TUM SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY

TECHNICAL UNIVERSITY OF MUNICH

Technical Report

# Agentic Business AI: Orchestrator-less Agentic Systems

**Bryan Alvin, Hamza Chaouki, Paul Friedrich Dallwig, Moritz Hjalmar Wiking Grüss, Xavier Igor Kamsu Nzodoum, Mohamed Nejjar**

| | |
|---|---|
| **Supervisor:** | **Prof. Dr. Ingo Weber, Prof. Dr. Helmut Krcmar** |
| **Advisors:** | **Ph.D. Hans Weytjens, Alisa Silke Mehler** |
| **Submission Date:** | **17.07.2026** |

# Acknowledgments

We thank the Chair of Information Systems and Business Process Management as well as the Chair of Information System Development and Operation for their ongoing support and valuable feedback. We also appreciate the people at SAP for their role as our customer.

**Declaration of AI Assistance**
We acknowledge the use of generative AI tools, specifically Claude Code and GitHub Copilot/-Codex, during the development of this project. These tools were used to assist in building the frontend components, implementing specific backend functions to optimize development time, and wiring together various system modules. Additionally, AI assistance was used to refine the linguistic clarity and wording of this technical report. All core ideas, system architectures, and final interpretations remain entirely our own.

# Contents

**Acknowledgments** **i**

**1. Introduction** **1**
1.1. From Language Models to Agents . . . 1
1.2. Agentic Systems and the Orchestrator . . . 1
1.3. Research Questions . . . 2
1.4. Contributions . . . 2

**2. Motivation** **3**
2.1. What Orchestrators Do Well . . . 3
2.2. Where Orchestrators Fail . . . 3
2.3. What a Fix Must Provide . . . 4

**3. System Design** **6**
3.1. Design Decisions . . . 6
3.2. Architectural Overview . . . 9
3.3. From Local to Cloud . . . 11
3.4. What the Platform Leaves Open . . . 12

**4. Naive Implementation: The Broadcast Baseline** **13**
4.1. Execution Model . . . 13
4.2. Engineering Properties . . . 14
4.3. Observed Limitations . . . 15

**5. Improving the Baseline** **16**
5.1. From Static to Dynamic Topology . . . 16
5.2. Consensus Mechanisms . . . 17
5.3. Moving Onto Bidding . . . 19
5.4. What Remained Unsolved . . . 21

**6. The Market-Hive** **22**
6.1. The Market Maker . . . 24
6.2. The Auction . . . 25
6.3. Promotion . . . 26
6.4. Termination . . . 30
6.5. Design Iteration: What a Live Run Taught Us . . . 32
6.6. Limitations . . . 33

**7. Evaluation** **34**
7.1. Benchmarking Up to the Midterm . . . 34

7.2. Evaluation on External Benchmarks . . . 37
7.3. Conclusion . . . 41

**8. Future Outlook** **42**
8.1. Limitations . . . 42
8.2. Out of Scope . . . 42
8.3. Directions . . . 43

**9. Reflection** **44**
9.1. On the Architecture . . . 44
9.2. On the Process . . . 44
9.3. Personal Perspectives . . . 45
9.4. Closing . . . 47

**A. Appendix** **48**
A.1. Configuration Constants . . . 48
A.2. API Surface . . . 48
A.3. Database Schema Model . . . 52
A.4. User Tour: Mycellium . . . 52

**Bibliography** **59**

# 1. Introduction

## 1.1. From Language Models to Agents

Large language models (LLMs) are the first general-purpose engines capable of deliberate, step-by-step problem solving across most domains. Built on the transformer architecture [1] and trained on a massive web corpus, they have shown competence at tasks they were never explicitly trained for through nothing but natural-language instructions [2, 3].

Instruction tuning aligned that capability with user intent [4], and prompting techniques such as chain-of-thought showed that models can be led through multi-step reasoning rather than single-shot pattern completion [5]. In the industry the effect was felt instantly: drafting, summarization, classification, and document analysis, tasks that previously required either specialized models or human effort, became almost instantaneous through API calls.

Consequently, the scope of LLMs has evolved beyond basic information retrieval. These models have been deployed in complex, dynamic workflows via Retrieval-Augmented Generation (RAG) for robust knowledge grounding [6] and programmatic API integration within different enterprise systems [7, 8].

A model that only *talks*, however, is limited to whatever fits in its context window. The step that turned language models into *agents* was expanding the environment beyond a mere input field: letting the model complete its reasoning with actions, calling search engines, databases, code interpreters, and APIs, and observe the results [9]. Models can learn to invoke tools [7], ground their claims in retrieved documents [6], and maintain memory across extended tasks [10]. An *agent*, in the sense used throughout this report, is therefore an LLM configured with an identity (a system prompt), private knowledge (attached files), and a set of tools it may call.

## 1.2. Agentic Systems and the Orchestrator

One agent, however well equipped, hits a ceiling on tasks that touch domains of expertise. For example, a commercial contract review touches law, finance, and regulation whereas an audit preparation touches processes, controls, and reporting. This triggered the introduction of *agentic systems*: compositions of several specialized agents that divide work among themselves [11]. Each agent is given a role (e.g. “you are a contracts lawyer”), roles are connected through conversation or a workflow, and the group solves what none of the agents can solve alone, such systems are demonstrated by frameworks like AutoGen [12], MetaGPT [13], CAMEL [14], and ChatDev [15].

This immediately raises questions about composition: Who decides which agents a task needs?

Who routes intermediate results between them? Who resolves disagreement between two agents? Who decides the group is done and what the final answer is? One answer emerges to answer all four questions: a central *orchestrator*, a privileged component, usually itself an LLM, that decomposes the task, assigns the parts, aggregates the outputs, and terminates the run. Orchestration is the default for good reasons: it is legible (one prompt describes the whole workflow), it is controllable, and it works. Every practical agentic system a business can buy today has one.

This report asks what happens when the orchestrator is removed, and whether the coordination functions it performs can instead *emerge* from the local interactions of the agents themselves. Chapter 2 argues that the orchestrator, for all its strengths, is also the scaling bottleneck, the single point of failure, and the systematic bias of the systems built on it, and that a rich pre-LLM literature suggests decentralised alternatives exist for every one of its functions.

## 1.3. Research Questions

The project was structured around three questions:

**RQ1** Can a group of AI agents converge to a correct shared answer using only local interactions and a pluggable consensus mechanism, without an orchestrating agent?

**RQ2** In what different ways can orchestrator-less communication be represented ?

**RQ3** Can a fully orchestrator-less structure perform well in different business settings without the need of a privileged node's guidance?

## 1.4. Contributions

The concrete contributions of this work are:

1. **A research platform for self-organising agentic systems.** A full-stack, open testbed in which agents, knowledge files, tools, communication topologies, and consensus mechanisms are user-configurable, and in which every run is persisted as a fully replayable trace (Chapter 3). The implementation is available in GitHub.

