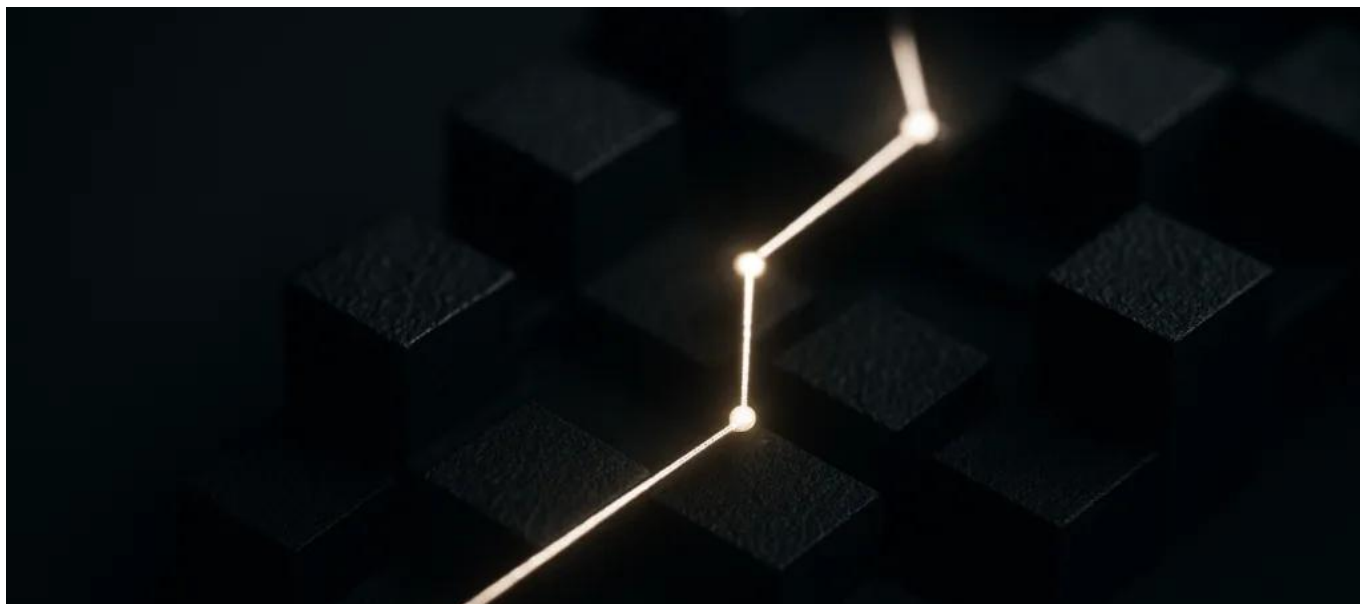




Agent Lightning v1.0 in Practice: What 3,500 Lines of Agentic RL Buy You, and What They Do Not

Can you train your existing LangGraph or Claude Code agent with reinforcement learning without rewriting it as an RL environment? Microsoft says yes, in about 3,500 lines. The install page says yes, on Linux with CUDA 12.9 or 13.0.

THE SHORT VERSION

Microsoft Research announced Agent Lightning v1.0 on October 7, 2026: a roughly 3,500-line MIT-licensed framework that trains agents through their real execution harnesses instead of rewritten RL environments. The architecture is light and the integration list covers nine frameworks. The training stack is not light: Linux, NVIDIA CUDA 12.9 or 13.0, verl 0.8.0 or 0.7.1, flash-attn built from source, no supported native Windows path.

The pitch and the prerequisites are two different documents, and both are worth reading



carefully. The [Microsoft Research announcement](#) frames v1.0 around a line count, which is a slightly cheeky move: it signals that the framework itself is small enough to read in one sitting. That part is true and it matters. The line count says nothing about the GPU training stack underneath, which is where most teams will stall.

Training-Agent Disaggregation keeps your agent out of the trainer

The core design idea in the [Agent Lightning documentation](#) is Training-Agent Disaggregation. Your agent keeps its own framework, prompts, tools and control flow. Agent Lightning sits around it and collects trajectories, events, tool interactions and rewards. The agent stays the thing you already shipped.

