PERSEUS: NON-BLOCKING PARALLEL EXPLORATION WITH A SPECULATIVE SWARM

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using agents face a breadth–coordination trade-off: a single agent follows a coherent task trajectory but discovers evidence mainly along its chosen call path, while multi-agent exploration broadens discovery through additional coordination. PERSEUS couples one authoritative main agent, the Actor, with a non-blocking speculative swarm. Single-round Speculators use constrained variants of native tools to explore complementary information needs in independent work copies, while the Actor retains task decisions and state changes. Persistent Futures carry unfinished acquisitions across Actor requests. A source-aware ledger admits completed observations with their provenance, allowing the Actor to continue while pending exploration proceeds asynchronously. Cooperation occurs through evidence: a Speculator's inspection can inform a different later action. On selected tasks from four benchmarks, PERSEUS achieves the highest overall end-to-end task success among 13 baselines, improving success from ReAct's 56.3% to 67.8% and Execution Speed from 2.43 to 3.20 ($10^{-3}\,\mathrm{s}^{-1}$). Temporal analysis, ablations, and scaling relate these outcomes to evidence flow, exploration structure, and persistence. Harness comparisons show fewer Actor turns with maintained or improved quality relative to single-agent controls. The results support broad exploration through asynchronous cooperation under one authority.

1 INTRODUCTION

Tool-using language agents alternate reasoning with environmental interaction to solve tasks (Yao et al., 2023). As tasks become longer and environments more complex, performance depends on both the evidence available to a decision and the coordination required to acquire it. Three recurring organizations expose this trade-off: the single-agent loop, centralized multi-agent systems, and decentralized multi-agent systems.

Single-agent loop: coherent authority, delayed discovery. A single agent chooses what to inspect from its history and reasons over the returned observations. Native parallel calls accelerate selected acquisitions; information outside those calls awaits a later decision. This couples an *execution dependency*, where an action needs a preceding result, to an *information-discovery order*, where relevant evidence is already accessible but has not been requested. Serial discovery adds avoidable latency when those acquisitions could proceed independently (Kim et al., 2024).

CMAS: broader expertise, centralized selection. Centralized multi-agent systems (CMAS) delegate work through a coordinator. AutoGen's manager-mediated group chat cycles through speaker selection, response collection, and broadcast (Wu et al., 2024). These exchanges add communication and inference steps to specialist contributions. Shared history coordinates acquired evidence; discovering missing information still depends on which investigation the coordinator activates next.

DMAS: distributed initiative, distributed coordination. Decentralized multi-agent systems (DMAS) distribute initiative through local routing and execution decisions, as in AgentNet (Yang et al., 2025). This broadens participation while creating a need to reconcile overlapping investigations, local conclusions, and decisions about how the task should proceed. Routing protocols organize those exchanges; their communication and reconciliation contribute to execution cost.

The common design question is how to broaden discovery while keeping task decisions coherent. We address it with *one authoritative main agent and a non-blocking speculative swarm*. The main

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

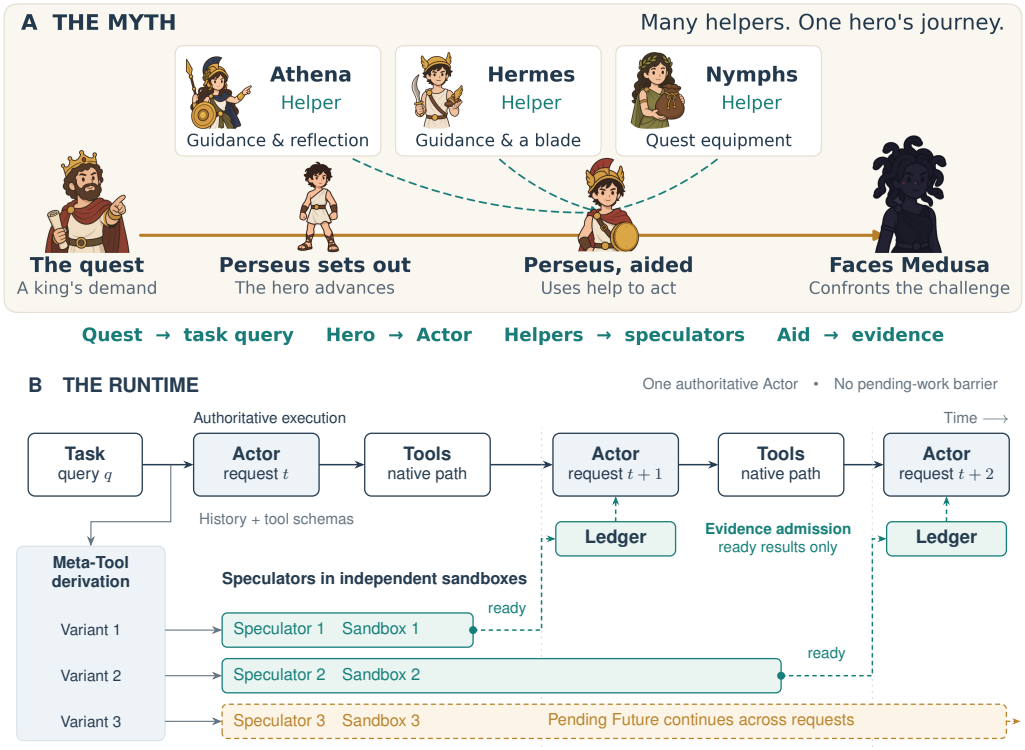


Figure 1: **Overview of PERSEUS. (A)** Naming analogy: the quest, Perseus, helpers, and aid correspond to the task query, Actor, Speculators, and evidence. **(B)** One exploration wave across Actor requests t through $t+2$. Meta-Tool variants guide concurrent Speculators whose calls execute in independent sandboxes. Ready results enter the source-aware ledger for admission at later request boundaries, while an unfinished Future remains active. The upper path shows authoritative execution; teal dashed arrows show evidence admission.

agent, or Actor, owns task decisions, state changes, and completion. Concurrent Speculators investigate complementary information needs and return observations through an evidence channel. Actor progress supplies fresh context to exploration, and completed observations inform later Actor decisions. This asymmetric exchange decouples the progress of evidence acquisition from authoritative execution: the Actor starts its next request as soon as its own work is ready, while unfinished Speculator calls continue in the background.

For this cooperation to be useful, exploratory results must inform the Actor's reasoning even when it chooses a different action. A Speculator inspecting a dependency constraint, for example, may reveal a configuration mismatch while the Actor diagnoses a failing function; the Actor can then choose a configuration repair. We call this *speculative evidence acquisition*: anticipating information needs through native tool executions with recorded provenance. Whereas LLMCompiler parallelizes a generated call plan (Kim et al., 2024) and Speculative Actions and PASTE reuse confirmed speculative actions (Ye et al., 2026; Sui et al., 2026), PERSEUS carries acquired information forward to whichever decision it informs. This creates three runtime requirements: distribute exploration, isolate its execution, and deliver evidence as it becomes ready.

We introduce **PERSEUS**—*Parallel ExploRation and Speculative Execution Unified Swarm*—a training-free runtime implementing this protocol (Figure 1). The name evokes Perseus's quest and the divine helpers who aided him against Medusa (Apollodorus, 1921, Library 2.4.2–3).

Parallel exploration partitions Meta-Tool argument domains into constrained variants so Speculators can concurrently pursue distinct acquisition directions beyond the Actor's issued calls, without an Actor-authored subtask plan. *Speculative execution* streams their native calls into independent work copies alongside the Actor, separating exploratory writes from authoritative state. *Unified swarm* connects single-round Speculators to the Actor through an asynchronous evidence channel built

from persistent Futures and a source-aware ledger. Futures retain unfinished acquisitions across requests and share repeated acquisition work; the ledger admits ready observations at later request boundaries while pending exploration continues. These stages connect broad acquisition to one decision process: variant assignment distributes the search, work copies isolate execution, and ready-only admission supplies observations at subsequent Actor requests.

Our contributions are threefold: **(i) parallel exploration under single authority**, where Meta-Tool variants allocate distinct acquisition directions to Speculators beyond the Actor's issued calls, broadening discovery without an Actor-authored subtask plan or distributed task ownership; **(ii) an asynchronous cooperation protocol**, where single-round Speculators stream calls into independent execution, persistent Futures carry work across requests, and source-aware admission connects ready evidence to the Actor while the swarm continues; and **(iii) an empirical evaluation** showing the highest overall end-to-end task success among 13 baseline configurations across four benchmarks (Section 3.1), complemented by temporal analysis, scaling studies, ablations, and harness comparisons that characterize how breadth, structure, and persistence translate into task quality and score-adjusted execution speed.

2 METHOD

2.1 PROBLEM SETUP AND OVERVIEW

Let q be a task, $\mathcal{T}$ its native tools, and x the authoritative environment. PERSEUS assigns task decisions and commits to one Actor, while a speculative swarm acquires observations in independent work copies. At integer-indexed Actor request t, history H_t contains the task, prior responses, tool observations, and previously admitted evidence. The Actor's fixed-parameter model π_θ receives H_t augmented by newly available evidence Δ_t:

$$\bar{H}_t = H_t \oplus \Delta_t, \qquad m_t \sim \pi_\theta(\cdot \mid \bar{H}_t), \tag{1}$$

where $\oplus$ denotes ordered message concatenation. Actor calls execute through the native authoritative path, preserving its ordering and parallelism. Their observations o_t update history as

$$H_{t+1} = \bar{H}_t \oplus (m_t, o_t). \tag{2}$$

The pair (m_t, o_t) appends the Actor response followed by its native tool-observation messages. Here o_t is empty when no tool is called; user input enters through the native session interface. Actor calls may update x, while auxiliary execution contributes observations to later Δ_t.

To supply Δ_t while the Actor advances, the runtime assigns constrained tool variants, launches asynchronous acquisitions, and retains their Futures for later evidence admission. Sections 2.2–2.4 define this protocol. It requires no additional training or Actor-facing coordination tools.

2.2 ORGANIZING EXPLORATION WITH META-TOOL VARIANTS

A *Meta-Tool* exposes operation families through arguments such as `method` or `operation`. For a native tool T with schema-valid argument space S_T, PERSEUS constructs variants $V_{T,i} = (T, D_{T,i})$, where $D_{T,i} \subseteq S_T$ restricts the acquisition domain while preserving the tool's native execution interface. Each Speculator receives one variant and chooses what to inspect within it from the current task history.

Declared action values and successful authoritative invocation heads supply the domain constraints. Ordered exclusion makes the domains disjoint, and a residual variant covers the remaining legal arguments. When no usable discriminator exists, the variant uses the original schema. Complete proposed calls are validated against the native schema and their assigned domain before execution. Appendix A.1 gives the construction and validation rules.

This decomposition distributes exploration over available operations while leaving the Actor's tool interface unchanged. Distinct argument domains can reveal overlapping information, so the runtime combines acquisition reuse with source-aware evidence filtering.

2.3 LAUNCHING AND EXECUTING THE SPECULATIVE SWARM

Launch on authoritative progress. At the initial request and each new progress revision r, PERSEUS dispatches one Speculator for each of n currently eligible Meta-Tool variants alongside one Actor ($n + 1$ workers per wave), with no default concurrency cap. Revisions advance on

genuine user input or authoritative observations; each domain launches at most once per revision. Evidence admission alone does not trigger renewal.

Execute calls as they become available. Each Speculator makes one model-generation round and proposes native calls for information acquisition. Complete streamed calls start after validation, overlapping the remaining generation and the Actor's work. Tool results feed the evidence path; adaptive acquisition requiring another model decision uses a later frontier. The runtime schedules these calls directly, without peer negotiation or task delegation among Speculators.

Each acquisition executes in its own host-managed work copy, with arguments, error status, and copy provenance attached to its result. Native commands can inspect data, run diagnostics, and make temporary changes inside that copy. Copy writes never merge into x, and invalid execution scopes fail closed. The host isolates all relevant mutable resources, including external services. Appendix A.2 specifies the execution contract.

2.4 Persistent Futures and Ready-Only Evidence Admission

Acquisitions finish on different schedules, so a session-owned Future registry preserves their lifetimes across Actor requests. Identical in-flight acquisitions share execution; completed acquisitions can be reused at the same authoritative revision and refreshed after progress. Each result has an originating request and a review flag.