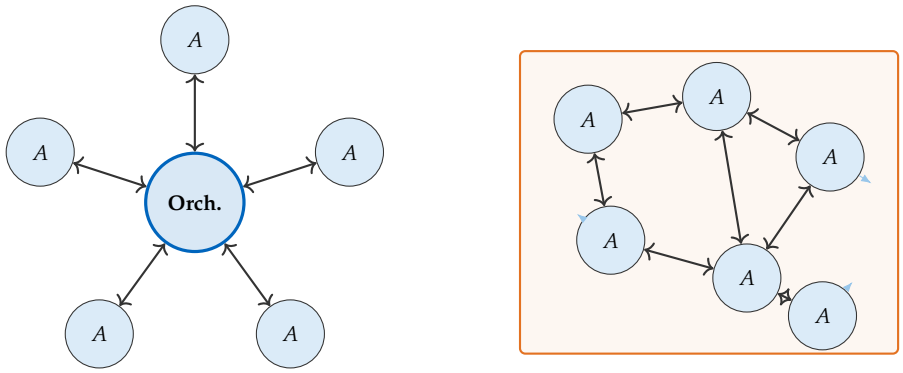
2. **Three different ways to represent self-organisation.** Starting from a baseline broadcast-and-discuss model over static graphs, we introduce dynamic topology rewiring and pluggable consensus mechanisms (trust-weighted opinion pooling and plurality voting), then evolve to the full Market-Hive architecture : an orchestrator-less system where agents bid for tasks based on reputation, deliberate on a shared message board, and reach consensus through emergent agreement patterns, all without a central controller or external judge. The system includes mechanisms for resolving deadlocks and ensuring efficient termination. (Chapters 4–6).

# 2. Motivation

The introduction ended by claiming that orchestrators are one of the bottlenecks of current agentic systems. This chapter delves into that in three steps. First of all, it gives orchestration its due by going over its strengths. Secondly, it examine where they fail. Finally, through literature that precede LLMs, it goes over what potential solutions could look like, motivating the design decisions of Chapter 5 and 6.

## 2.1. What Orchestrators Do Well

An orchestrated system concentrates four coordination functions in one place: *decomposition* (splitting the task), *allocation* (assigning parts to agents), *aggregation* (merging results, resolving conflicts), and *termination* (declaring the work done). Centralising them has genuine strengths, and any honest alternative has to concede them up front.




*(a) Orchestrated: all coordination traffic crosses one privileged hub*

*(b) Orchestrator-less: agents coordinate directly through local interactions*

Figure 2.1.: The two coordination patterns compared in this report. In (a) a privileged component decomposes, routes, judges, and terminates. In (b) those functions must instead emerge from local interactions among agents, with no privileged coordinator.

## 2.2. Where Orchestrators Fail

The orchestrator-controlled structures have five failure modes. All of them follow from the topology of Figure 2.1(a).

1. **Single point of failure.** If the orchestrator mis-plans, stalls, or crashes, the system fails, no matter how capable the agents are. Its errors are *silent*: a decomposition that omits

the regulatory angle for the orchestrator produces a confident answer with a regulatory hole, and no agent was ever asked the question that would have exposed it.

2. **Unflexible in vue of novel requirements.** Every new task family needs a new orchestration script or prompt. An enterprise with hundreds of document-heavy workflows must build and maintain hundreds of coordinators; adding a new specialist agent helps nothing until every relevant script is updated to know about it.

3. **Judge bias.** When one model both aggregates and evaluates contributions, its tendencies dominate the outcome. LLM judges show position bias, verbosity bias, and self-preference [16].

4. **The context bottleneck.** All coordination traffic flows through the orchestrator's context window. As agent count and task size grow, the hub must compress, and compression at the hub is exactly where interactions accross different domains (a pricing clause that is *legally* fine but *financially* toxic) get lost.

5. **The knowledge problem.** The orchestrator allocates work using its beliefs about what each agent can do. But the decisive information, an agent's actual fit for this task given its private knowledge files and persona, is distributed among the agents and not visible at the hub. This is precisely Hayek's argument against central planning: the knowledge relevant to an allocation decision "never exists in concentrated or integrated form, but solely as dispersed bits" held by the participants [17].

## 2.3. What a Fix Must Provide

Removing the orchestrator does not remove its four functions; it obliges the system to provide each of them decentrally. For every function, a pre-LLM literature supplies a tested mechanism, and together they form the blueprint this project implements.

### 2.3.1. Allocation: Markets Instead of Routing

Smith's Contract Net Protocol showed in 1980 how a distributed problem solver allocates tasks with no scheduler: tasks are announced, capable nodes bid based on self-assessment, and work goes to the best bidder [18]. Market-based control generalised the idea to price-like signals for distributed resource allocation [19, 20]. Applied to agents, bidding moves the allocation decision to where the knowledge lives: each agent judges its own fit, answering failure mode 5, and no single planner's omission can exclude a relevant specialist, answering failure mode 1 for the allocation step.

### 2.3.2. Deliberation: A Shared Workspace Instead of a Hub

The Hearsay-II blackboard architecture demonstrated that heterogeneous specialists can cooperate by reading and writing hypotheses on a shared, structured workspace, with control emerging from the data rather than a fixed call sequence [21]. The pattern removes the context bottleneck (failure mode 4): information lives in one shared medium that every specialist reads selectively, instead of being compressed through a hub. Nii's retrospective named the residual

difficulty, deciding who acts next without a scheduler [22], a problem this project answers with rationed *attention* rather than rationed *activation* (Chapter 6).

### 2.3.3. Aggregation: Endorsement Instead of a Judge

Collective decision theory offers judge-free aggregation with quality guarantees: Condorcet's jury theorem shows majority verdicts of independent, better-than-chance voters approach certainty with group size [23], Galton observed the effect empirically [24], and Surowiecki catalogued its preconditions, diversity, independence, honest aggregation [25]. Hong and Page add that diverse ensembles can beat uniformly stronger ones [26]. Multi-agent debate brings the idea to LLMs, improving factuality through symmetric mutual critique [27, 28, 29]. And stigmergy, Grassé's account of termite coordination through environmental traces [30], shows how endorsement can be *accumulated in the medium itself*: deposits reinforce good paths, evaporation removes stale ones, and the colony reads the decision off the environment [31, 32, 33]. A citation on a shared board is exactly such a deposit. This answers failure mode 3: no judge, no judge bias.

### 2.3.4. Termination: Equilibrium Instead of a Stop Order

The subtlest orchestrator function is knowing when to stop. Swarm systems stop when their dynamics reach equilibrium, when deposits stop changing the state [32]. For a deliberating swarm this translates to observable conditions, no new ratified findings; every participant voluntarily silent, that can be checked mechanically without granting any component stop authority. Chapter 6 shows this is the function that required the most engineering care: LLM agents, unlike termites, will happily keep talking.

### 2.3.5. The Remaining Doubts

Two honest caveats temper the blueprint. First, LLM agents built on the same base model are not the independent voters the jury theorem assumes, and deliberation itself erodes independence; the diversity precondition must be engineered (heterogeneous personas, providers, and knowledge) rather than assumed. Second, believable emergent behaviour has been demonstrated in open-ended societies [10, 34], but a business system must emerge to a *single, auditable answer within a budget*, a far harder target than believability, and one the cooperative-AI agenda identifies as a research problem in its own right [35]. These doubts define the evaluation questions of Chapter 7 and several limitations in Chapter 8.

