Apodex-1.0: A Verification-Centric Agent Team for Discoverative Intelligence

Apodex Team

We present **Apodex-1.0**, a verification-centric model for deep research. The trained model alone, Apodex-1.0, runs as a standard tool-using ReAct agent. Deploying it in our *heavy-duty mode*—an asynchronous agent team in which sub-agents specialize in retrieval and verification, route their reports through a shared evidence pool, and feed a global verifier that reasons over the assembled evidence graph to produce the final answer—yields **Apodex-1.0-H**, our flagship *heavy-duty solver*. Under the hood, a high-quality data pipeline and a three-stage post-training recipe (SFT, agentic DPO, RL) substantially raise the deep-research capability of the Qwen3.5 base while *preserving* its general knowledge, coding, reasoning, and instruction-following capabilities.
Apodex-1.0-H sets a new state of the art across both open and closed-source models on deep-research benchmarks—**BrowseComp**, **BrowseComp-ZH**, **HLE-text**, **DeepSearchQA**, **FrontierScience-Olympiad** and **FrontierScience-Research**. Every claim in the final report it produces is backed by an explicit evidence chain and independently audited before delivery. We open-source **Apodex-1.0-mini** (35B-A3B) alongside three smaller variants (0.8B, 2B, 4B); remarkably, the 4B variant outperforms every open-source 30B-class model on deep-research tasks.

🌐 **Online Service**: https://www.apodex.ai
🔌 **API Platform**: https://platform.apodex.ai
Apodex GitHub Repository: https://github.com/ApodexAI
🤗 **Model Weights**: https://huggingface.co/collections/apodex/apodex-1

Figure 1: Comparison of Apodex-1.0 with state-of-the-art agents and agentic foundation models.

1. Introduction

The hardest research problems an agent encounters today are not bounded by model capacity but by what the model is allowed to interact with. Long-horizon research tasks share a single structural feature: a single forward pass is not enough, and a single context window cannot hold the work. They demand reasoning interleaved with retrieval, tool use, and verification, sustained over hundreds of steps and many parallel branches of investigation. Reliability on such problems cannot come from a model's parametric memory alone. It must come from *discoverative intelligence*: the capacity to reason through active engagement with the external world, and to check that engagement against itself before committing to an answer.

The dominant approach today is the ReAct paradigm, in which a single agent executes a think–act–observe loop within one context window. This design carries a paradigm-level ceiling. The loop lengthens, the context becomes congested, exploration branches contaminate one another, and the only available verification mechanism—self-reflection—degrades with each additional turn. Empirically, a ReAct agent saturates after a few hundred steps regardless of the underlying model. Scaling parameters or context further does not lift this ceiling; it sits below them.

Apodex widens the frame: three jointly designed pieces sit behind the system.

Heavy-duty mode: an agent team with global verification. When the trained model (Apodex-1.0) is deployed in heavy-duty mode, an orchestrator *asynchronously* spawns specialized sub-agents for retrieval and verification, each running with its own context, prompt, and tool set. Sub-agent reports flow into a shared evidence pool whose status table the orchestrator reads asynchronously, never blocking on the slowest task. When two reports disagree, when a specific claim needs grounding, or when a draft is ready for a last pass, the orchestrator dispatches the work to a dedicated *verification team*—a conflict reviewer, a fact checker, and a draft-report reviewer. Once exploration completes, a *global verifier* reasons over the assembled evidence graph to produce the final answer. The resulting system is **Apodex-1.0-H**. In our deployment, this architecture coordinates up to 150 sub-agents executing over 15,000 steps within a single task—two orders of magnitude beyond the saturation point of a single-agent loop.

A three-stage post-training pipeline. A high-quality data pipeline together with a three-stage post-training recipe—SFT for behavioral initialization, agentic DPO for trajectory-level judgment, and RL on long agentic rollouts—substantially raises the deep-research capability of the Qwen3.5 base. Crucially, the recipe is designed to preserve rather than override: the base model's general knowledge, coding, reasoning, and instruction-following capabilities all survive the training process intact, so the post-training is additive on the deep-research axis rather than a trade across axes.

AgentOS, a task-agnostic runtime. The agent team runs on AgentOS, a runtime whose kernel knows nothing about any specific task. Workflow policy lives above a narrow runtime facade and task-agnostic execution mechanisms live below it, so adding a new application is a folder of plugin code rather than a patch to the kernel.

Running the trained model inside this agent team yields **Apodex-1.0-H**, our flagship system. Apodex-1.0-H sets a new state of the art across both open and closed-source models on deep-research benchmarks—**BrowseComp**, **BrowseComp-ZH**, **text-only HLE**, **DeepSearchQA**, and **FrontierScience-Research**. Every claim in the final report it produces is backed by an explicit evidence chain and independently audited by the verification team before delivery. We release **Apodex-1.0-mini** (35B-A3B) and three smaller variants (0.8B, 2B, 4B) as open weights; remarkably, the 4B variant outperforms every open-source 30B-class model

on deep-research tasks.

2. Related Works

Agentic Large Language Models Recent advances in LLMs have increasingly focused on enabling *agentic behavior*, where models autonomously decompose complex goals into sub-tasks, invoke external tools, and iteratively refine intermediate decisions based on environmental feedback [1]. Unlike conventional chatbots that rely primarily on single-step responses, agentic LLMs maintain persistent reasoning traces across many steps and dynamically coordinate tool execution.

Frontier models increasingly integrate such capabilities either during training or through tightly coupled inference frameworks. Representative examples include GPT-5.5 [2], Claude-Opus-4.8 [3], Gemini-3.1-Pro [4], DeepSeek-V4 [5], Kimi-K2.6 [6], Seed-2.0-Pro [7], and Muse Spark [8]. Collectively, these developments indicate a paradigm shift in which foundation models evolve from passive language generators into *general-purpose autonomous agents* capable of executing complex workflows and interacting with real-world environments.

Deep Research Agents Building on the emergence of agentic LLMs, recent work has introduced *deep research agents*, a class of LLM-based systems designed for open-ended knowledge synthesis tasks that require long-horizon reasoning and intensive information retrieval. Rather than answering questions solely from pre-trained knowledge, these systems actively acquire external information, iteratively refine hypotheses, and synthesize evidence from multiple sources to produce structured research outputs.

Industrial systems have begun deploying such capabilities at scale, including OpenAI Deep Research [9], Claude Research [10], Kimi-Researcher [11], and Grok DeepSearch [12], all of which couple LLMs with integrated web browsing and multi-step planning. In parallel, the open community has explored a variety of approaches for building open deep research agents, including Tongyi DeepResearch [13], WebThinker [14], DeepResearcher [15], and others. A common emerging thread is the use of *agentic mid-training* or post-training on synthetic research trajectories to instill long-horizon search-and-verify behavior directly into the model rather than at the prompt layer.

These efforts highlight a broader shift toward LLM systems that function as autonomous research assistants, capable of long-horizon information gathering, reasoning, and synthesis for complex open-ended tasks.

Multi-Agent Systems and Agent Teams A parallel line of work explores LLM-based multi-agent systems, in which several language-model agents cooperate, debate, or specialize to solve tasks that exceed the capacity of any single agent. Representative frameworks include AutoGen [16], MetaGPT [17], ChatDev [18], and AgentVerse [19], which compose role-playing agents through scripted protocols. A related thread investigates multi-agent debate and consensus [20, 21], where multiple completions are reconciled into a single answer. Most of these systems treat the multi-agent topology as an external scaffold layered around a generalist model. Apodex departs from this convention by training the team behavior—sub-agent spawning, asynchronous coordination, and self-verification—directly into the model, rather than imposing it through an outer harness.

Verification in LLM Agents A growing body of work targets verification as an explicit step in LLM reasoning. *Chain-of-Verification* (CoVe) [22] has a single model draft a response, plan independent verification questions, answer them, and revise the draft accordingly. *Self-Refine* [23] and *Reflexion* [24] iterate generation with self-critique signals across rounds. Process reward models [25] train a separate verifier to score intermediate reasoning steps under a learned reward. The common pattern across these approaches is that the verifier is either the generator itself in a different prompt, or a single model trained to score the generator's outputs.

Apodex differs along two axes: the verifier is an *independent agent* with its own context, prompt, and tool set rather than the generator in a different mode; and verification is dispatched as an *asynchronous team activity*—a conflict reviewer, a fact checker, and a draft-report reviewer operating in parallel—rather than a sequential self-refinement loop.

Reinforcement Learning for Agentic LLMs Reinforcement learning on long agentic rollouts surfaces problems largely absent from conventional RLHF: heavy-tailed trajectory lengths stall synchronous pipelines, off-policy drift between trainer and inference engine becomes structural rather than incidental, and on Mixture-of-Experts policies a small number of high-magnitude importance-sampling ratios can destabilize an entire update [26, 27]. Recent work addresses these along two complementary axes: infrastructure designs that disaggregate or asynchronously interleave training and inference (ROLL [28], ROLL Flash [29]), and algorithmic modifications that mask or reshape the heavy importance-ratio tail (IcePop [27], DAPO [30], DPPO [31], RPO [32]). Our training pipeline builds on this toolbox while introducing additional infrastructure and algorithmic refinements detailed in Section 7.3.

3. The Apodex Agent Team

As foreshadowed in Section 1, Apodex's deep-research system is organized as an *agent team*: an orchestrator (the main agent) decomposes the user query, dynamically spawns specialized sub-agents, collects their reports asynchronously, and dispatches a dedicated verification team against the assembled evidence before synthesizing the final answer (Figure 2). The team is assembled around each problem rather than configured as a fixed pipeline. Three properties distinguish this design from conventional multi-agent systems—self-verification as an intrinsic team behavior, asynchrony as the coordinating principle, and agent-level parallelism as the resulting scaling axis—examined in the subsections that follow.

3.1. Self-Verification

The defining capability of the agent team is self-verification. Consider a familiar failure mode of long-chain reasoning: a single trajectory commits early to a hypothesis and rationalizes downstream from that commitment, with no opportunity to recognize when an alternative direction would have been better. Maintaining multiple reasoning paths can surface such errors—agreement across paths increases confidence, divergence surfaces uncertainty, and inconsistencies signal errors no single chain could catch from within its own perspective.

Diversity alone, however, has a structural ceiling when bounded to a single agent. The alternative paths still share the same prompt, the same tools, the same accumulated context, and the same parametric biases. When the underlying reasoner holds a systematic error, every parallel path inherits it. Reliable verification must therefore be not merely diverse but genuinely *external*.

The agent team provides exactly this. Verification is dispatched to an independent agent whose role is structurally different from the reasoner's: the verifier does not share the reasoning trace it audits, is prompted not to continue the reasoning but to evaluate it, and may invoke grounding tools that the original reasoner did not. In practice, the verification team comprises three specialized roles, each engineered to probe a distinct failure mode:

- A **conflict reviewer** that reconciles contradictory reports between two or more expert sub-agents, identifies which claim is better supported by the underlying evidence, and surfaces unresolved disagreements when no resolution is possible.

