Apodex 1.1: Scaling Agentic Intelligence for Complex Work

Apodex Team

Official Website https://www.apodex.ai
API Platform https://platform.apodex.ai
GitHub Repository https://github.com/ApodexAI/FrontierAgent
Model Weights https://huggingface.co/collections/apodex/apodex-11

Abstract

General-purpose language models can reason and synthesize knowledge, but complex work also requires sustained interaction with files, information sources, and executable code, together with state maintenance, failure recovery, and verifiable delivery. We call this *working capability*: sustained, verifiable progress toward a real-world objective. Apodex 1.1 develops this capability along two complementary dimensions. *Environment Scaling* expands the diversity and verifiability of executable file, search, and code environments, while *Agentic Coordination Scaling* trains agents to decompose long-horizon tasks, delegate parallel work, integrate asynchronous results, and replan. A shared execution harness and AgentOS maintain task state and provenance across tools and agents, and training turns environment trajectories and coordination traces into reliable behavior. Across complex professional work, finance, scientific research, mathematics, coding, and search, Apodex 1.1 reaches the leading performance band despite using a substantially smaller model than many frontier systems. The 35B-parameter Apodex 1.1 Mini further retains strong working capability in a locally deployable form. These results ground agentic intelligence in useful, verifiable work completed over time and advance our goal of building a *Heavy-Duty Solver* for ambitious, long-running tasks.

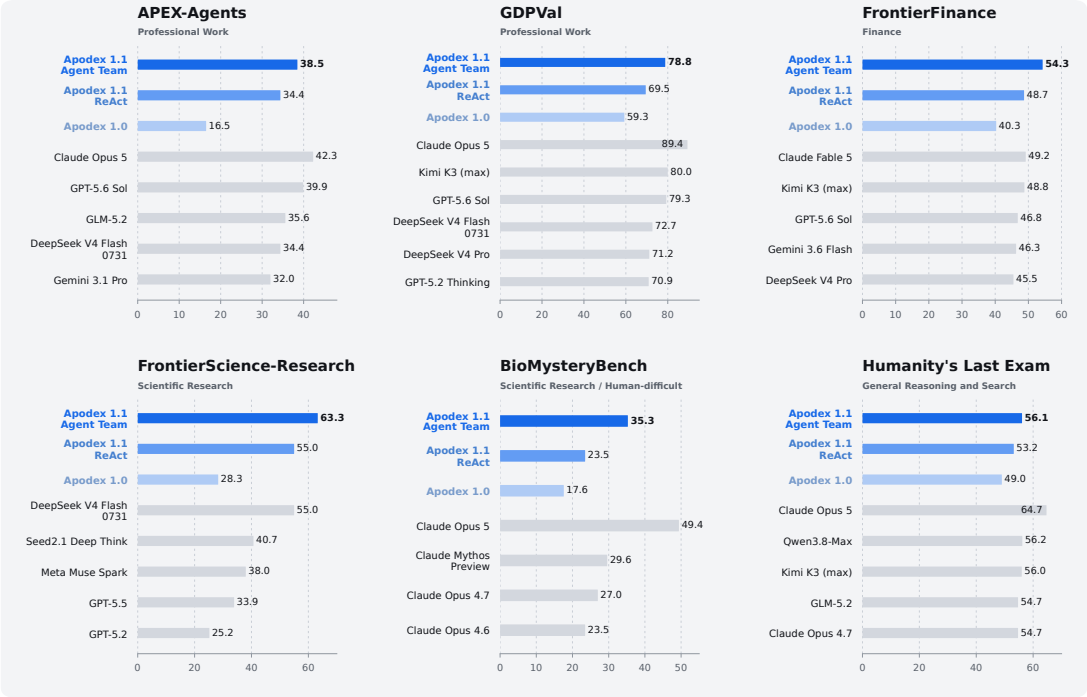

Figure 1: Apodex 1.1 reaches the leading performance band across professional work, finance, scientific research, and general reasoning.

Contents

1 Introduction

General-purpose language models have improved rapidly in knowledge, reasoning, mathematics, and coding. Yet many valuable tasks remain difficult even when the model can state the right answer, because the work unfolds over a long horizon. The model must find and interpret evidence, operate on heterogeneous files, execute and debug code, maintain and revise a coherent plan over many steps, recover from failed actions without discarding valid progress, and deliver artifacts that another person can inspect or continue using. Prior work on reasoning-and-acting, learned tool use, and interactive agent evaluation has exposed this gap between answer quality and reliable action (Yao et al., 2022; Schick et al., 2023; Liu et al., 2023; Xie et al., 2024). The limiting capability is therefore not reasoning in isolation, but the ability to turn reasoning into sustained work inside a changing environment.

Apodex 1.1 is a general-purpose model and execution system that scales agentic intelligence for complex work. We operationalize this objective as *working capability*: the ability to make useful progress over long horizons by understanding an objective, acting through tools and stateful environments, revising plans when observations change the problem, recovering from failure, and satisfying a delivery contract. The unit of this capability is completed work rather than an isolated response. It applies across long-horizon professional workflows, artifact-centric file and code tasks, open-ended scientific and financial investigations, and hard reasoning problems whose solutions must remain verifiable. Working capability is the model-level foundation of our longer-term objective: a *Heavy-Duty Solver* that can take responsibility for increasingly ambitious, long-running, and verifiable work.

We develop working capability along two complementary scaling dimensions. The first is *Environment Scaling*: expanding the diversity, fidelity, and verifiability of the file, search, and code worlds in which the model learns and acts. These environments specify state, tools, transitions, interaction budgets, failure conditions, and completion checks. They turn actions, observations, file changes, code outputs, recovery decisions, and artifacts into part of the learning distribution rather than treating tools as textual descriptions attached to otherwise static examples.

The second is *Agentic Coordination Scaling*: expanding how work is organized across agents, task branches, and time. Apodex 1.1 learns to decompose objectives, delegate work, incorporate asynchronous results, revise shared plans, and reorganize unfinished branches. Agent Team realizes these behaviors through adaptive parallel effort across evidence gathering, file analysis, implementation, verification, and counteranalysis. Existing systems demonstrate gains from conversational coordination, role specialization, and multi-agent hypothesis generation (Wu et al., 2023; Hong et al., 2023; Chen et al., 2023; Gottweis et al., 2025). The relevant scaling variable is therefore not simply the number of agents or samples, but the amount of useful work that can be coordinated as an objective evolves.

Both dimensions depend on a common execution harness. The harness binds the model to File, Search, and Code environments; maintains workspace, artifact, provenance, and branch state; defines observations and completion contracts; and supplies replay and verification hooks. AgentOS provides the persistent runtime beneath this harness, preserving authoritative state across tools, agents, context pressure, intervention, and partial failure. The same execution contract supports trajectory collection during training and sustained task execution at runtime, keeping environment interaction and agent coordination within a common model-and-system stack.

Training is organized to exploit both scaling dimensions. A unified SFT mixture establishes common behavior across reasoning, tools, recovery, delivery, and multi-agent coordination. Agentic RL then improves long-horizon decisions over executable environment trajectories and coordination traces. Real tasks, benchmark errors, and runtime failures are classified into capability gaps; Task Pipeline converts those gaps into new tasks; Environment Scaling supplies the corresponding worlds and verifiers; and evaluation, expert review, and user feedback determine the next allocation of training effort. We use *self-evolution* only for this managed engineering loop, not for unconstrained model self-modification.

Evaluation mirrors this decomposition. ReAct provides a lower-scaffold view of the model's working capability, while Agent Team measures the system-level lift obtained from trained coordination behaviors and additional organized computation. The full Apodex 1.1 Agent Team system reaches the strongest values shown in our FrontierFinance and FrontierScience-Research comparisons and remains competitive across professional work, reasoning, search, mathematics, and coding. The 35B-parameter Apodex 1.1 mini provides a complementary efficiency result, reaching the performance band of selected frontier systems and improving markedly over Apodex 1.0 mini on overlapping tasks. A shared main table is followed by capability analyses and end-to-end cases. Together, the model, scaled environments, coordination paradigm, harness, and training loop make Apodex 1.1 a concrete step toward a Heavy-Duty Solver.

Contributions. The report makes the following contributions:

- **Scaling agentic intelligence for long-horizon complex work.** Apodex 1.1 develops reasoning, tool use, file handling, code execution, state maintenance, recovery, and delivery in a common policy rather than as disconnected specialist modes.
- **Environment Scaling for working capability.** Diverse, faithful, and verifiable file, search, and code worlds expand the distribution of executable trajectories from which a unified policy can learn.
- **Agentic Coordination Scaling for organized work.** Apodex trains delegation, staged result integration, shared-state revision, and dynamic replanning as model behaviors, while Agent Team realizes them as adaptive asynchronous work at runtime.
- **A common execution harness for both scaling dimensions.** The harness connects the model to executable environments, persistent workspace and branch state, artifact provenance, replay, and verification, with AgentOS providing the underlying runtime.
- **Training across environment and coordination trajectories.** Unified SFT and agentic RL use executable task traces and Agent Team interactions to improve task execution, recovery, delivery, and coordination within one policy.
- **Evaluation from general capability to professional delivery.** A main benchmark table is followed by analyses of science, finance and professional file work, IMO gold-medal-level mathematics, coding, internal long-horizon tasks, and end-to-end cases.

2 Core Insights and Design Principles

Apodex 1.1 is built around a model-level observation: high-quality reasoning is necessary but insufficient for complex work, especially when the task unfolds over a long horizon. Scaling agentic intelligence requires a model to act in an environment, preserve authoritative state, turn intermediate results into better decisions, recover without losing valid progress, organize additional computation, and satisfy a checkable delivery objective. Our long-term objective is a *Heavy-Duty Solver* that can take responsibility for increasingly ambitious, long-running work. We organize the design around six principles in the same narrative order used throughout this report: define completed work, scale environments, scale agentic coordination, connect both through a common harness, train over the resulting trajectories, and evaluate the capabilities that emerge.

2.1 Completed Work Is the Unit of Agentic Intelligence

Conventional language-model evaluation often maps a prompt to an answer, whereas agent benchmarks increasingly place the policy in an interactive world and evaluate the resulting state (Liu et al., 2023; Yao et al., 2024; Xie et al., 2024; Vidgen et al., 2026). We use one task contract throughout this report:

$$\mathcal{E} = (\mathcal{W}, W_0, q, \mathcal{A}, \mathcal{T}, \Omega, \boldsymbol{B}, D, V_D). \tag{1}$$

Here $\mathcal{W}$ is the workspace-state space and $W_0 \in \mathcal{W}$ is the initial workspace; q is the objective; $\mathcal{A}$ is the set of actions available to the solver; $\mathcal{T}$ is the state-transition operator; Ω is the observation interface; $\boldsymbol{B}$ is the resource-budget vector; D is the delivery contract; and V_D is the task-level verifier associated with that contract. The transition operator may be stochastic; replay therefore requires the environment manifest to preserve the exogenous state, tool versions, and random seeds that the task exposes or controls. The budget vector may constrain turns, tool calls, tokens, wall-clock time, or concurrent executions. Later sections use named components such as B_{turn} or B_{tool} when a scalar budget is required.

The initial user input u_0 is the natural-language task request. The objective q is the normalized task objective induced from that request, while the delivery contract D states what must be delivered, which constraints the deliverables must satisfy, and how completion will be judged. Thus u_0 and q are related but not identical objects: the former is an input message, whereas the latter is part of the task specification. The acceptance clauses in D are task-specific and derived jointly from q, the requested deliverables, and applicable constraints. Both q and D are fixed within one task contract, although different tasks may instantiate different (q, D) pairs. A user message that clarifies evidence, priority, or method without changing the requested outcome remains part of the same task. A message that materially changes the objective or required deliverables instantiates a new task contract, even if the runtime reuses valid workspace state from the preceding task.

The workspace may contain local files, retrieved evidence, executable state, and previously generated artifacts. Let $\mathcal{U}$ be the space of user messages. During execution, $u_t \in \mathcal{U} \cup \{\varnothing\}$ for $t > 0$ denotes an asynchronous user intervention; it is empty at decision steps without an intervention. Admission of an intervention is a runtime update recorded in the trace rather than an action selected by the model. Before the next model action, the runtime admits the intervention and any accompanying artifacts into the solver-visible task state, so W_t denotes the workspace after this admission step. The model then selects $a_t \in \mathcal{A}$. An action may invoke an environment tool, update a coordination tool, dispatch or collect a subagent, or call a solver-visible verifier. The environment applies the action to the workspace and returns an observation:

$$W_{t+1} = \mathcal{T}(W_t, a_t), \tag{2}$$

$$o_{t+1} = \Omega(W_t, a_t, W_{t+1}). \tag{3}$$

Here u_t is reserved for user input, whereas o_{t+1} is the solver-visible response from a tool or the runtime. In particular, subagent reports, solver-visible verifier results, tool failures, timeouts, and execution-status updates are tool responses rather than user inputs. A no-op transition permits read-only actions. Coordination state such as a task board may live inside W_t and be read or mutated through ordinary tool calls; its schema is a harness choice described in Section 3.3, not part of the general task definition.

Let H denote the realized trajectory horizon and let

$$\tau_H = (u_0, a_0, o_1, \ldots, u_{H-1}, a_{H-1}, o_H) \tag{4}$$

denote the execution trace. Here u_0 is the initial request, and for $t > 0$, u_t records an asynchronous intervention or $\varnothing$ when none occurs. A natural-language answer or report can be an artifact in W_H, but it does not by itself define success. The task-level verifier returns a contract-specific outcome

$$S_D = V_D(W_0, W_H, \tau_H) \in \mathcal{Y}_D, \tag{5}$$

where $\mathcal{Y}_D$ is the outcome space induced by the clauses of D; S_D may be a scalar score, a vector of checks, or a structured verdict. This terminal verifier is conceptually distinct from a verifier exposed to the solver as a tool: the latter produces an ordinary observation o_{t+1} that may guide further work, whereas V_D judges the completed delivery. Individual clauses in D may use tests, exact recomputation, source alignment, structured rubrics, or human review. Success therefore requires both a useful result and a defensible path from input to delivery.

This formulation makes the title claim concrete. Scaling agentic intelligence for complex work requires an agent to sustain progress across a trajectory; preserve valid dependency and provenance relationships

among files, code, evidence, and artifacts; revise its plan when an observation changes the problem; recover from failed actions without erasing valid work; and satisfy executable and evidentiary constraints rather than optimize for plausible prose alone. The same abstraction applies to scientific analysis, repository work, professional document production, and command-line tasks.

2.2 Executable Environments Are a Scaling Surface

Conventional scaling expands model parameters, data, or inference compute. Working capability introduces another consequential surface: the environments in which the policy learns to act. An executable environment determines what states the model can observe, what actions it can take, how those actions change the world, which failures it encounters, and how completion is verified. Reproducible interactive benchmarks demonstrate that success depends on the evolving browser, desktop, repository, database, or file state rather than on tool descriptions alone (Zhou et al., 2023; Drouin et al., 2024; Xie et al., 2024; Yang et al., 2024).

Equation (1) separates the parts that environment construction can scale: initial workspaces and objectives (W_0, q), permitted actions $\mathcal{A}$, transition and observation structure $(\mathcal{T}, \Omega)$, resource contracts $\boldsymbol{B}$, and delivery-verification pairs (D, V_D). Scaling environments is therefore not equivalent to generating more prompts. It expands the distribution of executable task contracts across which the policy must generalize while preserving valid transitions and evaluable outcomes. Diversity without fidelity teaches behavior that fails in real tools; fidelity without sufficient coverage overfits a small number of workflows; and interaction without verification rewards plausible activity rather than completed work.

Apodex 1.1 therefore scales complementary file, search, and code worlds. File environments teach the model to inspect, transform, and preserve heterogeneous artifacts; search environments teach evidence acquisition and reconciliation under incomplete information; and code environments teach executable transformation, testing, and recovery. Their composition matters: evidence discovered by search must become input to code, code must operate on the authoritative file version, and the resulting artifact must retain a defensible relationship to both. Environment Scaling turns this coverage into training trajectories for a common policy rather than isolated tool specialists.

2.3 Agentic Coordination Is a Scaling Surface

Long-horizon work creates a coordination problem as well as a computation problem. A capable model must decide how to decompose an objective, which branches can proceed independently, when partial results should change the shared plan, and when obsolete work should stop. These are policy behaviors that can be represented in training trajectories, not merely properties of an inference-time wrapper. Multi-agent frameworks have explored role-based conversation, software-development organizations, debate, and dynamic scientific hypothesis generation (Wu et al., 2023; Hong et al., 2023; Qian et al., 2023; Du et al., 2023; Gottweis et al., 2025).

The important property of Agent Team is therefore not the number of agents or parallel samples. It is the continuous movement of results, decisions, and user feedback through an active task. A subagent returns a useful intermediate result as soon as it becomes available. The lead agent integrates that result into shared state, revises priorities, informs or terminates other branches, and creates new workstreams when evidence changes the problem. A slow or failed branch does not erase completed work elsewhere, and the user can change priorities or stop a path while other branches continue.

Apodex trains decomposition, delegation, result integration, and replanning as part of the model's working policy, then realizes those behaviors through Agent Team at runtime. Parallel execution supplies breadth; staged return and replanning supply feedback; shared state allows one branch to improve the value of another; and explicit termination prevents obsolete work from consuming the remaining budget. We call this joint training-and-runtime paradigm *Agentic Coordination Scaling*. It scales the organization of

work across agents, task branches, and time rather than merely increasing the number of samples.