# 3. System Design

The previous chapter argued that fixing the orchestrator's failure modes means decentralising its four coordination functions rather than adjusting how a single orchestrator is instructed. This chapter describes the system built to make that argument testable: Mycelium, a full-stack platform for running groups of LLM-backed agents, in which each of those four functions is a pluggable strategy rather than a hard-coded behaviour. The chapter stays deliberately at the platform level. The execution engines that run inside it, the broadcast baseline (Chapter 4), its topology and consensus refinements (Chapter 5), and the Market-Hive engine (Chapter 6), are covered separately; here we describe the vessel they all share, and the design rules that made it possible to swap allocation, deliberation, and aggregation mechanisms in and out without rebuilding the application around each one.

We first walk through the design decisions that shaped the platform (Section 3.1) and the resulting architecture (Section 3.2). Section 3.3 covers how the project is developed, tested, and deployed, since a research prototype that is expensive to run cannot be iterated on. Section 3.4 closes the chapter by making explicit what the next three chapters take for granted: which of Chapter 2's four coordination functions the platform leaves entirely to the engines, and which it constrains before any engine gets a say.

## 3.1. Design Decisions

Six decisions, taken early, shaped almost everything that came after. We describe them here with their reasoning rather than as a formal requirements list. On a project this size, the decisions effectively *were* the requirements. Each one also pulls some weight in the larger claim that a multi-agent system can be trustworthy without an orchestrator, and we flag that role as we go. Figure 3.1 groups the six by the strand of that argument they serve.

### 3.1.1. D1: One interface, pluggable engines

The project rested on a single bet: that we would not get the coordination mechanism right the first time. We did not. Chapters 4 through 6 walk through the versions it took. So rather than commit to a mechanism, we committed to a seam. Every run goes through one interface, `POST /api/simulation/run`, and a `mode` field selects the engine behind it. Everything around that engine (the agent roster, the knowledge files, persistence, the live event stream, the history views) is shared infrastructure that neither knows nor cares which engine is running.

That seam is what let the system grow from a plain broadcast baseline into a market-based swarm without ever being rebuilt around any one design. It also underwrites the blind, four-way comparison in Chapter 7: because every configuration takes the same request path, lands in the same schema, and is read back from the same trace format, a difference in the

results points to the coordination mechanism itself rather than to some incidental difference in plumbing.

### 3.1.2. D2: A modular monolith, not microservices

The backend is a single FastAPI process. Each engine sits in its own package (`server/topology/` and `server/market_hive/`), with its own logic, routes, and tools, so at a glance the two read as separate systems. Underneath, though, they ship as one unit, share one database schema, and talk to each other through ordinary function calls rather than network hops.

For a six-person team building a research prototype, that trade was an easy one. The operational simplicity of a single deployable outweighed any scaling argument for splitting things into services, and keeping the platform/engine boundary a matter of code organisation rather than network topology was one less thing to get wrong while the coordination logic was still being torn up and rewritten.

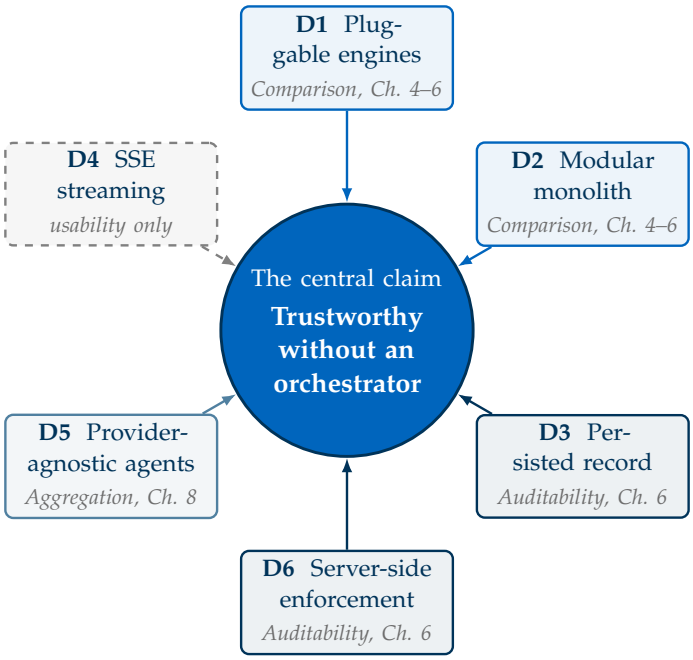



Figure 3.1.: The six design decisions radiating from the report's central claim.

### 3.1.3. D3: Every run a persisted record

Nothing about a run lives only in the UI. Each execution opens a *trial* and breaks it into rounds, connections, and messages, all written to the database as they happen. Replay, debugging, the history views, and the entire evaluation of Chapter 7 read back from that same trace. The rule behind it is deliberately strict: if an exchange between agents is not in the record, it did not happen.

This is what turns the application from a demo into an instrument, and it answers a worry that runs through the whole report. Coordination that emerges rather than being scripted is

only as trustworthy as it is auditable after the fact. An orchestrator at least leaves a log (its own decomposition script) of who was asked to do what; an orchestrator-less system has no such script, so the trace has to stand in for one. That is precisely the job done by the trust registry, the citation graph, and the audit endpoint of Chapter 6.

### 3.1.4. D4: Live progress over Server-Sent Events

A multi-agent run takes minutes (often dozens of LLM calls in sequence), and a spinner that sits there for five of them is indistinguishable from a crash. Every run therefore streams its own progress to the browser as it executes, emitting one event per meaningful step: an agent responding, a message posted, a decision reached. We used Server-Sent Events rather than WebSockets because the channel only ever runs one way (the client simply listens), and SSE delivers exactly that over plain HTTP, with no connection upgrade to manage and no trouble passing through proxies. It is the one decision here made purely for usability rather than for the argument.

### 3.1.5. D5: Provider-agnostic agents, user-supplied keys

An agent's profile declares *which* provider and model it runs on (OpenAI, Anthropic, Groq, or Gemini), and the model layer resolves credentials at request time. Keys arrive either in per-request headers sent by the browser, a bring-your-own-key scheme that lets the deployed instance hold no secrets of its own, or from server environment variables as a fallback. A mixed team, say a Claude analyst arguing against a GPT-4o counsel, is then a matter of configuration, not engineering.

The payoff reaches past convenience. Chapter 8 flags model-homogeneous swarms as a real threat to the independence assumption behind jury-theorem-style aggregation: agents built on the same base model do not behave like the independent voters that theorem requires. Because the providers are interchangeable, the remedy, a deliberately heterogeneous roster, becomes a configuration change rather than a research project.

### 3.1.6. D6: Invariants enforced, not prompted

We treat everything an LLM produces as untrusted input. Every action an agent can take is a *tool call* against a registered, schema-validated tool; guardrails check what goes into and comes out of each model call; and the rules the coordination logic actually depends on (length limits, one vote per agent, replies that point at messages that exist) are enforced in the server and the database, never merely requested in a prompt. Prompts steer behaviour; the platform *guarantees* it.

This is the precondition for everything Chapter 2 asks a decentralised system to deliver honestly. A market allocation is only a market if bids cannot be forged. A blackboard's endorsement signal means something only if a citation is a verified server-side event and not a claim made in prose. A run terminates trustworthily only if the conditions that end it are checked by machine rather than announced by a participant. Later chapters lean on this rule

repeatedly, for a plain reason: when the participants are stochastic, any invariant that is only asked for will eventually be broken.

## 3.2. Architectural Overview

The platform is a classic three-tier web application with one external dependency class: the LLM providers (Figure 3.2).