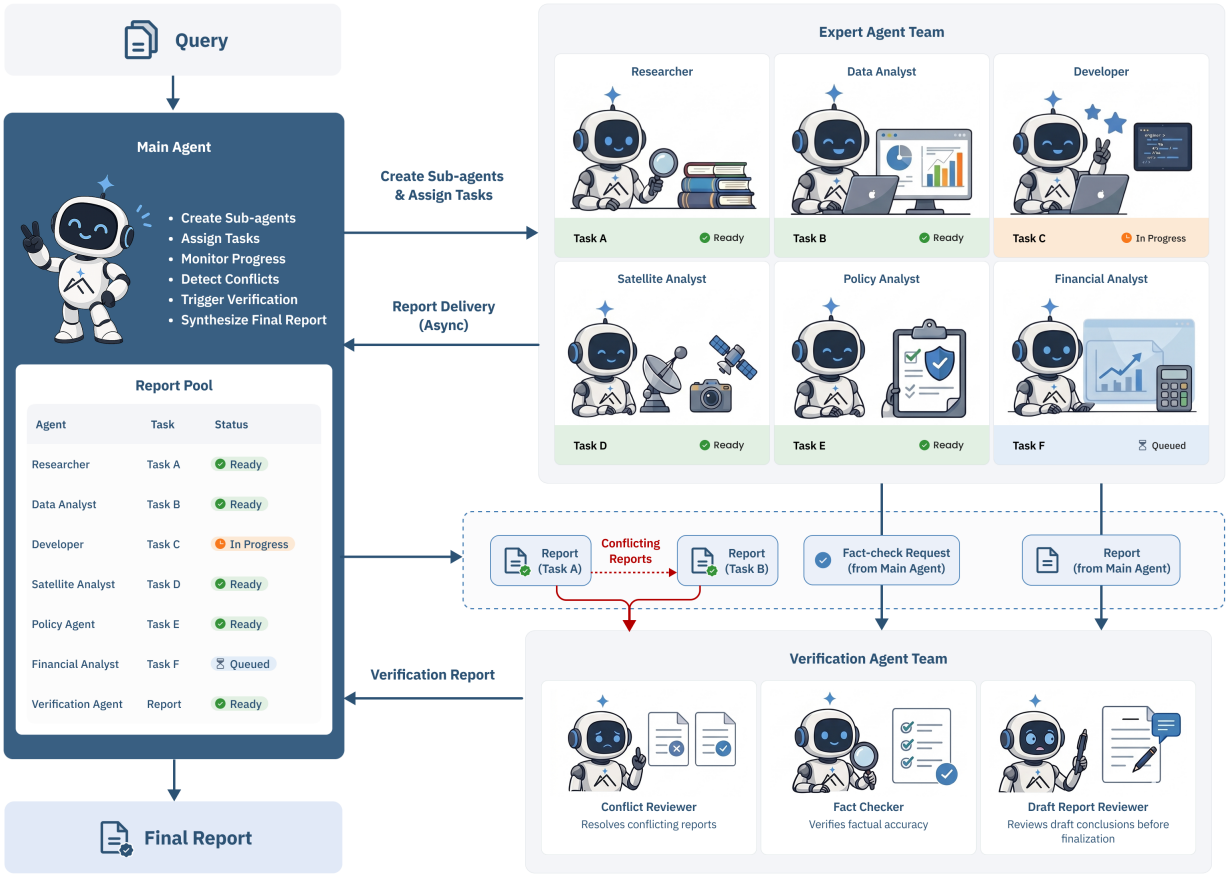

Figure 2: The Apodex agent team. A main agent dispatches expert sub-agents—researcher, data analyst, developer, and domain specialists—whose reports collect asynchronously in a shared report pool. When reports conflict, when a claim needs grounding, or when a draft is ready for a final pass, the main agent dispatches the work to a verification agent team consisting of a conflict reviewer, a fact checker, and a draft-report reviewer. The verified output becomes the final report.

- A **fact checker** that audits individual factual claims against fresh evidence, independent of the sub-agent that originally produced the claim.
- A **draft-report reviewer** that examines the synthesis as a whole—citation coverage, claim–evidence alignment, and logical coherence—before the report is delivered to the user.

Verification proceeds not as introspection but as an external check by agents uninvolved in the process under review. This is what distinguishes the agent team from a conventional multi-agent system: verification is an *intrinsic capability* of the team, a model-level behavior rather than a system-level workaround. This autonomy is made possible by the asynchronous coordination layer described next.

3.2. Asynchronous Coordination

With verification established as a team behavior rather than a pipeline stage, the surrounding architecture is organized around asynchrony. In a synchronous multi-agent design, every round waits for the slowest sub-agent, forcing verification into one of two undesirable regimes: it either blocks exploration or is omitted under time pressure. The agent team removes this barrier through a shared *report pool*.

Sub-agents spawn and return independently, depositing their outputs into the pool with a status flag (`queued`, `in progress`, `ready`). The main agent reads the pool's status table at its own pace, dispatching downstream work—verification, synthesis, or further sub-agent spawning—as reports become ready, rather than waiting on a round to complete. A fast retrieval is not stalled by a slow crawl, and a finding returned early can feed into verification while other branches continue executing. When a single sub-agent stalls or fails, the remainder of the team proceeds unaffected, and fault isolation emerges as a structural consequence of the architecture rather than as an explicit mechanism.

Beyond fault tolerance, asynchrony grants verification structural freedom. Exploration, verification, and synthesis become three independent control loops, each advancing at its own pace. The report pool, recording every finding, every verdict, and every user intervention, forms the causal backbone of the system. Conclusions become auditable, retractable, and forkable. The team returns not only an answer but a complete research history.

3.3. Agent-Level Scaling

The same architecture that liberates verification also renders parallelism native. Where a single agent processes one task at a time, an agent team processes many concurrently: a research problem requiring five distinct leads across three sources is pursued by five sub-agents executing simultaneously. This concurrency is inherent to the team structure rather than an optimization layered on top, yielding more directions explored per unit time, more findings cross-referenced, and more verification carried out in parallel.

This parallelism defines a new scaling curve. A ReAct agent reaches its ceiling at a few hundred steps, bounded by a single context window. The agent team distributes the cognitive load across sub-agents that each occupy an independent context. In our deployment, the architecture coordinates up to 150 sub-agents executing over 15,000 steps within a single task, two orders of magnitude beyond the saturation point of a single agent. This is not the familiar scaling law of larger models on larger clusters but a different axis entirely: scaling *agents* rather than parameters. The team behavior that drives this curve—sub-agent spawning, verification dispatch, asynchronous coordination—is not a framework wrapped around a passive model but a mode of reasoning the model has been trained to perform natively, through the training pipeline described in Section 7.

4. Heavy-Duty Verification

The agent team of Section 3 is itself a test-time scaling axis: many specialized sub-agents working under one orchestrator, with in-flight verification by a dedicated reviewer team. For the hardest workloads we add a further scaling axis on top—*heavy-duty mode*—in which Apodex runs *multiple agent teams in parallel* (or, in the math case, multiple iterative attempts) and a *global verifier* reasons over their combined output. Conceptually, heavy-duty mode is to the agent team what the agent team is to a single agent. The global verifier instantiates a different primitive per domain: evidence reasoning over an assembled claim–evidence graph for deep research, causal-evidence comparison for coding candidates, and generate–verify–revise for mathematical proofs. We describe each in turn.

4.1. Deep Research Verification via Evidence Reasoning

On real-world research tasks, the answer is not a single object to be scored but a composition of many evidence pieces that must fit together across sources; a single unsupported or contradicted piece can invalidate the whole. Selecting among candidate answers by agreement or aggregated score—which is the natural baseline when multiple agent teams produce multiple answers—collapses them into a count or an average and throws away the evidence that produced them. The approach saturates as soon as independent teams start duplicating the same findings rather than completing the missing ones.

Apodex reframes global verification as *evidence-based reasoning over a shared graph of evidences and claims*. After the parallel agent teams finish their explorations, the global verifier assembles all collected evidence into a meta-graph whose nodes are atomic findings and tentative claims and whose edges record support and contradiction. The verifier then reasons over the full graph rather than selecting among finished answers, weighing each claim against the support and contradiction it carries and judging corroboration strength jointly with source diversity. Every claim in the final answer traces back to a node in the graph, keeping the output auditable. Crucially, what the verifier reasons over is the assembled graph, not the population of teams that produced it—so the same weights operate from a single explorer up to a large parallel swarm without modification.

This verification reasoning behavior is learned rather than prompted: we add the global verification task to Apodex's RL training data, with data preparation and training following Argus [33].

4.2. Coding Verification via Causal-Evidence Comparison

A candidate patch can pass the visible tests bundled with a task, read as semantically reasonable, and come with the agent's own claim of success, yet still fail to causally fix the underlying bug. We call this failure mode *pseudo-correctness*. The default form of LLM verification—asking a judge which of two candidates "looks better" or to assign an absolute quality score—breaks down on coding tasks because perceived quality and ground-truth correctness can diverge arbitrarily: a patch that masks a bug with a `try/except` can read as cleaner than the correct fix, and an agent that fabricates a success message can sound more confident than one that actually ran a reproducer.

In heavy-duty mode, Apodex runs multiple coding agent teams in parallel and applies *causal-evidence comparison* across their candidates: the global verifier picks the trajectory or patch whose answer is most strongly supported by execution or implementation evidence, and least likely to be pseudo-correct. The question shifts from *which candidate looks better* to *which is most likely to have causally solved the task*. Coding quality is decomposed into three independently scored rubric axes, each engineered as a probe against a distinct pseudo-correctness mode: **Comprehension** asks whether the candidate identified the real problem rather than pattern-matching on surface features; **Causality** asks whether the solution addresses

the actual cause across the full input distribution rather than a visible slice; and **Empirical Grounding** asks whether claimed correctness is backed by observable execution rather than only asserted. The same conceptual framework spans trajectory-level tasks (SWE-Bench Verified [34], Terminal-Bench v2 [35]) and pure code-generation tasks (LiveCodeBench v6 [36]); only the instantiation differs depending on what kind of evidence is available.

The clearest empirical implication is that selection quality, not model scale, becomes the dominant lever on coding benchmarks: as we report in detail in Section 8.3, Apodex-1.0-mini (35B-A3B) in heavy-duty mode surpasses Apodex-1.0 (397B-A17B) without verifier on LiveCodeBench v6, matching the large model at roughly an order of magnitude fewer active parameters.

4.3. Math Verification via Generate–Verify–Revise

Mathematical proofs are a sharp testbed for verification: a proof is correct only if every step is justified, and a single unsupported claim invalidates the whole argument. Answer-only benchmarks miss this entirely. Unlike deep research or coding, where multiple agent teams can usefully be run in parallel and reconciled, mathematical proofs benefit more from *iterative refinement*: each new attempt can be steered by an explicit diagnosis of the previous one's weaknesses.