2.4 A Common Harness Connects Both Scaling Dimensions

Environment Scaling and Agentic Coordination Scaling require a shared execution contract. On the environment side, the harness defines tools, observations, workspace state, artifacts, interaction budgets, and completion checks. On the coordination side, it defines subagent execution state, staged returns, shared artifacts, lifecycle control, and verification hooks. Using one contract keeps trajectory collection, replay, training, and runtime execution aligned rather than allowing each capability to develop behind an incompatible interface.

Long-horizon work cannot be reduced to a larger context window. A task accumulates external state: files are created and revised, evidence is accepted or rejected, programs produce outputs, and intermediate artifacts become dependencies of later decisions. Extending the transcript does not by itself guarantee that these objects remain authoritative or causally consistent. AgentOS therefore provides the persistent runtime beneath the harness, maintaining search, file handling, code execution, artifacts, and Agent Team results in one task state.

Persistent state also makes intervention and recovery composable. New data may change a hypothesis, an intermediate result may expose a bad input, and user input may redirect the unfinished work without changing the task-level objective or delivery contract. The harness must retain unaffected artifacts, invalidate descendants whose dependencies changed, and revise pending work without discarding causally independent progress. If the user materially changes q or D, the runtime may carry valid state forward, but the formalism treats the continuation as a new task contract.

The same contract supports inspection and verification without exposing private chain-of-thought. Operational records expose plans, active and completed steps, artifacts, dependencies, failures, required decisions, and next actions. Statement Review then checks consequential claims against sources, computations, and artifact lineage. Related approaches use iterative self-feedback, verbal reinforcement, independent verification questions, or learned process rewards (Madaan et al., 2023; Shinn et al., 2023; Dhuliawala et al., 2023; Lightman et al., 2023); here these checks are attached to the persistent execution state shared by both scaling dimensions.

2.5 Training Exploits Both Scaling Dimensions

Runtime coordination alone cannot create reliable file handling, code execution, reasoning, or long-horizon control. Conversely, training on detached answers cannot build a model that operates reliably in executable environments. Figure 2 shows how the two scaling dimensions are converted into model capability and how observed failures determine the next development cycle.

Training is the mechanism that converts both scaling dimensions into model capability. Environment Scaling produces executable file, search, and code trajectories with explicit state transitions and verifiers. Agentic Coordination Scaling produces traces of decomposition, delegation, staged return, integration, replanning, and recovery. A unified SFT mixture establishes common task-execution and coordination behavior, while agentic RL improves long-horizon decisions over both trajectory families. Toolformer established self-supervised tool-call learning, while Search-R1, RAGEN, and recent agentic-RL systems optimize policies over search or multi-turn environmental interaction (Schick et al., 2023; Jin et al., 2025; Wang et al., 2025c;b; Yao et al., 2025).

The loop is deliberately controlled. Runtime failures and benchmark errors determine where the next environments, coordination examples, and tasks should be built; expert and user feedback determine which gaps matter; and subsequent training reallocates effort accordingly. Training is therefore not presented as a third independent scaling axis. It is how Apodex learns from the worlds and coordination structures expanded by the first two.

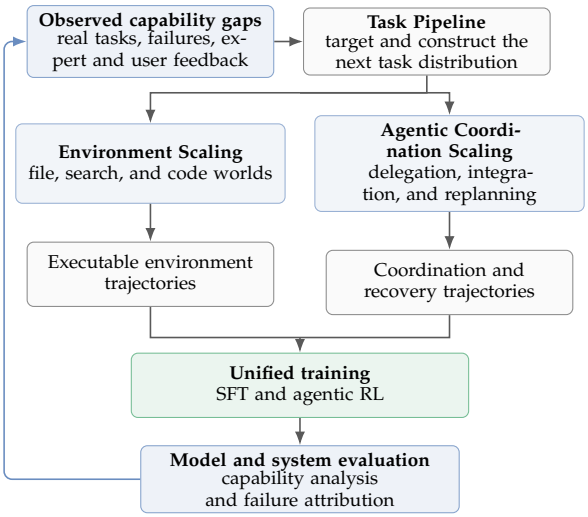

Figure 2: The controlled capability-development loop. Environment Scaling and Agentic Coordination Scaling create the executable trajectories used by training; evaluation, real failures, and human feedback determine which capability gaps the next Task Pipeline should target.

2.6 Evaluation Mirrors the Scaling Design

We measure agentic intelligence through working capability, which is inherently multi-objective. We distinguish task correctness, artifact completeness, provenance fidelity, recovery from tool or assumption failures, response to intervention, wall-clock time, and total compute. An answer-only score observes only a projection of this vector.

The evaluation design separates the layers introduced above. ReAct runs use a minimal scaffold to expose the underlying model policy, while Agent Team runs measure what trained coordination behaviors can achieve with additional organized computation. The main table provides a shared system snapshot; capability analyses then examine scientific research, professional work, mathematics, search, coding, and internal evaluations; end-to-end cases inspect workspace transitions, artifacts, and delivery. Section 4 reports these results without reducing the system to a single aggregate score.

Together, these principles imply the technical decomposition described next. Apodex 1.1 is the general-purpose model at the center; Environment Scaling expands the worlds in which it learns and acts; Agentic Coordination Scaling expands how it organizes work; the harness and AgentOS connect both dimensions through persistent execution state; SFT and RL learn from their trajectories; and evaluation measures the underlying policy, coordinated system, and completed work. This is the path from a capable language model toward a Heavy-Duty Solver developed in the remainder of the report.

3 Apodex 1.1

Apodex 1.1 is organized as a common policy and execution stack for long-horizon work. The model learns to reason, search, manipulate files, execute code, recover from failed actions, coordinate parallel work, and deliver verifiable artifacts without treating these behaviors as disconnected specialist modes. Its workspace may begin with papers, measurements, spreadsheets, images, or a repository and accumulate evidence, executable analyses, intermediate artifacts, and final deliverables under a fixed task objective q and delivery contract D.

The architecture follows the two scaling dimensions introduced in Section 2. Environment Scaling expands the executable file, search, and code worlds in which the policy learns and acts. Agentic Coordination Scaling expands how work is decomposed, delegated, integrated, verified, and revised

across agents and time. A common harness binds both dimensions to persistent state, typed observations, replay, and completion checks; AgentOS supplies the underlying execution substrate. The following sections describe the two scaling dimensions, their runtime support, and the training process that converts their trajectories into model capability.

3.1 Environment Scaling

Complex work is learned through interaction with worlds, not from answers alone. A model may possess the knowledge required by a task yet still fail because it cannot locate the authoritative file, preserve state across tool calls, recover from an execution error, reconcile conflicting evidence, or determine whether the delivered artifact is actually complete. These failures are not peripheral tool-use errors. They determine whether reasoning can be converted into reliable work over a long trajectory.

Environment Scaling treats the construction of such worlds as a primary scaling surface. Model scaling increases policy capacity, and inference scaling increases the computation spent on one task; Environment Scaling increases the coverage, fidelity, and structural depth of the executable situations from which the policy learns. The objective is not to attach more tools to a model or generate more prompts. It is to expose a common policy to increasingly diverse state transitions, information-acquisition problems, failure modes, and verifiable delivery requirements while preserving the causal structure of real work.

We use the task contract $\mathcal{E}$ defined in Eq. (1). Environment Scaling expands a distribution over its initial workspaces and objectives (W_0, q), action spaces $\mathcal{A}$, transition and observation structure $(\mathcal{T}, \Omega)$, resource budgets $\boldsymbol{B}$, and delivery-verification pairs (D, V_D). Apodex 1.1 develops this distribution across file, search, and code worlds, then applies common construction, verification, and replay requirements across all three families.

3.1.1 Environment Families

The three families emphasize different bottlenecks but share one task contract and can be composed within the same trajectory. File worlds center authority and transformation, search worlds center discovery and evidence alignment, and code worlds center executable transformation and verification.

File worlds: authority and transformation. Unlike tasks that concentrate information within the prompt, file-oriented work distributes the necessary information across a workspace containing nested directories, historical versions, heterogeneous formats, and cross-file references. The prompt specifies the objective, while the workspace carries the facts, rules, context, and evolution history. The agent must locate authoritative material, reconstruct relationships among files, filter duplicated or obsolete records, and materialize that understanding as a professional artifact.

Constructing a file world is the inverse of solving one. It begins by defining the underlying business state, authority relationships, derivation logic, and delivery requirements, then projects them into a workspace. For a file world, W_0 contains the supplied files and initialized execution runtime; $\mathcal{A}$ contains inspection, parsing, computation, and writing actions; $\mathcal{T}$ persists edits and executable state; and Ω exposes only the content requested by an action while consuming the relevant components of $\boldsymbol{B}$. The resulting artifacts must satisfy D, whose clauses are adjudicated by V_D.

File-world scaling widens both coverage and structural depth. Coverage is profession-conditioned: a maintained scenario registry expands domains into occupations, occupations into deliverable clusters, and clusters into per-task angles. The current registry spans 33 domains, 318 occupations, and 1,208 deliverable clusters, with the angle layer producing distinct tasks within an occupation. Structural depth varies the number and history of contributing systems, the length of the business-logic chain that must be reconstructed, and the portion of the delivery contract that must be inferred from context rather than copied from the prompt. These dimensions are independent of file count; enlarging a directory does not

by itself create a harder world.

Verification is anchored to independently recoverable evidence. A graded quantity must be re-derived by code from an authoritative source or connected through recorded provenance to an actual document. A task is admitted because independent derivations agree, not because a generated answer looks plausible.

Search worlds: discovery and evidence alignment. Search environments model open-web research as discovery, acquisition, and evidence synthesis. The agent uses structured search to discover candidate sources and fetch to inspect selected pages, with a constrained shell path for information that ordinary search and fetch cannot recover.

The agent must formulate and refine queries, triage candidate sources, follow references, reconcile evidence, and decide when support is sufficient. The gold object is therefore richer than a final answer: it includes the relevant source set, claim-to-evidence alignment, and explicit uncertainty where sources conflict. Retrieval provenance remains exact even when semantic support requires bounded model-based review.

Search-world scaling changes the structure of the acquisition problem rather than merely increasing corpus size. Evidence may be distributed across sources, separated from the initial query by intermediate entities, mixed with plausible but non-authoritative candidates, or exposed through heterogeneous access paths. Deeper structures require query reformulation, source triage, navigation, cross-source reconciliation, and disciplined stopping.

Code worlds: executable transformation and verification. A code world is a stateful environment in which an agent changes repositories, dependencies, and processes, then receives feedback from the resulting executable state. Construction separates reusable infrastructure from task-specific state: shared base images provide interpreters, toolchains, tests, and common dependencies, while each task contributes its repository state and requirements. Long-tail environments are built once and cached, making most new worlds an exercise in state assembly rather than repeated infrastructure construction.

Coverage combines harvested worlds grounded in real pull requests with synthesized worlds that extend beyond repository distributions. Synthesized tasks expand from verified seeds through composition, abstraction shifts, request variation, and adversarial input changes. As these transformations can invalidate an existing grader, verifier hardening precedes task perturbation; otherwise the result is a corrupted task rather than a new executable world.

Code verification is grounded in sandboxed execution. For harvested worlds, fail-to-pass tests must fail on the base state and pass after the reference change, while pass-to-pass tests must succeed in both states. Synthesized worlds have no external oracle, so passing the reference solution alone is not enough. We additionally test whether a solver can obtain reward without actually completing the task, and only attacks that succeed in the sandbox are treated as verifier failures. Scoring is isolated from the solver so that the solver cannot modify the verifier itself. Failed executions are also used to diagnose construction errors: tests passing before the fix point to incorrect test selection, while tests failing after the fix point to problems in the container, dependencies, or run command. This makes verification both a defense against reward hacking and a source of executable feedback for repairing the environment.

Table 1 summarizes the distinct construction and assurance boundaries of the three environment families.

3.1.2 Scaling Coverage and Difficulty

Scaling the environment distribution means allocating new worlds to capability gaps, not increasing volume uniformly. After each training round, unsuccessful trajectories are analyzed for recurring failure modes and abstracted into capability-level deficiencies. Those deficiencies become specifications for the next Task Pipeline, shifting construction toward weaknesses exposed by the current policy. Newly

Table 1: Environment families and their assurance boundaries. "Exact" means code-derived or independently reconciled, not accepted solely from a language-model judge.

Family	Construction	Verification boundary	Role
File worlds	Profession-conditioned multi-format workspaces	Code-derived values or recorded provenance	Authority and transformation
Search worlds	Indexed or open evidence sources	Provenance + claim review	Discovery and evidence alignment
Code worlds	Repositories and stateful sandboxes	Tests + artifact checks	Executable transformation

generated tasks re-enter the same family-specific verification pipeline before contributing training signal. Model-based diagnosis decides where to explore; environment verification decides which resulting worlds are trustworthy enough to train on.

Difficulty must also be calibrated against the behavior a world is intended to teach. File and search tasks are often dominated by constrained information acquisition: more files, pages, or tokens do not make a task harder when the authoritative path remains obvious. For these acquisition-heavy worlds, let $N_{\text{cand}}(\mathcal{E})$ be the number of plausible candidates requiring expensive inspection, $N_{\text{hop}}(\mathcal{E})$ the number of load-bearing evidence transitions, and $B_{\text{tool}}(\mathcal{E})$ the tool-call budget. We use

$$\rho_{\text{acq}}(\mathcal{E}) = \frac{N_{\text{cand}}(\mathcal{E}) + N_{\text{hop}}(\mathcal{E})}{B_{\text{tool}}(\mathcal{E})} \tag{6}$$

as a first-order acquisition-pressure coordinate, not as a universal measure of environment difficulty. Raising ρ_{acq} pressures triage, navigation, state tracking, and stopping discipline. Code worlds require different calibration coordinates, including dependency depth, state-transition depth, test observability, and the distance between a failure and its executable verification signal.

3.1.3 Verification, Isolation, and Replay

A scalable task distribution is useful only when its feedback remains trustworthy under interaction. Agreement among a generated task, its tests, and a reference solution is insufficient when all three may share the same error. Executable checks, independent derivations, provenance constraints, or bounded semantic review must establish the verifier outside the path used to produce the candidate result. Blind solver probes expose false negatives, and disagreements between a faithful solution and the grader are routed back to task construction rather than silently relabeled as model failures.

Each rollout receives an immutable world manifest and a fresh mutable sandbox. The harness materializes the initial state, preserves it throughout the session, and reclaims the sandbox on close or idle timeout. Sticky routing keeps a session on one worker. The learning service constructs dialogue and reward, while the harness owns physical execution; solver observations, hidden verifier state, and post-hoc labels remain separated.

A trajectory is retained only when its initial state can be reconstructed, tool execution is isolated, and the verifier can be replayed. The replay record includes the world seed and generator version, tool versions, action/observation sequence, file deltas, verifier version, and termination reason. Replay checks separate infrastructure failures from policy failures before a trajectory contributes training signal.

The resulting construction principle is "forward cheap, inverse expensive." A generator uses latent state or a reference program to construct and solve a world cheaply; the agent sees only the rendered workspace and must recover the relevant path under constraints. This asymmetry lets scale and verification coexist without pretending that every organic task has machine-proved semantics.

3.2 Apodex Agent Team 1.1: Interactive Self-Organizing Teams

Apodex 1.1 retains the central architecture of the Apodex 1.0 Agent Team (Apodex Team, 2026): a main (lead) agent first reasons over the problem globally, decomposes it into researchable subproblems, and then constructs specialized subagents on demand. The team is therefore induced by the problem rather than selected from a fixed catalogue of roles. Apodex 1.1 externalizes this architecture onto a persistent task board and adds four capabilities: asynchronous human intervention, asymmetric verification, adaptive Max Team Effort, and evidence-grounded synthesis.

3.2.1 From Latent Decomposition to an Explicit Task Board

In Apodex 1.0, decomposition was primarily an internal coordination decision: the main agent reasoned about the question and dispatched expert subagents. Apodex 1.1 makes that decision an explicit part of the task state. Before delegation, the main agent writes its decomposition to a task board whose entries record a bounded objective, dependencies, resolution state, assigned agents, and returned evidence or artifacts. The board is not merely a visualization of private reasoning. It is the shared coordination record between the model, runtime, and user: subagents receive their scope from it, completed work is attached back to it, and plan revisions are expressed as tool-mediated edits to it.

Externalizing the plan changes the semantics of coordination. A result can unlock a dependent task immediately; a failed or superseded premise can invalidate only its descendants; and a slow branch no longer prevents independent work from progressing. More importantly, the main agent must continuously reconcile its internal strategy with an inspectable task state. This turns decomposition from a one-shot preamble into a persistent control surface for long-horizon work.

3.2.2 Asynchronous Human Intervention

Real research tasks are underspecified at the start and better specified by execution. A source may reveal that the original premise is wrong, a preliminary analysis may trigger a new hypothesis, or the user may recognize that the emerging report optimizes the wrong objective. A plan that can be clarified only before execution forces the agent to remain faithful to an obsolete interpretation. Apodex 1.1 instead accepts a user message u_t during execution and lets the policy update the live task board through its coordination tools. The contribution is not an interrupt button; it is a policy trained to determine what the intervention changes, what remains valid, and how ongoing work should continue.

We construct intervention trajectories spanning requirement clarification, correction of task facts, priority changes, new files or evidence, revised hypotheses or methods, source and tool constraints, budget or deadline changes, pause–resume–cancel commands, progress queries, output-format and language preferences, and unrelated side questions. These messages require different behavior. Clarifications, priority changes, and method revisions that leave the acceptance clauses unchanged update the relevant board entries, invalidate affected descendants, and notify active subagents while preserving (q, D). If a message materially changes the objective, required deliverables, or acceptance constraints, the runtime begins a new task contract while carrying forward workspace state that remains valid. A task-preserving query receives a timely short response while independent executions continue. Training across these cases teaches the main agent to preserve causal continuity rather than restart all work or append every user message indiscriminately to the final report.