Let $\mathcal{F}_t$ be the registry snapshot immediately before Actor request t. The runtime selects completed, unreviewed results from earlier requests:

$$\mathcal{R}_t = \{f \in \mathcal{F}_t : \mathrm{origin}(f) < t,\ \mathrm{ready}(f),\ \neg\,\mathrm{reviewed}(f)\}. \tag{3}$$

With evidence ledger $\mathcal{L}_t$, it processes this set in deterministic order:

$$(\Delta_t, \mathcal{L}_t^+) = \mathrm{Admit}\big(\mathcal{L}_t, \mathrm{Sort}(\mathcal{R}_t)\big). \tag{4}$$

The ledger suppresses recognized source-qualified duplicates and superseded values, and returns complete retained observations with their provenance. Opaque outputs retain their native content. Every inspected result is marked reviewed, including results that contribute no new evidence. $\mathcal{L}_t^+$ denotes the ledger immediately after this admission step.

The Actor consumes Δ_t through Equation 1 and continues its native decision loop. Forming $\mathcal{R}_t$ never waits for pending acquisitions: unfinished work continues, and results completing after the snapshot become eligible at a later request. An empty ready set produces an empty Δ_t. Admitted evidence stays in history, while Actor-issued calls execute through the authoritative path. Task termination cancels pending work and settles cleanup. Appendix A.3 and Algorithm 2 specify ordering, provenance handling, and lifecycle events.

2.5 Using Evidence to Change Decisions

Admission makes an observation available to the Actor; its value comes from the decision it informs. For example, a Speculator's catalog inspection may reveal an eligibility constraint while the Actor prepares an update. The admitted observation can change which record the Actor selects or which condition it verifies, even though inspection and update are different calls. The Actor interprets the observation using its source and copy provenance, then executes the chosen action natively.

This interface supports two routes to utility. Timely evidence can remove a sequential information-gathering step; additional evidence can expose an overlooked constraint or alternative and improve task quality. The design therefore targets both the information available to a decision and the time spent obtaining it. Benchmark-native outcomes measure task quality, while execution speed relates the selected quality metric to execution time. The model in Appendix A.5 formalizes this trade-off: useful evidence must save more downstream work than it costs to acquire and process.

3 Experiments

3.1 Main Results

Evaluation setup. Table 1 compares PERSEUS with 13 baselines across four established agent benchmarks: AutomationBench for SaaS workflows (Shepard & Salimans, 2026), τ^2-Bench for

human–agent interaction (Barres et al., 2026), Terminal-Bench for engineering and coding (Merrill et al., 2026), and GAIA for web-centered assistance (Mialon et al., 2024). These domains cover major general-agent applications involving difficult, long-horizon tasks with strong environment dependencies. The selected subsets contain 36, 60, 31, and 56 cases, respectively (183 total), with three attempts per case; selection criteria appear in Appendix B.1. PERSEUS pairs a GPT-5.6 Terra Actor with GPT-5.6 Luna Speculators, both at high reasoning effort. Task success (TS) averages end-to-end success across attempts and cases. Execution Speed (ES) divides aggregate success by aggregate execution time. Overall pools all 183 tasks with task-count weighting. Baseline implementations follow official source-code references and default method configurations across single-agent, planning, workflow, multi-agent, speculative, and asynchronous strategies. ReAct is the single-agent control; CMAS and DMAS represent centralized and decentralized coordination. Appendix B provides model-role mappings, execution budgets, and mean times with SD.

Table 1: Main results. TS is end-to-end task success (%); subscripts show SD across three runs where available. ES is aggregate success per second ($10^{-3}\,\mathrm{s}^{-1}$). Overall pools 183 tasks with task-count weighting. Auto and Term denote AutomationBench and Terminal-Bench. Bold marks column-best values. Execution times and protocols appear in Table 4 and Appendix B.

Method	**TS (%) ↑**					**ES ↑**				
	Auto	τ^2	Term	GAIA	Overall	Auto	τ^2	Term	GAIA	Overall
ReAct	$27.8_{\pm 2.8}$	$66.7_{\pm 5.0}$	$69.9_{\pm 4.9}$	$56.0_{\pm 6.3}$	56.3	1.25	4.79	2.07	2.03	2.43
Plan-and-Execute	$14.8_{\pm 1.6}$	$71.7_{\pm 7.6}$	$72.0_{\pm 3.7}$	$48.2_{\pm 3.1}$	53.4	0.81	4.90	1.99	2.56	2.63
ReWOO	$0.9_{\pm 1.6}$	$53.3_{\pm 7.3}$	$25.8_{\pm 11.6}$	$34.5_{\pm 6.3}$	32.6	0.05	1.79	0.94	2.45	1.44
Multi-Persona	$16.7_{\pm 5.6}$	$68.3_{\pm 5.0}$	$68.8_{\pm 4.9}$	$31.0_{\pm 1.0}$	46.8	1.39	5.33	2.07	**3.54**	3.15
CMAS	$15.7_{\pm 4.2}$	$71.7_{\pm 4.4}$	$63.4_{\pm 6.7}$	$55.4_{\pm 9.9}$	54.3	0.93	4.94	1.37	1.81	2.14
DyLAN	$23.1_{\pm 1.6}$	$52.2_{\pm 5.4}$	$58.1_{\pm 5.6}$	$41.7_{\pm 4.1}$	44.3	0.20	0.87	0.66	0.83	0.61
MemGPT	$25.9_{\pm 5.8}$	$65.6_{\pm 1.0}$	$69.9_{\pm 4.9}$	$48.2_{\pm 6.4}$	53.2	1.77	5.83	3.15	2.88	**3.44**
LLMCompiler	$4.6_{\pm 1.6}$	$56.1_{\pm 2.5}$	$22.6_{\pm 6.5}$	$31.5_{\pm 4.1}$	32.8	0.08	3.23	2.59	3.35	1.51
AFlow	$23.1_{\pm 5.8}$	$67.8_{\pm 2.5}$	$75.3_{\pm 1.9}$	$41.1_{\pm 9.9}$	52.1	1.61	5.50	1.99	2.00	2.66
Magentic-One	$7.4_{\pm 3.2}$	$68.3_{\pm 3.3}$	$43.0_{\pm 6.7}$	$44.6_{\pm 0.0}$	44.8	0.12	2.10	0.66	1.41	1.04
DMAS	$20.4_{\pm 1.6}$	$73.3_{\pm 3.3}$	$78.5_{\pm 1.9}$	$41.1_{\pm 1.8}$	53.9	1.44	5.51	**3.18**	1.77	2.92
SA	$17.6_{\pm 7.0}$	$73.3_{\pm 4.4}$	$66.7_{\pm 1.9}$	$54.2_{\pm 3.7}$	55.4	1.24	**6.25**	2.57	2.73	3.24
AsyncFC	$27.8_{\pm 12.7}$	$65.6_{\pm 7.9}$	$65.6_{\pm 12.2}$	$58.3_{\pm 4.1}$	55.9	**2.17**	2.65	2.19	3.36	2.66
PERSEUS	**39.8**$_{\pm 1.6}$	**77.8**$_{\pm 2.6}$	**81.7**$_{\pm 4.9}$	**67.3**$_{\pm 6.3}$	**67.8**	2.06	4.81	2.44	3.22	3.20

Single-agent authority with broader discovery. PERSEUS improves both TS and ES over ReAct on all four benchmarks, raising overall TS from 56.3% to 67.8% and ES from 2.43 to 3.20. On Terminal-Bench, TS rises from 69.9% to 81.7% at similar mean execution times (338.1 versus 334.3 s). The gain therefore includes better completion at comparable execution time, matching the quality route described in Section 2.5: complementary evidence can improve the Actor's choices within one authoritative trajectory.

Evidence cooperation under unified authority. PERSEUS exceeds CMAS and DMAS in overall TS by 13.5 and 13.8 percentage points, with higher overall ES than both. This comparison tests the organization of cooperation: PERSEUS distributes acquisition across speculators while one Actor integrates their observations and selects actions. The joint quality and ES gains support the asymmetric exchange in Sections 2.3–2.4, where runtime dispatch and ready evidence connect parallel work to a continuing decision process.

Speculation and asynchronous execution. SA represents speculative action prediction; AsyncFC represents asynchronous execution through symbolic futures (Ye et al., 2026; Feng et al., 2026). PERSEUS improves TS over both on all four benchmarks, with overall TS/ES of 67.8/3.20 versus 55.4/3.24 and 55.9/2.66. Its evidence can guide actions beyond the call originally explored (Section 2.5), extending the role of speculation from executing anticipated work to informing decisions. SA's slightly higher ES and MemGPT's best overall ES (3.44 at 53.2% TS) locate the trade-off: PERSEUS prioritizes more completed tasks while preserving a high success rate per second.

3.2 Temporal Pattern of Speculative Evidence

To relate these task-level outcomes to the cooperation process, Figure 2 traces evidence admission across 549 task runs (183 tasks, three attempts each). It separates all runs with injection from those in which manual review identifies later use of the admitted information.

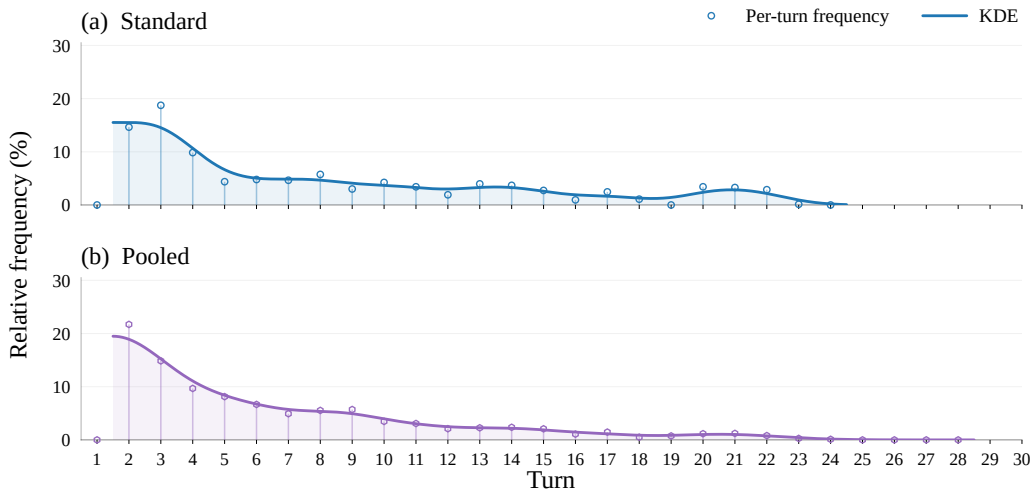


Figure 2: **Timing of speculative evidence injection. (a) Standard:** 162 task runs in which block matching and manual trajectory review identified later use of admitted evidence in reasoning or decisions. **(b) Pooled:** all 493 runs with evidence injection. Points and stems show each turn's share of cohort injections; curves show Gaussian KDE, with normalization and smoothing in Appendix B.1.

Early evidence with later contributions. Injection peaks at turns 2–4 place most exploratory evidence early in the decision sequence, when it can inform subsequent work. The lower-frequency tail shows that admission also continues as tasks develop; manual review identifies later reuse in the Standard cohort. These timings give the persistent channel in Section 2.4 a concrete role: ready observations enter the current request, while continuing acquisition can supply evidence to later decisions. The swarm supports early discovery and later information needs within one Actor trajectory.

3.3 Component Ablations

We next examine the components that produce and sustain this evidence flow, comparing full PERSEUS with configurations disabling Meta-Tool derivation, the speculative swarm, or cross-request Future persistence on an 85-case diagnostic subset. These ablations separate the organization, presence, and lifetime of auxiliary acquisition. Table 2 summarizes the component study; Appendix B.7 defines the study population, each configuration, and each metric.

Table 2: Aggregate component ablations. TS is end-to-end task success (%); PCS is Partial-credit Score; ES is fractional TS divided by mean time ($10^{-3}\,\mathrm{s}^{-1}$). TS, Turn, and Time subscripts show SD across three runs. Turn counts agent response turns; Tool is tool-call success (%). Equivalent tokens are in millions (M). Bold marks the best value in each metric. Configuration and metric details appear in Appendix B.7.