That inverts the usual bargain. Classic RL tooling asks you to reimplement your task inside a gym-style environment with a clean step function, so the thing you train is a model of your agent rather than your agent. Every behaviour that lives in your orchestration layer, your retry logic, your tool error handling, your prompt templates, either gets reimplemented or quietly dropped. Then you deploy the trained policy back into the real harness and discover the gap.

Three components carry the design. The API Gateway handles rollout intake and model services. The Rollout Controller launches rollouts locally or as Kubernetes Jobs. A Customized Trainer consumes the trajectories. The Kubernetes path is new in v1.0, and it is the piece I would look at hardest before using this for anything beyond a single-box experiment, because rollout throughput is usually where agentic RL dies. Your agent spends most of its wall clock waiting on tool calls and model responses, not on gradient steps.

The credit assignment piece is called LightningRL. It decomposes a multi-step trajectory into trainable units so reward attributes across intermediate actions and tool calls rather than landing as one opaque score on the whole episode. For a five-step agent where step three was the mistake, that is the difference between a usable signal and noise.

i The install reality

The full training stack installs with uv and verl: `bash scripts/setup_verl.sh 0.8.0 cu130` or `0.7.1 cu129`. The docs recommend CUDA 12.9 or 13.0 and prefer verl 0.8.0 with CUDA 13.0. flash-attn is built from source. Native Windows is not a supported primary platform. The package itself is `pip install agentlightning` under the MIT License.



Nine integrations, one command for coding agents, one CUDA toolchain for training

The documented integrations are LangChain, LangGraph, OpenAI Agents SDK, AutoGen, CrewAI, Claude Code, Codex, GitHub Copilot and Microsoft Agent Framework. That is a wide net, and the inclusion of Claude Code, Codex and Copilot changes who this is for. Coding agents install the skill through the GitHub CLI: *gh skill install microsoft/agent-lightning agent-lightning --agent <agent>*. So the entry point for a developer with a coding agent is one command, while the entry point for a training run is a CUDA toolchain.

On the optimization side the docs list GRPO, PPO, reward-model workflows, prompt optimization and supervised fine-tuning, with availability depending on which backend you select. That last clause does real work. Prompt optimization and SFT are a very different commitment from PPO on a self-hosted model, and the framework presents them in the same list. If you are scoping this, ask whether your chosen backend supports the method you want, because the headline list does not answer that.

[Release v1.0.2](#), dated September 29, 2026 at commit d381995, added Tinker as an alternative RL backend plus Azure OpenAI support for inference and supervised fine-tuning. My read: the Azure OpenAI path will get the most real-world use, because it removes the self-hosted GPU requirement for the SFT and inference cases, and the self-hosted GPU requirement is the actual barrier here. The verl path remains the one the docs treat as primary and the one the version pins are written for.



WHAT WORKS

- Your agent stays your agent. Training-Agent Disaggregation means no rewrite into an RL environment, and the framework, prompts, tools and control flow stay where they are.
- Per-step credit assignment via LightningRL, so reward attributes to intermediate actions and tool calls rather than one episode-level score.
- Nine documented framework integrations, including the three coding agents (Claude Code, Codex, Copilot) that most engineering teams already have in hand.
- Kubernetes Jobs as a native rollout target, which is the right abstraction for the part of agentic RL that is I/O-bound rather than compute-bound.
- MIT License. No commercial restriction, no cost, and a codebase small enough (~3,500 lines) that reading it is a realistic afternoon rather than a project.

ROUGH EDGES

- Hard version pins: CUDA 12.9 or 13.0, verl 0.8.0 (cu130) or 0.7.1 (cu129). If your cluster is on an older CUDA, you are not on the documented path.
- flash-attn compiled from source, which is the step most likely to eat a day.
- No supported native Windows. The documented path is Linux plus NVIDIA CUDA.
- v1.0 is a complete refactor. The repo tells pre-1.0 users to move to the legacy branch rather than assume API compatibility, so existing integrations do not carry forward for free.
- Optimization method availability depends on the backend, which means the headline feature list overstates what any single configuration gives you.