This formulation addresses cases that front-loaded clarification alone cannot. The user can supply information when it becomes relevant, and the model can answer progress questions without surrendering the compute already invested in the task. Intervention becomes part of the research process itself: it changes future actions while preserving completed work that remains causally valid.

3.2.3 Asymmetric Verification

Apodex 1.0 introduced conflict reviewers, fact checkers, and draft reviewers in contexts separated from the main agent (Apodex Team, 2026). Context separation is essential because a verifier that inherits the generator's full trajectory is easily anchored by the same assumptions. Independence alone, however, is insufficient. If a verification agent of comparable capability is asked to solve the entire problem again, its response can introduce a second, equally unconstrained chain of errors. The verifier then becomes another source of context pollution rather than a source of corrective signal.

Apodex 1.1 constructs *verification asymmetry*: verification is deliberately narrower than generation. Instead of routinely reproducing the whole solution, a verifier receives a specific claim, its supporting evidence, and the applicable delivery constraint. Its task is to attack that claim by searching for counterexamples, triangulating with a genuinely independent source class, checking atomic details such as names, dates, numbers, and formulas, or testing compliance with format and specification requirements. Targeted verification questions are already known to reduce coupling between generation and checking (Dhuliawala et al., 2023); here the same principle is applied inside a live agent team.

The asymmetry is in the task, not necessarily in model size. Falsifying a stated claim, locating a missing citation, or detecting a contract violation demands less open-ended synthesis than generating the report from scratch. It also produces a more actionable signal: the feedback names the contested claim, the disconfirming evidence or failed check, and the required repair. The main agent can therefore reopen one board item, dispatch a focused follow-up, and integrate the correction without allowing a free-form verifier narrative to overwrite stronger evidence elsewhere.

3.2.4 Adaptive Max Team Effort

Test-time scaling can improve difficult or ambiguous conclusions by investing more inference compute, but uniform scaling wastes budget on subproblems that are already settled. Adaptive Max Team Effort applies this principle at the team level: the main agent assigns additional, independently scoped investigations only to weak, contested, or load-bearing claims. The policy fans out along genuinely different hypotheses, methods, source classes, or query framings; allocates later waves to unresolved or disconfirmed branches; and requires focused verification before a load-bearing conclusion is finalized. Because allocation is revised after each return, the scaling variable is useful coordinated work rather than a fixed number of agents or samples. These behaviors are represented in training trajectories and activated at inference time through the Agent Team execution contract.

3.2.5 Evidence-Grounded Synthesis

Long-horizon team execution and final synthesis impose different demands. The lead agent must plan, dispatch, integrate partial findings, and react to new evidence, while the final deliverable must preserve decisive details, provenance, disagreement, and unresolved uncertainty accumulated across many trajectories. Compressing all of this directly from the lead agent's final context risks losing early evidence or turning a tentative branch result into an unsupported conclusion. Apodex 1.1 therefore adds a dedicated synthesis stage after team execution and verification. The stage consumes the terminal task board, subagent reports, retrieved evidence, produced artifacts, and verifier findings; it does not treat the last message of the lead agent as the report state.

The synthesis stage has two passes. **Evidence-graph construction** reconciles overlapping branch results into a claim–evidence graph and derives a writing outline, marking which claims are load-bearing, corroborated, disputed, or unresolved (Zhang et al., 2026). **Agentic synthesis** gives this graph to a writer agent, which produces the requested deliverable under the same tool, citation, and delivery constraints as the rest of the system. A claim that cannot be traced to evidence or computation is qualified or omitted; a missing load-bearing dependency is returned to the lead agent for further work rather than filled with

plausible prose.

Asymmetric verification makes this representation directly actionable. Because each verifier returns a targeted judgment about a claim and its support, graph construction can retain corroborated claims, isolate rejected ones, and expose disagreement before drafting begins. Verification therefore determines what may survive into the final evidence state, while synthesis determines how that state is communicated. This closes the Agent Team loop without making synthesis a separate product workflow or allowing the writer agent to overwrite the team's evidential decisions.

3.3 AgentOS: An Execution Substrate for Long-Horizon Work

AgentOS is the shared execution substrate for both single-agent and multi-agent work in Apodex 1.1. It maintains persistent workspace and tool state, converts runtime events into model-visible observations, preserves useful progress across long trajectories, and controls how intermediate artifacts become final deliverables. Agent Team builds an explicit coordination layer on this common substrate; it does not define the substrate itself. Apodex 1.1 retains the task-agnostic kernel–plugin architecture and generic ReAct loop introduced with AgentOS 1.0 (Apodex Team, 2026), while extending the runtime for persistent execution, asynchronous control, multi-agent coordination, and controlled artifact delivery.

3.3.1 Persistent Workspace and Tool Execution

AgentOS instantiates the general workspace $W_t \in \mathcal{W}$ of Eq. (1) as

$$W_t = (F_t, Q_t, C_t, I_t, G_t, K_t), \tag{7}$$

where F_t is file state, Q_t is retrieved evidence, C_t is executable state and logs, I_t is the artifact index, G_t is the dependency graph connecting sources, actions, and artifacts, and K_t is optional runtime control state. In a single-agent run, K_t may contain only lightweight plan and execution metadata; Agent Team instantiates it as explicit coordination state through the Task Board and Agent Bus. This is a harness-level instantiation of the workspace, not an extension of the task contract. Each application of $\mathcal{T}$ may update one or more components and records the corresponding artifact and provenance relationships in I_t and G_t.

Long-horizon execution requires stable names and explicit visibility rules for external state. Each run therefore receives three filesystem namespaces: `/inputs`, a read-only view of task-supplied files; `/workspace`, where agents keep calculations, notes, and candidate artifacts; and `/outputs`, the collection root for final deliverables. The distinction makes the delivery contract D of Section 2 checkable: a verifier can distinguish supplied material from intermediate work and from files that the run claims to deliver. Only non-scratch content admitted to `/outputs` is eligible to become a user-visible deliverable.

Not every namespace visible to a run is run-scoped. Deployments may additionally mount `/shares`, a read-only view of a durable personal and organizational document library that outlives any single run. Because the library is user-owned rather than run-produced, its contract is deliberately narrow: access is read-only, the mount is configured by the deployment rather than by the task request, and files the team consults remain citable in the final report alongside run-scoped evidence.

The visibility of `/workspace` depends on the isolation backend: under per-agent isolation each subagent works in a private worktree that the coordinator can inspect but siblings cannot, whereas under container isolation the workspace is shared and branches keep to disjoint paths by convention. In both modes, `/inputs` remains read-only and `/outputs` is shared. Figure 3 summarizes these state and visibility boundaries.

File access is capability-scoped rather than ambient. Execution profiles choose whether a model receives file readers, structured file constructors, editors, or a shell; in Agent Team, the coordinator and production subagents may receive different profiles. Candidate production and final publication are also separated:

ordinary execution writes under `/workspace`, whereas publication to `/outputs` follows the controlled delivery contract described below.

3.3.2 Coordination State for Agent Teams

Message history is a poor sole representation of a long-running plan: it is repeatedly reformatted, compacted, and interleaved with large tool results. AgentOS 1.1 therefore gives the coordinator a run-scoped task board outside the LLM context. In the notation of Eq. (7), the board is a tool-managed component of K_t: reading or mutating it is an ordinary action a_t, and the tool result is returned as o_{t+1}. Each item has a stable identifier, a concrete description, an owner set, optional dependency references, an optional group, optional references to returned evidence or artifacts, and a resolution state in {`open`, `in_progress`, `resolved`, `cancelled`}. Dependency references point to other board items and, through the workspace provenance graph G_t, to the evidence or artifacts on which the item depends; returned-result references address entries in the artifact index I_t. The owner set records which agents are responsible for the item; it does not grant runtime authority. Resolution state is the coordinator's lightweight semantic judgment about the work item. In particular, `resolved` means that the requested result has been returned and sufficiently checked, not merely that an associated process terminated.

The coordinator owns these semantic fields, while the runtime owns the status of each dispatched execution. This separation prevents a Task Board update from overwriting the state of a live asynchronous job. The runtime periodically re-injects a rendering of the board into the coordinator's history, so the plan remains visible after context compaction. Board mutations are also streamed as incremental observations for an interactive client.

When planning mode is enabled, the board forms an explicit phase boundary. Before `finish_planning`, the coordinator is restricted to read-only inspection and board operations; team creation, dispatch, file mutation, and finalization are denied by default. During execution the board remains mutable, allowing new subquestions to be registered as evidence changes the plan. A normal final answer is accepted only after every active item is resolved or cancelled. When planning mode is enabled, the finalization gate additionally requires at least one delegated branch to have run and an independent verifier to have been used. Thus the board is both external memory and a runtime finalization gate, rather than a presentation-only checklist.

Subagent execution lifecycle. The execution lifecycle belongs to a dispatched subagent assignment, not to a persistent subagent identity and not to the task item itself. Its normal runtime path is `created`, `queued`, `running`, and `reported`. Failure, cancellation, and timeout are recorded separately as termination reasons and may end an execution from the applicable non-terminal state. These values are runtime facts exposed through the subagent-management tools. They are intentionally distinct from Task Board resolution. A subagent execution may be `reported` while its item remains `open` because the report is incomplete, contradicted, or awaiting verification; conversely, the coordinator may resolve an item using evidence from several executions. This is a lightweight AgentOS harness definition rather than a new principle or a model of agentic intelligence.

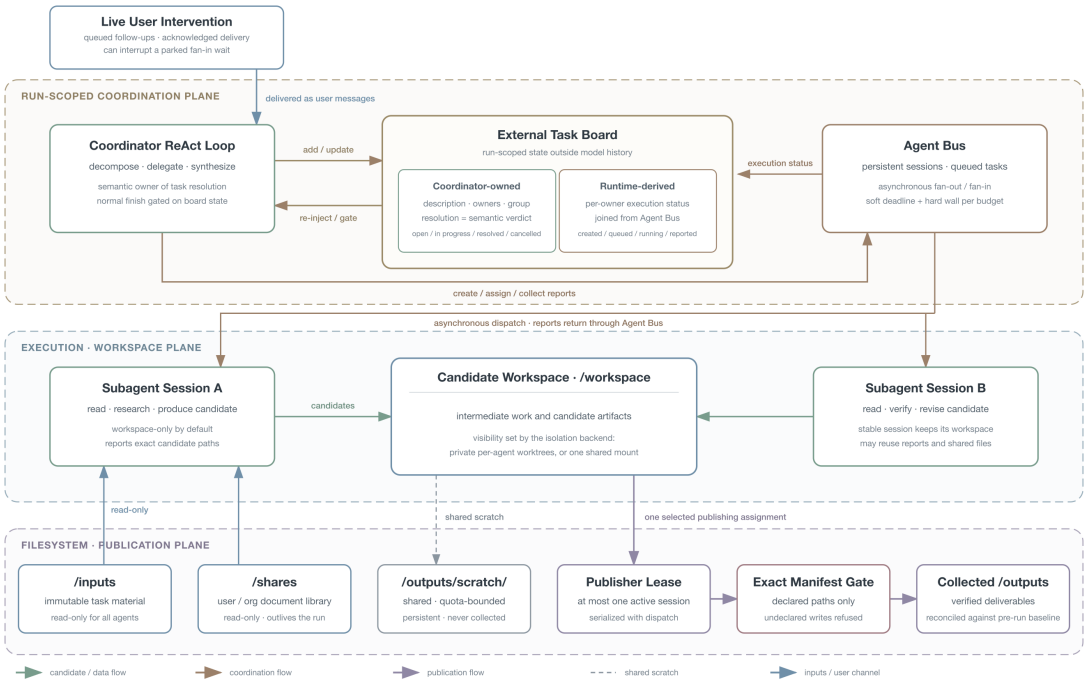

Figure 3: The Agent Team coordination extension over the shared AgentOS workspace. The Task Board stores coordinator-owned resolution state outside the model message history, while the Agent Bus exposes runtime execution state. Agents read immutable inputs and an optional cross-run document library, produce candidates in backend-dependent workspaces, and publish a declared manifest through a single authorized assignment.

3.3.3 Execution Continuity over Long Horizons

Long-running work must accept information that did not exist when the initial query was submitted. AgentOS therefore exposes an opt-in control channel backed by a run-scoped message queue: a client may add, update, or withdraw a pending follow-up, and the runtime acknowledges delivery only after the message has entered the model-visible history as a new user input. Pending interventions are normally drained before the next model call, allowing both ReAct and Agent Team runs to revise future actions without discarding valid workspace state.

Agent Team extends the same channel across asynchronous fan-in. A coordinator waiting for subagent reports can be woken while those subagents continue running, placing the intervention before its next model request instead of after the entire fan-in. Typed directives can also gate delegation or publication until a rejected plan is revised. Accepting a new follow-up renews the run's wall-time budget so that the additional request receives a fresh execution window rather than only the remainder of the original one.

Context pressure and graceful budget exhaustion. Long trajectories fail in two characteristic ways: the context grows beyond what the inference endpoint can accept, and useful work is cancelled while it is still being consolidated. AgentOS 1.1 addresses the two failures separately.

Tiered compaction is triggered by the token usage reported by the inference endpoint rather than by a local text estimate. A first, cheap tier evicts the bodies of older tool observations while preserving message structure, recent results, and protected fan-in reports; only if the measured relief is insufficient does a second tier summarize the older middle of the trajectory with an LLM. Escalation is therefore driven by the relief actually obtained, not by a fixed schedule.

For wall-clock exhaustion, each execution budget yields both a soft deadline and an outer hard timeout. As the soft deadline approaches, an observer asks the active agent to consolidate its findings, and model

retries and tool calls clamp their waits to the remaining time. Agent Team applies the same rule to subagent execution and blocking fan-in. Reaching the soft deadline ends the loop normally, after which a bounded, tool-free finalization call recovers a useful partial result from work already completed; hard cancellation remains only as a last resort.

Pause and cancellation are distinguished from this budget path: a cooperative pause is observed between completed turns rather than cancelling an in-flight one. On pause or abnormal termination, completed observations, evidence, and files remain in their respective stores; Agent Team additionally retains the most recent Task Board projection and completed subagent reports.

3.3.4 Controlled Artifact Delivery

AgentOS separates producing an artifact from declaring it as a final deliverable. A publishing execution declares an exact manifest of paths under `/outputs`; a run-scoped lease grants commit authority to at most one active session at a time. This distinction applies to any artifact-producing run and becomes essential when multiple Agent Team branches contribute to a shared deliverable. The lease may move to another session, and its holder may revise the manifest, only under controlled conditions; superseded entries cannot silently become undeclared deliverables.

The manifest is enforced before execution by the built-in file and shell writing surfaces: non-publishers are fail-closed on writes into `/outputs`, and the publisher may write only its declared paths. At termination, each manifest entry is reconciled against a baseline snapshot taken when the lease was granted, so an empty file—or a stale same-named file left by an earlier round—cannot satisfy the delivery contract. The terminal answer is likewise checked for delivery claims the manifest does not cover; an incomplete or unverified delivery is surfaced explicitly instead of being reported as success.

One namespace is intentionally outside the single-writer partition: `/outputs/scratch/` is a shared, quota-bounded area for intermediate products that must survive across rounds. It is writable by all assignments and excluded from collection, supporting collaboration without conflating reusable intermediate state with final delivery.

Table 2 summarizes the runtime mechanisms and the failure modes each one is designed to contain.

Table 2: AgentOS 1.1 mechanisms and the failure modes they address.

Concern	Failure mode	Mechanism
Workspace state	Ambiguous ownership or backend-dependent visibility	Stable three-region namespace with explicit private/shared workspace topology
Reference library	Durable user documents exposed to mutation	Read-only `/shares` mount, configured by the deployment
Coordination state	Plan and completion state disappear during compaction	External task board, periodic re-injection, and finalization gate
Live intervention	A follow-up waits behind a long fan-in operation or races a report	Message-addressable queue, interruptible waits, delivery acknowledgement, and lease renewal
Context pressure	Estimate-driven compaction fires too early or too late	Provider-reported trigger, tool-result eviction, and conditional LLM summary
Budget enforcement	Hard cancellation discards work in flight	Shared soft/hard budget, deadline-clamped waits, and bounded report recovery
Shared delivery	Concurrent, undeclared, stale, or incomplete output files	Single-publisher lease, exact manifest, scoped write policy, and baseline reconciliation

Operational Boundaries. The mechanisms above define a runtime contract for state, execution, and delivery; they do not guarantee that a retrieved source, computation, scientific method, or final conclusion is correct. Those claims still require task-appropriate verification, reproducible computation, and human review in high-stakes settings.

The publication guard mediates AgentOS's built-in file and shell writing surfaces; it is not a syscall-level filesystem monitor. Commands whose output paths cannot be determined are refused rather than admitted on trust, and where the isolation backend permits, an independent mount-level restriction backs the tool-level policy.

The coordination plane is also run-scoped rather than a durable distributed database. The current task board and Agent Bus sessions live in the worker process, and the runtime does not atomically checkpoint them together with the workspace filesystem. A client can retain the last streamed board on pause or abnormal termination, but process-restart recovery and historical filesystem rewind are outside the present contract. A resumed execution can only observe the workspace as it currently exists; it cannot return that filesystem to an earlier turn. Joint versioning of coordination state, message history, and the workspace is therefore the natural extension of the mechanisms described here.