Configuration	TS (%)↑	Turn↓	Time (s)↓	PCS↑	ES↑	Tool (%)↑	Eq. Tokens (M)↓
w/o Meta-Tool Derivation	$47.1_{\pm 10.2}$	$10.60_{\pm 1.32}$	$263.5_{\pm 63.1}$	0.660	1.79	96.8	14.56
w/o Speculative Swarm	$52.9_{\pm 5.9}$	$12.03_{\pm 3.42}$	$311.2_{\pm 35.6}$	0.660	1.70	96.2	6.14
w/o Persistent Futures	$49.0_{\pm 9.0}$	$10.77_{\pm 1.82}$	$266.2_{\pm 36.3}$	0.618	1.84	93.9	**5.51**
Full PERSEUS	$\mathbf{56.9}_{\pm 9.0}$	$\mathbf{9.96}_{\pm 4.88}$	$\mathbf{247.1}_{\pm 21.6}$	**0.825**	**2.30**	**97.2**	9.57

Exploration beyond the Actor's call path. Removing the swarm retains the Actor's native execution path and parallel calls. Full PERSEUS raises TS from 52.94% to 56.86% and PCS from

0.660 to 0.825, while reducing Actor turns from 12.03 to 9.96. This combination supports the pre-decision acquisition mechanism: speculators make complementary observations available alongside the Actor's chosen calls, reducing the sequential rounds needed to obtain useful information. The improvement concerns both task completion and progress captured by partial credit.

Structuring exploratory work. Without Meta-Tool derivation, speculators explore unmodified tool schemas. Restoring constrained variants raises TS from 47.1% to 56.9% and PCS from 0.660 to 0.825 while reducing equivalent tokens from 14.56M to 9.57M. More auxiliary computation in the unrestricted configuration accompanies worse outcomes. The result supports the purpose of domain assignment in Section 2.2: explicit acquisition scopes direct speculators toward complementary investigations, improving how exploration uses its computation budget.

Preserving evidence across decisions. Without persistent Futures, unfinished acquisitions are cancelled at the next boundary. Completion speed then determines which evidence reaches the Actor: quick observations survive, while slower investigations lose their opportunity to contribute later. TS falls to 49.0%, below the no-swarm control's 52.9%, and PCS falls to 0.618 versus 0.660. This pattern suggests that an incomplete stream of auxiliary observations can be less useful than the Actor's own acquisition sequence. Full persistence allows later evidence to complement early observations, reaching 56.9% TS, 0.825 PCS, and 2.30 ES versus 1.84 without persistence, with lower mean time (247.1 versus 266.2 s).

3.4 Scaling Speculative Acquisition

Having examined component removal, we next conduct an independent scaling study. Figure 3 varies the capacity allocated to acquisition through a Speculator concurrency cap and reasoning effort. Default width follows the eligible Meta-Tool variants without a fixed cap. Speculators acquire evidence in one model-generation round while the Actor continues task reasoning. The sweeps report native Partial-credit Score (PCS) under Terra/Luna; settings appear in Appendix B.4.

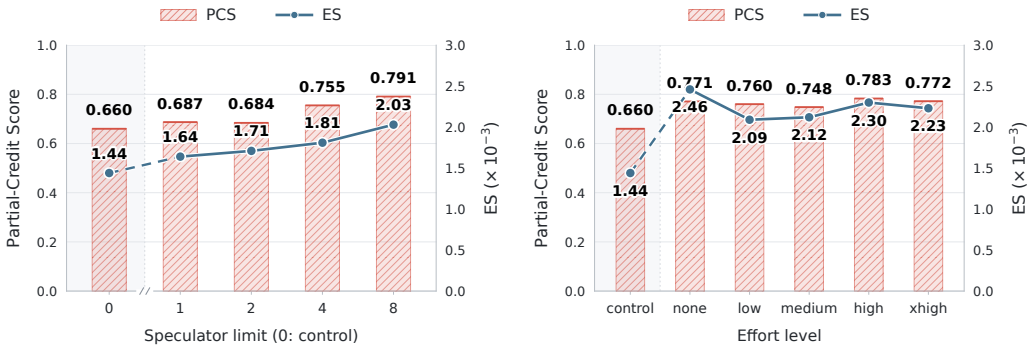


Figure 3: **Scaling speculative acquisition. Left:** the Speculator concurrency cap varies over $0, 1, 2, 4, 8$; zero disables Speculators and positive caps are spaced on a logarithmic scale. **Right:** Speculator reasoning effort varies from `none` (no thinking) through `xhigh`, with a separate no-Speculator control. The cap bounds the number of simultaneously active Speculators rather than fixing the swarm width; all other PERSEUS mechanisms remain unchanged.

Scaling parallelism. Raising the Speculator concurrency cap from zero to eight increases PCS by 0.131 and ES by 41.0%. ES improves at every increment, while PCS dips slightly from cap one to two. The concurrency cap bounds how many assigned acquisition directions can progress simultaneously. The joint improvement supports the breadth component of the design: additional exploratory capacity can improve task progress while the Actor retains one decision trajectory. Cap eight gives the highest PCS and ES within the tested range.

Scaling reasoning effort. All active settings exceed the no-Speculator control. PCS spans 0.748–0.783 without a monotonic trend: `none` gives the highest ES (2.46) at PCS 0.771, while `high` gives the highest PCS (0.783) at ES 2.30. Together with the width sweep, this pattern favors broad acquisition over uniformly increasing speculator reasoning. It fits the single-round division of labor

in Section 2.3: Speculators identify and execute useful inspections, while the Actor interprets their observations and decides how to proceed.

3.5 Comparison with Mainstream Harnesses

To test whether the reduction in sequential interaction persists against practical agent harnesses, Figure 4 places PERSEUS beside single-agent and workflow/multi-agent controls on AutomationBench. The experiment compares methods within the Opus+Sonnet and Terra+Luna model groups.

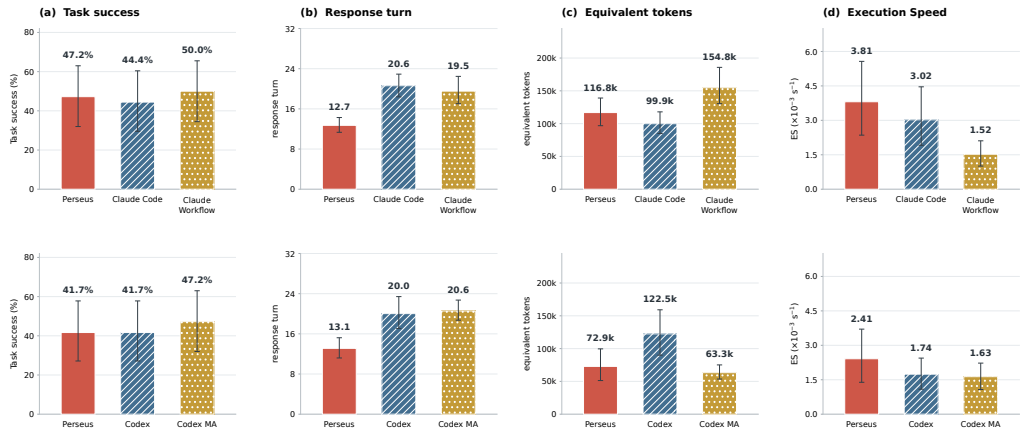


Figure 4: **Comparison with mainstream agent harnesses. Top:** PERSEUS, Claude Code, and Claude Workflow under Opus+Sonnet. **Bottom:** PERSEUS, Codex, and Codex MA under Terra+Luna. Columns report task success, response turns, equivalent tokens (K), and execution speed (ES, $10^{-3}\,\mathrm{s}^{-1}$). Metric definitions and token price conversion appear in Appendix B.5.

Single-agent controls: quality with fewer turns. PERSEUS matches Codex's 41.7% TS with 38.5% higher ES and 34.5% fewer turns; under Opus+Sonnet, TS rises from Claude Code's 44.4% to 47.2%, also with fewer turns and higher ES. Maintained or improved completion alongside fewer Actor rounds supports the latency route in Section 2.5: acquiring information ahead of demand can consolidate sequential inspection and reasoning steps. Both groups show this pattern.

Workflow controls: higher execution speed with fewer turns. PERSEUS achieves $2.51\times$ Claude Workflow's ES and $1.48\times$ Codex MA's ES, with fewer turns. The workflows attain higher TS, exposing different allocations of computation between task quality and execution time. PERSEUS's operating point reflects its lightweight cooperation path: speculators contribute observations while the Actor continues, raising the success-adjusted execution rate in both groups.

Equivalent-token trade-offs. PERSEUS uses more equivalent tokens than Claude Code but fewer than Codex, while reducing turns and raising ES against both. Across these usage patterns, moving acquisition off the Actor's decision sequence reduces sequential interaction while supplying evidence for its ongoing reasoning.

3.6 Case Study: Evidence Ahead of Demand

In AutomationBench's statement-delivery task `finance:4073`, Figure 5 shows how independent policy and accounting-interface discovery can overlap, while business-ID-dependent queries and delivery retain their order. PERSEUS can explore these independent branches concurrently and admit their ready observations at one decision boundary. Meta-Tool domains distribute acquisition, persistent Futures preserve unfinished work, and source-aware admission supplies new observations while suppressing repeated catalog entries. A speculator discovers the Wave invoice interface, and the ledger admits its catalog record for later Actor decisions. The Actor combines this evidence with customer records and sends the statement; the episode receives an official score of 1.0. The case illustrates how asynchronous exploration can shorten the information-acquisition chain while one Actor retains task authority. Appendix C provides the full trace.

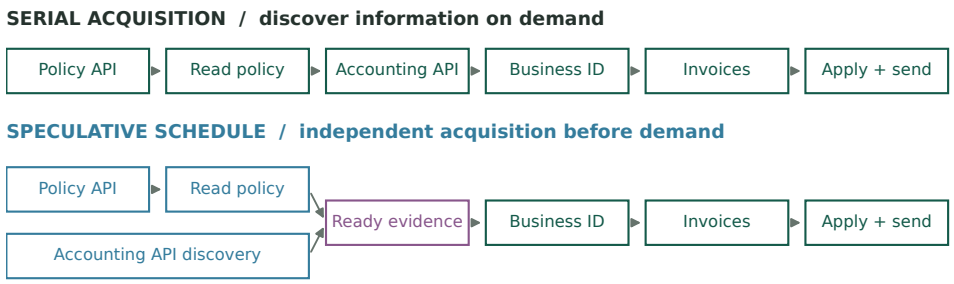


Figure 5: **Logical folding of independent acquisition.** Serial and speculative information-acquisition paths preserve the same downstream dependencies. Positions indicate logical stages.

4 Related Work

Tool use and agent workflows. Toolformer learns API invocation, while ReAct interleaves reasoning with actions and observations (Schick et al., 2023; Yao et al., 2023). ReWOO separates planning from tool feedback, LLMCompiler schedules independent calls from an explicit dependency graph, and AFlow searches over code-represented workflows (Xu et al., 2023; Kim et al., 2024; Zhang et al., 2025a). Beyond within-query execution, Sleep-time Compute anticipates future inference; Agentic Plan Caching and AgenticCache reuse plans or transitions across tasks; and MemGPT retains context for long-lived interactions (Lin et al., 2025; Zhang et al., 2025b; Kim et al., 2026; Packer et al., 2023). These approaches address action, scheduling, and memory.

Centralized and decentralized coordination. Multi-Persona elicits complementary roles, and AutoGen provides a programmable substrate for multi-agent conversations (Wang et al., 2024; Wu et al., 2024). Magentic-One and Puppeteer use centralized orchestration to assign or activate specialized agents, whereas DyLAN and AgentNet adapt participation and communication structure dynamically (Fourney et al., 2024; Dang et al., 2025; Liu et al., 2024; Yang et al., 2025). These systems organize agent participation and task delegation.

Speculative actions and asynchronous execution. Speculative decoding introduced predict-and-verify at the token level (Leviathan et al., 2023); agent-level methods extend it to actions. Speculative Actions predicts future actions with faster models, while Speculate While You Reason jointly trains agent and speculator modes for next-tool-call prediction (Ye et al., 2026; Ji et al., 2026). In parallel, Asynchronous Tool Usage models event-driven agents, AsyncFC exposes concurrency through symbolic futures, and Speculative Interaction Agents combines asynchronous I/O with speculative calls (Ginart et al., 2024; Feng et al., 2026; Hooper et al., 2026). AsyncTool evaluates the resulting challenges under delayed, interleaved calls (Shi et al., 2026). Action anticipation and asynchronous execution offer complementary ways to overlap work with ongoing interaction.

Positioning PERSEUS. PERSEUS brings these execution mechanisms into an asymmetric cooperation architecture: one authoritative main agent and a non-blocking speculative swarm. Futures and isolated work copies support a source-aware evidence channel. Actor progress renews exploration; persistent acquisitions proceed independently; completed observations inform later decisions. Cooperation centers on decision-relevant evidence, decoupling exploratory execution from task authority and collective synchronization.