Apodex's math recipe is **generate–verify–revise (GVR)**, an iterative refinement loop. The model first writes a full proof. A *self-grader*—the same model, given only the problem statement and the candidate proof—assigns a 0–7 score and a short written critique. Conditioned on its previous attempt and the critique, the model then writes a fresh proof, and the loop repeats for up to K rounds. The highest-scoring attempt is submitted. A deliberate design choice is that the in-loop grader is never shown the reference solution or the rubric: leaking either turns the grader into an oracle and inflates apparent scores. The grader's *written feedback*, not just its scalar score, is what distinguishes GVR from naive best-of-K sampling—each new attempt is steered by an explicit diagnosis of the previous one.

Across all three domains, heavy-duty mode shares the same architectural recipe: many attempts (parallel for deep research and coding, sequential for math) plus a domain-specific global verifier on top. Together with the in-flight verification team of Section 3, this realizes the claim made in the abstract: every claim in the final report Apodex-1.0-H produces is backed by an explicit evidence chain and independently audited before delivery.

5. AgentOS: A Task-Agnostic Runtime

The agent team of Section 3 and the heavy-duty verification machinery of Section 4 both require a runtime—a system that schedules work across many concurrent sub-agents, manages context, executes tools, records what happened, and enforces operational boundaries. Apodex-1.0 (as a standalone ReAct agent) and Apodex-1.0-H (with the full heavy-duty mode engaged) are both *workflows* hosted on this runtime, and they share the same kernel binary; only their plugin configuration differs.

The dominant failure mode at the runtime layer is *runtime–workflow coupling*. When each task embeds its own loop, state model, permission rules, tracing, and termination logic into the engine, adding a new benchmark, tool, agent topology, or verification strategy becomes a special case inside the kernel. The engine ossifies as a collection of task-specific branches, and new applications cost weeks of kernel surgery rather than a folder of new code. Reliable agentic compute at scale demands the opposite arrangement: a kernel that knows nothing about any specific task, and tasks that reach the kernel only through a stable, narrow surface.

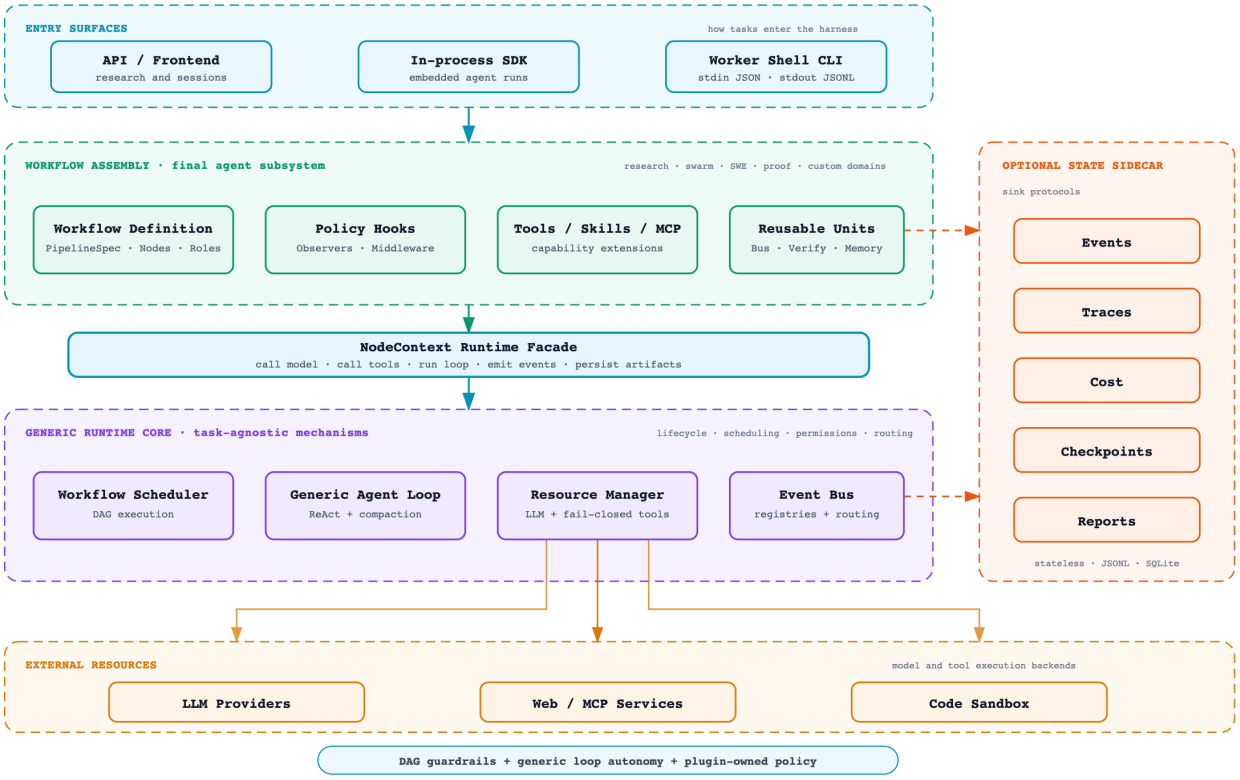

Figure 3: Overview of AgentOS. A task-agnostic execution kernel (generic agent loop, DAG scheduler, event bus, registries) sits below a thin node-runtime interface; workflows, observers, tools, and persistence backends attach as plugins through that interface.

We address this with **AgentOS**, the agent harness underlying Apodex. The kernel provides task-agnostic execution mechanisms—scheduling, model and tool routing, event streaming, checkpoints, traces, cost accounting, permission enforcement, and reusable agent components—while *workflows* loaded as plugins supply the task-specific policy: how to structure the run, which roles and prompts to use, which tools are visible, which observers and middleware are active, and how the final answer is judged (Figure 3). On the same kernel we have built more than a dozen workflows in production: the Apodex-1.0 and Apodex-1.0-H deep-research workflows, coding-agent variants, formal proof writing, heavy-duty reasoning, etc. Workflow policy lives above the runtime facade; task-agnostic execution mechanisms live below it. Adding a new application is a folder of plugin code, not a patch to the kernel.

5.1. Runtime and Plugin Separation

The kernel must not know what task it is running, and a workflow must not know how the kernel executes it. We enforce this through a single narrow facade—the *node context*—that every workflow node speaks to in order to invoke models, call tools, run the generic agent loop, emit events, persist artifacts, and reach the rest of the system. The facade is deliberately small. We resist adding methods to accommodate individual workflows, because each extension would otherwise become a future obligation of the kernel.

Above and below this facade the system is layered so that dependencies always point inward. *Entry surfaces*

(API routes, in-process SDK, worker CLI) submit tasks. *Workflow plugins* declare task-specific behavior: pipeline structure, nodes, roles, prompts, policy hooks, tools, skills, MCP servers, reusable components. The *node context* is the only surface they see. Below it sit the *generic runtime core* (scheduling, agent-loop execution, resource management, event routing, registries), the *state sidecars* that attach asynchronously through sink protocols for events, traces, cost accounting, checkpoints, and telemetry, and the *external resources* at the bottom—model inference engines, web and MCP services, sandboxed execution environments. No dependency runs from the kernel to a workflow, so adding a workflow never edits the kernel.

5.2. Generic Loop, Hooks, and Components

Inside the kernel is a single configuration-driven ReAct-style loop [1]: each turn calls the model, parses tool calls, executes them, appends observations, updates state, optionally compacts context, and repeats until a stop condition is reached. The loop itself is deliberately empty of task knowledge. Everything that varies across applications enters through two complementary surfaces, neither of which is allowed to grow a new branch inside the loop.

Policy hooks carry the cross-cutting behavior. *Observers* react to loop-level events—loop start, model response, tool result, turn end, loop end—and may collect evidence, track plans, synthesize assertions, stream progress, enforce budgets, detect repetition, or return interventions for the next turn. *Middleware* wraps the model and tool calls themselves, handling tracing, token accounting, skill injection, summarization, retries, fallback routing, and request/response transformation. Switching workflows amounts to swapping the set of hooks; the deep-research workflow that underlies Apodex-1.0-H attaches roughly a dozen of them, none of which the kernel is aware of.

Components carry the composable behavior. Where a hook reacts to events, a component is a reusable assembly that a workflow opts into when its task demands it: the *agent bus* through which Section 3's orchestrator spawns sub-agents, the *verifier components* that Section 4's in-flight reviewers and global verifier instantiate, *memory components* for long-running context, and *write-audit cycles* for iterative improvement. Components are more structured than individual tools but still live above the runtime core: they are workflow-level building blocks, not mandatory kernel machinery. Hooks and components together keep domain logic out of the loop while still letting each workflow customize what happens at every meaningful point inside it.

5.3. Workflow Scaffold

A workflow declares its control flow as a directed acyclic graph of nodes that the runtime compiles and schedules. The design principle is a *minimal scaffold with a free interior*: a few mandatory nodes provide macro-level guarantees about the run—typically a clarification node at the start, a verification gate, and a report node that always produces a final answer—while between them sits a single heavy node that runs the open-ended reasoning loop of Section 5.2. Inside that node the model is free to choose tools, spawn parallel sub-agents, and re-plan; the surrounding nodes enforce the task contract without dictating how the answer is reached. In the deep-research workflow that underlies Apodex-1.0-H this yields the flow $\mathsf{clarify} \rightarrow \mathsf{solve} \rightarrow \mathsf{verify} \rightarrow \mathsf{report}$, where the verification gate loops back to the solve node whenever citation coverage, average confidence, share of disputed claims, or unresolved-claim count fails its threshold.

The same scaffold absorbs very different applications without changing shape. A coding workflow focuses on sandboxed command execution and patch validation. A proof workflow emphasizes iterative writing and grading. A benchmark-adjudication workflow loops over candidate answers under a verifier-prompt template. None of them rewrites the kernel; they differ only in which nodes the DAG declares and which hooks and components those nodes invoke.

Table 1: The five extension points of AgentOS. Each is added as plugin code or configuration and discovered automatically at startup, with no change to the kernel.

Extension	How it is added	Examples
Tool	a decorated function registered with the tool registry	`web_search`, `bash`
MCP server	a single entry in the `mcpServers` config	`github`
Skill	a `SKILL.md` folder (description, allowed tools, prompt)	verification, finance
Component	a reusable composition unit	memory, write-audit cycle
Workflow	a `PipelineSpec` plus a `register()` hook	deep research, SWE, proof

5.4. Extending AgentOS

Every domain concern lives in the plugin layer, so extending AgentOS is a matter of adding plugins rather than modifying the runtime; bootstrap discovery picks them up automatically. The five extension points are summarized in Table 1. A *tool* is a decorated function registered with the tool registry. An *MCP server* is a single entry in the standard `mcpServers` config, whose tools are exposed at startup through the same interface as native tools. A *skill* is a folder containing a `SKILL.md` file whose header declares a triggering description and the tools it may use, and whose body is the prompt injected when the skill fires. A *component* is a reusable composition unit, such as a memory module or a write-audit cycle, that a workflow can pull in without writing it from scratch. A *workflow* is a folder declaring a pipeline specification, which may freely mix built-in nodes such as clarify and report with custom ones; it is selected at run time by its pipeline identifier. None of these requires a change to the kernel, and the same kernel binary runs every workflow we have ever written.