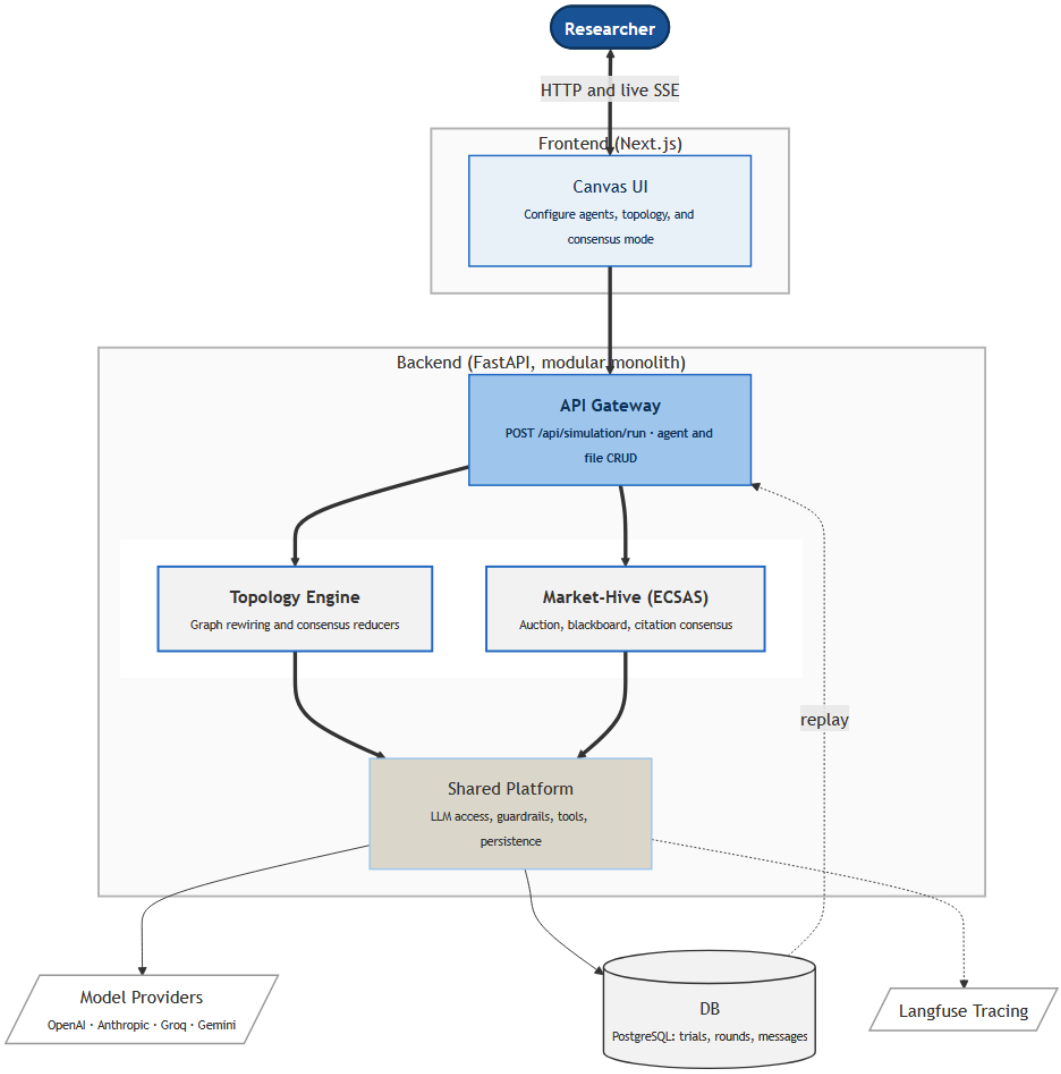



Figure 3.2.: Architecture overview. Both engines sit behind one mode-dispatched endpoint and share the platform's LLM layer, tool registry, and guardrails.

### 3.2.1. Client

A Next.js 16 / React 19 single-page application (TypeScript, Tailwind CSS, Framer Motion for animation). Its centrepiece is an infinite canvas on which users author their agent team; around it sit the run controls, the live activity surfaces, and the history and statistics views.

The client holds no authoritative state: it renders what the API tells it, either via REST for configuration data or via the SSE stream during a run.

This is more than an implementation detail. Because the client cannot originate coordination decisions of its own, it cannot quietly become an informal orchestrator: every decision a user sees was made by an agent, an engine, or a mechanical rule on the server. Live runs and replayed history are therefore the same view, both a projection of the same persisted trace, which is what lets any run be reconstructed exactly as it happened.

### 3.2.2. Server

A Python 3.12 / FastAPI application. The shared platform code sits flat under `server/`: `db/` (persistence via SQLAlchemy Core), `llm/` (the multi-provider model layer), `tools/` (the shared tool registry), `guardrails/` (input/output validation), `schemas/` (request/response models), and `routes/` (agent and file CRUD, model listing, and the mode-dispatched run endpoint). The tool registry deserves a specific note: every capability an agent can invoke during a turn is a registered tool with a validated input schema, and both engines draw their per-phase tool sets from it.

Every model call passing through the shared LLM layer is also traced in Langfuse, linking prompts, responses, token usage, latency, and cost to the corresponding simulation run. This provides fine-grained observability without coupling either execution engine to a specific tracing implementation.

The two engine packages sit alongside this shared code and hold everything specific to their own coordination strategy. One design rule keeps the boundary clean: engine *behaviour* lives in the engine packages, while anything both engines need lives in the shared platform and is owned by neither. The tool registry, the guardrail layer, and the persistence schema are therefore coordination-strategy-agnostic by construction, rather than Market-Hive code that the topology engine has quietly grown to depend on.

### 3.2.3. Database

PostgreSQL, accessed asynchronously (asyncpg). The schema for both engines is defined in a single module and created on startup, which keeps the two engines honest about sharing one data model. The test suite points the identical schema at an in-memory SQLite database, so tests run with no external services at all, a property Section 3.3 returns to.

Figure 3.3 summarises the technologies selected for each architectural layer, arranged top-to-bottom by proximity to the user.
The one architectural element worth singling out is the **engine dispatch**. When a run request arrives, the endpoint inspects `mode` and hands the request to the corresponding engine, together with the shared services it needs. Each engine returns the same thing: an asynchronous stream of typed simulation steps. Everything downstream (streaming, persistence, the UI) consumes that one step format. Adding the Market-Hive engine in Chapter 6, an engine with an entirely different coordination philosophy, a different allocation mechanism, a different aggregation