5 Conclusion

PERSEUS combines one authoritative main agent with a non-blocking speculative swarm through asynchronous, evidence-mediated cooperation. Across four benchmarks, it achieves the highest overall end-to-end task success and higher execution speed than ReAct. Ablations and scaling characterize how structure, breadth, and persistence support this design; harness comparisons show fewer sequential turns with maintained or improved task quality. Together, these results show that exploratory breadth need not require distributed task authority: asynchronous evidence acquisition can broaden discovery while preserving a single authoritative trajectory.

ETHICS STATEMENT

Speculative acquisition increases the scope of tool access, information processing, and resource consumption. Deployment requires authorized tool access, protection of credentials and observations, and isolation of external mutable services as well as local state. Tool output is treated as untrusted evidence, with current-state verification and application-specific oversight guiding consequential actions. PERSEUS retains one agent-side task authority, providing a single decision and execution path for applying these controls.

REPRODUCIBILITY STATEMENT

Section 2 and Appendix A specify the execution protocol and pseudocode. Appendix A.4 provides the Speculator prompt, history encoding, and evidence-message format; Appendix B gives benchmark populations, model assignments, execution budgets, and aggregation rules. The complete evaluation and validation code, together with the PERSEUS prototype implementation, will be released after the review process.

AI USE STATEMENT

Generative AI assisted literature search and synthesis, manuscript drafting and translation, refinement of method exposition and mathematical arguments, interpretation of aggregate results, and LaTeX and figure preparation. Responsibility for the final text, claims, and artifacts rests with the authors.

REFERENCES

Apollodorus. *The Library*. William Heinemann, 1921. Translated by James George Frazer. Book 2, sections 4.2–4.3.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=OC2z7iSQKa.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/f1320d2e2842169c6fc89dcbd80e94d0-Abstract-Conference.html.

Guangyu Feng, Huanzhi Mao, Prabal Dutta, and Joseph E. Gonzalez. Concurrency without model changes: Future-based asynchronous function calling for LLMs. *arXiv preprint arXiv:2605.15077*, 2026. URL https://arxiv.org/abs/2605.15077.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-One: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024. URL https://arxiv.org/abs/2411.04468.

Antonio A. Ginart, Naveen Kodali, Jason Lee, Caiming Xiong, Silvio Savarese, and John Emmons. Asynchronous tool usage for real-time agents. *arXiv preprint arXiv:2410.21620*, 2024. URL https://arxiv.org/abs/2410.21620.

Coleman Hooper, Minwoo Kang, Suhong Moon, Nicholas Lee, Eric Wen, John Wawrzynek, Michael W. Mahoney, Yakun Sophia Shao, Amir Gholami, and Kurt Keutzer. Speculative interaction agents: Building real-time agents with asynchronous I/O and speculative tool calling. *arXiv preprint arXiv:2605.13360*, 2026. URL https://arxiv.org/abs/2605.13360v2.

Jiabao Ji, Yujian Liu, Li An, Rohit Jain, Gungor Polatkan, Siyu Zhu, and Shiyu Chang. Speculate while you reason: Teaching agents to predict their next tool call via joint agent–speculator RL. *arXiv preprint arXiv:2607.25816*, 2026. URL https://arxiv.org/abs/2607.25816v1.

Hojoon Kim, Yuheng Wu, and Thierry Tambe. AgenticCache: Cache-driven asynchronous planning for embodied AI agents. In *Proceedings of Machine Learning and Systems*, volume 8, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/hash/c66a9db149261435664284a20b6f1d42-Abstract-Conference.html.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24370–24391. PMLR, 2024. URL https://proceedings.mlr.press/v235/kim24y.html.

Jacky Kwok, Shulu Li, Pranav Atreya, Yuejiang Liu, Yixing Jiang, Chelsea Finn, Marco Pavone, Ion Stoica, and Azalia Mirhoseini. LLM-as-a-Verifier: A general-purpose verification framework. *arXiv preprint arXiv:2607.05391*, 2026. URL https://arxiv.org/abs/2607.05391.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Kevin Lin, Charlie Snell, Yu Wang, Charles Packer, Sarah Wooders, Ion Stoica, and Joseph E. Gonzalez. Sleep-time compute: Beyond inference scaling at test-time. *arXiv preprint arXiv:2504.13171*, 2025. URL https://arxiv.org/abs/2504.13171v1.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=XII0Wp1XA9.

Mike A. Merrill, Alexander G. 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 Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert 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, Jesse Hu, 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. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/444a3737adaee10d86ad2ef5f74468e6-Abstract-Conference.html.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/25ae35b5b1738d80f1f03a8713e405ec-Abstract-Conference.html.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

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. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Daniel Shepard and Robin Salimans. AutomationBench. *arXiv preprint arXiv:2604.18934*, 2026. URL https://arxiv.org/abs/2604.18934.

Kou Shi, Ziao Zhang, Shiting Huang, Avery Nie, Zhen Fang, Qiuchen Wang, Lin Chen, Huaian Chen, Zehui Chen, and Feng Zhao. AsyncTool: Evaluating the asynchronous function calling capability under multi-task scenarios. *arXiv preprint arXiv:2605.27995*, 2026. URL https://arxiv.org/abs/2605.27995.

Yifan Sui, Han Zhao, Rui Ma, Zhiyuan He, Hao Wang, Jianxun Li, Kaiqiang Xu, Kai Chen, and Yuqing Yang. Parallelizing tool execution and LLM generation for low-latency agent serving. *arXiv preprint arXiv:2603.18897*, 2026. URL https://arxiv.org/abs/2603.18897v3.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2609–2634. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.147. URL https://aclanthology.org/2023.acl-long.147/.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 257–279. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.naacl-long.15. URL https://aclanthology.org/2024.naacl-long.15/.

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 conversations. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. ReWOO: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*, 2023. URL https://arxiv.org/abs/2305.18323.

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. AgentNet: Decentralized evolutionary coordination for LLM-based multi-agent systems. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3578. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/9a379c1b05793d1c42dc832269834515-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. Speculative actions: A lossless framework for faster AI agents. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/fb279d7a134ad6fead7953ac7fd77fc4-Abstract-Conference.html.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations*, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5492ecbce4439401798dcd2c90be94cd-Abstract-Conference.html.

Qizheng Zhang, Michael Wornow, and Kunle Olukotun. Agentic plan caching: Test-time memory for fast and cost-efficient LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/9549f7d06700f0966d5f938f1d11022a-Abstract-Conference.html.

A Runtime Details and Execution Properties

This appendix details the protocol defined in Section 2. Algorithms 1–3 specify domain construction, event scheduling, and evidence admission. Appendix A.4 gives the prompt and message interfaces; Appendix A.5 derives the execution properties and local utility condition.

A.1 Meta-Tool Decomposition and Variant Validation

Section 2.2 defines a variant as $V_{T,i} = (T, D_{T,i})$. The following construction determines its argument domain from the native schema and authoritative invocation records.

For tool T, let S_T be its schema-valid argument space. Declared action discriminators and successful authoritative invocation heads yield predicates $P_1, \ldots, P_K$, ordered by specificity with deterministic tie-breaking. A predicate fixes a declared action value or specializes a structured endpoint supported by an observed call. We define

$$\begin{aligned} D_{T,i} &= \{u \in S_T : P_i(u) \wedge \textstyle\bigwedge_{j<i} \neg P_j(u)\}, \\ D_{T,\mathrm{rest}} &= S_T \setminus \textstyle\bigcup_{i=1}^{K} D_{T,i}. \end{aligned} \tag{5}$$

Ordered exclusion makes the domains pairwise disjoint. The complement is retained unless coverage is established; with no usable discriminator, acquisition falls back to the original schema. Opaque shell strings are not heuristically split into program categories. The construction partitions schema-valid arguments; domains can differ in size and information value.

Algorithm 1 gives the construction. Each domain $D_{T,i}$ defines a Meta-Tool variant $V_{T,i} = (T, D_{T,i})$, realized by a restricted native schema and its fixed action bindings. The residual domain defines a corresponding residual variant. Each Speculator receives one such variant, retaining the original tool identity and schema constraints. A streamed call is checked against its assigned variant schema, then passed through native argument preparation and validation. The speculator retains its launch-time history and variant schema across subsequent Actor requests. Meta-Tool variant construction therefore decomposes acquisition scope while retaining native execution semantics.

Algorithm 1 Construct Meta-Tool variants from schemas and authoritative observations

Require: Native tool T, schema S_T, authoritative observation history H.
$C \leftarrow \emptyset$; covered $\leftarrow$ false.
Read the declared values of supported action fields.
Add equality predicates for declared scalar values and schema-branch action constants to C.
Set covered if a required action field has an exhaustive declared value set.
for each successful authoritative observation of T in H **do**
    Add supported action predicates and observed endpoint specializations.
    Add structured argument-vector head predicates when available.
end for
if $C = \emptyset$ **then**
    return one variant with the original schema S_T.
end if
Deduplicate predicates; sort by decreasing specificity, then canonical serialization.
$E \leftarrow \emptyset$; $V \leftarrow \emptyset$.
for each predicate P in this order **do**
    $D \leftarrow S_T \cap P \cap \neg(\bigvee E)$; append (T, D) to V.
    $E \leftarrow E \cup \{P\}$.
end for
if not covered **then**
    Append residual variant $(T, S_T \cap \neg(\bigvee E))$ to V.
end if
return V.

Action fields are `method`, `operation`, `action`, `op`, `mode`, `function`, and `command`. Specificity counts fixed fields (one for argument-vector heads); canonical serialization breaks ties and defines variant identities.

A.2 PROGRESS-TRIGGERED ASYNCHRONOUS ACQUISITION

Progress-triggered launch. At the initial Actor request and later request boundaries allowed by the refresh policy, the controller constructs Meta-Tool variants from the current authoritative records and asynchronously starts eligible Speculators. The default policy renews acquisition on new authoritative progress. Each wave dispatches one Speculator per eligible Meta-Tool variant alongside the single Actor. Its $n + 1$ organization follows the dynamically derived variants. Each domain is issued once per revision; a newer revision can launch another Speculator while older model generation or tool execution continues. The active population therefore follows launch and completion events across revisions. Auxiliary evidence admission does not advance r, preventing an "admit evidence, launch more Speculators" feedback loop. Initial-only, completed-step, and authoritative-information-growth policies provide alternative refresh heuristics.

Single-round proposals and streamed execution. A Speculator receives a recorded representation of the current history after admission, $\bar{H}_t$, its assigned tool variant, and an acquisition-only role. Acquisition uses a separately configurable pretrained model without additional runtime-specific training. The Speculator can propose zero or more independent native calls to obtain missing task information. The Actor produces the task plan and final deliverable. Each complete streamed call can enter execution as soon as its arguments pass validation; execution need not wait for the remainder of that Speculator's response.

Each Speculator makes one model-generation round. Tool returns do not feed a second reasoning round within the same Speculator. Local dependencies can instead be implemented inside one native program, such as a script that reads a file, computes a diagnostic, and returns the result. Adaptive exploration requiring another model decision after a tool return must wait for a later eligible acquisition frontier.

Context compaction and swarm restart. Context compaction resets the active exploration frontier. The runtime stops the existing speculative processes and their pending acquisitions, advances the execution epoch, and prevents late results from the previous epoch from entering the Actor context. It reorganizes the disclosure hierarchy into a single flat layer that directly exposes the retained source information, avoiding chains of nested indexes that obscure the underlying observations. The controller then refreshes the task objective and progress state from the compacted context and derives the remaining information needs. Exploration restarts differentially around those updated needs, so speculators pursue the current task state rather than continue from an outdated objective or progress snapshot.

Independent native execution. Acquisition requires tool-host support for independent execution; an invalid or closed scope must never fall back to authoritative execution. The external benchmark/tool host supplies the isolation layer. For each new executable acquisition, it opens a separate work copy with an independent scope identifier and provenance. The copy is created when that call starts, not necessarily when its Speculator context was recorded. Calls from the same Speculator do not share mutable copy state. Native commands may write temporary files or run local experiments inside their copies, but no copy writes or artifacts are merged into x.

Isolation is a host contract over *all relevant mutable resources*. Filesystem operations use private copies, and tools that mutate remote state require independently provisioned services or simulators. A work copy records its creation interval as provenance. Execution returns, errors, prepared arguments, execution intervals, and copy provenance are retained by the runtime. The ledger uses execution intervals as ordering metadata. A diagnostic obtained after a copy-local modification describes that private execution. The Actor uses this historical evidence to choose authoritative actions and continues without joining the acquisition frontier.