6. Training Data

Apodex's post-training data has two purposes: *teach* the model the long-horizon search-and-verify behavior that defines deep research, and *preserve* the general, coding, reasoning, and instruction-following capabilities it inherits from Qwen3.5. We accordingly construct two complementary data streams—one bespoke for deep research (Section 6.1), one mixed in to prevent catastrophic forgetting (Section 6.2).

6.1. Deep Research Data

Constructing training data for deep research demands more than general-purpose tool-use traces. Tasks must require genuine multi-step investigation—retrieval, cross-referencing, analysis, synthesis—with every answer grounded in authoritative sources. The two underlying faculties, complex open-ended reasoning and precise goal-directed search, must be trained in lockstep: reasoning alone cannot conjure missing evidence, search alone cannot impose structure on an unstructured problem. Data that exercises one and neglects the other produces agents that either think in an information void or retrieve without judgement. We address this through three principles.

Evidence-grounded knowledge extraction. Every atomic claim must be traceable to a citable origin. We restrict extraction to academic papers, technical reports, and other research literature, retaining the source paragraph, bibliographic provenance, and methodological or temporal qualifiers alongside each *(entity, statement)* pair. The resulting knowledge memory is smaller than a web-scale corpus but substantially denser in citable, research-grade facts.

Random-walk synthesis with joint reasoning and retrieval. Questions are synthesized via structured

random walks over the statement space, biased toward multi-hop trajectories crossing paper boundaries. Each walk yields a question and the ordered chain of statements it visited—the ground-truth evidence path, spanning 3–10 steps. At every step, a reasoning sub-step interprets the evidence so far and identifies what is still missing; a retrieval sub-step acquires the next piece based on that determination. The two capacities thus interleave: findings reshape the plan, and the evolving plan determines the next query.

Calibrated difficulty via model-in-the-loop filtering. A question is retained only if a panel of models spanning a capability range produces a meaningful spread: at least one must fail (excluding trivia) and at least one must succeed (excluding broken or ill-posed items). The filter evaluates the full reasoning–retrieval pipeline jointly, rejecting questions solvable by pure reasoning or by a single well-formulated query. Survivors sit at the current capability frontier and force both faculties to be exercised together.

Taken together, this pipeline yields data in which every answer is grounded in citable research, every question demands joint reasoning and search, and every question is calibrated to the frontier of what current models can solve.

6.2. Capability-Preservation Data

Deep research data alone trains an agent that has learned to investigate but has unlearned everything else. We therefore mix in two complementary streams whose sole purpose is to preserve, rather than extend, the base model's general capabilities.

General knowledge and instruction following. Breadth is enforced along three axes. *Subject breadth* spans STEM, humanities, social sciences, medicine, law, and professional knowledge, with explicit quotas in the weakest disciplines identified by per-axis diagnosis of the base model. *Linguistic breadth* treats English and Chinese as first-class languages, sourced natively rather than translated. *Format breadth* mixes multiple answer-presentation templates uniformly to prevent over-fitting to any single evaluation format. Layered on top, dedicated instruction-following supervision is added at both SFT and RL stages, drawing on diverse constraint types (format, length, keyword, compositional).

Coding-agent supervision. The coding stream is gated by execution rather than collected at scale. Each candidate trajectory is admitted only when its environment can be rebuilt, its specification is consistent with the executable harness, and its terminal outcome can be validated by running the corresponding checks, so the success signal is reliable before the trajectory enters training. The mixture combines naturally occurring software changes—which expose realistic dependencies, interfaces, and multi-file interactions—with controlled task synthesis that populates underrepresented behaviors and difficulty regimes, and varies the execution harness to induce different feedback latencies and recovery patterns. The final mixture is balanced across difficulty, interaction horizon, and failure mode, concentrating supervision in the regime where tasks require nontrivial reasoning but still yield clear, executable, verifiable feedback.

Together, these two streams feed the post-training pipeline described in Section 7.

7. Training Pipeline

The training pipeline turns Qwen3.5 [37] into **Apodex-1.0**—a model that sustains long-horizon search and self-verification natively, while keeping its inherited general capabilities intact. Using the data described in Section 6, we apply a three-stage post-training recipe: *SFT* establishes a clean behavioral initialization; *agentic DPO* supplies the trajectory-level judgment that single-demonstration SFT cannot express; and an *RL* stage on long agentic rollouts closes the gap between offline demonstrations and the team-coordinated

behavior the deployed system needs. The output of this pipeline is Apodex-1.0; deploying Apodex-1.0 in heavy-duty mode (Section 3, Section 4) yields Apodex-1.0-H.

7.1. Supervised Fine-tuning

SFT alone cannot teach the long-horizon search-and-verify behavior that the later stages target, but no later stage can recover from a model that has not seen the assistant-side dialogue format in the first place. The SFT stage therefore aims at exactly one thing: a clean behavioral initialization. Its mixture combines prompt–response examples (general reasoning, knowledge-intensive QA, coding, mathematics) with structured multi-turn agentic trajectories containing interleaved reasoning steps, tool calls, and tool observations. All examples are converted into a unified dialogue format in which tool observations are part of the input context and only assistant-side tokens contribute to the loss; the model is optimized with the standard autoregressive likelihood. This stage is deliberately shallow—it aligns output format, strengthens reasoning and instruction-following, and seeds the agentic priors—and explicitly defers long-horizon behavior to the preference and RL stages that follow.

7.2. Agentic Preference Optimization

SFT optimizes against a single demonstration per prompt and therefore cannot express the trajectory-level judgment that distinguishes a careful agent from a careless one: it has no way to say that one valid-looking trace is better than another. We close this gap before RL with Direct Preference Optimization [38] on a pairwise dataset $\mathcal{D}_{\text{PO}} = \{(x_i, H_i^+, H_i^-)\}_{i=1}^{M}$, where $H_i^{\pm}$ are complete agentic trajectories (reasoning, tool actions, observations, final answer). Preferences are assigned by *final-answer correctness* rather than by handcrafted structural heuristics—planning templates, step counts, tool-use patterns—so different tasks are free to admit different valid strategies. A trace-level filter removes trajectories with severe repetition, truncation, malformed tool calls, or invalid outputs on either side. The objective is the standard DPO loss against a frozen reference.

7.3. Reinforcement Learning

Building on the DPO-aligned trajectories, the final stage fine-tunes the model with RL on long agentic rollouts. Our RL stack[1] targets two axes that dominate end-to-end efficiency and stability on these workloads. The **infrastructure axis** combines a fully-asynchronous rollout pipeline with *partial-GPU collocation*, opportunistically materializing inference engines on training GPUs during their idle window. The **algorithm axis** pairs the IcePop bidirectional token-level mask with a *bypass-mode* importance ratio that elides the conventional old-log-probability recompute. The two are co-designed: collocation makes a non-trivial fraction of every rollout off-policy, and the algorithmic correction keeps the gradient well-behaved under that drift.

7.3.1. Infrastructure

Fully-asynchronous rollout. Following ROLL Flash [29], a single persistent rollout worker dispatches prompts through an SGLang router; the trainer consumes a group as soon as $n_{\text{samples-per-prompt}}$ trajectories are ready, while laggards continue to be served. A retry buffer holds partially-completed groups across weight versions, and a configurable bypass probability lets fresh prompts preempt buffer drain to prevent starvation on tool-call mis-fires. Each token carries its engine's `weight_version` in metadata, which is propagated end-to-end and feeds the importance-ratio correction in Section 7.3.2.

Partial-GPU collocation (hybrid-collocate). A fully-disaggregated deployment leaves the training pool idle between train(N) and the completion of rollout$(N+1)$. Building on ROLL's AutoDeviceMapping [28],

[1] Built on top of `radixark/miles` (https://github.com/radixark/miles).

we partition a single Ray placement group into a **remote pool** of dedicated inference GPUs (always registered with the router; serves evaluation) and a **colocate pool** of engines instantiated on the Megatron training GPUs (unregistered, offloaded during training via SGLang's `release_memory_occupation`, and onloaded only for the rollout tail). Two weight-sync channels are selected by a `target` broadcast: intra-node Gloo+IPC ($\sim$1 s) for colocate, cross-node NCCL (tens of seconds) for remote. After train(N), colocate onloads, takes an IPC update to v_N, and serves the rollout tail; it offloads before remote pauses for its NCCL update to v_N. The brief IPC–NCCL window in which colocate serves v_N while remote still serves v_{N-1} is the only mixed-version period within an iteration, with per-token provenance recorded in the metadata stream above. The critical invariant—colocate is offloaded whenever the training ranks need its memory—removes any per-call bookkeeping. Hybrid-collocate sits one step further along the collocate–disaggregate axis than ROLL's recipe: a small remote pool sustains steady-state throughput while training GPUs are recruited opportunistically, at the cost of one onload/offload cycle per iteration amortized across the tail.

7.3.2. Algorithm

Within a single rollout group, tokens may originate from up to three policy versions (v_{N-2}, v_{N-1}, v_N; see Section 7.3.1), so the naive importance ratio $r_t = \pi_\theta(y_t \mid s_t)/\pi_{\text{infer}}(y_t \mid s_t)$ develops heavy tails—markedly heavier on MoE models, where small routing differences amplify per-token log-probability gaps and a handful of tokens with $r \gg 1$ are sufficient to destabilize an update [26, 27].

Bypass-mode. Standard PPO recomputes an "old" log-probability snapshot via a fresh forward pass over every rollout sequence, doubling per-iteration training compute. *Bypass-mode* instead reuses the inference engine's own log-probabilities (already transported per-token, Section 7.3.1) as the denominator,

$$r_t = \exp\big(\log \pi_\theta^{\text{current}}(y_t) - \log \pi^{\text{rollout}}(y_t)\big), \tag{1}$$

which shortens the training step by 5–15 % on Qwen35-35A3 agentic configurations, scaling with response length. To control the resulting heavy tails we apply the IcePop bidirectional mask [27]

$$M(r_t) = r_t \cdot \mathbf{1}[r_t \in [\alpha, \beta]], \quad \alpha = 0.5,\ \beta = 5.0, \tag{2}$$

zeroing out-of-interval tokens entirely. Masking (rather than truncated importance sampling, which caps the ratio) was the operative ingredient on long MoE rollouts. We monitor two diagnostics: the IcePop mask fraction (a persistently high value signals that asynchrony should be tightened rather than the interval widened) and the r_t distribution, which is tighter than under the snapshot formulation because the rollout–trainer bridge is now a single hop. The mask is interchangeable with DPPO-style divergence masks [30, 31] without re-introducing the recompute, as both depend only on $(\pi_\theta, \pi^{\text{rollout}})$.

8. Evaluation

We evaluate Apodex along two complementary axes: *deep-research capability*—the main objective of the training pipeline and heavy-duty mode—and *capability preservation*, the requirement that boosting deep research must not cost the base model's general, coding, reasoning, and instruction-following abilities. Section 8.1 reports Apodex-1.0-H against open- and closed-source frontier systems on public deep-research benchmarks. Section 8.2 confirms that general-knowledge, mathematics, instruction-following, and long-context capabilities inherited from Qwen3.5 are preserved rather than traded off. Section 8.3 measures coding-agent and code-generation competence at two model scales and isolates how much of the gain on agentic coding comes from the heavy-duty verifier. Section 8.4 reports IMO-ProofBench under the generate–verify–revise loop.