3.4 Training

The preceding sections specify the environment, coordination, and runtime behaviors that training must strengthen. Delegation, staged return, synthesis, branch revision, and recovery are represented in the training trajectories rather than treated only as properties of an inference-time wrapper.

3.4.1 Supervised Fine-Tuning

The SFT stage provides the behavioral cold start for subsequent training. Its mixture spans general reasoning, agentic tool use, search, file interaction, coding, mathematics, scientific and financial reasoning, professional delivery, and multi-agent coordination. Examples from these domains are normalized into a common stateful interaction schema, allowing tool use, planning, recovery, and coordination behaviors to compose within a single policy.

Filtering prioritizes behavioral validity. We remove trajectories with invalid tool interactions, inconsistent state, ignored observations, or incomplete delivery, and relax role-alternation sequence constraints for asynchronous agent interaction data. Standard reject sampling is applied when gold answers are available; for rubric-scored tasks, we instead select task-relevant rubric dimensions and thresholds to retain high-quality demonstrations without relying on a single aggregate judge score.

To balance specialization with general capability, we train SFT variants over major capability domains, including general, agentic, and coding data, and combine them through model-soup merging. The resulting checkpoint serves as the unified behavioral initialization for subsequent optimization stages.

3.4.2 Reinforcement Learning

Our reinforcement learning stage targets sustained progress over long-horizon agentic tasks. File, code, search, and coordination environments induce heterogeneous interaction patterns, execution costs, trajectory lengths, and failure modes, while successful runs often compose several of these capabilities within one task. The primary algorithmic problem is assigning useful credit within trajectories whose terminal outcomes reveal little about which decisions should change. Recent work similarly shows that selecting informative intermediate turns can make agentic post-training substantially more compute-efficient (Yi et al., 2026). PIVOT-RL addresses this problem through localized trajectory optimization, while asynchronous optimization makes learning practical over the resulting irregular rollout stream.

PIVOT-RL: Localized Optimization at Consequential Decisions. Terminal outcomes provide only coarse supervision for long agentic trajectories, where useful intermediate work may precede a late failure and successful traces may still contain inefficient or weakly grounded decisions. PIVOT-RL uses hindsight-guided trajectory localization over a large policy-training corpus. Retrospective analysis identifies consequential decision points—*pivots*—where the model begins to follow an unproductive

strategy, rely on insufficient evidence, misuse a tool, or fail to revise an assumption.

At each pivot, we preserve the useful prefix and construct a localized continuation task with a short corrective hint. The hint provides directional guidance for the local correction, is never a prediction target, and is absent at inference time; for stateful tasks, we also restore the corresponding executable environment state. Localized continuations are mixed with unhinted full-task questions, combining efficient learning at failure-relevant states with autonomous end-to-end problem solving. This converts fragment-level credit assignment into targeted policy optimization: the completed prefix is retained, while learning is concentrated on the segment where a consequential correction is required. Figure 4 shows the resulting RL training dynamics on search, knowledge, and science tasks, all of which improve consistently as reinforcement learning scales.

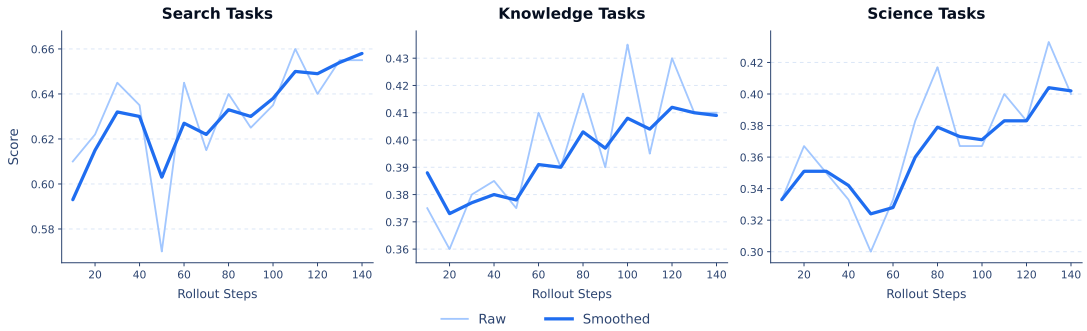

Figure 4: Scaling trends on three held-out agentic evaluations as RL compute increases.

To train efficiently over heterogeneous agentic workloads, completed trajectories enter optimization asynchronously without waiting for slower episodes. This is particularly useful for long-horizon tasks involving search, code execution, file processing, and coordination, whose wall-clock costs vary substantially even when their learning objectives are similar. This systems mechanism supports training throughput, while task-specific environments and verifiers continue to define the interaction state and rewarded behavior.

4 Evaluation

We evaluate Apodex 1.1 at three levels. First, public benchmarks measure the breadth of the model and the additional gains obtained from Agent Team coordination. Second, we report results on an internal structured-search benchmark and introduce a complementary benchmark for end-to-end research delivery. Third, HDS6 evaluates the quality of the execution process rather than only its final score. This progression moves from outcomes, to capability-specific evidence, to the reliability of the work used to produce them.

Across these views, Apodex 1.1 operates in the leading performance band of current agentic systems on complex work. The Agent Team system matches or exceeds strong reported reference points on several professional-work, finance, and scientific-research evaluations, while remaining broadly competitive in general reasoning, search, mathematics, and coding. The 35B-parameter Apodex 1.1 mini provides a direct test of model-scale efficiency: across representative work, finance, and scientific-research tasks, it reaches the performance band of selected frontier systems and improves substantially over Apodex 1.0 mini on overlapping evaluations.

4.1 Empirical Setup

We evaluate Apodex under two execution modes: *ReAct* and *Agent Team*. Scores are reported to one decimal place unless a benchmark's native unit requires otherwise. Semantic evaluations use the benchmark's stated judge or rubric, and captions identify internally reproduced comparison results.

ReAct. The model operates through a simple ReAct-style loop, alternating between reasoning, tool interaction, and observations. This setting minimizes additional orchestration so that performance primarily reflects the model's reasoning and tool-use policy.

Agent Team. The main agent can dynamically spawn subagents to parallelize or specialize work and can allocate focused verification when needed. This setting evaluates the trained ability to decompose tasks, delegate work, integrate results, and revise the shared plan under additional organized computation. Section 3.2.5 is only used for online products and is not included in offline evaluations.

4.2 Main Results

(a) Professional Work

Models	APEX-Agents	GDPVal
DeepSeek-V4-Pro (DeepSeek-AI, 2026)	24.3	71.2
Gemini-3.1-Pro (Google, 2026a)	32.0	–
Claude-Opus-4.6 (Anthropic, 2026b)	33.0	–
GPT-5.4 (OpenAI, 2026b)	33.3	–
DeepSeek-V4-Flash-0731 (DeepSeek-AI, 2026)	34.4	72.7
GLM-5.2 (Zeng et al., 2026)	35.6	–
GPT-5.5 (OpenAI, 2026c)	38.5	–
GPT-5.6-Terra (OpenAI, 2026d)	38.9	–
Claude-Opus-4.8 (Anthropic, 2026d)	39.4	80.2
GPT-5.6-Sol (OpenAI, 2026d)	39.9	79.3
Kimi-K3 (max) (Team et al., 2026)	41.0	80.0
Claude-Opus-5 (Anthropic, 2026)	**42.3**	**89.4**
Apodex 1.0 (Apodex Team, 2026)	16.5	59.3
Apodex 1.1 *w/ ReAct*	34.4	69.5
Apodex 1.1 *w/ Agent Team*	38.5	78.8

(b) Finance and Business Simulation

Models	FrontierFinance	YC-Bench Final Net Worth (USD)
Gemini-3.1-Pro (Google, 2026a)	30.5	$66,104
Gemini-3.5-Flash	36.1	$987,017
GLM-5.2 (Zeng et al., 2026)	42.8	$1,013,158
DeepSeek-V4-Flash-0731 (DeepSeek-AI, 2026)	44.2	–
DeepSeek-V4-Pro (DeepSeek-AI, 2026)	45.5	$1,066,426
Gemini-3.6-Flash (Google, 2026b)	46.3	–
GPT-5.6-Sol (OpenAI, 2026d)	46.8	$685,879
Kimi-K3 (max) (Team et al., 2026)	48.8	–
Claude-Fable-5 (Anthropic, 2026a)	49.2	**$1,977,573**
Apodex 1.0 (Apodex Team, 2026)	40.3	$47,966
Apodex 1.1 *w/ ReAct*	48.7	$1,038,255
Apodex 1.1 *w/ Agent Team*	**54.3**	–[†]

Table 3: Performance on professional work, finance, and business simulation benchmarks. Bold indicates the best reported result in each benchmark column. Apodex results are grouped below the horizontal rule. All GDPVal entries report win rate; all external-model GDPVal results shown here are our reproductions under the Apodex harness. The Claude Opus 5 APEX-Agents result is likewise reproduced internally. YC-Bench (He et al., 2025) reports mean final net worth in USD; external YC-Bench results are taken from its official leaderboard. The DeepSeek V4 entries on FrontierFinance are internally reproduced under the stated benchmark metric; the remaining external FrontierFinance entries are reported reference results. [†]YC-Bench uses the benchmark's default agent configuration; no separate Agent Team result is reported.

Tables 3 and 4 provide a common snapshot across public agentic and knowledge-intensive evaluations. Table 5 separately isolates the 35B-parameter Mini model to make model-scale efficiency explicit.

(a) Scientific Research		
Models	**FrontierScience Research**	**BioMysteryBench Human-difficult Set**
Gemini-3.1-Pro (Google, 2026a)	16.7	–
Claude-Opus-4.5 (Anthropic, 2025a)	17.5	–
Claude-Opus-4.7 (Anthropic, 2026c)	20.0	27.0
GPT-5.2 (OpenAI, 2026a)	25.2	–
Seed2.1-Turbo (ByteDance Seed Team, 2026)	33.3	–
GPT-5.5 (OpenAI, 2026c)	33.9	–
Meta-Muse-Spark (Meta AI, 2026)	38.0	–
Seed2.1-Deep-Think (ByteDance Seed Team, 2026)	40.7	–
DeepSeek-V4-Flash-0731 (DeepSeek-AI, 2026)	55.0	–
Claude-Opus-4.6 (Anthropic, 2026b)	–	23.5
Claude-Mythos-Preview (Anthropic, 2026b)	–	29.6
Claude-Opus-5 (Anthropic, 2026)	–	**49.4**
Apodex 1.0 (Apodex Team, 2026)	28.3	17.6
Apodex 1.1 *w/ ReAct*	55.0	23.5
Apodex 1.1 *w/ Agent Team*	**63.3**	35.3

(b) General Reasoning and Deep Search		
Models	**Humanity's Last Exam**	**DeepSearchQA**
DeepSeek-V4-Pro (DeepSeek-AI, 2026)	48.2	–
Gemini-3.1-Pro (Google, 2026a)	51.4	81.9
GPT-5.5 (OpenAI, 2026c)	52.2	94.0
Claude-Opus-4.6 (Anthropic, 2026b)	53.0	91.3
Qwen3.7-Max	53.5	–
Kimi-K2.6 (Moonshot AI, 2026)	54.0	92.5
GLM-5.2 (Zeng et al., 2026)	54.7	–
Claude-Opus-4.7 (Anthropic, 2026c)	54.7	91.7
Kimi-K3 (max) (Team et al., 2026)	56.0	**95.0**
Qwen3.8-Max	56.2	–
Claude-Opus-5 (Anthropic, 2026)	**64.7**	**95.0**
Apodex 1.0 (Apodex Team, 2026)	49.0	84.6
Apodex 1.1 *w/ ReAct*	53.2	88.2
Apodex 1.1 *w/ Agent Team*	56.1	92.4

Table 4: Performance on scientific-research, general-reasoning, and deep-search benchmarks. Bold indicates the best reported result in each benchmark column. Apodex results are grouped below the horizontal rule. DeepSearchQA reports F1, and BioMysteryBench uses the Human-difficult subset.

The cross-benchmark pattern is more informative than any single rank. The ReAct setting already places the underlying model near strong frontier references on multiple task families; Agent Team then converts additional coordinated computation into further gains, including the strongest values shown in our FrontierFinance and FrontierScience-Research comparison tables and competitive performance on GDPVal, APEX-Agents, HLE, and DeepSearchQA. This combination of breadth and model-scale efficiency is the main empirical result of Apodex 1.1.

35B model-scale efficiency. Apodex 1.1 mini shows that the gains are not confined to the largest model. On the overlapping FrontierFinance and APEX-Agents evaluations, ReAct improves over Apodex 1.0 mini from 33.2 to 40.0 and from 15.4 to 24.2, respectively. With ReAct alone, the 35B model reaches 40.0 on FrontierFinance, 45.0 on FrontierScience-Research, and 24.2 on APEX-Agents, establishing a strong underlying working policy before multi-agent coordination is added. Agent Team raises these scores to 50.2, 51.7, and 27.7, corresponding to gains of 10.2, 6.7, and 3.5 points. The resulting system leads the selected FrontierFinance comparison, approaches DeepSeek V4 Flash 0731 on FrontierScience-Research, and matches the performance band of Kimi K2.6 on APEX-Agents. This result is important beyond the individual rankings: it shows that broader executable environments, agentic training, and organized

Models	FrontierFinance	FrontierScience Research	APEX-Agents
Proprietary models			
Gemini-3.1-Pro (Google, 2026a)	30.5	16.7	32.0
GPT-5.6-Sol (OpenAI, 2026d)	46.8	–	**39.9**
Qwen3.7-Plus (Qwen Team, 2026)	–	–	22.4
Muse-Spark-1.0 (Meta AI, 2026)	–	38.0	–
Open-weight models			
Kimi-K2.6 (Moonshot AI, 2026)	32.3	–	27.9
DeepSeek-V4-Pro (DeepSeek-AI, 2026)	45.5	–	24.3
GLM-5.2 (Zeng et al., 2026)	42.8	–	35.6
DeepSeek-V4-Flash-0731 (DeepSeek-AI, 2026)	44.2	**55.0**	34.4
Apodex 1.0 mini *w/ ReAct*	33.2	25.0	15.4
Apodex 1.1 mini *w/ ReAct*	40.0	45.0	24.2
Apodex 1.1 mini *w/ Agent Team*	**50.2**	51.7	27.7

Table 5: Model-scale-efficient agentic performance of Apodex mini models (35B parameters). Models are grouped by availability; Apodex results appear below the final rule. Dashes denote unavailable matched results. Bold indicates the best result in each benchmark column among the selected comparison rows.

coordination can produce frontier-band complex-work capability at a compact 35B model scale. The comparison is deliberately stated as a performance-band result: parameter counts for several proprietary reference systems are not public.

4.3 Capability Analysis

The main tables establish breadth; the following analyses examine the corresponding capabilities in professional work, scientific research, general reasoning and search, mathematics, and software engineering.

4.3.1 Professional Work

Complex professional work combines domain reasoning with the production of artifacts that must survive practical review. We evaluate both broad professional deliverables and the specialized finance workflows needed to produce them.

Professional Work Domain. GDPVal and APEX-Agents capture complementary aspects of professional work. GDPVal emphasizes the quality of a completed work product across a broad occupational distribution, whereas APEX-Agents emphasizes sustained execution across high-context files and applications under strict completion criteria. Together, they evaluate both the quality of what the model produces and its ability to carry out the work needed to produce it.

GDPVal evaluates economically valuable, real-world knowledge work across 44 occupations in nine industries, with tasks created by experienced professionals and grounded in representative work products (Patwardhan et al., 2025). Prompts include reference files and context, and expected outputs span documents, slides, diagrams, spreadsheets, and other professional artifacts. We report GDPVal solely by win rate. Apodex 1.1 reaches 78.8 with Agent Team and 69.5 with ReAct; the 9.3-point gain shows that coordination improves the quality of complete professional deliverables.

APEX-Agents complements this view with 480 long-horizon, cross-application tasks constructed by investment banking analysts, management consultants, and corporate lawyers across 33 data-rich worlds (Vidgen et al., 2026). Agents must navigate realistic file systems and software, maintain context over extended workflows, and satisfy expert-defined criteria for task completion. Apodex 1.1 reaches 34.4 with ReAct and 38.5 with Agent Team, compared with 16.5 for Apodex 1.0. The ReAct result more than doubles the previous version, while Agent Team adds a further 4.1 points.

The two benchmarks expose different parts of the same improvement. GDPVal shows broader gains in artifact quality across occupations; APEX-Agents shows stronger persistence and completion in professional-services workflows that require files, tools, and long-horizon control. The improvement from Apodex 1.0 to Apodex 1.1 under ReAct indicates a stronger underlying working policy, while the additional Agent Team gains show that this policy can productively use coordinated computation. Taken together, these results establish Apodex 1.1 as a highly capable system for complex professional work.

Finance Domain. FrontierFinance measures support for the full investment workflow, from screening and discovery to company research, financial modeling, earnings analysis, and portfolio monitoring (Samaya Research, 2026). Its 220 open-ended queries are graded against 11,543 expert-written, source-attributed rubric items. The headline score is the rubric qualification rate macro-averaged over questions, with unanswered questions scored as zero. API baselines use the maintainers' Finance Agent v2 scaffold (Vals AI, 2026); Apodex runs use the same public data boundary, official grader, and metric, while retaining their stated ReAct or Agent Team harness.