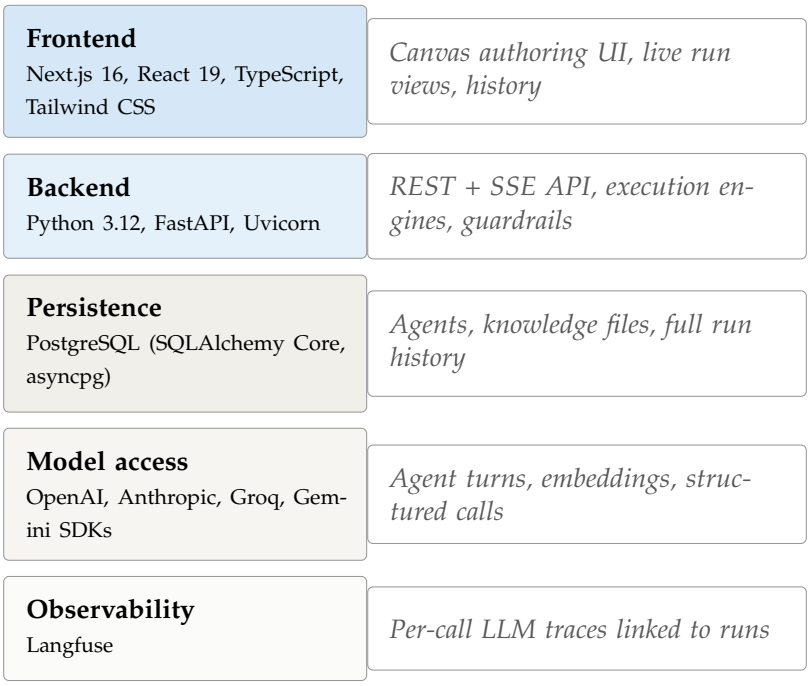



Figure 3.3.: Technology stack, arranged from user-facing (top) to infrastructure (bottom).

signal, and eleven tables of its own, required no changes to this contract. That the platform absorbed a change of that magnitude without touching its own architecture is the strongest evidence that Sections 3.1 and 3.2 drew the platform/engine boundary in the right place.

## 3.3. From Local to Cloud

The platform is a single monorepo: one setup command installs everything and starts the client and server together against a bundled PostgreSQL, and a Docker Compose file offers the same stack in containers for anyone who would rather not install a toolchain at all.
Two properties of the workflow are worth recording, because both shaped how quickly the engine iterations of Chapters 4 to 6 could happen. The project's core question, whether a swarm of LLM agents can coordinate without central authority, can only be answered by running the swarm many times over many configurations, which is affordable only if a run is cheap to set up and cheap to execute. To further facilitate this, we are using Langfuse to monitor the usage and collect relevant data.

### 3.3.1. The test suite runs offline

Over 600 backend tests run against in-memory SQLite with the model layer replaced at well-defined seams, so a complete run needs no API keys, no network, and finishes in seconds. The engines accept injectable agent runners and embedders precisely so that a full multi-round run can be driven deterministically in a test. This is what made it affordable to redesign the consensus mechanism three times over (Chapters 4 to 6): whether a new engine dispatches, persists, and streams correctly could be verified without spending a single token, leaving the

limited real-model budget for the only question tokens were actually needed for, whether the coordination behaviour itself was any good.

### 3.3.2. Quality gates are automatic

Every pull request must pass linting, type checks, and the full test suite in CI before it can merge, and formatting is fixed by git hooks rather than left to discipline, so review can argue about design instead of style. Deployment is equally hands-off: merging to main builds the same containers used locally and ships them, together with a managed PostgreSQL, with no manual database step. The deployed instance holds no model credentials of its own; visitors supply their own provider keys in the UI, which travel as per-request headers (Section 3.1), so the public instance can be shared freely without anyone's budget attached to it.
The result is that the distance from `git clone` to a running local instance is a single command, and the distance from a merged pull request to the live deployment is zero. For a research prototype whose purpose is to be *demonstrated and experimented with*, we consider this operational lightness a design feature on par with the architecture itself.

## 3.4. What the Platform Leaves Open

This chapter has stayed one level below the coordination argument on purpose: nothing described so far allocates a task, aggregates an opinion, or decides that a run is finished. That is deliberate, and it is worth stating plainly what the platform does and does not settle before Chapter 4 introduces the first engine.

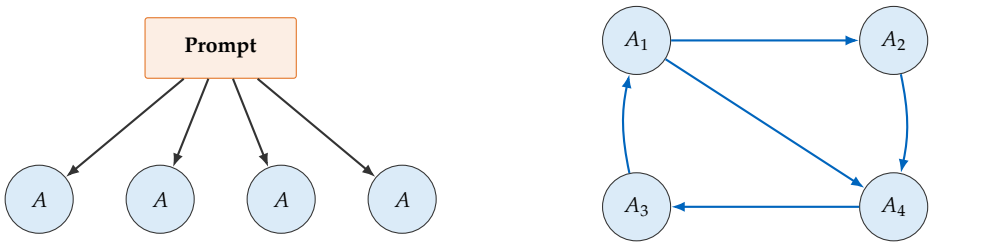
The platform settles *how* coordination is recorded and enforced: every coordination act is a schema-validated tool call, every such call is a persisted, replayable database row, every run streams live rather than blocking, and no engine can smuggle in a shortcut that only works because a model happened to behave (Section 3.1). What the platform leaves entirely open is *what* the coordination mechanism is. The engine dispatch of Section 3.2, and the fact that every round of every run is persisted as its own set of connections and messages, exist precisely so that allocation, deliberation, aggregation, and termination, the four functions Chapter 2 argued an orchestrator-less system must provide decentrally, can each be redesigned independently of the other three without touching the platform underneath them. That separation is what allows Chapter 4 to study a static, orchestrator-free graph in isolation; Chapter 5 to add dynamic topology, voting, pooling, and bidding as alternative, swappable mechanisms on the same infrastructure; and Chapter 6 to replace all three with a market, a blackboard, and citation-based stigmergy, while the run interface, the persistence schema, and the audit trail stay exactly as described here. The platform is the constant, the next three chapters are the experiment.

# 4. Naive Implementation: The Broadcast Baseline

Every claim about self-organisation needs a control. Our first working system, referred to as V0 or, in the evaluation harness, mode B-02, is deliberately the simplest multi-agent arrangement that is still recognisably a system rather than a prompt: several agents, a fixed communication graph, and repeated rounds of mutual reading. This chapter describes its execution model, what it got right, and the specific failure modes that motivated everything after it.

## 4.1. Execution Model

A run proceeds in two phases, sketched in Figure 4.1.




*(a) Broadcast: independent, parallel answers* *(b) Discussion: fixed directed graph, r rounds*

Figure 4.1.: The naive execution model. Every agent receives the user prompt and answers it independently (a); agents with incoming edges then reply to their senders' latest responses over a static communication graph for $r$ rounds (b).

**Broadcast phase.** Every agent receives the original user prompt and responds independently, in parallel. No agent sees any other agent's output in this phase; it establishes the diversity of initial positions that collective-decision theory says the system should preserve [25, 26].

**Discussion phase.** For a fixed configured number of rounds $r$, each agent that has at least one *incoming* edge in the communication graph receives a context message assembled from its senders' most recent responses and is asked to reply. This communication could motivate agents to change their first answers during the run, e.g., due to missing information which got filled in the discussion. Agents without incoming edges stay silent during discussion. The graph is *static*: whatever edges the user drew on the canvas persist unchanged for the whole run.

In its naive configuration the run ends with no consensus operator (`broadcast_no_consensus`) at all: after the final round, the system simply presents every agent's last response in a list. This is not an oversight but a measurement choice, the mode preserves all outputs for analysis, making it the right instrument for observing raw deliberation dynamics before any aggregation machinery is layered on top.
Figure 4.2 shows the corresponding authoring experience: agents with their provider/model badges, hand-drawn communication edges, and knowledge files attached as source nodes. Since the edges correspond to bilateral communication (message send in direction of the edge, response send in the opposite direction), the canvas showcases them as undirected edges. In static mode this hand-drawn graph is the communication structure for the whole run, i.e., all $r$ rounds of discussion. A more detailed walkthrough of a complete run on the canvas is provided in Appendix A.4.