Table 2: Performance comparison on agentic **search** benchmarks. We report the latest publicly available results for competing models, with scores taken from the technical reports or model cards of other organizations. To minimize the impact of randomness from agent-environment interactions, we report the average performance of our Apodex models on each benchmark. For Humanity's Last Exam, Apodex-1.0 series and DeepSeek-V4-Pro-Max were tested on **text-only** subset, and other models were tested on the entire set which includes some multi-modal problems.

Benchmarks	Browse Comp	Browse Comp-ZH	DeepSearchQA	Humanity's Last Exam w/ Tools
DeepSeek-V4-Pro-Max [5]	83.4	–	–	48.2
Kimi-K2.6 [6]	86.3	–	92.5	54.0
Seed-2.0-Pro [7]	77.3	82.4	77.4	54.2
Muse Spark [8]	–	–	74.8	58.4
OpenAI-GPT-5.5 [2]	84.4	–	–	52.2
OpenAI-GPT-5.5-pro [2]	90.1	–	–	57.2
Gemini-3.1-Pro [4]	85.9	–	81.9	51.4
Claude-Opus-4.8 [3]	84.3	–	93.1	57.9
Apodex-1.0-mini	71.5	80.6	82.2	46.8
Apodex-1.0	75.5	82.6	84.6	49.0
Apodex-1.0-H	**90.3**	**84.1**	**94.4**	60.8

8.1. Deep Research

We evaluate Apodex under a Deep Research setting, where models are required to solve complex deep research tasks through multi-step reasoning, web browsing, retrieval, and evidence synthesis. These agentic deep research benchmarks include Humanity's Last Exam (HLE) [43], BrowseComp [44], BrowseComp-ZH [45], DeepSearchQA [46], FrontierScience-Olympiad [47], FrontierScience-Research [47], and SuperChem[48].

Following standard protocols, we report results on the 2,158 text-only subset of HLE. For SuperChem, we report text-only subset. For BrowseComp, we use Claude-Opus-4.6 as the judge model. For all other benchmarks, we evaluate on the full test set. To reduce potential data contamination, such as directly retrieving benchmark answers from public repositories, we block access to relevant benchmark-hosting websites in the tool environment.

Unless otherwise specified, we use fixed inference settings across benchmarks: temperature $= 1.0$, top-p $= 0.95$, context length $=$ 256K tokens, and maximum output length $=$ 32K tokens. We use LLM-as-a-Judge evaluation for all benchmarks. Under ReAct mode, all experiments are conducted with a simple ReAct-style loop, so that the results primarily reflect the model's own reasoning and tool-use capabilities rather than additional system engineering. The maximum number of interaction turns is set to $T_{\max} = 600$ for all benchmarks in ReAct mode. We set the context retention budget to $K = 5$ for BrowseComp and BrowseComp-ZH. Note that we *do not apply aggressive context management strategies, such as "discard-all"*, in our experiments. Under Agent Team mode, the main agent dynamically spawns sub-agents that can in turn autonomously spawn verifiers, exercising the model's native team behavior beyond what the single-agent loop supports. We set the maximum number of interaction turns to $T_{\max} = 100$ for both the main agent and each sub-agent, with a context length of 256K tokens at each level. Each sub-agent is equipped with the full tool set, including Python code execution, web search, and web fetch. Because each sub-agent occupies its own context window, no context management strategies (e.g., keep-last-K) are applied to either the main

Table 3: Performance comparison on agentic **science** benchmarks. We report the latest publicly available results for competing models, with scores taken from the technical reports or model cards of other organizations. To minimize the impact of randomness from agent-environment interactions, we report the average performance of our Apodex models on each benchmark.

Benchmarks	FrontierScience Research	FrontierScience Olympiad	SuperChem
Seed-2.0-Pro [7]	25.0	74.0	53.0
Muse Spark [8]	38.3	–	–
OpenAI-GPT-5.2 [39]	25.2	75.0	58.0
OpenAI-GPT-5.4 [40]	33.0	–	–
OpenAI-GPT-5.4-pro [40]	36.7	–	–
Gemini-3.0-Pro [41]	15.0	73.0	63.2
Gemini-3.1-Pro [4]	23.3	–	–
Claude-Opus-4.5 [42]	21.7	71.0	43.2
Apodex-1.0-mini	25.0	77.2	61.4
Apodex-1.0	28.3	80.3	69.0
Apodex-1.0-H	**46.7**	**87.4**	**74.2**

agent or the sub-agents. For Apodex-1.0-H, we use 16 rollouts per query for BrowseComp and HLE and 8 samples for all other benchmarks.

Deep-research benchmarks. Apodex-1.0-H sets a new state of the art across both open- and closed-source frontier systems on the public deep-research suite. On the search benchmarks of Table 2, it leads on **BrowseComp** (90.3, narrowly ahead of GPT-5.5-pro at 90.1), **BrowseComp-ZH** (84.1), **DeepSearchQA** (94.4, ahead of Claude-Opus-4.8 at 93.1 and Kimi-K2.6 at 92.5), and **text-only HLE** (60.8). On the science benchmarks of Table 3 the lead is more decisive: **FrontierScience-Research** (46.7 vs. next-best Muse Spark at 38.3), **FrontierScience-Olympiad** (87.4 vs. GPT-5.2 at 75.0), and **SuperChem** (74.2 vs. Gemini-3.0-Pro at 63.2)—margins of 8–12 absolute points over the strongest competitor in each case.

Comparing within the Apodex family quantifies what heavy-duty mode contributes: on BrowseComp, the base Apodex-1.0 reaches 75.5 and heavy-duty mode lifts it by $+14.8$ points; on FrontierScience-Research the lift is $+18.4$ ($28.3 \rightarrow 46.7$). Even without heavy-duty mode, Apodex-1.0-mini (35B-A3B) is competitive with mid-tier closed-source systems on BrowseComp-ZH (80.6), DeepSearchQA (82.2), and FrontierScience-Olympiad (77.2), confirming that a substantial amount of deep-research capability lives in the trained model itself rather than only in test-time scaling.

Apodex-1.0 Smol Series. To facilitate research in the open-source community, we also train a series of smaller models using our Deep Research SFT data. These models are designed to provide accessible starting points for studying long-horizon search, tool use, and agentic reasoning under limited model capacity. Interestingly, we find that small models can already acquire strong Deep Research capabilities from high-quality SFT data alone. For example, Apodex-1.0-4B-SFT outperforms every open-source 30B-class model on both BrowseComp and BrowseComp-ZH, demonstrating that effective data construction can substantially improve the research ability of compact models.

Table 4: Performance comparison of smaller open-source Deep Research models. We report results on four representative agentic benchmarks: BrowseComp, BrowseComp-ZH, Humanity's Last Exam (text-only), and DeepSearchQA. Competing model scores are collected from publicly available technical reports or model cards.

Model	**Params**	**BrowseComp**	**BrowseComp-ZH**	**Humanity's Last Exam**	**DeepSearchQA**
AgentCPM-Explore-4B [49]	4B	24.1	29.1	19.1	32.8
DR-Venus-4B-RL [50]	4B	29.1	37.7	–	39.6
ReTrac-4B [51]	4B	30.0	36.1	22.2	–
LiteResearcher [52]	4B	27.5	32.5	22.0	–
MarcoDR-8B [53]	8B	31.4	47.1	–	29.2
WebExplorer-8B-RL [54]	8B	15.7	32.0	17.3	17.8
Tongyi-DeepResearch [13]	30B	43.4	46.7	32.9	–
OpenSeeker-v2-30B-SFT [55]	30B	46.0	58.1	**34.6**	–
RedSearcher-30B [56]	30B	42.1	49.8	34.3	–
Apodex-1.0-0.8B-SFT	0.8B	13.9	10.7	11.2	25.8
Apodex-1.0-2B-SFT	2B	27.9	35.0	18.2	49.9
Apodex-1.0-4B-SFT	4B	**48.8**	**63.5**	32.9	**69.9**

Table 5: Benchmark results. *Apodex-1.0-mini* and *Apodex-1.0* columns are our independent evaluations; Qwen3.5 numbers are taken from the official release.

Benchmark	Apodex-1.0-mini	Qwen3.5-35B-A3B	Apodex-1.0	Qwen3.5-397B-A17B
AIME 2026	94.7	93.3	93.3	91.3
HMMT Feb 25	92.7	89.0	100.0	94.8
GPQA Diamond	84.7	84.2	87.9	88.4
IFEval	91.5	91.9	93.2	92.6
IFBench	69.9	70.2	75.7	76.5
C-Eval	90.8	90.2	93.4	93.0
MMLU-Redux	92.9	93.3	94.2	94.9
MMLU-Pro	85.9	85.3	87.9	87.8
SuperGPQA	62.2	63.4	66.7	70.4
LongBench v2	58.3	59.0	61.1	63.2
AA-LCR	55.8	58.5	67.6	68.7

8.2. General Capabilities

We adopt a suite of public benchmarks that collectively examine the general capabilities of Apodex-1.0-mini and Apodex-1.0. These benchmarks are categorized along the following dimensions:

- **General Tasks.** We report MMLU-Pro [57], MMLU-Redux [58], and C-Eval [59] to measure general knowledge recall.
- **Scientific Reasoning.** We use SuperGPQA [60], and GPQA-Diamond [61], to probe graduate-level scientific knowledge and reasoning.
- **Mathematical Reasoning.** We employ AIME 2026 [62] and HMMT February 2025 [63].
- **Instruction Following.** We report the strict-prompt accuracy of IFEval [64], and the constraint pass rate of IFBench [65].
- **Long-Context Understanding.** We evaluate AA-LCR [66] and LongBench v2 [67].

For all Apodex models we use a sampling temperature of 0.6, top-p of 0.95, and top-k of 20. The maximum

Table 6: Coding-agent and code-generation performance. The best score within each scale and benchmark is shown in **bold**.

Model	SWE-bench Verified	Terminal-Bench v2	LiveCodeBench v6
35B-A3B scale			
Qwen3.5 35B-A3B	69.2	40.5	74.6
Apodex-1.0-mini	71.5	39.3	77.8
397B-A17B scale			
Qwen3.5 397B-A17B	76.4	52.5	83.6
Apodex-1.0	76.5	51.7	82.0
Apodex-1.0-H	**79.0**	**58.4**	**85.1**

output length is set to 81,920 tokens for reasoning benchmarks, and to 32,768 tokens for knowledge, instruction following, and long-context benchmarks.

Across this suite, Apodex-1.0-mini and Apodex-1.0 track the general capabilities of their Qwen-3.5 counterparts of matching size. Apodex-1.0-mini is within roughly one point of Qwen3.5-35B-A3B on every benchmark (AIME 2026, HMMT Feb 25, GPQA Diamond, IFEval, C-Eval, MMLU-Redux/Pro, LongBench v2, AA-LCR), and Apodex-1.0 likewise tracks Qwen3.5-397B-A17B, leading on HMMT Feb 25 (100.0 vs. 94.8), C-Eval, MMLU-Pro, and IFEval, and trailing within a few points on GPQA Diamond, IFBench, SuperGPQA, and LongBench v2. The pattern indicates that the agentic-focused post-training pipeline preserves the underlying general knowledge, reasoning, and instruction-following capabilities of the base model rather than trading them off for deep research performance.