As shown in Table 3, Apodex 1.1 reaches 48.7 with ReAct. Agent Team raises the score to 54.3, a gain of 5.6 points, showing that financial research benefits when evidence collection, quantitative analysis, and verification are coordinated across workstreams.

4.3.2 Scientific Research

We evaluate scientific capability from two complementary perspectives. FrontierScience-Research measures cross-domain, expert-level scientific reasoning, whereas BioMysteryBench measures end-to-end analysis of heterogeneous biological data. Together, they test whether the system can sustain evidence gathering, analysis, and verification beyond short-form scientific question answering.

FrontierScience. FrontierScience-Research comprises 60 research-level tasks spanning physics, chemistry, and biology. Each response is graded against a ten-point rubric, and a task passes when it receives at least seven rubric points (Wang et al., 2025a). Apodex 1.1 reaches 55.0% with ReAct and 63.3% with Agent Team, an 8.3-point gain from agentic coordination, compared with 28.3% for Apodex 1.0. The corresponding mean rubric scores are 6.7 and 6.9.

BioMysteryBench. BioMysteryBench evaluates bioinformatics research over realistic, noisy datasets rather than self-contained questions (Anthropic, 2026a). On the revised 17-task Human-difficult Set, Apodex 1.1 scores 23.5% (4/17) with ReAct and 35.3% (6/17) with Agent Team, while Claude Opus 5 reports 49.4%. The Claude 4.x values in Table 4 use the original 23-task set and are included as historical references.

Across both benchmarks, Agent Team improves over ReAct, with particularly large gains on the biomedical tasks. The results support agentic coordination for decomposing research problems, maintaining parallel lines of evidence, and verifying intermediate conclusions.

4.3.3 General Reasoning and Search

Humanity's Last Exam tests broad expert-level reasoning with tools, while DeepSearchQA emphasizes multi-step evidence seeking and synthesis. Apodex 1.1 with Agent Team reaches 56.1 on HLE and 92.4 F1 on DeepSearchQA, compared with 53.2 and 88.2 for ReAct.

4.3.4 Mathematical Reasoning

Mathematics provides a compact test of whether the same agent system can sustain search-free formal reasoning rather than only retrieve and synthesize evidence. As summarized in Table 6, the matched

Agent Team comparison shows a large generational gain: Apodex 1.1 raises the MathArena-derived[1] score over Apodex 1.0 from 12.5 to 36.5 on the evaluated IMO 2025 set, from 13.0 to 30.5 on IMO 2026, and from 5.8 to 26.5 on USAMO 2026. Within Apodex 1.1, Agent Team further improves over ReAct from 24.3, 18.5, and 16.0 on the three sets, respectively. The corresponding reference thresholds are 35, 29, and 25. IMO-ProofBench provides a separate proof-oriented check: Agent Team reaches 96.7% on Basic and 63.3% on Advanced, compared with 80.0% and 46.4% for ReAct. Apodex 1.1 with Agent Team exceeds the stated reference threshold on all three evaluated sets, including the IMO gold-medal cutoffs.

Models	**IMO 2025**	**IMO 2026**	**USAMO 2026**	**ProofBench Basic**	**ProofBench Advanced**
Reference threshold	*35*	*29*	*25*	–	–
Apodex 1.0 *w/ Agent Team*	12.5	13.0	5.8	63.3	20.0
Apodex 1.1 *w/ ReAct*	24.3	18.5	16.0	80.0	46.4
Apodex 1.1 *w/ Agent Team*	**36.5**	**30.5**	**26.5**	**96.7**	**63.3**

Table 6: Mathematical reasoning results on competition-level proof tasks. IMO 2025, IMO 2026, and USAMO 2026 are scored under the MathArena protocol.

4.3.5 Coding and Software Engineering

Coding evaluations test a different form of complex work: the model must inspect an existing repository or terminal state, make executable changes, and use feedback from tests or commands to repair failures. We focus the report on two established, environment-grounded evaluations (Jimenez et al., 2024; Merrill et al., 2026). Terminal-Bench 2.1 measures sustained execution in realistic command-line environments, while SWE-bench Verified measures repository-level issue resolution against executable tests.

Table 7: Coding-agent results on Terminal-Bench 2.1 and SWE-bench Verified. One result is retained per model on each benchmark; where multiple settings are available, the highest reported score is shown.

Benchmark	Model	Score ↑
Terminal-Bench 2.1	Gemini 3.6 Flash	91.9
	Kimi K3	88.3
	DeepSeek V4 Flash 0731	82.7
	Claude Opus 5	77.3
	Apodex 1.1	70.8
SWE-bench Verified	Claude Opus 5	92.2
	Kimi K3	80.8
	DeepSeek V4 Pro	80.6
	Apodex 1.1	77.7

Apodex 1.1 reaches 70.8 on Terminal-Bench 2.1 and 77.7 on SWE-bench Verified.

4.4 Internal Evaluation

Public benchmarks provide breadth, but they do not fully capture structured search deliverables or open-ended research execution. We therefore report FrontierSearchBench results for structured evidence acquisition and outline FrontierResearchBench as a complementary evaluation of end-to-end scientific delivery.

[1] https://github.com/eth-sri/matharena

4.4.1 FrontierSearchBench

FrontierSearchBench is an internal benchmark of 41 verifiable deep-search tasks (Apodex Team, 2026). Each task specifies a structured deliverable, such as an enumerated set, an ordered list, or a numeric summary, whose components must be gathered and reconciled across many sources. Scoring checks a set of ground-truth dimensions per task rather than a single answer string. Tasks are constructed so that the correct answer does not drift with retrieval date, and task construction, ground-truth annotation, and scorer implementation were completed before and independently of all evaluated runs.

Scoring proceeds in three stages per task: an extraction stage converts the delivered report into structured claims, the claims are aligned to the frozen ground-truth dimensions by a fixed panel of judge models, and a rubric deterministically assigns the normalized task score $r_i \in [-1, 1]$, with incorrect assertions penalized below zero to discourage exhaustive guessing. Each r_i is the scalar outcome for the i-th task contract in the sense of Eq. (5). FrontierSearchBench aggregates the task-level outcomes as

$$s_{\mathrm{FSB}} = \frac{100}{N} \sum_{i=1}^{N} r_i, \tag{8}$$

where $N = 41$ is the number of task contracts in the suite. The reported number is thus an unweighted mean on a 100-point scale, with mathematical range $[-100, 100]$; a negative aggregate is possible when hard-negative penalties exceed positive credit. We additionally report the fractions of tasks with positive, zero, and negative r_i.

Table 8: FrontierSearchBench results: mean normalized task score over 41 tasks on a 100-point scale, with the proportion of tasks scored positive, zero, and negative. Negative aggregate scores are possible. Column best values are in bold.

Model	Positive (%) ↑	Zero (%) ↓	Negative (%) ↓	Avg. score ↑
DeepSeek-V4-Flash-0731 (DeepSeek-AI, 2026)	75.6	19.5	4.9	54.9
Kimi-K3 (Team et al., 2026)	75.6	22.0	2.4	60.1
DeepSeek-V4-Pro (DeepSeek-AI, 2026)	85.4	14.6	**0.0**	61.3
Claude-Opus-5 (Anthropic, 2026)	85.4	12.2	2.4	64.4
GPT-5.6-Sol (OpenAI, 2026d)	85.4	14.6	**0.0**	67.4
Apodex 1.1 *w/ ReAct*	75.6	19.5	4.9	57.0
Apodex 1.1 *w/ Agent Team*	**87.8**	**9.8**	2.4	**69.1**

Table 8 presents the results on FrontierSearchBench. Apodex 1.1 with ReAct is competitive with the open-weight systems, and Agent Team lifts the average score past the strongest proprietary reference shown, achieving the best result in the comparison. Agent Team roughly halves the share of tasks earning no credit relative to ReAct, the lowest of any system, while keeping incorrect assertions rare.

4.4.2 FrontierResearchBench

FrontierResearchBench targets high-difficulty, end-to-end scientific workflows rather than isolated scientific question answering. We build FrontierChallenge, our internal benchmark collection, with 97 executable tasks across materials science, chemistry, chemical engineering, life science, bioinformatics, medical imaging, environmental analysis, computational chemistry, molecular simulation, and physical modeling. Given a fixed objective and input data in a task-specific Docker environment, an agent must complete the analysis and deliver a mutually consistent set of research artifacts, such as executable code, structured data, figures, domain-specific files, and a written report. A complete example is provided in Appendix A, Case 1, where the agent analyzes the prognostic role of EASIX after allogeneic transplantation and delivers the full statistical workflow as Excel and Word artifacts.

The benchmark operationalizes research rigor through executable and cross-artifact verification. Each task has a custom Grader that checks required files, numerical results, formats, executable outputs, and

consistency across artifacts; a plausible narrative cannot compensate for an invalid analysis or mutually inconsistent deliverables. Deterministic rules are combined with rubric-defined semantic judgments from GPT-5.6-Sol when qualitative scientific assessment is required, while the task Grader, rather than the Judge model, computes the final outcome. We report *Pass Rate*, the fraction of tasks receiving the full task score, so any unmet requirement results in a non-pass. Each row in Table 9 is a model–harness system; Apodex 1.1 *w/ Agent Team* is evaluated in the FrontierAgent harness.

Table 9: FrontierResearchBench results. Pass Rate is the fraction of tasks awarded full score. Column best values are in bold.

Model	Harness	Pass Rate (%) ↑
GPT-5.6-Sol	Codex	**20.6**
Grok-4.6	Claude Code	**20.6**
Kimi-K3	Claude Code	17.5
Claude-Opus-5	Claude Code	17.5
Qwen3.8-Max	Claude Code	15.5
DeepSeek-V4-Flash-0731	Claude Code	12.4
DeepSeek-V4-Pro-0813	Claude Code	13.4
Qwen3.5-397B-A17B	Claude Code	4.1
GLM-5.2	Claude Code	3.1
Apodex 1.1 *w/ Agent Team*	FrontierAgent	12.4
Apodex 1.1	Claude Code	10.3

On these high-difficulty workflows, Apodex 1.1 *w/ Agent Team* achieves a Pass Rate of 12.4%. Although Apodex 1.1 remains behind the frontier systems, even GPT-5.6-Sol with Codex and Grok-4.6 with Claude Code receive full credit on only 20.6% of tasks. Reliably completing a specified scientific workflow and delivering every required artifact therefore remains challenging for all evaluated systems. This capability is central to AI-enabled scientific discovery, and we will continue to strengthen both the model and its scientific execution system.

4.5 Heavy-Duty Solver *(HDS6)* Analysis

Outcome scores establish whether a task was completed; they do not show whether the underlying work was coherent, grounded, or recoverable. We therefore close the evaluation with **HDS6**, a process-verification framework aligned with the working-capability objective introduced in Section 1. HDS6 evaluates six process capabilities: long-horizon state coherence, evidence fidelity, hypothesis management, boundary and failure reasoning, tool use and execution-state management, and self-correction under verification. Each capability contains four scored rubric items, producing 24 items in total, with a separate integrity gate applied to the recorded trajectory. The framework therefore tests whether a successful delivery is supported by a defensible execution process rather than only a plausible final artifact.

Process-level judging protocol. A trajectory spanning hundreds of steps across several concurrently active agents is too long, and too structurally irregular, for a single forward pass to score reliably against all 24 scored items. We therefore adapt **HDS6** into an agentic, process-level judge for asynchronous multi-agent trajectories: tool calls, subagent dispatches, and intermediate observations are stitched into a single ordered log, then processed by role-specialized mapping, judging, review, and arbitration stages. Each rubric item receives a 0/1/2 band (fails, meets, or exceeds compliance) supported by citations to the visible execution log. Grading is outcome-blind and excludes private reasoning; every cited event is re-grounded against the recorded trajectory before it can support a verdict. A single integrity gate sits above the rubric: a fabricated tool result or a narrated action that never occurred zeroes the entire trajectory outright, bypassing the weighted score.

Findings. Figure 6 presents the HDS6 comparison between Apodex 1.0 and 1.1. The three panels are evaluation slices rather than stages of one pipeline. *Deep Discover* evaluates the 397B model with Agent

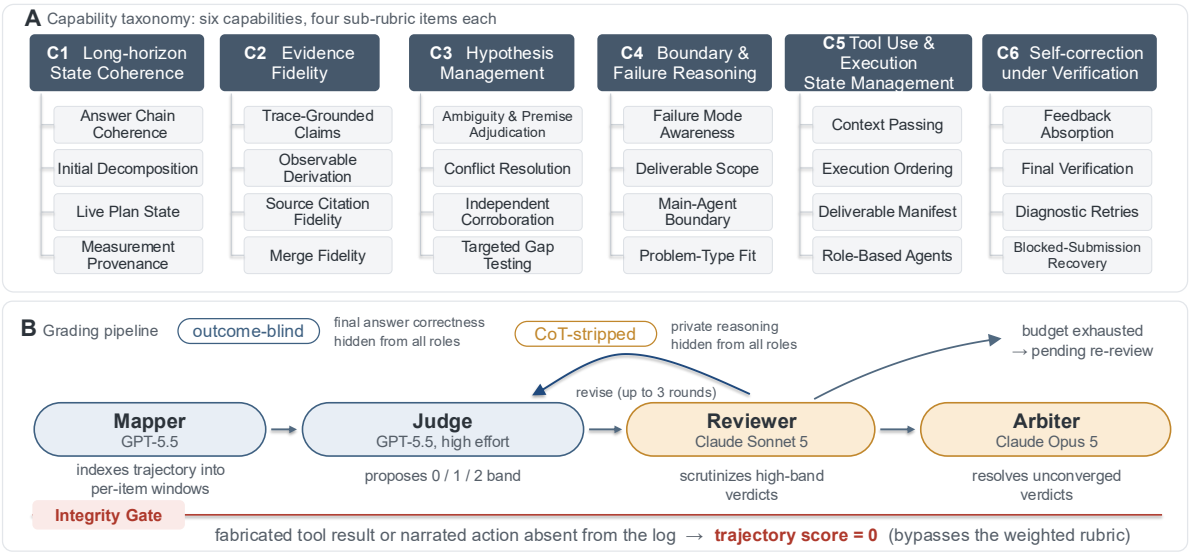

Figure 5: The HDS6 capability taxonomy and process-grading pipeline. The six capability groups contain four rubric items each; an integrity gate is applied independently of the weighted rubric.

Team: the Apodex 1.0 result aggregates eight independent runs, whereas the Apodex 1.1 result is obtained from a single run. *Deep Solve* evaluates the 397B model with ReAct, and *Deep Research* evaluates the 35B model with ReAct. The panels share the HDS6 rubric but differ in task family, model scale, execution mode, and, for Deep Discover v1.0, aggregation protocol. Scores should therefore be interpreted as within-panel system comparisons rather than compared directly across panels; Deep Discover should not be read as a strictly matched single-run model ablation. C1–C6 correspond to the six process-capability categories defined in Figure 5.

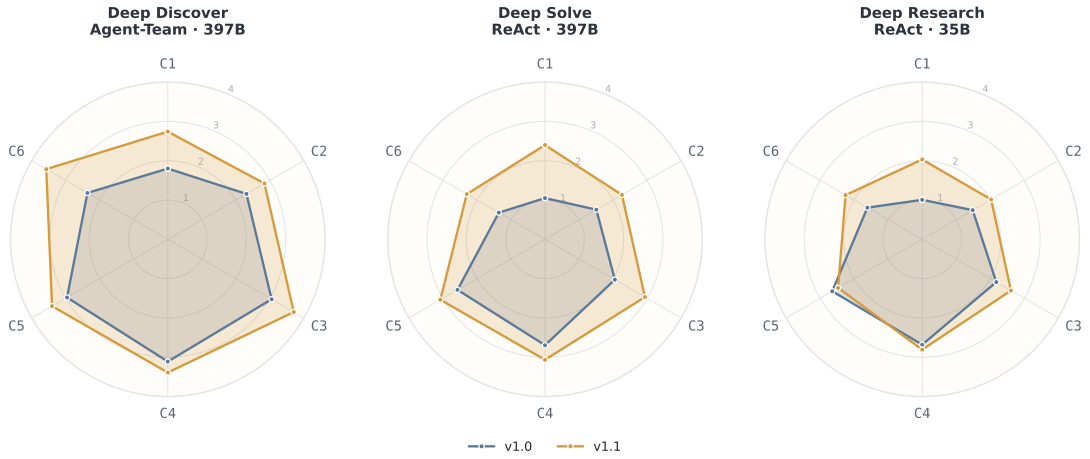

Figure 6: HDS6 comparisons between Apodex v1.0 and v1.1 across three evaluation settings. In Deep Discover, v1.0 uses the 397B Agent Team with eight-run aggregation, whereas v1.1 uses a single run. C1–C6 follow the process-capability taxonomy in Figure 5; comparisons are intended within, rather than across, panels.

The pattern is consistent with gains from both the model and the execution harness. The two largest single-item deltas are *Initial Decomposition* (+1.3) and *Final Verification* (+0.8), which align respectively with the harness-supported organization of work and stronger verification behavior. Evidence fidelity and hypothesis management also improve across their constituent rubric items, indicating broader gains in how Apodex 1.1 grounds, revises, and completes extended execution. HDS6 does not isolate the

causal contribution of an individual system component; instead, it localizes where the combined model, environment, coordination, and harness changes become visible in the work process.

Qualitative cases. Aggregate scores are complemented by three end-to-end cases covering clinical survival analysis, molecular docking, and electrochemical corrosion analysis. Appendix A documents their inputs, division of work, tool execution, intermediate checks, and delivered artifacts without interrupting the main quantitative progression.

