Sequence length	53,760
Epochs	1
Learning rate	1×10^{-4}
Schedule	cosine
Warmup ratio	0.05
Weight decay	0
Per-device batch	1
Gradient accumulation	32
Effective batch	128
Gradient checkpointing	on
Fused cross-entropy	Liger
Seed	42
Merge weight	0.10
Framework	LLaMA-Factory (Zheng et al., 2024b)
Hardware	NVIDIA H200

Table 6: Reinforcement learning.

Parameter	**Value**
Algorithm	GRPO (Shao et al., 2024)
Framework	verl (Sheng et al., 2025)
Steps	20
Evaluated checkpoint	step 20
Train batch	64
Mini-batch	16
Micro-batch per GPU	1
Rollouts per prompt	16
Max prompt length	26,624
Max response length	32,768
Max model length	59,392
Reward	task score
Loss aggregation	sequence mean, token mean
Clip ratio	0.2
Entropy coefficient	0
Overlong penalty	off
Tensor parallel	1
GPU memory utilization	0.90
Reference model	offloaded
Actor parameters and optimizer	offloaded
Hardware	NVIDIA H200

Table 7: Evaluation sampling, shared by all arms.

Parameter	**Value**
Thinking	on
Temperature	1.0
Top-p	0.95
Top-k	20
Presence penalty	1.5
Max output tokens	32K
Samples per problem, FrontierCS and ALE-Bench	5
Execution feedback, FrontierCS and ALE-Bench	none
Agent runs per task, MLS-Bench	1
Hardware	NVIDIA H200 or A100

F THE RL PROBLEM SET

RL in Section 3 trains on 1,300 single-turn problems that we designed from scratch, each scored by a deterministic checker.

Problems and scoring. Each problem asks for a program that reads an instance and writes a solution, and a checker scores the solution with a ratio in $[0, 1]$: an infeasible output scores 0, a feasible output no better than a trivial construction that the checker builds itself scores about 0.1, and an output ten times better than that construction reaches the maximum of 1.0. FrontierSmith (He et al., 2026) builds such problems from closed-ended contest problems by turning the goal into an optimization objective, restricting the output or generalizing the input. Its generator is not released, so we rebuilt it and widened the range of problems it draws from. The problems come in the five formats of Table 8, all under this scoring rule. So that every score is deterministic and independent of hardware, kernel problems are scored by operation count after an exact equivalence check, and GPU kernels, interactive tasks, heavy simulators, metrics that require training a model and rewards from a judge model are excluded.

Table 8: The five problem formats. Each is deterministic and prints one ratio in $[0, 1]$.

Format	Shape	Scored by	Count
A	combinatorial optimization, C++ generator and checker (testlib)	C++ checker	319
B	evolve a heuristic against a frozen evaluator over seeded instances	Python evaluator	258
C	constructive artifact (packing, matrix, point set, schedule)	Python verifier	448
D	straight-line program or decomposition, scored by operation count	equivalence gate, then count	144
E	symbolic law discovery from a training sample, scored on a held-out extrapolation split	Python verifier	131

F.1 SEEDS AND THEIR DISTRIBUTION

Every problem starts from a seed, which specifies a problem without yet being one. A seed names: (1) a **family**, the kind of mechanism the problem instantiates; (2) a **theme**, the setting in which the statement is written; (3) the **objective direction**; (4) the instance **scale**; (5) two or three **mechanisms** that the objective must couple; (6) an **innovation hook**, one sentence naming the insight that should beat the obvious approach; and (7) a **trap**, a kind of case on which the obvious approach falls far short of the insight. Families come from a survey of deterministic task forms in prior work on automated discovery, grouped into the four tiers that head Table 9, and from seed-writing agents, each of which proposes a new family. A candidate seed is admitted only if its family differs from every admitted family under a token-set match, its pair of family and theme is unused, its format quota is not full, and no admitted seed couples the same mechanisms, since two such seeds would pose the same problem in different settings. Table 9 gives the distribution of the resulting 1,300 problems by tier and format. They span 1,001 families, 965 of which have a single problem, and the largest family has 14. Of the 1,300 problems, 307 are in C++ and 993 in Python. The median statement has 3,968 characters, and the median full prompt has 1,293 tokens, with a 10th to 90th percentile range of 1,056 to 1,520 and a maximum of 2,282. None of the problems is an evaluation task, since the authoring contract forbids copying any task of the evaluation benchmarks.