8.3. Coding

Apodex-1.0 is designed to substantially improve deep-research capability while preserving strong coding-agent competence. This section evaluates two questions: whether the deep-research gains reported in Section 8.1 come at the cost of coding performance, and whether the heavy-duty mode with verification can convert multiple candidate trajectories into stronger single-answer performance.

Benchmarks and baselines. We evaluate coding ability along two complementary axes. The first is *agentic coding*, where a model must interact with a repository, shell, or execution environment over multiple steps. We measure this setting with SWE-bench Verified [34] and Terminal-Bench v2 [35]. The second is *single-shot code generation*, measured by LiveCodeBench v6 [36], whose contamination-controlled, time-windowed problems provide a cleaner test of generation quality. We report results for two model scales, Apodex-1.0 (397B-A17B) and Apodex-1.0-mini (35B-A3B), each compared with its matched-size Qwen3.5 model.

SWE-bench Verified is evaluated under our internal SWE-agent harness [68], using temperature 1.0, top-p 0.95, top-k 20, a 256K context window, and a 32K maximum output length. Terminal-Bench v2 uses our internal mini-swe-agent harness with temperature 0.6 and otherwise the same sampling and context settings. LiveCodeBench v6 is evaluated in a single-shot setting with temperature 1.0, top-p 0.95, and a 64K maximum output length.

Single-sample results report average pass@1. We additionally evaluate a *heavy-duty mode* in which the model samples multiple candidate trajectories and selects one using an adaptive pairwise verifier. We use 4 candidates for SWE-bench Verified and Terminal-Bench v2, and 8 candidates for LiveCodeBench v6.

Table 7: IMO-ProofBench results under our generate–verify–revise recipe. We report the official IMO-ProofBench metric: mean score as a percent of the total possible points, with each problem graded 0–7 against a reference solution and a problem-specific rubric [69]. "Basic" covers the 30 pre-IMO to IMO-Medium problems, "Advanced" the 30 novel IMO-Hard problems, and "Overall" the full 60. Apodex-1.0 (397B) numbers are still in flight.

Model	Basic	Advanced	Overall
Apodex-1.0	49.05	12.38	30.71
Apodex-1.0-H	80.48	34.29	57.38

Table 6 shows that Apodex-1.0 preserves strong coding performance under the same research-oriented training recipe whose deep-research gains are reported in Section 8.1. In the single-sample regime, both Apodex models remain competitive with their matched Qwen3.5 bases across the evaluated benchmarks. At the 35B-A3B scale, Apodex-1.0-mini improves SWE-bench Verified from 69.2 to 71.5 and LiveCodeBench v6 from 74.6 to 77.8, while staying close on Terminal-Bench v2. At the 397B-A17B scale, Apodex-1.0 matches the base on SWE-bench Verified (76.5 vs. 76.4) and remains within a small margin on Terminal-Bench v2 and LiveCodeBench v6. These results indicate that the research-oriented training recipe does not erase the model's coding-agent competence.

This preservation is consistent across both agentic and single-shot settings. On benchmarks requiring repository or shell interaction, Apodex remains competitive in single-sample decoding and improves further under the heavy-duty mode with verification, reaching 58.4 at the 397B-A17B scale. On LiveCodeBench v6, which evaluates single-shot generation rather than agent-environment interaction, Apodex remains broadly competitive with the base models and also improves under the heavy-duty mode with verification at both scales. Together, these results suggest that Apodex retains general coding ability while benefiting from the stronger heavy-duty model.

8.4. Mathematical Proofs

We evaluate proof-writing capability on **IMO-ProofBench** [69], which contains 30 basic problems (pre-IMO to IMO-Medium) and 30 advanced problems (novel, IMO-Hard). Each submission is graded 0–7 against a reference solution and a problem-specific rubric, and the official metric is the mean score as a percentage of the total possible points.

Table 7 reports results for Apodex-1.0 and for the same model under the generate–verify–revise loop with 10 rollouts (yielding Apodex-1.0-H). The largest gain is on the Advanced subset, where GVR lifts the score from 12.38 to 34.29—a near-tripling, and the strongest within-system improvement we observe on any benchmark in the report. On the Basic subset, GVR raises the score from 49.05 to 80.48 (+31.4 points absolute, +64% relative). Aggregated over the full 60-problem suite, Apodex-1.0-H scores 57.38 versus the base Apodex-1.0 at 30.71, an 87% relative improvement.

The pattern is what GVR is designed to produce: the Basic subset already has a non-trivial base score that revision rounds polish into a near-ceiling number, while the Advanced subset starts from a low base where iterative critique-and-revise has the most room to operate. Because the in-loop grader never sees the reference solution or the rubric, the gain comes purely from the model's ability to score and steer its own proofs across rounds—not from oracle access.

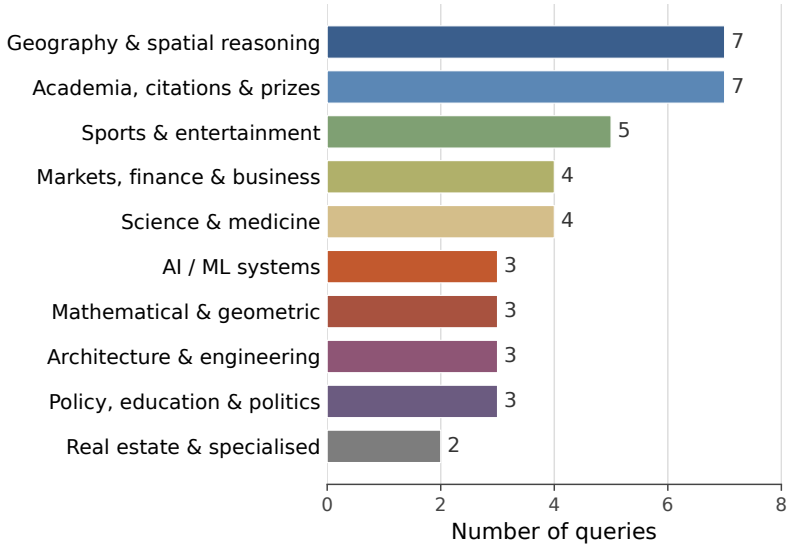

Figure 4: Coverage of the *Hard Deep Research* benchmark. The 41 queries grouped into ten content domains, with each query assigned to exactly one domain by inspecting its reference answer.

9. Internal Evaluation

To evaluate beyond public retrieval benchmarks, we construct *Hard Deep Research*, a benchmark of 41 queries requiring multi-source retrieval, multi-step reasoning, and evidence synthesis. Each query is hand-authored by domain reviewers, and its answer takes the form of a unique entity, a top-k list, an enumerable set, or a structured derivation. As the answer shape is closed, every query is scored programmatically and deterministically against locked references, yielding an exact measure of factual retrieval and reasoning correctness.

9.1. Dataset Composition

Figure 4 illustrates the thematic coverage of the benchmark, which spans ten domains. This distribution shows that the benchmark is not concentrated on a small set of topics, but instead covers a wide range of knowledge areas and task contexts.

We design every query so that its answer can be checked deterministically: a single entity or value, a top-k ranking under a stated metric, an enumerable multi-field set, or a structured derivation against a closed reference solution such as a map path. Internal domain reviewers curated 41 such queries. Each is programmatically scorable yet still demands genuine research: rather than recalling a memorized fact or relying on a single search hit, a model must locate, interpret, and synthesize evidence across multiple sources.

For each query, reviewers identify the supporting evidence, derive the ground-truth answer, and freeze it as a versioned, lock-dated reference. A task-specific scoring function then extracts the required structured claims from a model response and compares them against this reference, with the resulting query-level score normalized by the maximum attainable score. Some queries additionally include hard-negative cases, where mentioning specified incorrect items incurs deductions, discouraging exhaustive guessing. The headline metric is the arithmetic mean of these normalized scores over all 41 queries.

Model	Pos. (%) ↑	Zero (%) ↓	Neg. (%) ↓	Average ↑
Qwen3.6-Plus [72]	24.4	61.0	14.6	−1.4
MiniMax-M2.7 [71]	58.5	29.3	12.2	17.0
GLM-5.1 [73]	63.4	31.7	4.9	40.1
Gemini-3.1-Pro [4]	68.3	26.8	4.9	40.8
OpenAI-GPT-5.4 [40]	70.7	22.0	7.3	41.3
Claude-Opus-4.7 [70]	75.6	17.1	7.3	44.0
Apodex-1.0-mini	73.2	24.4	2.4	49.8
Apodex-1.0	75.6	22.0	2.4	52.9
Apodex-1.0-H	**85.4**	**14.6**	**0.0**	**63.6**

Table 8: Average scores of the nine candidate models on *Hard Deep Research*, together with the proportion of queries scored positive, zero, or negative. Apodex-family rows are highlighted; column-best values are in bold. Negative scores arise from the hard-negative penalty, which deducts points for mentioning specified incorrect items.

9.2. Evaluation of *Hard Deep Research*

Setup. We report aggregate results using the scoring protocol defined in Section 9.1. Each baseline is evaluated in its deep-research configuration: Claude-Opus-4.7 [70] in Research mode with web search and external thinking enabled; OpenAI-GPT-5.4 [40] in Deep Research mode with the Pro tier and thinking effort set to extended; Gemini-3.1-Pro [4] in Deep Research mode with web browsing enabled; MiniMax-M2.7 [71] in Research mode; Qwen3.6-Plus [72] in Deep Research mode; and GLM-5.1 [73] in ChatGLM-5.1 Agent mode. All scores are reported on a 0–100 scale, with negative averages possible when hard-negative penalties exceed positive credit.

Results. Table 8 reports the average scores of the nine candidate models on the *Hard Deep Research* benchmark. The Apodex-1.0 family claims the top three positions, led by the Apodex-1.0-H. Notably, even the smallest variant, Apodex-1.0-mini, ranks third overall, surpassing the strongest commercial baseline, Claude-Opus-4.7. These higher averages arise first from the Apodex models retrieving a larger number of correct items, and are further reinforced by their robustness to the hard-negative penalty: they incur far fewer net-negative responses than the baselines, reflecting their ability to accurately surface relevant items while avoiding spurious ones. We provide a detailed case study in Section 9.3 for analysis.

9.3. Case Studies

To complement the aggregate metrics in §9.2, this section presents two case studies.

9.3.1. Case 1: Inverse Fact Retrieval from Cryptic Clues with Hallucination Resistance

This is an inverse-retrieval problem driven by indirect clues: the letters “zy” inside the brand name, the letters “tech” inside the applicant’s name, and a “first-ever” qualifier through an unnamed expedited pathway. Solving it requires mapping these soft constraints onto a single 2013 biologic and then recovering its exact license number and approval date. The rubric scores four independent facts at $+1$ or -1 each, so a confident but wrong guess is punished to -4 while a correct answer earns $+4$. Table 9 summarizes the Apodex-1.0-H output, naming Gazyva (obinutuzumab), BLA 125486, approved 1 November 2013 by Genentech, with the clarifications that fix the program (Breakthrough Therapy Designation) and rule out the tempting “zy” distractors.