5 Related Work

General-purpose agentic models. Recent frontier model releases increasingly treat agentic work as a core expression of general intelligence rather than an application-specific layer. Kimi K3, GPT-5.6, Claude Fable 5, GLM-5.2, and DeepSeek V4 extend general-purpose models toward different combinations of long-context reasoning, coding, tool use, information seeking, and sustained task execution (Team et al., 2026; OpenAI, 2026d; Anthropic, 2026a; Zeng et al., 2026; DeepSeek-AI, 2026). Although their architectures, training recipes, and runtime systems differ, they reflect a shared transition from optimizing isolated responses toward models that can operate through tools and environments over longer horizons. Apodex 1.1 follows this general-purpose direction, but organizes its development around two explicit scaling surfaces: Environment Scaling expands the executable worlds in which working capability is learned, while Agentic Coordination Scaling expands how that capability is organized across agents, task branches, and time.

Executable environments and real work. ReAct establishes the reasoning–action–observation loop for environmental interaction (Yao et al., 2022), while Toolformer studies how models can learn when and how to invoke external APIs (Schick et al., 2023). WebArena provides reproducible websites and functional task checks for long-horizon web agents (Zhou et al., 2023); WorkArena and BrowserGym extend environment-based evaluation to enterprise knowledge work (Drouin et al., 2024); and OSWorld evaluates cross-application tasks in real operating systems with execution-based graders (Xie et al., 2024). In software engineering, SWE-bench measures issue resolution against repository tests (Jimenez et al., 2024), and SWE-agent shows that the agent–computer interface materially affects repository-level performance (Yang et al., 2024). Terminal-Bench extends this direction to realistic command-line tasks (Merrill et al., 2026). APEX-Agents further evaluates long-horizon, cross-application work designed by lawyers, investment bankers, and management consultants, including the files, rubrics, and gold deliverables needed for professional evaluation (Vidgen et al., 2026). Apodex's File, Search, and Code environments follow the same execution-grounded principle but are used jointly for task synthesis, training, replay, and evaluation.

Search and deep-research agents. Deep-research systems extend reasoning-and-acting to open-ended information seeking, evidence reconciliation, and report synthesis. Product systems include OpenAI Deep Research, Claude Research, Kimi-Researcher, and Grok DeepSearch (OpenAI, 2025; Anthropic, 2025b; Moonshot AI, 2025; xAI, 2025). Open research has examined supervised and reinforcement-learning recipes for this behavior. WebThinker integrates autonomous search with long-form reasoning (Li et al., 2025b); DeepResearcher trains in real-world web environments (Zheng et al., 2025); WebDancer and WebSailor study autonomous information seeking and difficult web navigation (Wu et al., 2025; Li et al., 2025a); Search-R1 learns multi-turn retrieval with outcome-based reinforcement learning (Jin et al., 2025); and Tongyi DeepResearch presents an end-to-end agentic training recipe with broad benchmark analysis (Team et al., 2025). Apodex incorporates search as one environment within a broader working process: retrieved evidence may become input to code, file transformations, parallel investigation, and delivery-level verification rather than terminating in a report by default.

Agents for scientific discovery. Scientific-agent research demonstrates why useful work must include executable methods and artifacts. ChemCrow augments a language model with chemistry tools for synthesis, drug discovery, and materials tasks (Bran et al., 2023). The AI Scientist joins idea generation, code execution, experimentation, visualization, paper writing, and simulated review in an automated research loop (Lu et al., 2024); its successor introduces agentic tree search and a dedicated experiment manager (Yamada et al., 2025). The AI co-scientist uses an asynchronous coalition of specialized agents to generate, debate, rank, and evolve scientific hypotheses under researcher guidance (Gottweis et al., 2025). These systems emphasize different parts of the scientific process. Apodex instead scales agentic intelligence in a general-purpose model and execution runtime rather than targeting a domain-specific laboratory agent: scientific workflows are a primary use case, but the same file–search–code policy is also evaluated on software and professional work.

Multi-agent coordination and test-time scaling. AutoGen coordinates agents through programmable conversations (Wu et al., 2023); MetaGPT and ChatDev organize specialized roles around structured software-development workflows (Hong et al., 2023; Qian et al., 2023); and AgentVerse studies collaborative groups and emergent behavior (Chen et al., 2023). Multi-agent debate instead uses independent solutions and critique to improve reasoning or factuality (Du et al., 2023; Liang et al., 2023). More recent scientific systems demonstrate dynamic task allocation and asynchronous execution for hypothesis generation (Gottweis et al., 2025). These results also caution that additional agents are not uniformly beneficial: collaboration is most useful when work can be decomposed and information can be integrated reliably. Apodex's Agent Team is designed around staged result return, shared task state, subagent execution control, and user intervention rather than a fixed dialogue graph or majority-vote ensemble.

Verification and process supervision. Self-Refine iterates generation and self-feedback (Madaan et al., 2023), and Reflexion stores verbal feedback from prior attempts to guide later decisions (Shinn et al., 2023). Chain-of-Verification generates targeted verification questions before revising an answer (Dhuliawala et al., 2023), while process-supervision work trains reward models to assess intermediate reasoning steps (Lightman et al., 2023). These methods differ in whether critique is produced by the generator, a separate prompted role, or a learned verifier. Apodex combines three verification levels: environment checks over executable outcomes, artifact and provenance checks over the completed workspace, and Statement Review over consequential claims with an independent role and context.

Environment scaling and agentic reinforcement learning. Recent work increasingly treats environment construction itself as a scaling dimension for agent learning. Environment Scaling studies the path from broader interactive worlds to general agentic intelligence (Fang et al., 2025); ScaleEnv scales synthesized tool-use environments from scratch (Tu et al., 2026); Agent-World develops real-world environment synthesis for continually evolving general agents (Dong et al., 2026); and TaskCraft and Agent Learning via Early Experience study task generation and early interaction experience as sources of agent capability (Shi et al., 2025; Zhang et al., 2025). Complementary training work addresses the optimization challenges created by these environments. Continual pre-training can establish broad tool-use behavior at an earlier stage (Su et al., 2025); Search-R1 and RAGEN optimize multi-turn interaction (Jin et al., 2025; Wang et al., 2025c); DAPO and related methods improve policy-optimization stability (Yu et al., 2025; Qi et al., 2026); and ROLL and ROLL Flash provide scalable asynchronous RL infrastructure (Wang et al., 2025b; Yao et al., 2025). Apodex integrates these directions through File, Search, and Code worlds with a shared delivery contract, artifact and provenance state, replay requirements, independent verification, trained Agent Team coordination, and a failure-driven task-construction loop.

6 Conclusion

Apodex 1.1 advances a model-level view of agentic intelligence for complex work: capability should be measured by whether a model can sustain progress through a changing task, use tools and evidence

effectively, recover from failure, and deliver a verifiable result. We develop this capability along two complementary scaling dimensions. Environment Scaling broadens the executable file, search, and code worlds in which the model learns and acts, while Agentic Coordination Scaling broadens how work is decomposed, delegated, integrated, and reorganized across agents and time. A common harness and AgentOS provide the persistent execution state required by both dimensions, and unified training converts environment and coordination trajectories into a stronger working policy.

The evaluation is designed around the same objective. Results across scientific research, professional work, mathematics, search, coding, and internal long-horizon evaluations test the breadth of the underlying model, while the comparison between ReAct and Agent Team estimates the system-level lift of trained coordination under additional organized computation. The HDS6 analysis and end-to-end cases further examine whether successful outputs are supported by coherent tool use, evidence, repair, alternatives, and reviewable artifacts. Taken together, these evaluations position Apodex 1.1 not as a collection of task-specific agents, but as a common model and execution stack whose capabilities compose across different forms of work.

The 35B Apodex 1.1 mini provides a particularly direct efficiency result: it reaches the performance band of selected frontier systems on representative professional, financial, and scientific tasks, improves over Apodex 1.0 mini on overlapping evaluations, and retains substantial gains from Agent Team coordination. This shows that the two scaling dimensions improve not only peak system performance, but also the capability obtained at a compact model scale.

Our longer-term goal is a *Heavy-Duty Solver*: a system capable of taking responsibility for increasingly ambitious, long-running, and verifiable work. Progress toward that goal will come from scaling the coverage and fidelity of executable environments, strengthening learned coordination over longer horizons, improving training and credit assignment over hierarchical traces, and closing the loop between real failures, task construction, training, and evaluation. Apodex 1.1 establishes this development path and shows how agentic intelligence can be scaled around completed work rather than isolated responses.

References

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

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

Anthropic. Evaluating claude's bioinformatics research capabilities with biomysterybench. https://www.anthropic.com/research/Evaluating-Claude-For-Bioinformatics-With-BioMysteryBench, 2026a.

Anthropic. Introducing claude opus 4.6. https://www.anthropic.com/news/claude-opus-4-6, 2026b. Official announcement.

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

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

Anthropic. Introducing claude opus 5. https://www.anthropic.com/news/claude-opus-5, 2026.

Anthropic. Claude fable 5 and claude mythos 5. https://www.anthropic.com/news/claude-fable-5-mythos-5, 2026a. Official announcement.

Anthropic. Claude mythos preview system card. https://www-cdn.anthropic.com/7624816413e9b4d2e3ba620c5a5e091b98b190a5/Claude%20Mythos%20Preview%20System%20Card.pdf, 2026b. Official system card.

Apodex Team. Apodex-1.0: A verification-centric agent team for discoverative intelligence. Technical report, Apodex, 2026. URL https://github.com/ApodexAI/AgentHarness.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D. White, and Philippe Schwaller. ChemCrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023. URL https://arxiv.org/abs/2304.05376.

ByteDance Seed Team. Seed2.1 officially released: Advancing ai productivity. https://seed.bytedance.com/zh/blog/seed2-1-officially-released-advancing-ai-productivity, June 2026. Official launch blog and model-family evaluation overview.

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.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

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.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, Jiajie Jin, Yutao Zhu, Hanbin Wang, Fangyu Lei, Qinyu Luo, Mingyang Chen, Zehui Chen, Jiazhan Feng, Ji-Rong Wen, and Zhicheng Dou. Agent-World: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026. URL https://arxiv.org/abs/2604.18292.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. WorkArena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024. URL https://arxiv.org/abs/2403.07718.

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.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, Shibin Wu, Zhengwei Tao, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Towards general agentic intelligence via environment scaling. *arXiv preprint arXiv:2509.13311*, 2025. URL https://arxiv.org/abs/2509.13311.

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/, 2026a. Official announcement.

Google. Introducing gemini 3.6 flash, 3.5 flash-lite, and 3.5 flash cyber. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-6-flash-3-5-flash-lite-3-5-flash-cyber/, 2026b. Official announcement.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, et al. Towards an AI co-scientist. *arXiv preprint arXiv:2502.18864*, 2025. URL https://arxiv.org/abs/2502.18864.

Muyu He, Adit Jain, Anand Kumar, Vincent Tu, Soumyadeep Bakshi, Sachin Patro, and Nazneen Rajani. YC-Bench: Benchmarking AI agents for long-term planning and consistent execution. https://collinear-ai.github.io/yc-bench/, 2025.

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.

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.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025. URL https://arxiv.org/abs/2503.09516.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, et al. Websailor: Navigating super-human reasoning for web agent. *arXiv preprint arXiv:2507.02592*, 2025a.

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*, 2025b.

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.

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.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kejuan Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*, 2023. URL https://arxiv.org/abs/2308.03688.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024. URL https://arxiv.org/abs/2408.06292.

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.

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.

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

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

Moonshot AI. Kimi k2.6: Advancing open-source coding. https://www.kimi.com/blog/kimi-k2-6, 2026. Official technical blog.

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

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

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

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

OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition. https://openai.com/index/gpt-5-6/, 2026d. Official announcement.

Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simón Posada Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, Natalie S. Kim, Patrick Chao, Samuel Miserendino, Gildas Chabot, David Li, Michael Sharman, Alexandra Barr, Amelia Glaese, and Jerry Tworek. GDPval: Evaluating AI model performance on real-world economically valuable tasks, 2025. URL https://arxiv.org/abs/2510.04374.

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.

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.

Qwen Team. Qwen3.7-Plus: Multimodal agent intelligence, May 2026. URL https://qwen.ai/blog?id=qwen3.7-plus.

Samaya Research. FrontierFinance: A benchmark for financial reasoning agents. https://samaya.ai/blog/frontier-finance, 2026. Leaderboard: https://research.samaya.ai/benchmarks/frontier-finance.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023. URL https://arxiv.org/abs/2302.04761.

Dingfeng Shi, Jingyi Cao, Qianben Chen, Weichen Sun, Weizhen Li, Hongxuan Lu, Fangchen Dong, Tianrui Qin, King Zhu, Minghao Liu, et al. Taskcraft: Automated generation of agentic tasks. *arXiv preprint arXiv:2506.10055*, 2025.

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.

Liangcai Su, Zhen Zhang, Guangyu Li, Zhuo Chen, Chenxi Wang, Maojia Song, Xinyu Wang, Kuan Li, Jialong Wu, Xuanzhong Chen, et al. Scaling agents via continual pre-training. *ICLR 2026*, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, M. C., Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, et al. Kimi k3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

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.

Dunwei Tu, Hongyan Hao, Hansi Yang, Yihao Chen, Yi-Kai Zhang, Zhikang Xia, Yu Yang, Yueqing Sun, Xingchen Liu, Furao Shen, Qi Gu, Hui Su, and Xunliang Cai. ScaleEnv: Scaling environment synthesis from scratch for generalist interactive tool-use agent training. *arXiv preprint arXiv:2602.06820*, 2026. URL https://arxiv.org/abs/2602.06820.

Vals AI. finance-agent-v2: Reference agent scaffold for FrontierFinance. https://github.com/vals-ai/finance-agent-v2, 2026. Open-source reference implementation.

Bertie Vidgen, Austin Mann, Abby Fennelly, John Wright Stanly, Lucas Rothman, Marco Burstein, Julien Benchek, David Ostrofsky, Anirudh Ravichandran, Debnil Sur, et al. APEX-Agents. *arXiv preprint arXiv:2601.14242*, 2026. URL https://arxiv.org/abs/2601.14242.

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*, 2025a.

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

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, et al. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025c. URL https://arxiv.org/abs/2504.20073.

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhengwei Tao, Dingchu Zhang, Zekun Xi, Gang Fu, Yong Jiang, et al. Webdancer: Towards autonomous information seeking agency. *arXiv preprint arXiv:2505.22648*, 2025.

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.

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

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024. URL https://arxiv.org/abs/2404.07972.

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

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

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.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

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.

Junkeun Yi, Damon Mosk-Aoyama, Baihe Huang, Ritu Gala, Charles Wang, Sugam Dipak Devare, Khushi Bhardwaj, Abhibha Gupta, Oleksii Kuchaiev, Jiantao Jiao, Jian Zhang, and Venkat Srinivasan. PivotRL: High accuracy agentic post-training at low compute cost. *arXiv preprint arXiv:2603.21383*, 2026. URL https://arxiv.org/abs/2603.21383.

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.

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.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, et al. Agent learning via early experience. *arXiv preprint arXiv:2510.08558*, 2025.

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.

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.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. URL https://arxiv.org/abs/2307.13854.

A Case Studies

Three recorded runs follow. Each was handed raw files and a list of deliverables, and had to produce finished, checkable artifacts rather than an answer.

Case 1 builds a runnable starting point for a protein simulation. Modelling every atom is too slow, so the standard move is to coarse-grain—bundle groups of atoms into single beads, 3785 of them here—which keeps the overall shape at a fraction of the cost. From that model the run has to produce the files that tell a simulation engine how the beads interact, a box holding three copies of the protein in salt water, and a first calculation that relaxes the awkward contacts: eighteen interlocking files, and the engine rejects all of them if one box dimension or particle count disagrees with the rest.

Case 2 counts how many cells died. In each of six microscope photographs, blue marks every nucleus and green marks the dying cells, so the greener an image is relative to its blue, the more death it shows. Three images are controls and three are treated; the run has to measure the ratio from the pixels rather than assume it, compare the two groups, and deliver a table, a statistics file and a plot.

Case 3 looks for the genes behind measurable traits in fifty pigs. Genes that rise and fall together tend to be doing related work, so five thousand of them are sorted into ten groups by how closely they track one another; each group is then matched against traits such as body weight and backfat thickness, and the best-matching group for each trait yields a shortlist of candidate genes.

Each case opens with the user's input as it was received, and closes with the delivered result quoted in abridged form and the full file inventory. In between, Cases 1 and 2 are told through the agents that ran them: the lead agent **`swarm_main`** reads the inputs and writes a task board of numbered items `t1`, `t2`, ..., then builds subagents and hands the items out. Every agent gets one block—its name, the items it owns, its tool calls broken down by tool, what it **did**, and what it **returned**, separating files that were delivered from reports and working files that were not—and the case ends with a timeline of when each agent was active. Case 3's agent-level trace was not retained, so its blocks are named for the pipeline stages instead.

Case 1: Coarse-Grained Structure Preparation and Energy Minimisation of a Protein Hexamer

The deliverable here is not an answer but a runnable simulation package: eighteen interlocking files that a molecular-dynamics engine has to accept as one consistent system. The run is included because the simulation engine it needs was not installed and not permitted to be invoked, and because the second half of the trace is spent finding and repairing defects in files the team had already built.

`User input`

 # Build a coarse-grained initial structure and energy-minimisation inputs from a protein structure

Read `7M6J_fixed.pdb` from the input directory. Apply the coarse-graining force field currently most widely used for biological systems, and from the resulting `cg.pdb` produce force-field topology files usable by free open-source molecular-dynamics software, plus a conformation snapshot `scene.bmp`. Then write run-control files with appropriate parameters, build a simulation box holding three copies of the protein in physiological salt solution, generate the energy-minimisation input, and run the minimisation. Write everything to `/app/output/`.