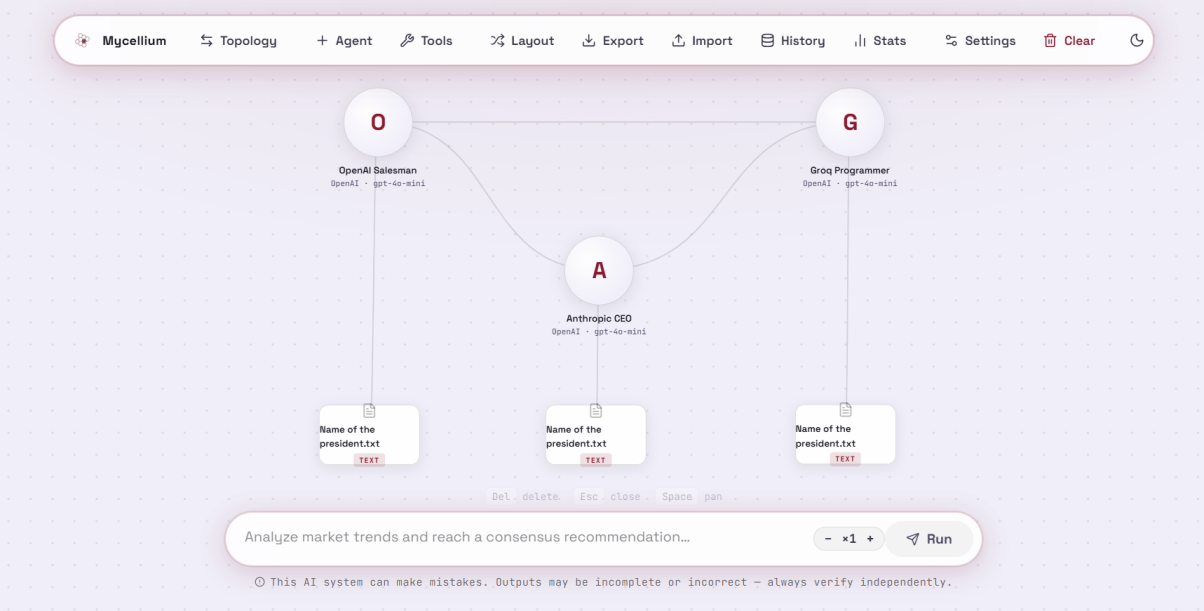



Figure 4.2.: The topology canvas with an imported demo configuration: three agent personas connected by communication edges, each grounded in an attached knowledge file.

## 4.2. Engineering Properties

Three properties of even this minimal engine carried through the whole project.

**Parallelism by default.** All agent calls within a phase are issued concurrently with asynchronous provider calls. Round latency is bounded by the slowest agent, not the sum of agents, which is what keeps ten-agent experiments responsive enough for interactive use.

**Failure as data.** An agent whose provider call fails during any phase has the error text recorded as its response for that round. Subsequent rounds include this text as context, so other agents remain aware that a participant failed rather than silently losing an opinion. This

turned out to be the correct primitive for the robustness ablations in Chapter 7, where we deliberately fail agents and observe whether the collective compensates.

**Complete traces.** Every message along every edge in every round is persisted as it happens (Trial → Rounds → Connections → Messages), so post-hoc analysis of who influenced whom never depends on logging discipline in engine code.

## 4.3. Observed Limitations

Running the baseline on multi-domain document-analysis prompts exposed five limitations, each of which maps onto a mechanism introduced later.

1. **No outcome.** Without a reducer, the system's "answer" really is $N$ answers. For exploration this is a feature; for a business task it pushes the aggregation burden back onto the human, or onto exactly the kind of judge model the design set out to avoid.
   ⇒ pluggable consensus mechanisms (Chapter 5).

2. **Uniform attention.** Every agent reads all of its senders' full responses every round, regardless of relevance. Token cost grows with (agents × rounds × verbosity) and most of the context an agent re-reads is text it has already processed.
   ⇒ relevance-ranked attention over a shared board (Chapter 6).

3. **Conscription, not allocation.** Every agent on the canvas participates in every prompt, including agents whose expertise is irrelevant. The user is the de-facto task router, which is precisely the manual orchestration step we want to remove.
   ⇒ auction-based self-selection (Chapter 6).

4. **No memory across runs.** An agent that performs consistently well confers no advantage on its future participation; every run starts from an identical prior.
   ⇒ per-domain trust scores (Chapters 5 and 6).

5. **No termination signal.** The engine runs exactly the configured number of rounds. Whether the discussion converged in round two or was still productive at the cap, the budget spent is identical.
   ⇒ convergence detection and quiescence (Chapters 5 and 6).

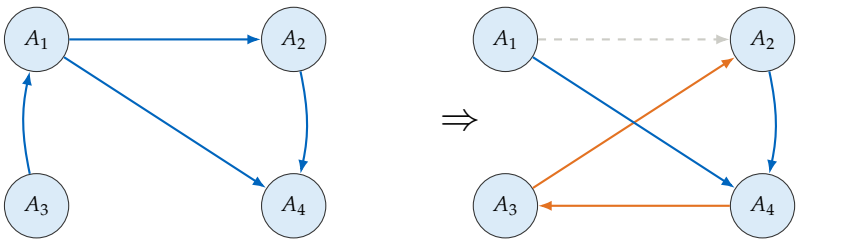
One further observation deserves recording because it shaped our reading of the debate literature [27, 28]. With symmetric, static topologies, discussion rounds exhibited rapid *opinion homogenisation*: agents converged on a shared phrasing within two to three rounds, sometimes around a position no agent had originally held strongly. Agreement produced by mutual paraphrase is not the independence-preserving aggregation that jury-theorem arguments require [23]; it is closer to an echo chamber. This observation is why the later engines separate *content* (posts) from *endorsement* (citations or votes): a countable, deliberate endorsement action is a far more honest consensus signal than textual similarity.
The baseline, then, is not a strawman. It is the system a competent team builds in week one, it produces genuinely useful multi-perspective output, and its trace infrastructure outlived every subsequent redesign. Its limitations are structural, not incidental, and the next two chapters address them in increasing order of ambition.

# 5. Improving the Baseline

With the limitations stated in Chapter 4, we explore possible improvements from three different standpoints. First, a dynamic instead of static communication method is explored via a method we refer to as *Dynamic Topology*. Second, *Consensus Mechanisms* enable the agent pool to converge on a single answer instead of a hand-picked one. Lastly, a Bidding Mechanism makes participation itself optional. Each step is plugged in and selected by the user per run for flexibility and honest ablation.

## 5.1. From Static to Dynamic Topology

The baseline's graph is fully controlled by the user. Once defined, the communication becomes static. In the *Dynamic Topology*, agents declare, each round, which peers they want to hear from next. This selection of contactable peers is enabled via a *Phone Registry* tool with semantic search. A controller assembles the declared edges into the next round's graph, optionally capping fan-out so that no agent becomes a hub that must read everyone (Figure 5.1). Because connections are persisted per round, the emergent communication structure is itself an experimental observable: one can watch specialists find each other, or watch the graph collapse into a single agent, from the connection history alone.




*Round r* — *Round r+1: declared edges (orange) added, undeclared ones (grey) dropped*

Figure 5.1.: Dynamic topology. Agents declare desired peers each round; the controller rewires the graph between rounds, subject to fan-out caps. Every intermediate graph is persisted.