Query	Answer	Clarifications & Exclusions
A certain country's drug regulatory agency has a special "fast track" program. The first drug ever approved for market through this fast track has the letters "zy" in its brand name, and the company that submitted the application has "tech" in its name. What is the official approval number of this drug, and on exactly which date was it approved?	**Gazyva** (obinutuzumab), approval number **BLA 125486**, approved on **1 November 2013**, submitted by **Genentech, Inc.** It was the first drug ever approved under the FDA's Breakthrough Therapy Designation. The clues fit exactly: the brand "Ga**zy**va" contains "zy", and "Genen**tech**" contains "tech".	The unnamed "special fast track" is the FDA Breakthrough Therapy Designation (created by the FDA Safety and Innovation Act of 2012), not the FDA's separately named "Fast Track" program. Tempting wrong matches are ruled out: other "zy"-brand drugs such as **Zytiga** or **Zydelig** come from companies (Janssen/Ortho, Gilead) whose names lack "tech", and none was the first Breakthrough Therapy approval.

Table 9: Apodex-1.0-H response excerpt for the fast-track drug-identification query.

Model	Drug identified	Company	Appr. no.	Date	Score	Norm.
OpenAI-GPT-5.4 [40]	Zytiga (wrong)	No	No	No	-4	-1.00
GLM-5.1 [73]	none	No	No	No	-4	-1.00
MiniMax-M2.7 [71]	none	No	No	No	-4	-1.00
Qwen3.6-Plus [72]	Zurnai (invented)	No	No	No	-4	-1.00
Claude-Opus-4.7 [70]	Gazyva	Yes	Yes	Yes	4	1.00
Gemini-3.1-Pro [4]	Gazyva	Yes	Yes	Yes	4	1.00
Apodex-1.0-mini	Gazyva	Yes	Yes	Yes	4	1.00
Apodex-1.0	Gazyva	Yes	Yes	Yes	4	1.00
Apodex-1.0-H	Gazyva	Yes	Yes	Yes	**4**	**1.00**

Table 10: Per-model results on the fast-track drug-identification query. The rubric scores four independent facts at $+1$ or -1 each: *Drug* identity (the named brand/generic), *Company*, approval *number*, and approval *date*; *Score* is their sum (range $[-4, +4]$) and *Norm.* its normalization to $[-1, 1]$. All three Apodex configurations recover every field, whereas four of the six baselines instead report a wrong drug (or none) and are penalized to -4. Apodex-family rows are highlighted.

Table 10 reports the per-model breakdown. All three Apodex configurations recover every field exactly for a perfect 4/4. The discriminating behavior is hallucination resistance: four of the six baselines fail, and they fail loudly. GPT confidently reports Zytiga and Qwen invents "Zurnai", each wrong on all four fields and penalized to -4, while MiniMax and GLM-5.1 return nothing. Only the two strongest baselines, Claude-Opus-4.7 and Gemini-3.1-Pro, also reach 4/4. The case shows that when a query rewards precision and punishes guessing, the Apodex family reliably converts a set of cryptic clues into a verifiable identifier rather than a plausible-sounding fabrication.

9.3.2. Case 2: Precision-Constrained Enumeration under a Strict Definitional Boundary

The query admits a closed, objectively checkable answer set whose membership is fixed by a single criterion, namely a *formal government act* of expulsion. All three Apodex models answer correctly, returning exactly the three qualifying laureates with no false positives. As a representative example, Table 11 reports a summarized version of the Apodex-1.0 output along three axes: the query, the answer that lists the laureates satisfying the strict criterion, and its clarifications & exclusions that rule out the tempting distractors.

Query	Answer	Clarifications & Exclusions
Among writers who have won the Nobel Prize in Literature, which ones were formally expelled or exiled by the government of their own country?	Under the strict criterion of a *formal government act* (deportation, banishment, and/or stripping of citizenship), exactly **three** laureates qualify: 1) **Aleksandr Solzhenitsyn** (1970, USSR): arrested, stripped of citizenship, and deported to Frankfurt in Feb. 1974; 2) **Joseph Brodsky**(1987, USSR): forcibly put on a plane to Vienna in June 1972 and rendered stateless; 3) **Miguel Ángel Asturias** (1967, Guatemala): stripped of citizenship and expelled in 1954 by the post-coup Castillo Armas regime.	Many laureates experienced exile or censorship without ever being subject to a *formal expulsion order*, and are excluded on that basis: 1) **Pablo Neruda** fled Chile under an arrest warrant; 2) **Thomas Mann** had already emigrated before his denaturalization by the Nazi regime; 3) **Wole Soyinka** escaped across the border to evade arrest; and 4) **Gao Xingjian** entered self-imposed exile and was only later declared *persona non grata*, among others.

Table 11: Apodex-1.0 response excerpt for the Nobel-laureate expulsion query.

Model	Correct ↑	Over-claimed ↓	Score ↑	Norm. ↑
Claude-Opus-4.7 [70]	3	8	−5.0	−1.7
Gemini-3.1-Pro [4]	3	5	−2.0	−0.7
OpenAI-GPT-5.4 [40]	3	4	−1.0	−0.3
MiniMax-M2.7 [71]	0	0	0.0	0.0
Qwen3.6-Plus [72]	2	1	1.0	0.3
GLM-5.1 [73]	3	0	3.0	1.0
Apodex-1.0-mini	3	0	3.0	1.0
Apodex-1.0	3	0	3.0	1.0
Apodex-1.0-H	3	0	3.0	1.0

Table 12: Per-model results on the Nobel-laureate expulsion query, scored by counting *correct* (+1) and *over-claimed* (−1) items. **Score** is the raw sum and **Norm.** is its normalized value. Apodex-family rows are highlighted. Negative scores arise from the over-claim penalty, which deducts points for asserting specified incorrect items.

This query rewards recall and precision simultaneously: the rubric awards +1 for each laureate removed by a formal state act, and −1 for each over-claimed name: a writer who self-exiled, defected, or fled persecution but was never formally expelled. Table 12 reports the per-model breakdown under this rubric. The ground-truth answer set contains only three laureates, so the dominant failure mode is over-enumeration. The Apodex family, all three models, reports exactly the three defensible cases and quarantines the tempting distractors into an explicit exclusion list, attaining a perfect score with zero penalties. The highest-recall baselines do the opposite: aiming for exhaustive coverage, they conflate “de facto exile under pressure” with “formal expulsion” and enumerate as many as fifteen names, as shown in Table 12. For example, Claude-Opus-4.7 [70] incurs up to eight penalty points and Gemini-3.1-Pro [4] five, driving their normalized totals well below zero. The case shows that Apodex strikes the right balance: it pursues coverage as fully and accurately as the evidence allows while avoiding over-claims, which is exactly what the scoring rewards.

10. Conclusions

We introduced **Apodex-1.0**, a deep-research model, together with **Apodex-1.0-H**, the same model deployed in our verification-centric heavy-duty mode. One thesis underwrites the design: reliability on hard, open-ended problems is not a function of model scale or context length, but of a team that engages the external world and audits its conclusions before committing.

Three jointly designed pieces carry that thesis. A high-quality data pipeline and a three-stage post-training recipe (SFT, agentic DPO, RL on long agentic rollouts) substantially raise the deep-research capability of the Qwen3.5 base while preserving its general knowledge, coding, reasoning, and instruction-following capabilities. When the trained model is deployed in *heavy-duty mode*, it operates inside an asynchronous *agent team*: an orchestrator dispatches specialized sub-agents for retrieval and verification, routes their reports through a shared evidence pool, runs an in-flight verification team (conflict reviewer, fact checker, draft reviewer) over individual reports, and finally hands the assembled evidence to a domain-tailored *global verifier*—evidence reasoning for deep research, causal-evidence comparison for coding, generate–verify–revise for math. **AgentOS**, a task-agnostic runtime, hosts both deployment modes above a single narrow facade, so adding a new application is a folder of plugin code rather than a patch to the kernel.

Apodex-1.0-H sets a new state of the art across open- and closed-source models on the public deep-research suite—BrowseComp, BrowseComp-ZH, text-only HLE, DeepSearchQA, FrontierScience-Research, FrontierScience-Olympiad, and SuperChem—and on our internal Hard Deep Research benchmark of 41 reviewer-curated queries, where it leads the strongest commercial baselines by a substantial margin. The smaller Apodex-1.0-mini surpasses every external baseline on the internal benchmark at a fraction of the active parameters. We release **Apodex-1.0-mini** alongside three smaller variants (0.8B, 2B, 4B); remarkably, the 4B variant outperforms every open-source 30B-class model on deep-research tasks.

References

[1] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2022.

[2] OpenAI. Introducing gpt-5.5. https://openai.com/index/introducing-gpt-5-5/, 2026. Official announcement.

[3] Anthropic. Introducing claude opus 4.8. https://www.anthropic.com/news/claude-opus-4-8, 2026. Official announcement.

[4] Google. Gemini 3.1 pro: Announcing our latest gemini ai model. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/, 2026. Official announcement.

[5] DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

[6] Moonshot AI. Kimi-k2.5. https://www.kimi.com/blog/kimi-k2-6, 2025. Official model card.

[7] ByteDance Seed Team. Seed 2.0 official launch. https://seed.bytedance.com/en/blog/seed-2-0-official-launch, 2026. Official launch blog for the Seed 2.0 series, including Seed 2.0 Pro.

[8] Meta AI. Introducing muse spark: Scaling towards personal superintelligence. https://ai.meta.com/blog/introducing-muse-spark-msl/, 2026.

[9] OpenAI. Introducing deep research. https://openai.com/index/introducing-deep-research/, 2025. Official product announcement.

[10] Anthropic. Introducing claude research. https://www.anthropic.com/news/research, 2025. Official product announcement.

[11] Moonshot AI. Kimi-Researcher: End-to-end rl training for autonomous research agents. https://moonshotai.github.io/Kimi-Researcher/, 2025. Official release blog.

[12] xAI. Grok DeepSearch. https://x.ai/news/grok-deepsearch, 2025. Official product announcement.

[13] Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, et al. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*, 2025.

[14] Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. Webthinker: Empowering large reasoning models with deep research capability. *arXiv preprint arXiv:2504.21776*, 2025.

[15] Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*, 2025.

[16] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

[17] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

[18] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.

[19] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen-Ming Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. *arXiv preprint arXiv:2308.10848*, 2023.

[20] Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

[21] Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*, 2023.

[22] Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-verification reduces hallucination in large language models. *arXiv preprint arXiv:2309.11495*, 2023.

[23] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

[24] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

[25] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

[26] Sheng Liu et al. Stabilizing MoE reinforcement learning by aligning training and inference routers. *arXiv preprint arXiv:2510.11370*, 2025. URL https://arxiv.org/abs/2510.11370.