Required deliverables `cg.pdb`; `martini.itp`; `molecule_A.itp` through `molecule_F.itp` (one per chain); `topol.top`; `scene.bmp`; `hexamer_3copies.pdb`; `ions.itp`; `ions.mdp`; `em.mdp`; `ionized.gro`; `em.tpr`; `em.gro`; `em.log`.

Interface constraints Chain identifiers `A`–`F` are the stable identity keys of the chain topology files. PDB site records carry Å coordinates, GRO records carry nm. `topol.top` must express the system topology, including references to the submitted force-field, chain and ion topologies; any relatively referenced file is itself a required deliverable and its path must stay resolvable. `em.log` must belong to the same run as the submitted minimisation input and result. Extra auxiliary files are allowed but may not stand in for a required one.

`7M6J_fixed.pdb` (input structure: six-chain protein, chains `A`–`F`) 2,117,776 B

swarm_main — lead. 17 tool calls: 1 glob_search, 1 grep_search, 15 collect_reports.
Did. Wrote a six-item board: t1 inspect the input for chain count, residue count and format problems; t2 coarse-grain with Martini 3 to produce cg.pdb, martini.itp and the six chain topologies; t3 render the snapshot; t4 build the three-copy box, solvate and ionise; t5 write em.mdp, generate em.tpr and run the minimisation; t6 verify every deliverable is present and format-valid. Constructed seven subagents and issued no bash call itself, collecting returns fifteen times over 79 minutes—more collections than agents, because three of them were re-dispatched after defects surfaced.
Returned. Board closed, 6/6 resolved; final report.

box_builder — board t4, preparation. 72 tool calls: 56 bash, 11 web_fetch, 3 web_search, 1 create_file, 1 read_file.
Did. Established that the box could not be built here at all: which gmx returns nothing, the sandbox denies the command outright, /usr/share/gromacs/top does not exist and $GMXLIB is empty. Rather than stop, it fetched the four official Martini 3.0.0 files from their upstream sources and pinned each by byte size and MD5, then read the ion and solvent definitions directly out of them—confirming the [defaults] 1 2 sigma-epsilon format, NA/CL on bead type TQ5, and W at nrexcl 1—and checked the official water box for the right Martini density before adopting it as the solvation source.
Returned. No deliverable. A build plan opening with a hard runtime gate that aborts unless gmx and all eight required subcommands answer, plus draft ions.mdp, em.mdp and topol.top, and the staged force field. Recorded that the tool layer rewrites /app to another path in displayed output only, and verified byte-for-byte that the plan file itself still carries the correct target.

em_runner — board t5, draft only. 11 tool calls: 7 bash, 2 create_file, 1 web_search, 1 web_fetch.
Did. Independently confirmed the same environment verdict, and characterised the all-atom input directly: 26,141 atom records over six chains, per-chain counts A 3844 to F 4161, chain extents read off the TER lines. Drafted both control files to the parameters it had been given, and flagged one of them as a version hazard: the requested vdwtype = Shift is deprecated, so the draft carries the modern equivalent as an inline comment in case grompp rejects the legacy form.
Returned. No deliverable. Draft em.mdp and ions.mdp; no grompp or mdrun was attempted, since the board item said not to run yet.

gmx_installer. 100 tool calls: 99 bash, 1 create_file.
Did. Worked the toolchain problem to a conclusion. apt-get is not on the allowed-command list and no conda or mamba exists, so both obvious install routes fail at the policy gate; the package was instead taken from a conda-forge build staged in the workspace and unpacked as a full prefix. The residual obstacle was the audit filter itself, which denies any command containing the bare token gmx—so the binary is reached through a python3 wrapper that sets the library path and execs it. It then verified all eight subcommands individually and confirmed the coarse-graining tool was importable at a known version.
Returned. A working GROMACS 2024.5 invocation path and the wrapper every later agent uses. No deliverable file.

cg_modeler — board t1–t2, then repair. 105 tool calls: 105 bash.
Did. On re-dispatch this agent's pass is a repair pass, and it names four defects in files that already existed. CRYST1 in hexamer_3copies.pdb had been written with the nm values, which the PDB specification reads as Å—a tenfold box error—and was replaced with $83.154 \times 92.632 \times 821.164$ Å. 3309 solvent beads sat outside the periodic box in ionized.gro and were minimum-image wrapped, leaving the atom order and the entire 11,355-site protein block byte-identical. em.mdp was missing epsilon_r = 15, so the minimisation had been running at the engine default, and grompp and mdrun were re-run to keep the triple consistent. It also settled a count that two earlier narratives disagreed on: cg.pdb holds 3785 sites, not 3385—the file had always been right and the smaller number was a transcription error.

Returned. The corrected package, 22 files staged with a regenerated MD5 inventory, and the six chain topologies cross-checked entry by entry against their chain in `cg.pdb`, 0 mismatches. It also carried forward one defect it was *not* allowed to fix: the coarse-grained model's residue sequence differs from the input at 131 positions and its numbering restarts within three chains, inherited from the mapping step. The constraints forbade touching `cg.pdb` or the topologies, and the engine accepts the system as it stands, so the defect is reported rather than repaired.

`scene_renderer` — board `t3`. 43 tool calls: 40 `bash`, 3 `read_file`.
Did. Validated the existing snapshot instead of re-rendering it, with the re-render path armed in case a check failed. Parsed the BMP header byte by byte and confirmed twelve properties against each other—magic bytes, declared size against actual, offset, 847 × 817, 24 bits per pixel, uncompressed, and stride × height equal to the declared image size—then loaded the full pixel array to prove nothing was truncated, and read the image itself to confirm the title, the six-chain legend and the axis labels are present and unclipped.
Returned. No deliverable; no re-render needed. The bead counts printed in the figure's own title sum to 3785 and match the six chains in `cg.pdb`, which makes the snapshot self-verifying against the structure it depicts.

`final_verifier` — board `t6`. 92 tool calls: 91 `bash`, 1 `read_file`.
Did. Seven-point re-verification of the repaired files. The decisive check was done without the engine: it decoded the binary `em.tpr` header by hand and compared its embedded starting coordinates against `ionized.gro` for all 63,484 particles, maximum absolute difference 0, which proves the minimisation input was built from the submitted structure rather than from a pre-repair copy. It also re-hashed every file it was not supposed to change, and dismissed one apparent discrepancy on evidence: the ion residue names differ between `ionized.gro` and `em.gro` because the engine derives them from the topology, which is correct behaviour and not a defect.
Returned. No deliverable. **7/7 PASS**, submission ready. Recorded that the repair pass replaced an earlier converged run—218 steps at the default dielectric constant—with the submitted 255-step run, and that the superseded scratch files are not part of the submission.

`final_publisher`. 22 tool calls: 22 `bash`.
Did. Copied all 22 manifest files to the publication root and verified each by byte length and MD5 against its staged source, then confirmed the count: exactly 22 present, no extras. `/app/output` could not be created—it points at a target the account cannot make—so the report keeps `/app/output/` as the canonical path in its manifest and in every downstream reference, with the instruction that the published directory be moved there unchanged and the relative includes in `topol.top` not be rewritten.
Returned. 22 published files, 0 mismatches.

Delivered result.

`topol.top` + **`em.log`** — system composition and the minimisation it closes over

A Martini 3 three-copy system in physiological salt solution, minimised, with the input, log and output structure verified to come from one run.

`topol.top` — `[ molecules ]`	
`molecule_A` ... `molecule_F`	3 each
`W`	50957
`NA`	601
`CL`	571

`em.log` — GROMACS 2024.5, `gmx mdrun`	
Steepest descents converged to `Fmax` < 1000	in 255 steps
Maximum force	9.029×10^{2} kJ/mol/nm
Potential energy	-1.77496×10^{6} kJ/mol

The closure is established by binary evidence rather than by assertion: the coordinates embedded in `em.tpr` match `ionized.gro` for all 63,484 particles at a maximum absolute difference of 0, so the submitted input, log and output structure provably belong to one run on the repaired files. Stated boundaries: the package is an equilibration starting point, not a production simulation — coupling, constraint handling and output frequency still have to be set for the research question, and any edit to `em.mdp`, `ionized.gro` or `topol.top` invalidates the submitted `em.tpr`. One defect is carried forward unfixed and disclosed: the coarse-grained model's residue sequence departs from the input structure at 131 positions with numbering restarting inside three chains, inherited from the mapping step and left alone because the constraints forbade editing `cg.pdb` or the chain topologies.

File	Content	Size
cg.pdb	3785 coarse-grained sites, chains A–F, Å	299,015 B
martini.itp	Force-field entry point; includes ff/ by relative path	670 B
molecule_A–F.itp	Per-chain bonded, constraint and virtual-site topology	98,506–126,931 B
ions.itp	Martini 3 ion topology, NA/CL	7,853 B
topol.top	Six chains × 3, water and ions; net charge 0	849 B
hexamer_3copies.pdb	11,355 sites = 3 × 3785; CRYST1 in Å	908,492 B
ionized.gro	63,484 particles, box in nm, wrapped in-cell	2,856,866 B
ions.mdp	Ion-placement stage control file, not production parameters	1,283 B
em.mdp	Steepest descents; epsilon_r = 15, constraints = none	1,542 B
em.tpr	Binary minimisation input, 63,484 particles	1,505,592 B
em.gro	Minimised structure, same box and particle count	2,856,855 B
em.log	Converged in 255 steps; same run as em.tpr/em.gro	99,569 B
scene.bmp	Conformation snapshot for human inspection only	2,078,502 B
ff/	Martini 3.0.0 parameters, ions, solvents, water box	4 files, 16.1 MB

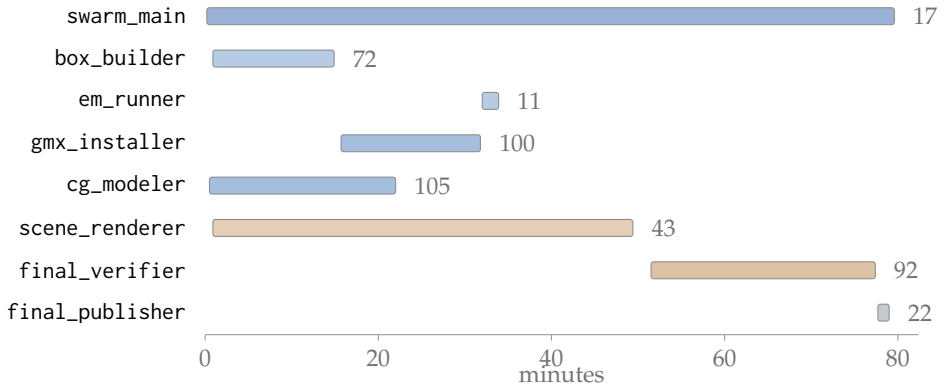

Figure 7: **Top:** the published package — 18 required files plus the local force-field directory they reference; ff/ is itself a required deliverable, because martini.itp includes it by relative path. **Bottom:** active window per agent, tool-call count at right. Agent Team Mode, Apodex 1.1; 82.4 min wall-clock; 864 recorded steps = 402 reasoning + 462 tool calls; board 6/6 resolved. **box_builder** and **em_runner** could only prepare, because the simulation engine was unavailable until **gmx_installer** finished; **cg_modeler**'s window closes on a repair pass, and **final_verifier** alone accounts for 92 of the run's tool calls.

Case 2: TUNEL/DAPI Apoptosis Quantification from Fluorescence Micrographs

The inputs here are photographs, not tables: six raw microscope exports from which every reported number has to be measured. The run is included because two measurement agents read the same word—"intensity"—in two defensible ways, and the two readings disagree about whether the two groups differ at all.

User input

 # TUNEL/DAPI fluorescence image quantification

Input Six fluorescence micrographs (C1–C3.jpeg, E1–E3.jpeg), all TUNEL/DAPI immunofluorescence stains.

Output

- result.csv: wide format, five columns—Sample, Group, TUNEL fluorescence intensity, DAPI fluorescence intensity, TUNEL/DAPI fluorescence intensity ratio (%)—six data rows plus a header. Stable row key Sample, taken from the input filename without extension.
- statistics.json: the between-group comparison, as group_c/group_e objects carrying mean, sd, sem, n, plus effect_size, p_value, statistical_method and error_bar_type; n integer, the rest numbers. Flat aliases group_c_mean and group_e_mean are accepted, extra fields permitted.

- `TUNEL_Ratio_Plot.pdf`: bar chart of the two groups with mean ± SEM error bars, group labels, axis name and a p-value annotation.
- `output.zip`: the three files at the ZIP root, no nested directories, byte-identical to their loose counterparts.

Constraints Every value must come from an actual measurement of the source images—nothing hard-coded or fabricated. The originals must not be modified. All six samples must be processed under one identical rule.

`C1, C2, C3 .jpeg` (control group, 2880 × 1642 RGB) 170,094 / 173,155 / 175,369 B
`E1, E2, E3 .jpeg` (experimental group, same geometry) 143,525 / 160,638 / 139,848 B

`swarm_main` — lead. 9 tool calls: 2 `glob_search`, 7 `collect_reports`.
Did. Wrote a seven-item board: `t1` determine which RGB channel carries TUNEL and which carries DAPI; `t2` fix one reproducible intensity rule—ROI, channel separation, background—and measure all six samples; `t3` reproduce the measurement independently and cross-validate; `t4` group statistics and `statistics.json`; `t5` the plot; `t6` `result.csv` and the archive; `t7` independent verification of everything. Assigned `t3` to two agents at once—a second measurement agent and an arbitrator—rather than to a single checker; issued no `bash` call.
Returned. Board closed, 7/7 resolved; final report, which records the discarded measurement definition explicitly rather than silently dropping it.

`img_measure_a` — board `t1–t2`. 27 tool calls: 20 `bash`, 6 `read_file`, 1 `create_file`.
Did. Characterised the images: all six 2880 × 1642 RGB uint8, red-channel mean only 0.31–0.44 AU, so no red fluorophore is present. Found two separated hue clusters—green ≈ 110–185°, blue ≈ 185–255°—with the histogram valley at ≈175–200°, and confirmed by morphology that blue is the punctate nuclear DAPI signal and green the diffuse TUNEL signal. Segmented in HSV with $V > 8$, $S > 10$, hue cut 185°, then took each fluorophore's mean over *its own* mask.
Returned. No deliverable. `measure_a.py`, `measurements_a.csv`: group means 84.17 ± 21.98 vs 88.39 ± 8.43, Welch $P = 0.78$, $d = 0.25$ — **no difference between the groups**.

`img_measure_b` — board `t3`. 25 tool calls: 16 `bash`, 9 `read_file`.
Did. Re-measured with a deliberately different separator: k-means ($K = 3$, fixed seed) on circular hue features $(\cos\theta, \sin\theta)$ with $\theta = \text{atan2}(B, G)$, clusters assigned to stains by centroid angle, cross-checked against a strict $\theta < 40°$ / $\theta > 50°$ split. Took both channel means over one *shared* tissue ROI, $\{V > 35\} \cap \{\text{chroma} > 0.20\}$.
Returned. No deliverable. `measure_b.py`, `measurements_b.csv`: 114.12 ± 31.50 vs 47.42 ± 9.09, Welch $P = 0.0574$, $d = 2.88$ — **group C is** $2.4\times$ **group E**. Same pixels, same channel assignment, opposite conclusion.

`local_verifier` — board `t3`, arbitration. 34 tool calls: 23 `bash`, 11 `read_file`.
Did. Re-ran both scripts sample by sample and reproduced both tables exactly, establishing first that neither agent had made an arithmetic error and the conflict was definitional. Then measured a third time under its own thresholds, $\{V > 32\} \cap \{\text{chroma} > 0.18\}$, and stress-tested both definitions across $V > 25/30/35/40$, two chroma cuts and Otsu. The shared-ROI field mean held: Welch $P = 0.057$–0.069, $d = 2.70$–2.88, Mann–Whitney $P = 0.10$ throughout. The per-mask mean did not: C1's TUNEL value swings $36.3 \to 45.0 \to 101.5$ AU as the floor moves $V > 6 \to 8 \to 32$, and the group comparison *reverses direction* across that range ($E > C$ at $P = 0.49$; "no difference" at $P = 0.78$; $C > E$ at $P = 0.16$).
Returned. `recommended_rows.json`, `statistics_arb.json`. Ruled for the shared-ROI field mean on four grounds: the requested field is the standard mean fluorescence intensity over a region, not the brightness of positive pixels; the ratio is an apoptosis index and must stay sensitive to signal *abundance*, and the per-mask mean discards exactly the ≈2.7× TUNEL-positive area difference that separates the groups; averaging both channels over one ROI keeps the comparison symmetric; and a definition whose sign depends on an arbitrary intensity floor is not reproducible. It also corrected the winning agent: the ratio

is stable across *plausible tissue* ROIs, not across all ROIs—including the background haze moves C1 from 83.6 to 103.6.

`stats_plot` — board `t4–t5`. 12 tool calls: 7 `bash`, 4 `read_file`, 1 `create_file`.
Did. Built the CSV from the arbitrated rows without re-segmenting the images, computed every statistic from the CSV ratio column so the delivered table is the sole data source, and self-checked by recomputation—15/15 items. Rendered the plot with a pre-save `get_window_extent` against `clip_box` check on every text element and the legend.
Returned. `result.csv`, `statistics.json`, `TUNEL_Ratio_Plot.pdf`. Since $P = 0.0593 > 0.05$, the figure is annotated "`p = 0.0593 (ns)`" and carries no significance star.