F.2 FROM A SEED TO A PROBLEM

FrontierSmith turns a seed into a problem in four steps: it mutates the seed, filters by the diversity of sampled solution ideas, has agents write the tests and the checker, and has other agents cross-check them. We keep the first and third steps and replace the second and fourth, which rest on the judgment of agents, with execution. One agent takes each seed through five steps.

1. **Authoring.** The agent reads the seed and the contract of its format and writes the statement, the instance generator, the checker and four reference solutions: **trivial**, the baseline

Table 9: The 1,300 synthetic problems. Left: count by tier and format. Right: harness-certified mean ratio of each reference solution over the problem's cases, median and 10th to 90th percentile across problems.

Tier	Count
graph and combinatorial optimization	247
mathematical discovery and heuristic evolution	469
engineering and scientific optimization	307
ML-method design and exotic construction	160
breadth fill across domains	91
bespoke composite mechanisms	26
Total	1,300

Format	Count
A combinatorial, C++	319
B evolve a heuristic	258
C constructive artifact	448
D operation count	144
E symbolic law	131

Solution	median	p10 to p90
trivial	0.100	0.100 to 0.100
greedy	0.208	0.138 to 0.510
strong	0.445	0.221 to 0.788
infeasible	0.000	0.000 to 0.000
strong − trivial	0.344	0.121 to 0.682

construction against which the checker normalizes; **greedy**, the approach a strong programmer writes first; **strong**, a program that embodies the insight of the seed; and **infeasible**, an output that violates the constraints on purpose. The contract fixes what makes a problem open-ended: a graded objective with no known reachable optimum, several plausible strategies, a generator whose cases range from tiny to large and include planted, trap and needle instances, a checker that strictly validates feasibility with bounded reads, and the scoring rule above. It also rules out by name the known closed forms that would make a family trivial, and a symbolic-law problem may not reveal its law through the example in the statement, the output of the generator or an importable module.

2. **Validation.** The agent runs the harness on its own problem, which passes only if it clears eight gates: the generator, the checker and all reference solutions compile; every case is generated within its size budget; every score lies in $[0, 1]$; a rerun reproduces every score bit for bit; the infeasible solution and a battery of empty, garbage, oversized, non-finite and score-injecting outputs all score about 0; the trivial solution scores in $[0.03, 0.35]$; the strong solution beats the trivial one by a margin; and the three feasible solutions give different per-case scores. Candidate programs run under bubblewrap in fresh namespaces, with the problem directory hidden and a private `/proc`, and an isolation gate rejects any evaluator that a program can read by walking stack frames. A program therefore cannot read the judge's stack, its process memory or the generator source beside it.

3. **Repair.** If a gate fails, the agent reads its diagnostics and the recorded scores, fixes the failing part and reruns the harness, for up to six rounds. A problem that still fails gets one more pass of the same loop from a second agent, and a problem that never passes is dropped.

4. **Headroom.** Authoring also targets room for RL to improve: strong − greedy ≥ 0.06, so that the insight clearly beats the obvious approach; greedy − trivial ≥ 0.03, so that the reference solutions are properly ordered; and strong ≤ 0.92, so that room remains above the strong solution. Table 9 gives the resulting scores of the reference solutions.