The published evaluation covers three tasks with programmatic rewards

The evidence that exists comes from the original paper, [arXiv:2508.03680](https://arxiv.org/abs/2508.03680), which evaluated the approach on Spider text-to-SQL (roughly 200 databases and over 10,000 questions), retrieval-augmented generation with MuSiQue over a Wikipedia-scale corpus of about 21 million documents, and Calc X math question answering. The paper reports stable reward improvement across all three.

That is a real result on three real benchmarks, and it is also the limit of what I can tell you. “Stable reward improvement” is the claim in the research data; I do not have the magnitude, the baselines, or the compute budget behind it. Those three tasks also share a



property that flatters agentic RL: each has a crisp, cheap, automatic reward. SQL either executes and returns the right rows or it does not. Math answers are checkable. Multi-hop QA has gold answers.

So the benchmark set tells you the method works where reward is programmatic. It tells you nothing about the case most teams actually have, where output quality is judged by a human or by another model. If your reward function is an LLM judge, you are stacking one source of variance on another, and the published evaluation does not speak to that. I would treat Spider and Calc X as proof that the plumbing is sound, not as evidence that your customer-support agent will improve.

Install it if your reward is machine-checkable, skip it if your CUDA is old

Install it if you have a GPU box or cluster on CUDA 12.9 or 13.0, an agent built on one of the nine supported frameworks, and a task where you can compute a reward automatically and cheaply. That combination is narrower than the announcement implies, though it is not rare: text-to-SQL, code generation with tests, structured extraction with validators, anything machine-checkable. For those teams the value is concrete, because the alternative is building the trajectory collection and credit assignment layer yourself, and 3,500 MIT-licensed lines is a reasonable starting point.

Close the tab if your evaluation story is “a human looks at it and says it feels better”. The framework will happily train against whatever reward you supply, and ****a weak reward signal with a strong optimizer is a fast way to get an agent that games your metric****.

Close the tab also if your team is on Windows without a Linux GPU environment, or if your production clusters are pinned to an older CUDA. The version requirements are specific enough that treating them as suggestions will cost you a week. This pattern shows up often enough in Microsoft’s agent tooling that I have written about it before, in the context of [what the Agent Framework docs promise versus what the install flags tell you](#): the conceptual layer ships polished and the operational layer ships honest, and you have to read both.

One more thing about the refactor. v1.0 is a complete rewrite and the repo directs pre-1.0

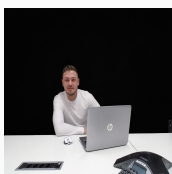


Agent Lightning v1.0 in Practice: What 3,500 Lines of Agentic RL Buy You, and What They Do Not

users to the legacy branch. If you built anything against the earlier releases, budget for a port rather than an upgrade. That is the correct call by the maintainers (a clean break at 1.0 beats years of compatibility shims) but it does mean early adopters pay for the architecture change.

My take: the disaggregation idea is the genuinely good part here, independent of the implementation. Separating the agent from the training loop is the right architectural boundary, and most RL tooling gets it wrong by pulling the agent inside the trainer. If Agent Lightning does nothing else, it shows the boundary is drawable in ****a small amount of code****, and the MIT License means anyone can take that design whether or not they take the dependency.

What I am less sure about is the Kubernetes rollout story. It is new in v1.0, it is the right idea, and I have no evidence about how it behaves under real load: how rollout failures are retried, how partial trajectories are handled, what happens when a tool call hangs. The available documentation does not answer any of that. Unconfirmed, but my expectation is that the next several releases will spend their effort there. If you are weighing whether your reward function is strong enough to be worth training against, that is the conversation worth having before you touch the install script, and I am [happy to talk it through](#).



WRITTEN BY

Artur Markus

Software engineer in Basel building AI infrastructure and agentic systems, with a background in pharmaceutical automation, GxP compliance, and validation. I help teams turn AI from a chatbot into a reliable operational layer.

Book a meeting →