A.3 PERSISTENT FUTURES AND EVIDENCE ADMISSION

Work survives request boundaries. A session-owned Future registry $\mathcal{F}$ tracks acquisitions by canonicalized pairs of native tool names and prepared arguments. Identical in-flight acquisitions share work; completed acquisitions can be reused at the same authoritative revision, while new progress permits refreshing them. Each execution retains a distinct observation identity, originating request, launch revision, and review state, so a refresh does not overwrite an older unreviewed result.

This reuse applies to acquisition Speculators only. If the Actor later emits the same native invocation, the host still executes it normally rather than substituting the Future's result.

Invocation equality identifies reusable acquisition work. An in-flight execution retains its original work copy and launch provenance across authoritative revisions. Progress enables a fresh acquisition, while admission presents retained results as historical observations. This separation keeps execution reuse, refresh eligibility, and the Actor's assessment of applicability explicit.

Snapshot and review semantics. At a request boundary, $\mathcal{F}_t$ in Equation 3 is a snapshot of the session registry $\mathcal{F}$. Completion changes readiness; request boundaries change review state. A result that completes after the snapshot remains unreviewed and can enter the next eligible snapshot. An empty $\mathcal{R}_t$ leaves the ledger unchanged and yields empty evidence.

In Equation 4, Sort orders results by completion time with deterministic tie-breaking. Admit processes them sequentially against the ledger, so later results in the same batch see earlier admission decisions. Each processed observation ID is marked reviewed once, including errors and results fully suppressed by structural filtering. The operation can emit multiple evidence units at a boundary. $\mathcal{L}_t^+$ is the post-admission ledger; subsequent authoritative observations update its source records before the next boundary.

Source-aware record processing. For supported structured outputs, the ledger extracts complete source-addressed units, such as records, rows, or extents, preserving their original values. It incorporates authoritative observations, previously admitted evidence, and other ready results processed at the same boundary. Source keys distinguish authoritative observations from work-copy observations; equality checks compare source-qualified records. Known duplicate values at the current source frontier and observations ordered before newer known values are suppressed. Incomparable conflicting values remain distinguishable. Comparable source versions establish value order; execution intervals provide temporal provenance when source versions are unavailable. The authoritative progress revision controls acquisition renewal.

Unknown formats, images, opaque shell output, ambiguous JSON, and values that cannot be parsed losslessly retain their complete native observation instead of a model-generated summary. Supported record-level filtering carries complete retained units and their provenance. Collection-wide properties such as cardinality, order, and absence belong to the original response as a whole; decisions requiring those properties use the complete response or an authoritative verification. The ledger handles structural repetition and source ordering, while the Actor performs task-level relevance and validity judgments.

Historical delivery and termination. Admitted content is appended to ordinary persistent conversation history and the session log, retaining the tool, arguments, error status, and explicit non-authoritative provenance. It does not impersonate an Actor-issued tool call or replace an authoritative return. Once admitted, a message stays at its historical position rather than being republished each turn. Pending acquisitions remain active across Actor requests. Termination, cancellation, or an explicit reset stops unfinished work, invalidates late publication through epoch checks, and settles resource cleanup. Algorithm 2 summarizes the event flow.

Registry and admission operations. Algorithm 3 expands acquisition reuse and evidence selection. Canonicalization sorts object keys recursively and preserves array order. The ledger registers source aliases from the ready batch before processing observations, allowing records with recognized source identities to be compared across calls. Native Actor observations update the same ledger through their ordinary completion path.

A.4 Prompt and Evidence-Message Protocol

Speculator input. Figure 6 gives the complete acquisition-role prompt and its dynamic inputs. The controller converts the post-admission history into speculator-facing user messages, each carrying its original role and mainline ownership. Text and images are preserved, mainline tool calls are serialized as observed actions, and model-specific opaque reasoning blocks are omitted. Each speculator sees one tool with its native name and description and the assigned variant schema. The Actor retains the task harness's native system prompt.

Algorithm 2 PERSEUS: event-driven non-blocking exploration with a speculative swarm

Initialize history H, ledger $\mathcal{L}$, Future registry $\mathcal{F}$, revision r, epoch e.
Before Actor request t:
  Snapshot $\mathcal{R}_t$ using Equation 3; do not await pending work.
  Apply Equation 4; form $\bar{H}_t = H_t \oplus \Delta_t$; mark inspected results reviewed.
  If refresh is eligible, construct Meta-Tool variants and launch eligible Speculators in the background.
  Start the native Actor request using the augmented history (Equation 1).
On a complete streamed Speculator call:
  Validate the assigned variant schema; prepare arguments and validate the native schema.
  Reuse identical in-flight or same-revision completed acquisition if available.
  Otherwise capture epoch e, register a Future, and execute in a new work copy.
On acquisition completion:
  Settle copy cleanup; record the native result or error and provenance.
  Make the observation ready only if its captured epoch equals e and the session is active.
On authoritative tool completion or genuine user input:
  Commit through the native history path; update authoritative records and revision r.
On context compaction:
  Advance e; stop active Speculators and pending acquisitions; settle cleanup.
  Flatten the disclosure hierarchy; refresh the task objective and progress from compacted context.
  Update r and restart exploration for the remaining information needs.
On task termination, cancellation, or reset:
  Advance $e \leftarrow e + 1$, cancel pending acquisitions, and settle cleanup.

Algorithm 3 Persistent Future reuse and source-aware evidence admission

Acquire(T, u, t, r)**, after variant and native validation:**
  $k \leftarrow \mathrm{Canonical}(T, u)$; $f \leftarrow \mathcal{F}[k]$.
if f exists and (unfinished(f) or revision$(f) = r$) **then**
    return f to the requesting Speculator.
end if
  Preserve an older unreviewed f under its observation identity.
  Register a new Future with origin t, revision r, captured epoch, and review flag false.
  Start host-managed acquisition asynchronously; publish completion under the epoch check.
Before Actor request t:
  $R \leftarrow$ completed, unreviewed Futures with origin less than t.
  Sort R by completion time, then observation ID; register source aliases from R.
  $\Delta \leftarrow [\,]$.
for each $f \in R$ **do**
  $o \leftarrow \mathrm{Observation}(f)$; $U \leftarrow [\,]$.
  If o is already processed, mark f reviewed and continue.
  for each complete unit $u \in \mathrm{Extract}(o)$ **do**
    $a \leftarrow \mathrm{Canonical}(u.\mathrm{source}, u.\mathrm{location}, u.\mathrm{kind})$.
    Compare o with the known frontier at a using source versions or execution intervals.
    if o is older than a known observation at a **then**
      Suppress u.
    else
      Retain u in U if its complete value is new; record overlapping conflicts.
      Update the frontier, replacing older entries and retaining incomparable values.
    end if
  end for
  Mark o processed and f reviewed.
  if $U \neq \emptyset$ **then**
    Append the historical-evidence message for (o, U) to Δ.
  end if
end for
Append Δ to Actor history; continue the next request while pending Futures run.

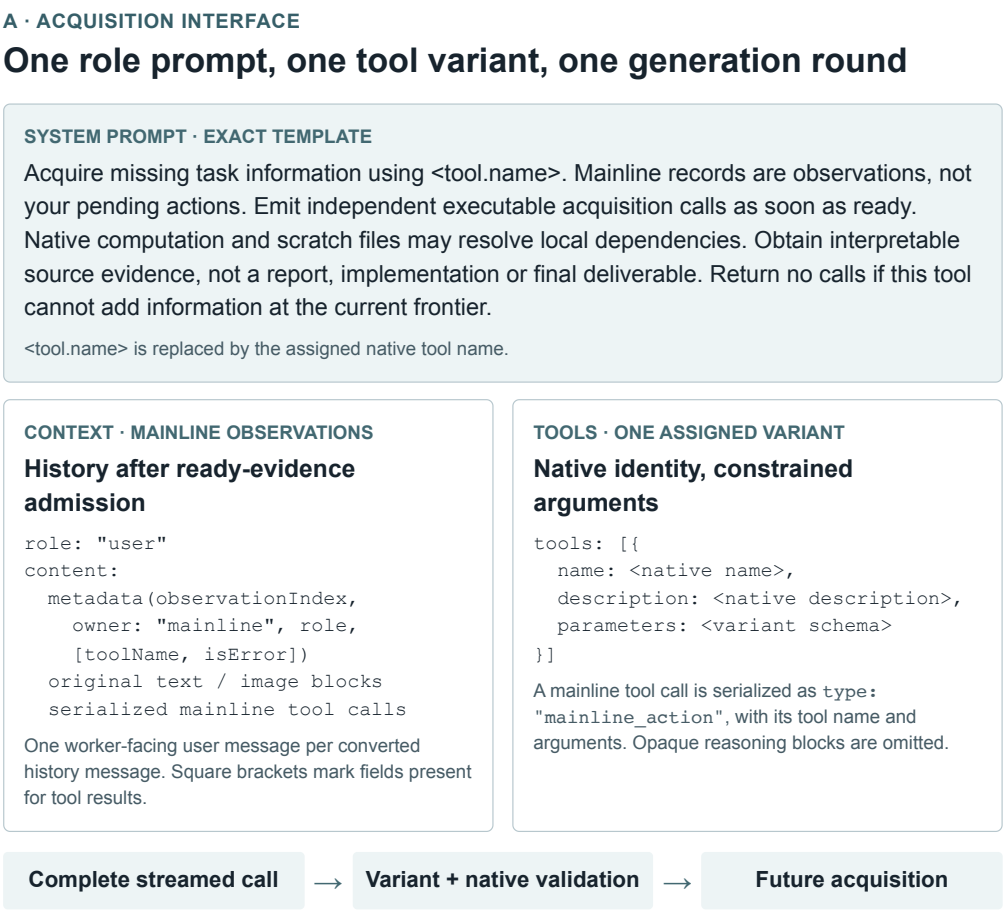


Figure 6: **Speculator prompt and input construction.** The role prompt is the implementation's complete template. The history and tool panels specify its dynamic inputs; calls enter acquisition as their complete arguments become available.

Evidence delivery. Figure 7 specifies the channel from completed acquisitions to the Actor. Each admitted message uses the user role, a historical-observation header, and complete retained content. Structured records carry source, location, kind, pointer, and value; opaque observations carry their original native content. The session stores the original observation and retained units with the message so the ledger can be reconstructed from persisted history. Observation intervals remain runtime metadata used for ordering. This interface exposes source evidence for the Actor's next decision and leaves unfinished acquisitions active.

A.5 Execution Properties and Resource Trade-offs

Direct execution isolation. Assume the host isolates every relevant mutable resource, auxiliary execution can modify only its private scope, and no merge path exists. Then any interleaving of auxiliary executions leaves the authoritative state unchanged *through direct auxiliary effects*. Each auxiliary transition preserves the authoritative projection of the joint state; composition preserves this property. Subsequent changes to the Actor's decisions occur through the evidence channel in Equation 1, followed by native authoritative execution. The contract therefore separates direct state mutation from information-mediated decision making.



Figure 7: **Future ledger and evidence-message protocol.** Ready Futures pass through source-aware admission before entering persistent Actor history. Angle-bracket fields denote runtime values. The two payload forms preserve complete structured units or native observations.

At-most-once admission and no acquisition barrier. Within a normal, crash-free run, processed observation identifiers and reviewed Futures give at-most-once admission for each observation ID. Distinct IDs undergo the ledger's source-aware structural checks. Equation 3 allows the next Actor request to proceed while acquisitions remain unfinished. Admission processing, longer model inputs, copy management, resource contention, and final cleanup contribute to the remaining runtime overhead.

A local utility condition. To distinguish useful evidence from a mere acquisition hit, consider a fixed decision boundary where a baseline requires a downstream information-gathering segment of duration L, with all other comparable work taking T_0. Let p be the joint probability that an auxiliary observation arrives in time, is applicable and relevant, and actually eliminates that segment. Let h be expected additional latency from acquisition-related overhead, including any contention and

cleanup attributable to this comparison. Under this local model,

$$\begin{aligned} \mathbb{E}[T_{\text{base}}] &= T_0 + L, \\ \mathbb{E}[T_{\text{PERSEUS}}] &= T_0 + (1-p)L + h, \\ \mathbb{E}[T_{\text{base}} - T_{\text{PERSEUS}}] &= pL - h. \end{aligned} \tag{6}$$

Net local latency benefit requires $pL > h$: expected downstream work saved must exceed acquisition-related overhead. Productive acquisition therefore depends on both the information obtained and its delivery before demand. Task score separately measures improvements from decision changes, including cases that introduce additional useful verification.