5. **Review and admission.** A second model reviews the finished problem for scoring loopholes, non-determinism, mismatches between statement and code, trap cases that fail to penalize the greedy solution, and strong solutions that are only the greedy one with tuning, and the agent repairs what it flags and reruns the harness. Finally, the problem is compared with every admitted problem by an exact hash of its code skeleton and by the token 5-gram similarity of its checker logic, with comments and literals stripped. Of any near-duplicate pair, only the problem with the larger headroom is kept.

For example, one seed has the family **circuit-breaker tuning**, the mechanisms failure threshold, half-open probing and flapping suppression, and the hook "a tight symmetric threshold trips fast and flaps; the insight is asymmetric hysteresis, trip on a short window and recover on a long one, with the probe rate tied to observed recovery time". It became a Format C problem. The input is a deterministic timeline of dependency health with request costs, the output is a set of breaker parameters that a fixed replay engine applies tick by tick, and the score counts successful requests minus wasted calls, relative to the checker's baseline breaker that never trips. The greedy solution, the textbook symmetric breaker, handles hard outages cleanly but flaps on the six gray-failure timelines that the generator plants. The strong solution gates recovery on a longer, independent window and sets the

probe interval from the last observed recovery. The harness scores the trivial, greedy and strong solutions at 0.100, 0.352 and 0.418.

G MLS-BENCH EVALUATION

MLS-Bench (Lyu et al., 2026) gives an agent the code of a research baseline in a fixed scaffold and asks it to improve the method by editing the code, running the task's tests and submitting. We use the 20 tasks of Table 10, all of which run on CPU, and run the benchmark's agent loop once per task with at most 20 steps. The score of a task is the benchmark's own: 0 at the weakest published baseline, 50 at the strongest and 100 at the bound of the metric. Tables 1 and 2 report the mean over the 20 tasks.

All arms of the same model size run with the same context window and time limit per task.

Table 10: The 20 MLS-Bench tasks, by task ID.

Area	**Task ID**

Causal inference	`causal-discovery-discrete`
	`causal-observational-linear-gaussian`
	`causal-observational-linear-non-gaussian`
	`causal-observational-nonlinear`
	`causal-treatment-effect`

Machine learning	`ml-active-learning`
	`ml-anomaly-detection`
	`ml-calibration`
	`ml-clustering-algorithm`
	`ml-dimensionality-reduction`
	`ml-ensemble-boosting`
	`ml-missing-data-imputation`
	`ml-selective-deferral`
	`ml-subgroup-calibration-shift`
	`ml-symbolic-regression`

ML systems	`mlsys-moe-load-balance`

Optimization	`optimization-evolution-strategy`
	`optimization-hyperparameter-search`
	`optimization-multi-objective`
	`optimization-nas`

H TECHNIQUE FAMILIES

Section 4.2 counts the technique families that a proposal considers. We compiled the 84 families of Table 11 from about 9,800 public C++ and Python programs that frontier models wrote for algorithm and research problems. Each family is a set of keywords, the names of the technique and the identifiers that implement it in C++ or Python, so a family is found whether the reasoning names it or sketches its code.

A proposal considers a family if its reasoning contains any keyword of that family, where the reasoning is the text before `</think>`. Matching ignores case, allows a space, hyphen or underscore between the words of a keyword, and counts a family once per proposal however often it appears. The same keywords, applied to the submitted program, give the families that the proposal puts into code, and a family that the reasoning considers but the program does not use is one that the model weighed and set aside.

The counts in Figure 8 cover all five proposals for every problem of the three benchmarks: the 172 problems of FrontierCS Algorithm, the 64 of FrontierCS Research and the 40 of ALE-Bench.

Table 11: The 84 technique families and their keywords. Identifiers are set in `typewriter`.