Its research value is that it demonstrates genuine structural self-organisation: edges arise from each agent's own assessment of who is relevant, with no central component assigning partners. The controller only enforces validity and fan-out constraints. What the mechanism deliberately does not answer is who should be in the conversation at all: every agent on the canvas still participates each round and may contact any peer for information. That question is deferred to Section 5.3, where agents gain the ability to self-evaluate and abstain from participation.

## 5.2. Consensus Mechanisms

With communication self-organising, the second gap is the convergence to an answer. We define mechanisms to allow the agents to converge on a single answer. This is represented by Plurality Voting and Weighted Opinion Pooling.

### 5.2.1. Plurality Voting

The voting mode enables the agents to present a vote option and grants them the right to vote on other agents. Every agent must end its turn with structured `REASON:` and `VOTE:` lines. Every voting round, a deterministic tally function normalises the votes, computes the majority, and broadcasts the tally with all submitted reasons back to every agent. The run converges when the plurality is stable across two consecutive vote rounds. This is Condorcet-style aggregation [23, 25] with a deliberation loop: agents see the distribution of positions and the reasons behind them before voting again, in the spirit of multi-agent debate [27] but with a mechanical, bias-free reducer in place of a judge model [16].

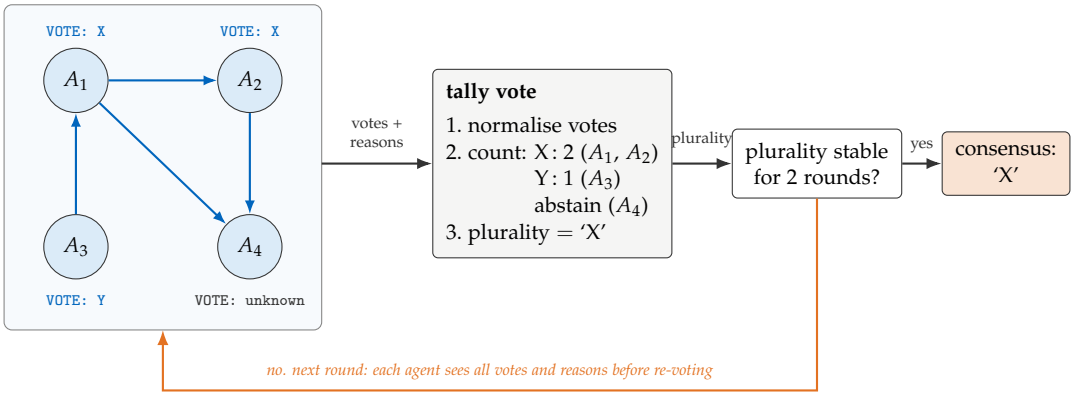



Figure 5.2.: Example of plurality voting with four agents. The agents converge on the answer 'X', proposed by $A_1$ and $A_2$, while $A_4$ abstains with `unknown`.

This mechanism enables each agent to reflect upon its answer based on the other agents' vote and reasoning. An agent that does not propose an answer can also still contribute by voting.

The voting mode has its own weaknesses. The most important one is baked into its format: a vote must fit on one line as a bare answer, so the mechanism works well for questions with factual, concise answers. However, it cannot produce a comprehensive open answer, a comparison, or a plan. Second, the tally compares votes as text. The normalisation strips filler words, but two answers that mean the same thing in different words still count as two different answers, splitting the vote. Third, every vote counts the same: an agent quoting a source has exactly as much say as an agent guessing, so a confident wrong majority simply wins. Showing agents the current tally makes this worse, since it invites them to follow the crowd, and because the plurality only needs to hold for two rounds, an early bandwagon can lock in the result before dissenting evidence spreads.

### 5.2.2. Weighted Opinion Pooling

Each agent maintains a trust-weight vector over its incoming neighbours. When an agent receives its neighbours' replies, each reply is labelled with the current weight, so the model sees not just *what* a neighbour said but *how much* past agreement that neighbour has earned. After each round, the weights are updated from the semantic agreement (embedding cosine similarity) between the agent's own answer and each neighbour's. The run converges when the weights stop changing across a round, and the final answer is the reply of the agent the group trusts most in aggregate.

Figure 5.3 shows the loop. On the left, four agents answer over the communication graph; $A_4$ weighs $A_2$'s reply at 0.6 against $A_1$'s 0.4, while an agent with a single neighbour trusts it fully. After each round the system compares the answers, blends the result with the old weights, and renormalises. If the weights still move, the next round starts with freshly labelled replies; once they are stable, the run ends. Here $A_1$ collects the most incoming trust, so its answer X' becomes the consensus over $A_3$'s dissenting Y'.

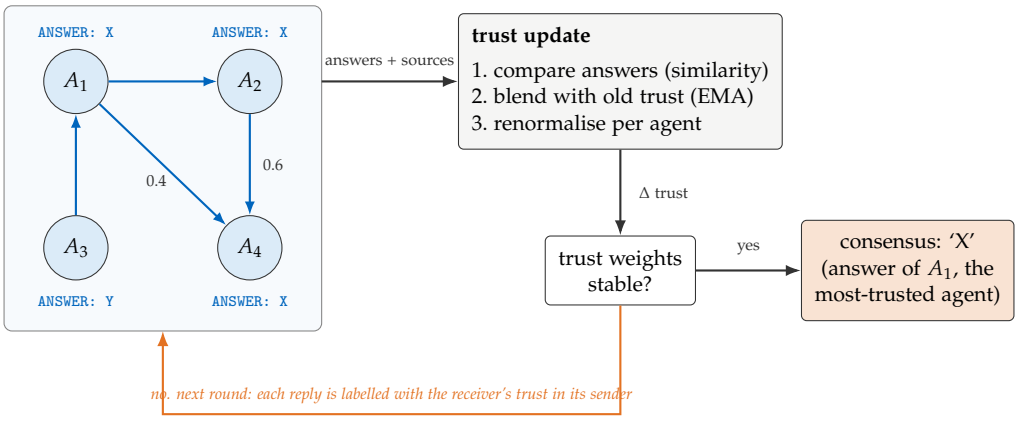



Figure 5.3.: Example of weighted opinion pooling with four agents.

The mechanism has clear weaknesses. The first is a *cold start*: every score begins uniform, so in the early rounds the trust labels carry no real information, and first impressions are formed from a single exchange of answers. Because the run stops as soon as the scores settle, it can also stop too early, after one quiet round in the worst case, before trust had time to mean anything. Second, trust here rewards *agreement* rather than *correctness*. An agent earns a higher score by saying what its neighbours already believe and loses trust by disagreeing, even when its answer is right and backed by a source. A confident majority therefore reinforces itself, while a lone dissenter like $A_3$ is pushed to the margin regardless of evidence. Third, the similarity signal itself cannot tell agreement apart from answers that merely talk about the same things, a softer form of the sameness problem noted in Chapter 4.

However, the mechanism introduced two ideas the Market-Hive engine later reused in sharper form: trust between agents kept as run state, and convergence detected from that state settling rather than from a fixed round count.

## 5.3. Moving Onto Bidding

The previous iterations address the convergence into an answer, maintain a trust-vector across sessions, and enable a dynamic communication method between agents. These solutions, however, have several limitations such as the cold-start problem, mandatory participation, rigid answer formats, and the wrong-majority problem.