Additional evidence can reveal a missing constraint and lead to more necessary verification, improving completion quality while increasing execution time. Irrelevant, stale, or copy-specific observations also impose interpretation and verification work, making provenance and acquisition budgeting important parts of the design. Speculator generation, native tool execution, and copy management contribute to runtime work. Acquisition width and persistence determine how this work is distributed across Actor requests, shaping the quality–latency trade-off.

B Evaluation Protocol and Supplementary Results

B.1 Benchmark Coverage and Aggregation

Benchmark coverage. PERSEUS uses 183 tasks: 36 AutomationBench (Shepard & Salimans, 2026), 60 τ^2-Bench (Barres et al., 2026), 31 Terminal-Bench (Merrill et al., 2026), and 56 GAIA (Mialon et al., 2024). Terminal-Bench uses revision 2.1. Table 1 compares PERSEUS with 13 baselines. The supplementary BFCL V4 evaluation comprises 156 function-calling cases and four methods (Appendix B.6).

Task selection. Within each benchmark, we screened candidate tasks for five properties: (1) demands within the practical capability range of the Actor model; (2) an environment compatible with, and portable to, the evaluation host; (3) continued availability of required dynamic resources; (4) sustained interaction over a sufficiently long task lifecycle, excluding simple single-round dispatch and short retrieval-only workflows; and (5) a tractable validation duration, excluding cases such as model training or other processes expected to require more than 3000 s. Selected tasks use the 1800 s per-run execution budget in Table 3. Instance-level execution checks and manual verification then determined the main-evaluation subsets. The selected task IDs will be released through GitHub in the subsequent artifact-release stage.

Repeated-attempt aggregation. The main comparison reports mean TS and execution time across three runs for the repeated-run evaluations. Scaling and component ablations also use three attempts per task. Each reported quality metric averages the three corresponding evaluator outcomes for each task, then averages across tasks. Mean execution time uses the same attempts. The 60-task τ^2-Bench evaluation therefore contains 180 attempts per configuration.

Temporal-analysis cohorts. Figure 2 uses the 549 task runs from the main evaluation. Pooled includes the 493 runs with at least one injection of speculative evidence into the Actor history. Standard comprises 162 runs in which admitted information was carried forward in later reasoning, decisions, or task execution. We first match text blocks from admitted evidence against subsequent Actor turns, then manually inspect the matched trajectory segments for continuity of reasoning and substantive reuse of the information. Manual review validates the candidate links by checking how later reasoning or decisions use the admitted information. Cohort sizes count task runs, with each attempt counted separately.

Realized swarm concurrency. For each task run in the main evaluation, we record the maximum number of simultaneously active Speculators and average these per-run peaks across the 549 runs. The mean peak swarm concurrency is 9.37, alongside one Actor. The scaling sweep applies explicit concurrency caps up to eight to examine how restricting exploratory capacity affects performance.

Injection frequencies and density smoothing. For each cohort g, let $c_{g,t}$ be the number of evidence injections at Actor turn t and $C_g = \sum_t c_{g,t}$ the total injection count. Open markers and vertical stems show $100c_{g,t}/C_g$, so the discrete frequencies sum to 100% within each cohort. The denominator counts injection events; the cohort sizes of 162 and 493 count task runs. Continuous curves use Gaussian kernel density estimation weighted by $c_{g,t}$, with a common bandwidth of one turn and left-boundary correction. A density estimate $\hat{f}_g(x)$ is displayed as $100\hat{f}_g(x)\Delta x$ with $\Delta x = 1$ turn, placing it on a scale comparable to the percentage frequencies. The probability assigned to a turn-width interval is the integral of the density over that interval; the curve height is a scaled continuous density rather than an exact probability at an integer turn.

Precision and aggregation. The Overall columns in Table 1 pool the four benchmarks with task counts 36, 60, 31, and 56. Overall TS weights each benchmark by its task count; overall ES divides total success by total execution time. Aggregation uses the recorded means before display rounding. Rankings and ties use displayed precision, with decimal ties rounded to even.

B.2 Models and Baselines

Model and effort assignment. PERSEUS uses GPT-5.6 Terra as the Actor and GPT-5.6 Luna as the Speculator model, both at high reasoning effort. This configuration applies to the main comparison, BFCL, and component ablations. The scaling experiments use the same models and vary the labeled acquisition parameter. The harness experiment includes Terra+Luna and Claude Opus 5+Sonnet 5 model groups (Figure 4). Appendix C illustrates the execution protocol through a statement-delivery workflow. Role-asymmetric multi-agent baselines assign Terra to the principal decision maker or coordinator and Luna to auxiliary roles. Single-agent baselines and role-symmetric systems use Terra for all agent roles.

Execution configuration. Table 3 specifies the default PERSEUS configuration. Exploration width follows the available Meta-Tool variants: every eligible variant receives one Speculator, with no additional width cap. A task run ends on normal completion, at 50 Actor response turns, or after 1800 s. The evaluation host enforces these episode budgets and provides independent acquisition environments. The evaluated configuration imposes no output-token budget, prompt slicing, or evidence-count limit and disables automatic context compaction; admitted observations remain in history. Appendix B.4 specifies the width and effort sweeps.

Table 3: Default PERSEUS execution configuration.

Setting	Value
Actor / Speculator	GPT-5.6 Terra / GPT-5.6 Luna; high / high effort
Worker organization	n Speculators for n eligible Meta-Tool variants, plus one Actor; uncapped
Acquisition renewal	Initial request and new authoritative progress
Future lifetime	Persistent across Actor requests
Episode limits	50 Actor response turns; 1800 seconds
Output-token budget	None imposed by the runtime
Stopping conditions	Normal completion, turn limit, or time limit
Repeated attempts	3 per task

Baselines. We implement the baselines with reference to their official source code, following the default method configurations and the model assignments above. ReAct is the single-agent baseline (Yao et al., 2023). CMAS and DMAS represent centralized and decentralized multi-agent coordination, respectively, with AutoGen's manager-mediated group chat (Wu et al., 2024) and AgentNet's local routing (Yang et al., 2025) as their architectural references.

Plan-and-Execute separates planning from tool-based execution, following the plan-first, solve-second strategy of Plan-and-Solve (Wang et al., 2023). Multi-Persona denotes multi-persona self-collaboration (Wang et al., 2024), and SA denotes Speculative Actions (Ye et al., 2026). The remaining baselines are MemGPT (Packer et al., 2023), AFlow (Zhang et al., 2025a), Dynamic LLM-Powered Agent Network (DyLAN) (Liu et al., 2024), Magentic-One (Fourney et al., 2024), LLMCompiler (Kim et al., 2024), and Reasoning WithOut Observation (ReWOO) (Xu et al., 2023).

B.3 End-to-End Task Success and Execution Speed

Execution speed. For end-to-end evaluation, let $s_{i,r} \in \{0, 1\}$ be the benchmark's success indicator and $t_{i,r}$ the execution time in seconds for case i and attempt r. For N cases and R attempts per case, define

$$\bar{s} = \frac{1}{NR} \sum_{i=1}^{N} \sum_{r=1}^{R} s_{i,r}, \qquad \overline{T} = \frac{1}{NR} \sum_{i=1}^{N} \sum_{r=1}^{R} t_{i,r}. \tag{7}$$

End-to-end task success is $\mathrm{TS} = 100\bar{s}$ in percent. Execution speed is the ratio of mean success to mean execution time:

$$\mathrm{ES} = v_{\mathrm{eff}} = \frac{\bar{s}}{\overline{T}} = \frac{\sum_{i,r} s_{i,r}}{\sum_{i,r} t_{i,r}}, \qquad \overline{T} > 0. \tag{8}$$

For end-to-end evaluation, ES measures aggregate success per second. The displayed value in $10^{-3}\,\mathrm{s}^{-1}$ is $1000\bar{s}/\overline{T} = 10\,\mathrm{TS}/\overline{T}$.

Overall aggregation. For benchmark b, let n_b be its task count, $\bar{s}_b$ its mean binary success, and $\overline{T}_b$ its mean execution time. Pooling all $N = \sum_b n_b = 183$ tasks gives

$$\mathrm{TS}_{\mathrm{all}} = 100 \frac{\sum_b n_b \bar{s}_b}{N}, \qquad \mathrm{ES}_{\mathrm{all}} = \frac{\sum_b n_b \bar{s}_b}{\sum_b n_b \overline{T}_b}. \tag{9}$$

Equal repetition counts cancel in these ratios. The table displays $1000\,\mathrm{ES}_{\mathrm{all}}$ in $10^{-3}\,\mathrm{s}^{-1}$.

Execution times. Table 4 reports the execution-time means used to compute ES in Table 1, together with their run-level SD. ES is computed from the ratio of aggregate success to aggregate time, as in Equation 8.

Table 4: Mean execution time in seconds for the main comparison. Entries show mean ± SD across three runs where available.

Method	Automation	τ^2	Terminal	GAIA
ReAct	$223.0_{\pm 29.0}$	$139.2_{\pm 10.9}$	$338.1_{\pm 59.6}$	$275.8_{\pm 28.1}$
Plan-and-Execute	$182.9_{\pm 23.0}$	$146.4_{\pm 12.2}$	$361.1_{\pm 92.9}$	$188.6_{\pm 7.4}$
ReWOO	$201.3_{\pm 16.2}$	$297.9_{\pm 17.2}$	$275.3_{\pm 32.5}$	$140.9_{\pm 9.1}$
Multi-Persona	$119.9_{\pm 7.0}$	$128.2_{\pm 3.8}$	$331.6_{\pm 38.2}$	$87.4_{\pm 6.3}$
CMAS	$169.9_{\pm 19.1}$	$145.2_{\pm 10.6}$	$463.4_{\pm 122.8}$	$306.0_{\pm 29.4}$
DyLAN	$1144.7_{\pm 32.9}$	$599.1_{\pm 22.6}$	$886.4_{\pm 13.1}$	$500.5_{\pm 48.7}$
MemGPT	$146.5_{\pm 10.8}$	$112.4_{\pm 5.8}$	$222.0_{\pm 31.6}$	$167.2_{\pm 6.1}$
LLMCompiler	$594.8_{\pm 44.1}$	$173.8_{\pm 18.4}$	$87.2_{\pm 38.2}$	$94.2_{\pm 10.9}$
AFlow	$144.1_{\pm 20.1}$	$123.2_{\pm 4.1}$	$378.0_{\pm 54.2}$	$205.3_{\pm 84.6}$
Magentic-One	$594.8_{\pm 12.1}$	$325.3_{\pm 2.8}$	$650.3_{\pm 68.0}$	$317.7_{\pm 88.3}$
DMAS	$141.7_{\pm 10.8}$	$133.1_{\pm 6.3}$	$246.5_{\pm 30.9}$	$232.6_{\pm 60.4}$
SA	$142.0_{\pm 5.6}$	$117.2_{\pm 16.0}$	$259.1_{\pm 68.9}$	$198.5_{\pm 9.5}$
AsyncFC	$128.3_{\pm 23.7}$	$247.2_{\pm 99.0}$	$299.9_{\pm 105.7}$	$173.6_{\pm 27.1}$
PERSEUS	$193.4_{\pm 19.5}$	$161.5_{\pm 24.7}$	$334.3_{\pm 49.9}$	$208.8_{\pm 21.0}$

B.4 Scaling Setup

The scaling study uses an 85-case long-horizon diagnostic subset drawn from the main evaluation: 24 AutomationBench, 25 Terminal-Bench, 26 GAIA Level 3, and 10 τ^2-Bench cases. The subset targets long-horizon tasks with pronounced task structure and high component activation for mechanism analysis.

Partial-credit scoring. Scaling uses Partial-credit Score (PCS) from each benchmark's native evaluator. Let $c_{i,r} \in [0, 1]$ be that evaluator's score for task i on attempt r. We retain the benchmark's own partial-credit calculation where available; settings without partial credit, including GAIA, contribute their native binary success outcome. All scores are taken directly from benchmark evaluators,

with no additional LLM rubric. Across N tasks and $R = 3$ attempts,

$$\mathrm{PCS} = \frac{1}{NR}\sum_{i=1}^{N}\sum_{r=1}^{R} c_{i,r}, \qquad \mathrm{ES}_{\mathrm{PCS}} = \frac{\mathrm{PCS}}{\overline{T}}. \tag{10}$$

Figure 3 labels this PCS-based rate as ES and displays it in $10^{-3}\,\mathrm{s}^{-1}$. The bars report PCS on its native $[0, 1]$ scale.

The parallelism sweep applies a Speculator concurrency cap of $0, 1, 2, 4, 8$ with high reasoning effort. The effort sweep fixes the Actor at high and varies Speculator effort over `none`, `low`, `medium`, `high`, and `xhigh`, with a no-Speculator control.