Family	**Keywords**
Search and metaheuristics	
Simulated annealing	simulated annealing, anneal, temperature update (`temperature *=`, `temp *=`), acceptance `exp(-delta)`
Hill climbing	hill climbing, local search, neighbour solution, `improved = true`
Beam search	beam search, beam width, beam
Genetic algorithm	genetic, evolutionary, mutation, crossover, population, `generation <`
Tabu search	tabu
Monte Carlo tree search	Monte Carlo, MCTS, playout, rollout
Random restarts	restart, random seed, `mt19937`, `rand_r`, `random_device`, `rng`, `uniform_int_distribution`, `np.random`
Branch and bound	branch and bound, prune, `bound <`
Greedy	greedy
Graph algorithms	
Breadth-first search	BFS, breadth-first, `queue<...> q`, `deque([`
Depth-first search	DFS, depth-first, `void dfs`, `def dfs`
Dijkstra	Dijkstra
Bellman–Ford / Floyd–Warshall	Bellman–Ford, Floyd–Warshall
Minimum spanning tree	Kruskal, Prim, minimum spanning
Topological sort	topological
Connectivity	Tarjan, strongly connected, SCC, articulation, bridge
Maximum flow	max flow, Dinic, Ford–Fulkerson, min cut, push–relabel
Matching	Hungarian, Kuhn, Hopcroft, bipartite matching, matching
Data structures	
Union–find	union–find, disjoint set, DSU, `find_set`, `parent[`
Segment tree	segment tree, segtree, lazy propagation
Fenwick tree	Fenwick, binary indexed, BIT
Heap	heap, `priority_queue`, `heapq`, `make_heap`
Hash map	`unordered_map`, `unordered_set`, `dict(`, `defaultdict`, `std::map<`, `HashMap`
Trie	trie, prefix tree
Sparse table	sparse table, RMQ
Balanced search tree	treap, splay, AVL, red–black, policy-based, `order_of_key`
Bitset	bitset, `__builtin_popcount`, `bit_count`, popcount
Dynamic programming and optimization	
Dynamic programming	dynamic programming, memo, `dp[`, `dp =`
Bitmask DP	bitmask, mask, `1 << n`, `(1<<`

Family	Keywords
Knapsack	knapsack
Linear programming	linear programming, simplex, LP, `linprog`, `pulp`
Mathematics and geometry	
Sieve	sieve, Eratosthenes, `is_prime`
Modular exponentiation	`modpow`, `power_mod`, `pow_mod`, fast pow, `binpow`
GCD / LCM	`__gcd`, `gcd(`, `math.gcd`, lcm
Chinese remainder theorem	Chinese remainder, CRT
Matrix exponentiation	matrix exponentiation, matrix power, matrix multiplication, `matmul`
FFT / NTT	FFT, NTT, fast Fourier, convolution
Gaussian elimination	Gaussian elimination, row reduction, LU, `rank =`, `linalg.solve`
Combinatorics	binomial, nCr, factorial, Catalan, inclusion–exclusion, `comb(`
Probability	probability, expected value, expectation
Convex hull	convex hull, Graham scan, Andrew's monotone chain
Computational geometry	cross product, `dot(`, `atan2`, `hypot`, `orientation(`, `ccw(`
String algorithms	
KMP	KMP, Knuth–Morris, failure function, prefix function
Z-algorithm	Z-algorithm, Z-function
Suffix structures	suffix array, suffix automaton, suffix tree, SA-IS
String hashing	rolling hash, polynomial hash, string hash
Aho–Corasick	Aho–Corasick
Algorithmic paradigms	
Sorting	`sort(`, `sorted(`, `.sort(`, `stable_sort`, `nth_element`
Two pointers	two pointers, sliding window, `l++ ... r++`
Binary search	binary search, `lower_bound`, `upper_bound`, `bisect`
Ternary search	ternary search, golden section
Meet in the middle	meet in the middle
Divide and conquer	divide and conquer, merge sort
Coordinate compression	coordinate compression, `compress(`
Prefix sums	difference array, imos, prefix sum, `cumsum`, `accumulate(`
Square-root decomposition	sqrt decomposition, Mo's algorithm, `block_size =`
Simulation	simulate, simulation
Caching	memoize, `lru_cache`, `cache[`
Performance engineering	
Parallelism	thread, OpenMP, `#pragma omp`, `multiprocessing`, `concurrent.futures`
SIMD	SIMD, AVX, `__m256`, `#pragma GCC target`
Triton	Triton, `tl.load`, `tl.store`, `@triton.jit`
CUDA	CUDA, `__global__`, `threadIdx`, `blockIdx`, `cudaMalloc`
Quantization	quantize, quantization, int8, fp8, bfloat16
Machine learning and scientific computing	
NumPy	`import numpy`, `np.`
PyTorch	`import torch`, `torch.`, `nn.Module`
scikit-learn	scikit, `sklearn`
pandas	`import pandas`, `pd.`
SciPy	`import scipy`, `scipy.`
Gradient descent	gradient descent, SGD, Adam, `optimizer.`, `backward()`, `lr_scheduler`
Neural networks	MLP, BatchNorm, LayerNorm, ReLU, activation, `Linear(`, `Conv1d` / `Conv2d`
Attention	attention, transformer, QK, QKV, rotary, RoPE
Regression	least squares, `LinearRegression`, `Ridge`, `Lasso`, `polyfit`, `curve_fit`
Clustering	cluster, `KMeans`, `DBSCAN`, `AgglomerativeClustering`
PCA / SVD	PCA, SVD, eigen, `truncated_svd`
Nearest-neighbour index	nearest neighbour, FAISS, HNSW, IVFFlat, Annoy, NMSLIB