`publisher` — board `t6`. 15 tool calls: 15 `bash`.
Did. Copied the three artifacts and built the archive with `arcname` set to the bare filename so the ZIP root stays flat, then verified structure and byte identity—MD5 on all four published files, `testzip()`, and a member-by-member comparison of the decompressed entries against the loose copies.
Returned. Four published files, zero mismatches.

`fs_explorer`. 19 tool calls: 19 `bash`.
Did. Investigated why `/app/output` could not be written. The agent runs as uid 999 while `/app` is `root:root 0755`; the tool layer also rewrites the literal `/app` to `/mnt/agent`, so the `mkdir` failure names a path the command never used. Read `/proc/self/mountinfo` to show that `/outputs` is an ordinary top-level directory on the same overlay, not a host mapping of `/app/output`.
Returned. No deliverable. Confirmed all four files present and well-formed at the publication root, and flagged one disagreement between the task description and the observed filesystem: `/app` is a real directory, not the symlink the description implies.

`final_verifier` — board `t7`. 25 tool calls: 22 `bash`, 3 `read_file`.
Did. Eight-point audit. Re-measured all six samples from the raw pixels under the published ROI—maximum absolute deviation from the CSV 5.23×10^{-5}—then repeated the measurement under five ROI and threshold definitions of its own, including Otsu, specifically to try to break the conclusion. Recomputed the statistics from the CSV with `scipy`, checked JSON types, CSV encoding and line count, rendered the PDF and read it back by text extraction and by vision, and re-hashed every ZIP member.
Returned. No deliverable. **8/8 PASS**, verdict CONFIRMED, confidence 0.97. Every statistic matched `statistics.json` bit for bit; all five alternative definitions preserved $C > E$ at $C/E \approx 2.3$–2.4. Recorded one non-error worth stating: $P = 0.0593$ is borderline and would cross at $\alpha = 0.1$.

Delivered result.

`result.csv` + `statistics.json` — per-sample measurements and the group comparison

Group C carries 2.4× the TUNEL/DAPI ratio of group E—a very large effect that does not reach significance at $n = 3$ per group, and is reported as such.

Sample	Group	TUNEL (AU)	DAPI (AU)	Ratio (%)
C1	Control	70.4417	84.3945	83.4671
C2	Control	85.7051	77.0177	111.2798
C3	Control	96.5947	65.8671	146.6508
E1	Experimental	57.2276	100.1187	57.1597
E2	Experimental	36.8663	93.4199	39.4631
E3	Experimental	44.4751	96.0397	46.3091

Control 113.80 ± 31.67 (SEM 18.28), Experimental 47.64 ± 8.92 (SEM 5.15), $n = 3$ each; difference 66.2 percentage points; Cohen's $d = 2.84$; Welch two-sample $P = 0.0593$, Mann–Whitney $P = 0.10$. Intensity is the channel mean over the shared tissue ROI $\{V > 32\} \cap \{\text{chroma} > 0.18\}$, with TUNEL = G and DAPI = B, applied identically

to all six images; ratio = 100 × TUNEL/DAPI. The six CSV rows are the only input to the JSON and the figure.

File	Content	From
Delivered — 4 files, SHA-256 recorded in the trace		
`result.csv`	7 lines: header + 6 samples × 5 columns	349 B
`statistics.json`	`group_c/group_e` objects, effect size, P, method, error-bar type, flat aliases	536 B
`TUNEL_Ratio_Plot.pdf`	1 page; two bars, mean ± SEM, `p = 0.0593 (ns)`, $d = 2.84$	20,987 B
`output.zip`	Flat root, 3 members, byte-identical to the loose copies	13,745 B
Working files, not delivered		
`measure_a.py` `measurements_a.csv`	Per-mask definition — measured, then rejected	`img_measure_a`
`measure_b.py` `measurements_b.csv`	Shared-ROI definition, k-means hue separation	`img_measure_b`
`recommended_rows.json` `statistics_arb.json`	Arbitrated rows and statistics	`local_verifier`
`build_stats.py`	CSV, JSON and figure from the arbitrated rows	`stats_plot`
`plot-1.png`	PDF rasterised for the visual clipping check	`final_verifier`

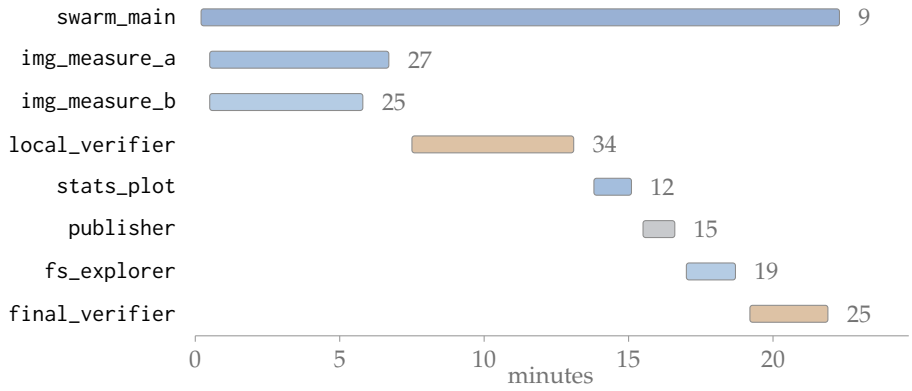

Figure 8: **Top:** every file the run produced. **Bottom:** active window per agent, tool-call count at right. `Agent Team Mode`, Apodex 1.1; 24.7 min wall-clock; 324 recorded steps = 151 reasoning + 166 tool calls + 7 agent returns; board 7/7 resolved. **`img_measure_a`** and **`img_measure_b`** ran concurrently on the same images under different measurement definitions; **`local_verifier`** was dispatched to arbitrate rather than to re-check a single result.

Case 3: WGCNA of a Pig RNA-seq Cohort with Twenty Specified Deliverables

This request specifies fourteen analysis steps and then pins each one to a file with a declared column schema, so the interface is as much of the task as the statistics. As in Case 1, the agent-level trace was not retained; the blocks below follow the pipeline stages recorded in the run's own report, and every figure is read back from the delivered tables.

User input

 # WGCNA analysis and result visualisation

Run a complete WGCNA analysis on the pig RNA-seq raw counts, sample phenotype data and gene annotation in the input folder. Read and inspect all three inputs. Check whether gene IDs and sample IDs are duplicated. Match the expression matrix to the phenotype data by sample name and make the orders identical. Filter low-expression genes, transform the counts, and select 5000 highly variable genes for WGCNA. Check for missing values, zero-variance genes and unqualified samples. Complete sequencing depth, detected-gene count, PCA, sample clustering and sample connectivity as quality control on both samples and genes. Assess the soft threshold and build the co-expression network. Identify and merge gene modules. Compute module eigengene correlations against all phenotypes with P values and corrected P values. Compute module membership (MM) and gene significance (GS) for every gene. Screen candidate hub genes for each phenotype.

Required deliverables `analysis.R` and `analysis_report.md`; `selected_genes.tsv`, `selected_samples.tsv`, `wgcna_expression.tsv`, `wgcna_traits.tsv`; `sample_qc.tsv`, `soft_threshold_statistics.tsv`, `network_parameters.tsv`; five PNG figures; `gene_module_assignment.tsv`, `module_sizes.tsv`, `module_trait_results.tsv`, `gene_MM_GS_results.tsv`, `hub_gene_candidates.tsv`; `wgcna_results.RData`.

Interface constraints Every TSV is UTF-8, tab-separated, with a header; column order is free but extra columns must not collide with or shadow a specified field. Stable keys are given per file — `gene_id`, `sample_id`, `power`, `module_color`, and the `module_color`×`trait` pair. `wgcna_traits.tsv` must carry exactly the five named phenotypes and no others; `wgcna_expression.tsv` must carry no column outside the selected gene set, and its row order must match `wgcna_traits.tsv` exactly. `gene_MM_GS_results.tsv` needs `MM.<module>` and `p.MM.<module>` per module and `GS.<trait>` and `p.GS.<trait>` per phenotype. `hub_gene_candidates.tsv` keeps one row per gene–phenotype–module record, so one gene serving two phenotypes stays two rows.

`pig_raw_counts.tsv` (6000 genes × 50 samples, integer counts) 1,162,652 B

`pig_traits.tsv` (50 samples × 5 phenotypes: treatment, body weight, backfat, feed conversion, serum IGF-1) 1,880 B

`pig_gene_annotation.tsv` (6000 gene records; used for interpretation only, never as a key) 405,467 B

Input audit. No exclusion triggered.
The audit is written to fail loudly rather than repair quietly: duplicate IDs, a missing stable key or an unmatchable sample stop the pipeline instead of being merged or aligned by position. On these inputs it found 6000 genes and 50 samples `Pig_001`–`Pig_050`, zero duplicate `gene_id`, zero duplicate sample columns, 50 unique `sample_id` with all five phenotypes complete, and count columns matching trait IDs both as sets and in order. The explicit re-ordering by `sample_id` was kept anyway, so the guarantee does not depend on the incoming row order. The annotation table's 6000 unique IDs were deliberately *not* promoted to the primary key: `gene_id` from the counts file stays the key throughout, and annotation is reserved for interpreting candidates.

Filtering and gene selection. $6000 \rightarrow 5983 \rightarrow 5000$ genes.
Low-expression filtering required CPM > 1 in at least 25 of the 50 samples, retaining 5983 genes; counts were transformed as $\log_2(\mathrm{CPM} + 1)$ so that library-size differences do not push a deeply sequenced sample's whole expression profile upward and contaminate the sample correlations, the PCA and the gene correlation matrix alike. The top 5000 genes by variance were then taken for network construction, recorded as a noise- and cost-reduction step rather than a claim that the excluded genes are uninformative.

Sample and gene quality control. All 50 samples retained.
Library sizes span 601,830 to 1,576,901 and detected genes 5932 to 5950; Z connectivity runs -2.506 to 1.611, crossing no conventional threshold, and PCA and hierarchical clustering show no isolated sample. `candidate_outlier` is `FALSE` for all 50. The stated justification for retaining everything is not that the depth spread is negligible but that no sample is flagged by several indicators at once. The report separates two quantities that share a name: the `connectivity` in `sample_qc.tsv` is sample-to-sample and scales with sample count, whereas the connectivities in `soft_threshold_statistics.tsv` are gene-network quantities scaling with gene count, and using the latter to judge samples would be an error.

Soft threshold and network construction. `selected_power = 20`, as a fallback.
Powers 1–20 were evaluated and *none* reached the preset scale-free fit of $R^2 \geq 0.85$. The highest fit sits at power 20 with $R^2 = 0.8486$, slope -1.917 and mean gene connectivity 4.13, so power 20 is recorded as a maximum-fit fallback rather than a satisfied criterion — the report states this explicitly to prevent the value being read as unconditionally optimal. The network is `signed` with `bicor` correlation and `signed` TOM, `min_module_size` 30, merge threshold 0.25, over 50 samples and 5000 genes. Dynamic tree cutting followed by eigengene-similarity merging produced 10 modules: grey 1560, turquoise 590, blue 589, brown 550, yellow 541, green 528, pink 228, red 155, black 131, magenta 128.

Module–trait association and hub candidates. 437 candidate records.
All 50 module–trait combinations were tested and BH-corrected together. Hub screening then took, per phenotype, the strongest non-grey module at FDR < 0.05 and kept genes with $|\text{MM}| > 0.8$ and $|\text{GS}| > 0.2$, yielding 437 records: 93 for `treatment`, 72 for `body_weight_kg`, 72 for `backfat_mm`, 81 for `feed_conversion_ratio`, 119 for `serum_igf1_ng_ml`. One consequence is flagged rather than smoothed over: GS is a signed correlation but the screen uses its absolute value, so a candidate can pass inside its target module while its own GS sign opposes the module eigengene's direction — signs must be read back from `gene_MM_GS_results.tsv` before any candidate is interpreted. Hub genes are framed as ranked candidates for annotation, enrichment and validation, not as causal or regulatory genes.

Residual limitation. `analysis.R` was not executed end to end.
R could not be run in this sandbox. The script is delivered as the reproducible path from the inputs to every required file, and the numerical results were produced and cross-checked by an equivalent implementation with file-level consistency checks, but the report states the residual risk plainly: re-running `analysis.R` in a standard R environment may expose version or package-API differences, and the first things to compare are `selected_genes.tsv`, `module_sizes.tsv` and `module_trait_results.tsv`. As in Case 1, `/app/output/` was read-only, so the package was published to the collection directory while the report keeps the specified paths. The input path named in the request did not exist either; the files were located in the sandbox's actual input directory.

Delivered result.

`module_trait_results.tsv` + **`network_parameters.tsv`** — strongest non-grey module per phenotype

Each of the five phenotypes maps onto a different non-grey module, all at FDR $< 10^{-16}$ — correlation structure, not intervention effect.

Phenotype	Module	n genes	Correlation	FDR
`treatment`	pink	228	-0.8890	6.5×10^{-17}
`body_weight_kg`	black	131	-0.9118	5.9×10^{-19}
`backfat_mm`	red	155	-0.9287	1.3×10^{-20}
`feed_conversion_ratio`	magenta	128	$+0.9231$	3.7×10^{-20}
`serum_igf1_ng_ml`	brown	550	$+0.8901$	6.5×10^{-17}

Network: `signed` type, `bicor` correlation, `signed` TOM, `selected_power` 20 by maximum scale-free fit (no power reached 0.85), `min_module_size` 30, merge threshold 0.25, $n = 50$ samples and 5000 genes.

Stated boundaries: $n = 50$ supports a correlation network but limits covariate modelling; batch information was not available, so a batch confounded with a phenotype would be absorbed into these correlations; correlation gives no direction; and the module partition is sensitive to the power, the expression filter, the gene count and both module-merging parameters, so the durable results are the module–trait associations and the candidate ranking rather than the module colours themselves. Grey's 1560 genes are unassigned and are not interpretable as one biological process.

Table 10 lists the complete delivered set for the WGCNA case.

Table 10: Twenty required deliverables plus three auxiliary files. Every TSV row count was read back from the published file.

File	Content	Size
`analysis.R`	Reproducible pipeline, inputs to every required file	39,191 B
`analysis_report.md`	QC, outlier reasoning, network diagnostics, grey handling, limits	19,097 B
`selected_genes.tsv`	5000 genes entering the network	95,008 B
`selected_samples.tsv`	50 retained samples	410 B
`wgcna_expression.tsv`	50 rows $\times$ 5000 gene columns, transformed	3,044,699 B
`wgcna_traits.tsv`	50 rows, exactly the 5 specified phenotypes, row order matched	1,829 B
`sample_qc.tsv`	50 rows: library size, detected genes, connectivity, Z, outlier flag	3,337 B
`soft_threshold_statistics.tsv`	20 candidate powers, fit and connectivity diagnostics	1,636 B
`network_parameters.tsv`	Single row: type, correlation, power, TOM, module parameters	237 B
`gene_module_assignment.tsv`	5000 genes with module number and colour	135,505 B
`module_sizes.tsv`	10 modules summing to 5000 genes	124 B
`module_trait_results.tsv`	50 module–trait rows: correlation, P, FDR	3,264 B
`gene_MM_GS_results.tsv`	5000 genes $\times$ per-module MM and per-trait GS with P values	2,189,810 B
`hub_gene_candidates.tsv`	437 gene–phenotype–module records	17,492 B
`sample_QC_PCA.png`	Sample PCA with the outlier view	151,904 B
`sample_dendrogram_traits.png`	Sample clustering with phenotype bands	132,931 B
`soft_threshold.png`	Network diagnostics across the 20 candidate powers	144,196 B
`gene_dendrogram_modules.png`	Gene clustering with the module partition	63,318 B
`module_trait_heatmap.png`	Module–phenotype correlation map	144,196 B
`wgcna_results.RData`	Objects for continued analysis and reproduction	4,413,697 B
Auxiliary, not required		
`wgcna_TOM.npz`	Topological overlap matrix, for further inspection	100.0 MB
`hub_gene_candidates_detail.tsv`	Candidates with MM and GS values and signs	54,270 B
`pipeline_summary.json`	Counts, parameters and headline results in one file	2,823 B

B Contributors

Contributors are ordered alphabetically by their given-name initials.

B. An, B. Li, B. Wang, B. Zhang, B.L. Wang, C. Feng, C. Wei, C. Xue, C. Zhang, D. Ng, D. Ye, E. Min, F. Chen, F. Liu, F. Yang, F. Ye, G. Sun, H. Ji, H. Xu, H. Yang, H. Ye, H. Zhang, H. Zhao, J. Li, J. Lin, J. Xia, K. Jin, K. Wang, K. Yang, L. Bing, L. Lei, L. Su, Le. Wang, Lu. Wang, N. Wang, Q. Ren, Q. Yang, R. Li, S. Bai, S. Du, S. Li, S. Lin, S. Nie, S. Wang, S. Zhang, S.Z. Wang, T. Ge, Ta.Q. Fang, Ti.Q. Fang, W. Fang, W. Li, W. Zhang, X. Chen, X. Li, X. Tang, X. Wang, X. Xu, X. Zhang, X.Q. Wang, X.Y. Wang, Y. Deng, Y. Gao, Y. Hu, Y. Li, Y. Sui, Y. Wang, Y. Xiao, Y. Zhang, Y. Zhou, Z. Chen, Z. Cheng, Z. Feng, Z. Liang, Z. Liu, Z. Zhang.