[27] Ant Ling. Ring-flash-2.0: Conquering the RL stability challenge in MoE with the IcePop discrepancy reduction method. https://ant-ling.medium.com/ring-flash-2-0-4bf5b62204f5, 2025. Blog post.

[28] Shuo Wang et al. ROLL: Reinforcement learning optimization for large-scale learning. *arXiv preprint arXiv:2506.06122*, 2025. URL https://arxiv.org/abs/2506.06122.

[29] Zhenyu Yao et al. ROLL Flash: Accelerating RLVR and agentic training with asynchrony. *arXiv preprint arXiv:2510.11345*, 2025. URL https://arxiv.org/abs/2510.11345.

[30] Qiying Yu et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025. URL https://arxiv.org/abs/2503.14476.

[31] Hao Qi et al. Rethinking the trust region in LLM reinforcement learning. *arXiv preprint arXiv:2602.04879*, 2026. URL https://arxiv.org/abs/2602.04879.

[32] Zhihan Liu, Miao Lu, Shenao Zhang, Boyi Liu, Hongyi Guo, Yingxiang Yang, Jose Blanchet, and Zhaoran Wang. Provably mitigating overoptimization in rlhf: Your sft loss is implicitly an adversarial regularizer. *Advances in Neural Information Processing Systems*, 37:138663–138697, 2024.

[33] Zhen Zhang, Liangcai Su, Zhuo Chen, Xiang Lin, Haotian Xu, Simon Shaolei Du, Kaiyu Yang, Bo An, Lidong Bing, and Xinyu Wang. Argus: Evidence assembly for scalable deep research agents, 2026. URL https://arxiv.org/abs/2605.16217.

[34] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024.

[35] Mike A Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu,

Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Kwesi Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, 2026.

[36] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

[37] Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, February 2026. Official release blog.

[38] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

[39] OpenAI. Introducing gpt-5.2. https://openai.com/index/introducing-gpt-5-2/, 2026. Official announcement.

[40] OpenAI. Introducing gpt-5.4. https://openai.com/index/introducing-gpt-5-4/, 2026. Official announcement.

[41] Google. Gemini 3: Introducing the latest gemini ai model from google. https://blog.google/products-and-platforms/products/gemini/gemini-3/, 2025. Official announcement for Gemini 3 Pro.

[42] Anthropic. Introducing claude opus 4.5. https://www.anthropic.com/news/claude-opus-4-5, 2025. Official announcement.

[43] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

[44] Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

[45] Peilin Zhou, Bruce Leon, Xiang Ying, Can Zhang, Yifan Shao, Qichen Ye, Dading Chong, Zhiling Jin, Chenxuan Xie, Meng Cao, et al. Browsecomp-zh: Benchmarking web browsing ability of large language models in chinese. *arXiv preprint arXiv:2504.19314*, 2025.

[46] Nikita Gupta, Rishav Chatterjee, Liane Haas, Chaofan Tao, Anqi Wang, Chun Liu, Hideto Oiwa, Elena Gribovskaya, Jan Ackermann, John Blitzer, Shachar Goldshtein, and Dipanjan Das. Deepsearchqa: Bridging the comprehensiveness gap for deep research agents. *arXiv preprint arXiv:2601.20975*, 2025.

[47] Miles Wang, Robi Lin, Kat Hu, Joy Jiao, et al. Frontierscience: Evaluating ai's ability to perform expert-level scientific tasks. *arXiv preprint arXiv:2601.21165*, 2025.

[48] Zehua Zhao, Zhixian Huang, Junren Li, Siyu Lin, Junting Zhou, Fengqi Cao, Kun Zhou, Rui Ge, Tingting Long, Yuexiang Zhu, Yan Liu, Jie Zheng, Junnian Wei, Rong Zhu, Peng Zou, Wenyu Li, Zekai Cheng, Tian Ding, Yaxuan Wang, Yizhao Yan, Tingru Wei, Haowei Ming, Weijie Mao, Chen Sun, Yiming Liu, Zichen Wang, Zuo Zhang, Tong Yang, Hao Ma, Zhen Gao, and Jian Pei. Superchem: A multimodal reasoning benchmark in chemistry. *arXiv preprint arXiv:2512.01274*, 2025.

[49] Haotian Chen, Xin Cong, Shengda Fan, Yuyang Fu, Ziqin Gong, Yaxi Lu, Yishan Li, Boye Niu, Chengjun Pan, Zijun Song, Huadong Wang, Yesai Wu, Yueying Wu, Zihao Xie, Yukun Yan, Zhong Zhang, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Agentcpm-explore: Realizing long-horizon deep exploration for edge-scale agents, 2026. URL https://arxiv.org/abs/2602.06485.

[50] Venus Team, Sunhao Dai, Yong Deng, Jinzhen Lin, Yusheng Song, Guoqing Wang, Xiaofeng Wu, Yuqi Zhou, Shuo Yang, Zhenzhe Ying, Zhanwei Zhang, Changhua Meng, and Weiqiang Wang. Dr-venus: Towards frontier edge-scale deep research agents with only 10k open data. *arXiv preprint arXiv:2604.19859*, 2026.

[51] Jialiang Zhu, Gongrui Zhang, Xiaolong Ma, Lin Xu, Miaosen Zhang, Ruiqi Yang, Song Wang, Kai Qiu, Zhirong Wu, Qi Dai, Ruichun Ma, Bei Liu, Yifan Yang, Chong Luo, Zhengyuan Yang, Linjie Li, Lijuan Wang, Weizhu Chen, Xin Geng, and Baining Guo. Re-trac: Recursive trajectory compression for deep search agents, 2026. URL https://arxiv.org/abs/2602.02486.

[52] Wanli Li, Bince Qu, Bo Pan, Jianyu Zhang, Zheng Liu, Pan Zhang, Wei Chen, and Bo Zhang. Literesearcher: A scalable agentic rl training framework for deep research agent. *arXiv preprint arXiv:2604.17931*, 2026.

[53] Bin Zhu, Qianghuai Jia, Tian Lan, Junyang Ren, Feng Gu, Feihu Jiang, Longyue Wang, Zhao Xu, and Weihua Luo. Marco deepresearch: Unlocking efficient deep research agents via verification-centric design. *arXiv preprint arXiv:2603.28376*, 2026.

[54] Junteng Liu, Yunji Li, Chi Zhang, Jingyang Li, Aili Chen, Ke Ji, Weiyu Cheng, Zijia Wu, Chengyu Du, Qidi Xu, et al. Webexplorer: Explore and evolve for training long-horizon web agents. *arXiv preprint arXiv:2509.06501*, 2025.

[55] Yuwen Du, Rui Ye, Shuo Tang, Keduan Huang, Xinyu Zhu, Yuzhu Cai, and Siheng Chen. Openseeker-v2: Pushing the limits of search agents with informative and high-difficulty trajectories. *arXiv preprint arXiv:2605.04036*, 2026.

[56] Zheng Chu, Xiao Wang, Jack Hong, Huiming Fan, Yuqi Huang, Yue Yang, Guohai Xu, Shengchao Hu, Dongdong Kuang, Chenxiao Zhao, Cheng Xiang, Ming Liu, Bing Qin, and Xing Yu. Redsearcher: A scalable and cost-efficient framework for long-horizon search agents. *arXiv preprint arXiv:2602.14234*, 2026. URL https://arxiv.org/pdf/2602.14234.

[57] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark, 2024. URL https://arxiv.org/abs/2406.01574.

[58] Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, Claire Barale, Robert McHardy, Joshua Harris, Jean Kaddour, Emile van Krieken, and Pasquale Minervini. Are we done with mmlu?, 2025. URL https://arxiv.org/abs/2406.04127.

[59] Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu, Maosong Sun, and Junxian He. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models, 2023. URL https://arxiv.org/abs/2305.08322.

[60] P Team, Xinrun Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, King Zhu, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, Chujie Zheng, Kaixin Deng, Shawn Gavin, Shian Jia, Sichao Jiang, Yiyan Liao, Rui Li, Qinrui Li, Sirun Li, Yizhi Li, Yunwen Li, David Ma, Yuansheng Ni, Haoran Que, Qiyao Wang, Zhoufutu Wen, Siwei Wu, Tyshawn Hsing, Ming Xu, Zhenzhu Yang, Zekun Moore Wang, Junting Zhou, Yuelin Bai, Xingyuan Bu, Chenglin Cai, Liang Chen, Yifan Chen, Chengtuo Cheng, Tianhao Cheng, Keyi Ding, Siming Huang, Yun Huang, Yaoru Li, Yizhe Li, Zhaoqun Li, Tianhao Liang, Chengdong Lin, Hongquan Lin, Yinghao Ma, Tianyang Pang, Zhongyuan Peng, Zifan Peng, Qige Qi, Shi Qiu, Xingwei Qu, Shanghaoran Quan, Yizhou Tan, Zili Wang, Chenqing Wang, Hao Wang, Yiya Wang, Yubo Wang, Jiajun Xu, Kexin Yang, Ruibin Yuan, Yuanhao Yue, Tianyang Zhan, Chun Zhang, Jinyang Zhang, Xiyue Zhang, Xingjian Zhang, Yue Zhang, Yongchi Zhao, Xiangyu Zheng, Chenghua Zhong, Yang Gao, Zhoujun Li, Dayiheng Liu, Qian Liu, Tianyu Liu, Shiwen Ni, Junran Peng, Yujia Qin, Wenbo Su, Guoyin Wang, Shi Wang, Jian Yang, Min Yang, Meng Cao, Xiang Yue, Zhaoxiang Zhang, Wangchunshu Zhou, Jiaheng Liu, Qunshu Lin, Wenhao Huang, and Ge Zhang. Supergpqa: Scaling llm evaluation across 285 graduate disciplines, 2025. URL https://arxiv.org/abs/2502.14739.

[61] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

[62] Mathematical Association of America. AIME 2026: American invitational mathematics examination. https://maa.org/aime, 2026. Competition problem set.

[63] Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. *arXiv preprint arXiv:2605.00674*, 2026. URL https://arxiv.org/abs/2605.00674.

[64] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

[65] Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following, 2025. URL https://arxiv.org/abs/2507.02833.

[66] Artificial Analysis Team. Artificial analysis long context reasoning benchmark(lcr), 2025.

[67] Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks, 2025. URL https://arxiv.org/abs/2412.15204.

[68] 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, pages 50528–50652, 2024.

[69] Google DeepMind. IMO-ProofBench: A benchmark of olympiad-level mathematical proofs. https://imobench.github.io/, 2025.

[70] Anthropic. Introducing claude opus 4.7. https://www.anthropic.com/news/claude-opus-4-7, 2026. Official announcement.

[71] MiniMax-AI. Minimax-m2.7. https://github.com/MiniMax-AI/MiniMax-M2.7, 2026. Official repository and model release.

[72] Qwen Team. Qwen3.6-Plus: Deep research-native agentic model. https://qwen.ai/blog?id=qwen3.6-plus, 2026. Official release blog.

[73] Aohan Zeng et al. Glm-5: From vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026. URL https://arxiv.org/abs/2602.15763.