B.5 Harness Comparison and Usage Metrics

Figure 4 compares PERSEUS with Claude Code and Claude Workflow under Opus+Sonnet, and with Codex and Codex MA under Terra+Luna, on AutomationBench. Each comparison uses the methods in the same model group. Task-success error bars in Figure 4 show 95% Wilson confidence intervals.

Response turns measure sequential interaction. Token usage is computed from complete usage records for successfully completed Actor and Speculator model requests.

Equivalent tokens and price conversion. Equivalent tokens (Eq. Tokens) express model usage in Actor-model token units using official input and output list prices. For model m and token category $c \in \{\mathrm{in}, \mathrm{out}\}$, let $N_{m,c}$ denote its recorded token count and $p_{m,c}$ its price per million tokens. With Actor model A as the reference, we compute

$$N_{\mathrm{eq}} = \sum_{m}\sum_{c\in\{\mathrm{in},\mathrm{out}\}} N_{m,c}\frac{p_{m,c}}{p_{A,c}}. \tag{11}$$

Input and output counts are converted separately and then summed. The reference model is Terra for the GPT group and Opus for the Claude group, with the same conversion applied to all methods within each group.

The adopted standard list prices and conversion weights are shown below (USD per million tokens).[1]

Model	Input price	Output price	Input weight	Output weight
GPT-5.6 Terra	\$2.00	\$12.00	1.00	1.00
GPT-5.6 Luna	\$0.20	\$1.20	0.10	0.10
Claude Opus 5	\$5.00	\$25.00	1.00	1.00
Claude Sonnet 5	\$2.00	\$10.00	0.40	0.40

The input and output ratios coincide within each model pair. Thus, with $N_m = N_{m,\mathrm{in}} + N_{m,\mathrm{out}}$, the conversions simplify to

$$N_{\mathrm{eq}}^{\mathrm{GPT}} = N_{\mathrm{Terra}} + N_{\mathrm{Luna}}/10, \qquad N_{\mathrm{eq}}^{\mathrm{Claude}} = N_{\mathrm{Opus\,5}} + 0.4N_{\mathrm{Sonnet\,5}}.$$

Figure 4 and Figure 8 report equivalent tokens in thousands (K); Table 2 reports millions (M).

B.6 BFCL Function-Calling Results

Evaluation setup. We evaluate PERSEUS, ReAct, CMAS, and DMAS on 156 cases from thirteen BFCL V4 categories (Patil et al., 2025). The tasks cover single-turn function selection, argument construction, call composition, and relevance judgments. Accuracy is the official grader's strict pass fraction, and time is mean agent execution time. PERSEUS uses Terra+Luna with high reasoning effort for both models.

[1] Official prices, accessed September 21, 2026: GPT-5.6 Terra, GPT-5.6 Luna, and Anthropic model pricing.

Task-profile construction. Figure 8 groups the cases into four disjoint task profiles. The *simple* group contains `simple_java`, `simple_javascript`, `simple_python`, and `live_simple` (48 cases). The *relevance* group contains `irrelevance`, `live_irrelevance`, and `live_relevance` (36 cases). The *static* group contains `multiple`, `parallel`, and `parallel_multiple` (36 cases); *live* contains their three `live_` counterparts (36 cases). Group accuracy and mean time aggregate the cases in each group; group ES is the ratio of the corresponding score and time aggregates. Overall results weight the groups by 48, 36, 36, and 36 cases.

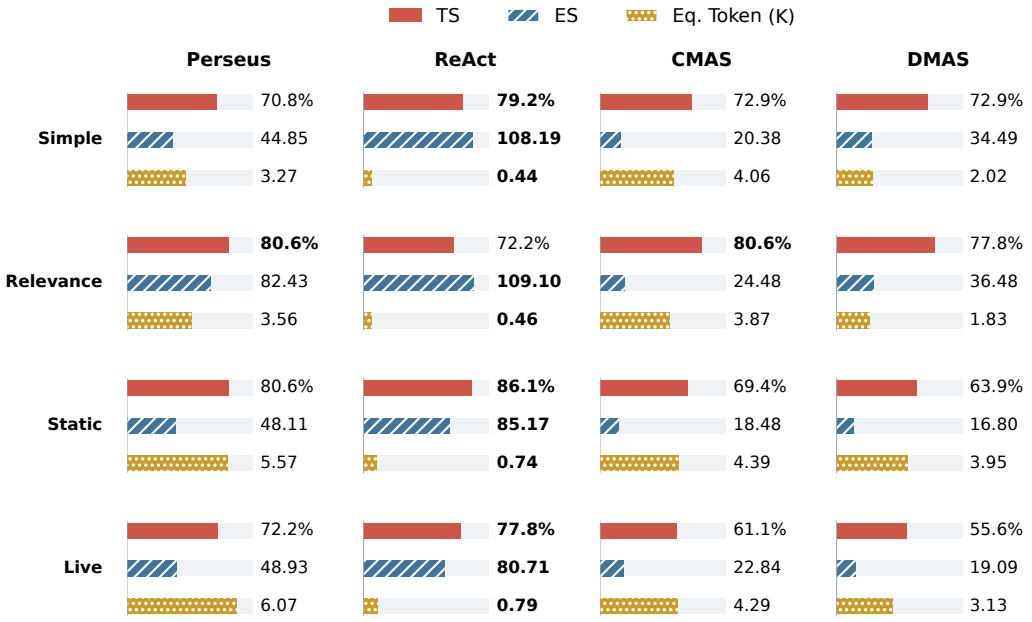


Figure 8: **BFCL evaluation by task profile and architecture (156 cases).** Red bars show strict task accuracy (TS, %); blue hatched bars show ES ($10^{-3}\,\mathrm{s}^{-1}$); gold dotted bars show price-normalized equivalent-token usage (K). Higher TS and ES and lower equivalent-token usage are better. Bar scales are shared across cells within each metric. Group definitions and aggregation appear in Appendix B.6.

Comparison with ReAct: the role of task horizon. PERSEUS improves relevance accuracy by 8.3 percentage points for an additional 3.15 s, while accuracy falls by 8.3 points on simple calls and 5.6 points on both static and live multi-call combinations. ReAct achieves higher ES in all four groups. These short function-calling tasks offer few later decisions in which to reuse speculative observations: supplementary information helps relevance judgments, but acquisition and evidence processing add work to otherwise short execution paths. The aggregate result in Table 5—75.64% accuracy at 14.38 s for PERSEUS versus 78.85% at 8.34 s for ReAct—illustrates the utility condition in Section A.5: useful downstream work saved must cover acquisition overhead.

Comparison with CMAS and DMAS. PERSEUS executes faster than both controls in all four groups. It exceeds DMAS accuracy in three groups and matches or exceeds CMAS accuracy in three; both controls score higher on simple calls. Overall, CMAS achieves 71.15% accuracy at 33.45 s and DMAS 67.95% at 26.92 s, placing PERSEUS ahead on both measures. This comparison connects the short-task results to the cooperation design: the asymmetric evidence channel supports broader acquisition with less execution time than the evaluated multi-agent coordination paths.

B.7 Component Ablations

The component study evaluates 85 long-horizon cases: 24 AutomationBench, 25 Terminal-Bench, 26 GAIA Level 3, and 10 τ^2-Bench cases. We selected these cases from the main-evaluation tasks:

Table 5: BFCL results (156 cases). Each method entry reports strict accuracy (%) / mean agent execution time (s). Bold marks the highest accuracy and lowest time within each row, including ties. Overall results use case-weighted aggregation.

Group	Cases	PERSEUS	ReAct	CMAS	DMAS
Simple	48	70.83 / 15.79	**79.17 / 7.32**	72.92 / 35.79	72.92 / 21.14
Relevance	36	**80.56** / 9.77	72.22 / **6.62**	**80.56** / 32.90	77.78 / 21.32
Static	36	80.56 / 16.74	**86.11 / 10.11**	69.44 / 37.59	63.89 / 38.03
Live	36	72.22 / 14.76	**77.78 / 9.64**	61.11 / 26.75	55.56 / 29.10
Overall	156	75.64 / 14.38	**78.85 / 8.34**	71.15 / 33.45	67.95 / 26.92

a case qualified when at least two of its three runs contained an injection of speculative evidence into the Actor history, with the injection records manually verified. The admission criterion follows the Pooled event definition in Section B.1. This diagnostic subset focuses the ablations on tasks with recurring speculative activity, exposing the roles of exploration structure, concurrency, and persistence.

The study reports end-to-end task success (TS), Partial-credit Score (PCS), mean execution time, Agent Turn, Tool Success Rate, and equivalent-token usage. TS averages binary task-completion outcomes; PCS averages benchmark-native scores, retaining binary outcomes for settings without partial credit, including GAIA. Both metrics average three attempts per case and then average across cases. Table 2 compares each restricted configuration with full PERSEUS.

Supplementary interaction and usage metrics. Agent Turn measures agent response turns. Tool Success Rate measures the percentage of successful tool calls, distinct from task-level TS and PCS. ES follows Equation 8: mean end-to-end success divided by mean execution time, reported in $10^{-3}\,\mathrm{s}^{-1}$. TS is converted from percentage to a fraction before division. Equivalent-token usage is reported in millions, using the price-ratio normalization defined in Appendix B.5.

Configuration semantics. Disabling Meta-Tool derivation restores one unmodified-schema Speculator per native tool while retaining auxiliary acquisition. Disabling the speculative swarm removes auxiliary Speculators while retaining the Actor's native execution path and native parallelism. Disabling cross-request Futures retains ready observations at the next boundary and cancels unfinished acquisitions, while the source-aware ledger and previously admitted evidence remain active. These controls target acquisition scope, concurrent execution, and unfinished-work lifetime, respectively.

Tool-call outcomes. The full system achieves a tool success rate of 97.2%, compared with 93.9% without persistent Futures. The failed-call share is 2.8% for full PERSEUS and 6.1% without persistent Futures, a relative reduction of 54.1%.

Complementary component roles. Meta-Tool variants organize acquisition, the swarm broadens discovery, and persistent Futures preserve unfinished work for later admission. Full PERSEUS achieves the highest TS, PCS, and ES with the lowest mean execution time (247.1 seconds), supporting the complementary roles of structured acquisition and persistent evidence collection.

B.8 Best-of-n Sampling Comparison

Best-of-n sampling spends additional inference compute on complete candidate trajectories. LLM-as-a-Verifier (Kwok et al., 2026) studies this approach through verifier-based selection, including DeepSeek self-verification. We use an oracle best-of-n comparison to examine the alternative of spending additional compute on repeated ReAct execution, alongside PERSEUS's acquisition of evidence within a continuing Actor trajectory.

Budget and selection. For benchmark b, let $C_{P,b}$ and $C_{R,b}$ denote the three-run mean equivalent-token totals of PERSEUS and ReAct. The pooled ratio is $\sum_b C_{P,b} / \sum_b C_{R,b} = 1.122$; the equal-benchmark mean ratio is $\frac{1}{4}\sum_b C_{P,b}/C_{R,b} = 1.685$. The ReAct mean bar follows the main table, while ReAct@1 takes the first recorded run on each benchmark. ReAct@2 averages per-case oracle success across all three run pairs, and ReAct@3 uses the three-run per-case oracle. For parallel attempts, wall time is the maximum of their benchmark-run mean durations, averaged over the three pairs for @2 and pooled across benchmarks by case count. Every candidate contributes to equivalent-token usage; ES divides pooled success by pooled parallel wall time.

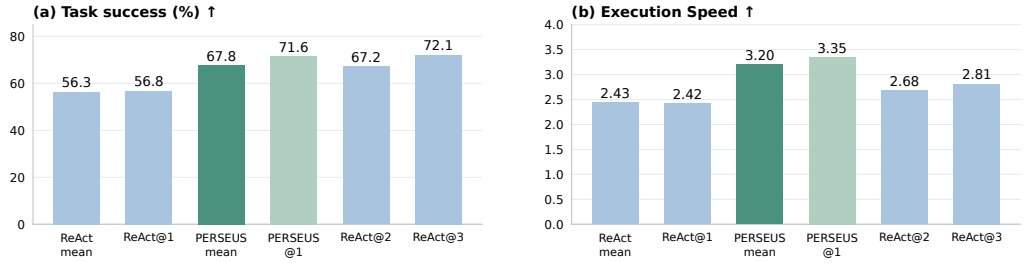


Figure 9: **Best-of-n sampling and PERSEUS.** Task success (left) and Execution Speed (right, $10^{-3}\,\mathrm{s}^{-1}$) across 183 tasks. Bars follow increasing price-normalized usage cost. PERSEUS combines higher success than ReAct mean with higher ES; additional ReAct attempts improve oracle success at greater cumulative cost. Selection and accounting follow the accompanying settings.