Family	Keywords
Tree models	`DecisionTree`, `RandomForest`, `GradientBoosting`, XGB, LightGBM, CatBoost
Ensembles	ensemble, bagging, voting, stacking
Cross-validation	cross-validation, `KFold`, `train_test_split`
Bayesian optimization	Bayesian optimization, Gaussian process, Optuna, Hyperopt
Symbolic regression	symbolic regression, PySR, SymPy
Methods from other fields	
Physics simulation	n-body, Barnes–Hut, Verlet, leapfrog, force field, octree
Information theory	entropy, mutual information, Kullback, KL divergence
Control theory	PID, Kalman, Lyapunov, model predictive
Economics and game theory	auction, Nash, regret minimization, multiplicative weights, game theory

I MORE CASE STUDIES

This appendix gives more detail on the case of Section 4.3 and one more case. We recompute every score with the official tools of each benchmark and quote model outputs verbatim, with [...] marking omitted text. All draws are those of the evaluation in Table 1.

I.1 ALE-BENCH AHC041

The official generator and visualizer, run on the 150 private test cases, reproduce the evaluation's scores for both programs in Figure 9. Table 12 changes only the processing order of the program from InnoPrior+RL. Processing the lightest vertices first puts the lightest fifth of the vertices at a mean depth of 0.6 and the heaviest fifth at 6.5; processing the heaviest first reverses this profile and loses 43% of the score.

Table 12: Official score of the prior arm's AHC041 program with only the processing order changed, averaged over the 150 private cases. The last row is the output Base+RL submitted.

Processing order	Score per case
Lightest first, as submitted	289,053
Random, mean of five shuffles	227,036
Heaviest first	164,006
Every vertex a root	50,562

In another draw, Base+RL first says "we want vertices with high A_v to be close to roots", then corrects itself, "So we really want to maximize the depth of vertices with high beauty values", and then plans to "Sort vertices by A_v (beauty value) in descending order", the reverse of the order that InnoPrior+RL reaches. At 4B, InnoPrior+RL also submits the lightest-first construction and reaches the same score.

I.2 FRONTIERCS 83: A COMPLETELY MULTIPLICATIVE SIGN SEQUENCE

FrontierCS problem 83 asks for a completely multiplicative $f : \{1, \dots, 10^6\} \to \{\pm 1\}$ that minimizes the discrepancy $D = \max_k |\sum_{i \le k} f(i)|$, the completely multiplicative case of the Erdős discrepancy problem, with a lower D scoring higher. Figure 10 shows the reasoning of each 9B model after RL. Every 9B model reaches the Liouville function $\lambda(n) = (-1)^{\Omega(n)}$, whose partial sums grow like $\sqrt{n}$; on this instance it gives $D = 1253$, which scores 0.6. InnoPrior+RL goes further and reaches a score of 70.

