Arbor: How a Hypothesis Tree Beats Top AI Coding Agents by 2.5x on the Same Compute

01

Picture this. Your engineering team ships an AI agent that searches internal company documents and answers employee questions. In development it behaves beautifully. In production it does the opposite — hallucinating answers and skipping over the constraints that matter most. The fix is almost never a one-line patch. It means grinding through a trial-and-error cycle of adjusting chunking strategies, retrieval methods, and system prompts all at once. And because those changes are tangled together, working out which adjustment actually helped becomes nearly impossible.

02

A new framework called Arbor is built to break that deadlock. Created by researchers at Renmin University of China's Gaoling School of Artificial Intelligence and Microsoft Research and released on June 10, 2026, Arbor reframes AI-driven optimization as a process of cumulative learning rather than a string of disconnected guesses. It organizes hypotheses, experiments, and findings into a tree, so the system can learn from earlier failures and make verified improvements that build on one another over time.

The headline result is hard to ignore. In real-world engineering tests, Arbor produced more than 2.5 times the verifiable performance gains of standard AI coding agents — while running on the very same resource budget.

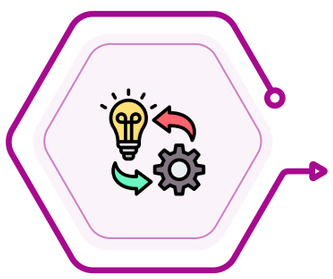
Why throwing more compute at the problem does not work

As AI systems grow more capable, the industry expects them to take on heavier work, including autonomous optimization (AO): improving software artifacts such as agent harnesses or model-training pipelines with little human oversight. AO is the core loop of autonomous research. An agent is handed a starting artifact — a machine-learning codebase, say, or a data pipeline — along with a goal, and it has to keep improving that artifact through experimental feedback on its own.

The instinct is to give the agent more time and more compute and let it grind. In practice, that rarely produces better results, and the reason is widely misunderstood. As Jiajie Jin, a co-author of the paper, put it to VentureBeat, a loop is not the same as progress. When the objective is vague or the metric is easy to game, a long-running agent mostly just generates throwaway “improvements” faster — output nobody actually wanted. Hard problems take many attempts to crack, and raw persistence without structure does not get you there.

Isolation is the whole trick

Arbor's central idea is to stop changing everything at once. Return to that internal AI assistant and imagine the goal is to optimize its retrieval-augmented generation (RAG) pipeline. Ask a single agent such as Claude Code or Codex to “improve accuracy,” Jin explained, and it will typically rewrite several things in one pass — the chunking, the prompt, and the retrieval method together. Those edits are now entangled, so there is no way to tell which one helped, and the agent has mutated the repository directly, with no isolation between experiments.



Arbor treats each lever as its own hypothesis instead. Chunking becomes one branch, retrieval another, the prompt a third — and each is implemented and tested in its own isolated git worktree. The payoff is clean attribution: the system can state, in effect, that reworking the retrieval side delivered a measurable gain while a different tactic actively hurt. When an executor agent finishes a task and returns its report, the coordinator records that evidence in the tree and propagates the insight back up to the parent nodes, so later decisions inherit what earlier ones learned.

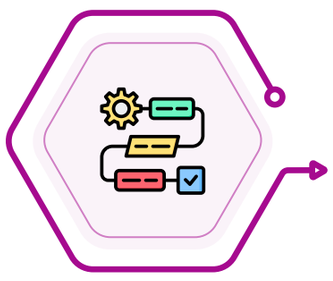
How it performed

The team evaluated Arbor two ways: on an autonomous optimization suite drawn from real research settings, and on MLE-Bench Lite, a standardized machine-learning-engineering benchmark. The AO suite spanned several corners of AI development, including model training, harness engineering, and data synthesis. The researchers mixed and matched backbone models for the coordinator and executor roles — among them Claude Opus 4.6, GPT-5.5, and Gemini-3-Flash — and pitted Arbor against the strongest available coding agents, Codex and Claude Code, with every system handed the same resources. On the MLE-Bench Lite tasks, Arbor was also measured against leading agentic research systems including AI-Scientist, ML-Master, and AIDE.

Arbor came out ahead consistently. Across six autonomous optimization tasks covering model training and data synthesis, it delivered over 2.5 times the average relative held-out gain of both Codex and Claude Code, and posted the best held-out test results on every task evaluated. On MLE-Bench Lite, Arbor running on GPT-5.5 reached an Any-Medal score of 86.36 percent — the share of tasks where it performed well enough to earn at least a bronze-level result. On the BrowseComp accuracy comparison it scored 67.67 against Claude Code's 53.33.

What it means for enterprises, and how to get it

For companies, the appeal is direct: Arbor points toward automating the continuous improvement of complicated, real-world engineering systems — the kind of ongoing tuning that usually eats senior engineers' time. Rather than paying for an agent to churn indefinitely, teams get a method that compounds verified gains and shows its work, attributing each improvement to a specific change.



Arbor is open source and available on GitHub under the RUC-NLPIR/Arbor repository. It ships with a command-line runtime and a set of skills designed to plug into other coding agents, so teams can fold it into workflows they already run rather than rebuilding around it.

The takeaway

The deeper lesson sits underneath the benchmark numbers. The bottleneck in autonomous optimization was never a shortage of compute or patience — it was the absence of structure that lets a system learn from what it already tried. By turning a flat, entangled grind into a branching tree of isolated, attributable experiments, Arbor converts more attempts into more knowledge. On equal hardware, that structure is the difference between an agent that loops and an agent that actually improves.