Usage and selected-run times. Accounting includes input, output, and cache reads in Actor-model equivalent-token units. Relative to ReAct mean, the six budgets are 1.000, 1.015, 1.122, 1.122, 2.000, and 3.000; both PERSEUS bars use its mean usage. The PERSEUS@1 records are 80.00% / 189.5 s on τ^2, 73.21% / 194.14 s on GAIA, 41.67% / 172.6 s on Automation, and 87.10% / 344.5 s on Terminal. Pooling their 131 successes over 183 tasks gives 71.58% TS; dividing by 39,134.94 task-seconds gives 3.35 ES.

Cooperation and cost. PERSEUS allocates more tokens to inexpensive Speculators while reducing calls to the expensive Actor. At 1.122 times the ReAct mean normalized cost, its mean TS rises from 56.3% to 67.8% and ES from 2.43 to 3.20. It also achieves slightly higher TS than ReAct@2 (67.2%) at 56.1% of the normalized cost, with ES 3.20 versus 2.68 under parallel timing. PERSEUS@1 reaches 71.6% TS and 3.35 ES; ReAct@3 reaches 72.1% TS and 2.81 ES. The comparison shows the benefit of investing auxiliary computation in evidence for one continuing Actor trajectory: asynchronous exploration improves the observed success–cost trade-off while authoritative reasoning and execution advance together.

C Case Study: Asynchronous Speculative Cooperation

Asynchronous exploration under one task authority. PERSEUS combines internal action-space decomposition, asynchronous speculative execution, and a persistent evidence ledger around a single authoritative Actor. We examine the speculative swarm runtime on AutomationBench `finance:4073`, a monthly account-statement workflow illustrating the PERSEUS execution protocol. The case illustrates the complete lifecycle of exploration, evidence admission, and authoritative execution.

C.1 Task and Information Dependencies

The task requires sending statements with unpaid invoices, balances, and aging information while respecting the organization's distribution policy. Three native tools, `api_search`, `api_fetch`, and `base64_encode`, expose a broad invocation space spanning service discovery, policy retrieval, accounting queries, and statement delivery.

The case progresses from interface discovery and distribution-policy retrieval to Wave accounting records and a statement sent through Gmail. Independent observations can be acquired in advance, whereas customer-specific queries and the final send must respect their data dependencies. Speculator branches execute in independent work copies; their observations are evidence, not authoritative commits. The Actor retains responsibility for the actual statement delivery. The recorded episode finishes with an official task score of 1.0: the authoritative Gmail message contains the required balance and invoice numbers.

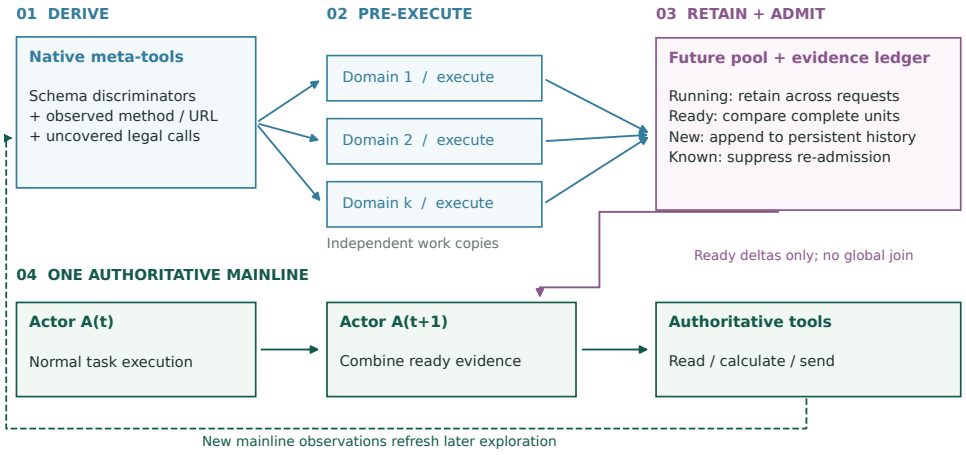


Figure 10: **One authority with asynchronous exploratory branches.** The runtime derives internal action domains from the native tool schemas and observed invocation heads. Branches execute independently and feed a persistent Future pool and evidence ledger. Only ready, newly admitted evidence enters the Actor history; unfinished work can outlive multiple requests. Solid arrows carry work or evidence; the dashed loop carries new authoritative observations.

C.2 Three Complementary Mechanisms

Internal action domains. Schema discriminators and observed method–endpoint heads define disjoint internal execution domains, with a complement domain for uncovered legal invocations. The Actor sees no extra tools.

Asynchronous speculative swarm. Branches explore while the Actor continues ordinary work. Runtime dispatch and admission require no manager LLM, peer negotiation, or global speculator-join barrier. Later decisions can consume observations not individually requested by the Actor.

Persistent Futures and evidence. The Future pool retains unfinished and completed pre-executions across request boundaries. The ledger separately tracks observed content and provenance, admitting complete new units to persistent history while suppressing recognized duplicates. This separates the lifetime of exploratory work from the duration of a single model request.

C.3 Global Execution Sequence

Figure 11 summarizes the case through its task stages: discover interfaces, obtain policy, establish accounting records, complete invoice details, and deliver the statement. The mainline advances while independent exploration contributes ready evidence through the Future ledger.

Open the frontier, then refine it. At task entry, schema-derived domains expose alternative invocations behind the native tools. The Actor searches for usable accounting and messaging interfaces while branches perform local exploration. As the mainline reads the policy and locates Wave, observed method–endpoint heads refine the available domains. Later exploration uses the updated mainline context, not an unrelated task plan. The runtime can launch a new wave when the authoritative-information-growth policy triggers; deferring a new wave does not discard existing Futures.

Retain unfinished work; admit only a delta. At each Actor boundary, the runtime inspects completed observations without waiting for all branches. Pending Futures remain available for subsequent boundaries. The ledger compares complete evidence units with known content and provenance: newly admitted units enter persistent history, while recognized duplicates are not appended again. Independent-copy results remain marked as non-authoritative evidence rather than being treated as committed mainline state.

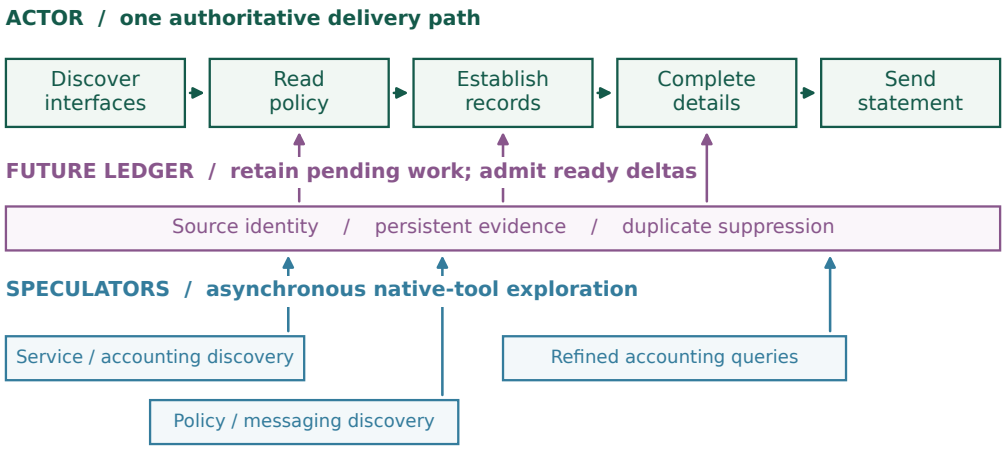


Figure 11: **Task progression and persistent evidence flow.** The Actor follows one delivery path. Exploratory work feeds the ledger, which retains pending work and admits ready deltas at decision boundaries. Horizontal positions represent logical stages, not elapsed time.

Converge on one delivery path. The Actor establishes the business, invoice, and customer records, issues the required detail queries, and encodes and sends the statement. Parallel branches do not select another task owner or independently commit the delivery. The Actor's completion decision closes the remaining exploratory work. Thus the workflow expands its observation frontier while retaining one authoritative state trajectory.

Logical folding. When independent observations are already available, a later Actor decision can combine them rather than alternate between requesting and inspecting each one. The folded object is the information-acquisition chain, not an entire Actor turn taken over by another agent. Queries that require a newly established business ID still depend on that ID. The architecture preserves genuine task dependencies while moving eligible exploration ahead of demand.

C.4 Evidence Flow and Logical Folding

From acquisition to persistent evidence. During interface discovery, a Speculator executes `api_search({"query":"invoices unpaid"})` in an independent work copy. Its native response includes `wave.invoices.list`, with method `POST` and a GraphQL request schema requiring a business ID. The ledger admits complete catalog records, retaining their source association. When another branch returns the same Wave entries, their identities allow the ledger to suppress repeated admission while keeping newly discovered entries.

Table 6: **Evidence flow in the statement-delivery case.**

Stage	Information or action	Runtime role
Acquire	Search for unpaid-invoice interfaces; receive Wave catalog entries	Execute a native tool through an independent exploratory branch
Admit	Complete catalog entries become available to later decisions	Preserve source identity and append new evidence
Deduplicate	Wave entries reappear in another search result	Retain new records without re-appending recognized duplicates
Execute	Obtain the business ID, invoices, and customer details	Keep data-dependent queries on the authoritative mainline
Deliver	Encode and send the statement with the required balance and invoices	Commit through the Actor and complete the task

The Actor and exploratory branches can discover overlapping interfaces. The ledger preserves evidence identity across these encounters, while the Actor combines available information with the business records needed for delivery.

Serial acquisition and speculative acquisition. Figure 5 uses the same task dependencies to illustrate the folding opportunity. A demand-driven serial schedule first discovers the policy interface and reads its policy, then discovers the accounting interface. PERSEUS can explore these independent branches concurrently and admit their ready observations at one decision boundary. Reading the policy after discovering its interface requires a later acquisition frontier inside the same speculator. Business-ID-dependent invoice queries, policy application, and delivery remain ordered.

C.5 Discussion: Three Representative Design Alternatives

The case highlights three structural distinctions: pre-decision exploration relative to ReAct, evidence-mediated cooperation relative to centralized MAS, and a single authority relative to decentralized MAS. Magentic-One and AgentNet serve as representative reference points for the latter two designs.

ReAct: move observation acquisition ahead of demand. ReAct interleaves reasoning with task actions and environmental feedback (Yao et al., 2023). PERSEUS preserves this authoritative loop while acquiring independent observations ahead of the Actor's requests. In the statement task, service discovery and policy retrieval proceed concurrently, making complementary evidence available to a later decision. The Actor can then combine those observations in one reasoning step while preserving the order of data-dependent queries.

Centralized MAS: evidence-mediated cooperation. Magentic-One illustrates manager-led planning, delegation, and integration with progress tracking (Fourney et al., 2024). PERSEUS assigns exploration through runtime scheduling and delivers completed observations through the ledger. In the case, the interface search contributes evidence while the Actor advances the delivery workflow; the search's unfinished branches remain active across later decisions. The cooperation channel carries observations directly into the continuing task trajectory.

Decentralized MAS: keep authority separate from breadth. AgentNet distributes task routing and specialization across a dynamic network (Yang et al., 2025). PERSEUS distributes acquisition while the Actor retains state commitment and completion. Exploratory branches discover interfaces and constraints; the same Actor selects the recipient, retrieves dependent records, and commits delivery. Each Speculator uses one model-generation round, and later Actor progress can launch fresh acquisition. This connects broad discovery to a single authoritative workflow.

Table 7: **Structural contrast with representative agent designs.**

Design	Information / coordination path	PERSEUS distinction
ReAct	Acquire observations after choosing corresponding actions	Acquire independent evidence before demand
Centralized MAS	Manager-mediated delegation and integration	Runtime dispatch; consume ready deltas without a collective join
Decentralized MAS	Distributed routing, handoffs, and local decisions	Many exploratory branches; one authoritative commit path

Design contribution. The case connects the three parts of the design: Meta-Tool domains distribute exploration, persistent Futures carry acquisition across decisions, and the ledger delivers new evidence with its provenance. Their asynchronous cooperation brings complementary observations into one Actor trajectory, broadening discovery while preserving the task's state and dependency order.