Base, InnoPrior and Base+RL all raise the non-principal character modulo 4 and drop it. Base and InnoPrior drop it because a character vanishes at even numbers: "Non-trivial characters take 0 values sometimes. Condition $|f(i)| = 1$ forbids 0. [...] So it CANNOT be a Dirichlet character of modulus > 1." Base+RL drops it because it expects the character's partial sums to grow like $\sqrt{n}$. InnoPrior+RL resolves the objection about even numbers: a completely multiplicative function is free on the primes, so the character's values on the odd primes can be kept and $f(2)$ set by hand. The submission sets $f(p) = +1$ for $p \equiv 1 \pmod 4$ and $f(p) = -1$ for every other prime. On odd numbers this is the character itself, whose partial sums stay in $\{0, 1\}$, and the whole sequence reaches $D = 10$. The construction is one line inside a linear sieve, and replacing that line with $f(p) = -1$ gives back the Liouville function.

Problem (FrontierCS 83)
Output $f(1), \ldots, f(n)$ for $n = 10^6$, each ± 1, with f completely multiplicative: $f(xy) = f(x)f(y)$.
So f is fixed by its signs on the primes. Minimise $D = \max_k |S_k|$, where $S_k = f(1) + \cdots + f(k)$.
Scored by the official checker as $100 \cdot \min(1, 7/D)$, where 7 is the D of its reference answer.

Base + RL (9B) — **D = 1253, 0.56 points**

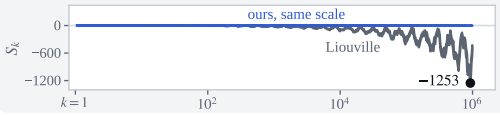
assumes λ — *"Okay, let's assume the Liouville function is the intended solution. Why? Because $f(p) = -1$ is the most common way to make it oscillate."*

raises mod 4 — *"[…] the solution is to set $f(p) = 1$ if $p \equiv 1 \pmod 4$ and -1 if $p \equiv 3 \pmod 4$? No, that is the Dirichlet character χ_4. The partial sums of χ_4 are $O(\sqrt{n})$."*

keeps λ — *"The partial sums of $\lambda(n)$ are also $o(n)$. Which one is better? Actually, $\lambda(n)$ is more random-like. So $\lambda(n)$ should be the answer."*

Submitted program — **Liouville λ**

```
Ω(k) = # prime factors of k, with repeats
f(k) = +1 if Ω(k) is even, else -1,
  i.e. f(p) = -1 for every prime p
```



InnoPrior + RL (9B), ours — **D = 10, 70 points**

raises mod 4 — *"Or 'Assign $f(p) = 1$ for $p \equiv 1 \pmod 4$ and $f(p) = -1$ for $p \equiv 3 \pmod 4$.' This is the non-principal character modulo 4."*

undefined at 2 — *"Dirichlet characters $f(x)$ are defined for $(x, n) = 1$. Here we need $f(2)$ defined."*

free on primes — *"If f is completely multiplicative, $f(2k) = f(2)f(k)$. So $f(p)$ can be chosen arbitrarily."*

Submitted program — **mod-4 rule**

```
prime p: f(p) = +1 if p % 4 == 1,
         else -1  (p = 2 included)
f(i*p) = f(i) * f(p), in a linear sieve
```

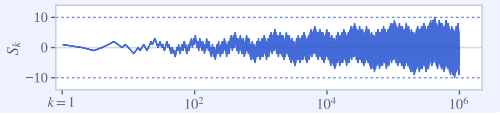


Figure 10: FrontierCS problem 83: excerpts of each 9B model's reasoning after RL, the rule it submitted and the partial sums of its output.