To address these issues, we introduce the *Bidding Mechanism*. The bidding is a pivotal move from previous iterations. Instead of forming connections between agents, we strip away all previous communication methods and switch to a *single, shared canvas,* which is referred to as the *bidding board*. Additionally, the bidding introduces *self-abstaining*: An agent that deems itself unable to contribute does not participate in an answer. Lastly, we introduce *proof of participation*: An agent that cannot prove its capability to contribute by citing a tool or a knowledge file is also banned from participating, despite its willingness.

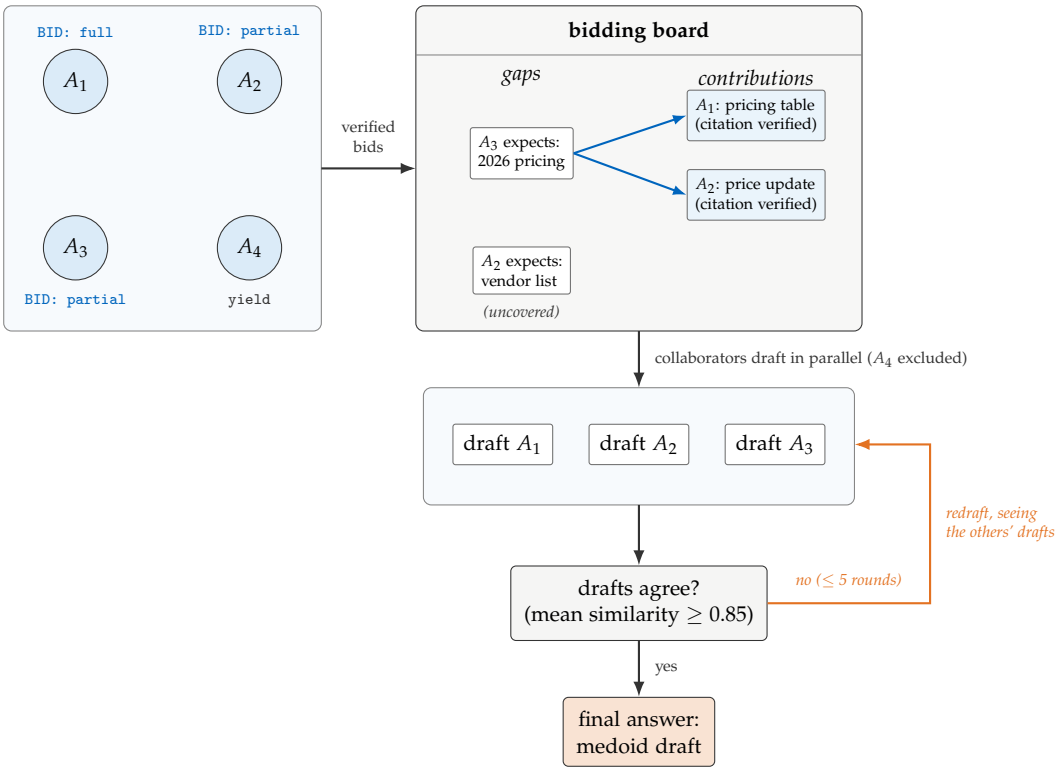



Figure 5.4.: Example of bidding consensus with four agents.

1. **Proof bids (round 1).** Each agent submits a structured JSON bid declaring its estimated completeness for the task (`full/partial/none`), the tools and knowledge files that support its claim, and the gaps it cannot fill alone. The bid must be *verifiable*: an agent may only cite a knowledge file it actually read during the turn, and every citation is cross-checked server-side against the agent's real tool grants and tool-call log. Invalid citations are stripped and the rejection is recorded; the proof cannot be faked. An agent with no verified evidence left after this check yields and takes no further part.

2. **Gap coverage (round 2).** The declared gaps are posted on the bidding board. Each agent reads the other agents' gaps and declares which ones it can cover, either by reusing one of its own verified findings or by reading a file and adding a new contribution. The new evidence is verified the same way as in round 1. A gap may attract several contributions, and gaps nobody covers are reported to the user rather than silently absorbed.

3. **Selection.** Agents that made a verified contribution in either round form the collaboration set; everyone else is excluded, whether they yielded on their own or failed the evidence check.

4. **Parallel consensus.** The collaborators each draft a full answer from the bidding board in parallel. The system measures how much the drafts agree (mean pairwise semantic similarity); if they agree closely enough, the most representative draft, the medoid, becomes the final answer. Otherwise every collaborator redrafts after reading the others' drafts, for at most five rounds (Figure 5.4).

The bidding board also changes how conflicts between sources are handled. It is common for two agents to hold different files that describe the same fact, for example an old price list and a newer price update. In the earlier mechanisms such a conflict was resolved pairwise and in private: only agents connected by an edge ever saw each other's claim, the outcome depended on who trusted whom, and the losing version simply disappeared from the conversation. On the shared board, both claims sit side by side, each with its verified citation naming the file and, where available, its date (Figure 5.4, where $A_1$ and $A_2$ both contribute to the same gap). Every collaborator drafts from the same board, so the conflict is visible to all of them at once, and they can resolve it by comparing the sources rather than by trusting a neighbour. When the sources genuinely disagree, the agents are instructed to keep the conflict and explain it in the final answer instead of papering over it, so the resolution stays transparent to the user.

However, bidding is not without weaknesses. The board is written in a fixed number of passes: one round of bids and one round of gap coverage. An agent must therefore bid with what it can prove at that moment; it cannot wait for a peer's data and then build on it, and a gap discovered late, while covering another gap or while drafting, can no longer be posted. The pipeline is exactly two steps deep, so any task needing a longer chain of dependent steps either fits into that depth or ends with uncovered gaps. The proof requirement also cuts a second way. A bid only survives when it cites a file actually read or a tool actually invoked, so an agent with no files and no tools can never pass the check, even when it could genuinely help, for example as a strong general reasoner or as a critic of the others' drafts. The system verifies access to sources, not ability to contribute: the same rule that stops bluffing also bars the honest generalist, whose knowledge leaves no tool-call log to check. These limitations are later addressed in Chapter 6.

Three ideas make their first appearance here and become load-bearing in the Market-Hive engine: *self-selection by self-assessment* (the contract-net insight [18], with the allocation decision moved to where the knowledge lives [17]), *a shared board* that agents read instead of point-to-point context stuffing [21], and *evidence-backed claims* enforced by the server rather than trusted from prose.

## 5.4. What Remained Unsolved

The improved baseline answers RQ1 in the affirmative, at least for well-posed questions: agents, a self-organising graph, and an honest reducer do converge on correct shared answers without an orchestrator. But the three improvements are alternatives, not layers. The user picks one per run, and each one fixes a weakness of the others while bringing back one of its own.

The strengths never combine. The dynamic topology lets specialists find each other, but says nothing about who should take part or how an answer is agreed. Voting aggregates honestly, but only over one-line answers. Pooling handles free text, but rewards agreement over correctness. Bidding produces open, evidence-backed answers, and then throws the rest of the machinery away: once the collaborators are selected, every agent simply reads every other draft, so neither a graph nor trust plays any role in the discussion they were built for.

Beneath this, three problems survived every mechanism. First, agreement stands in for correctness. Every convergence signal in this chapter, matching vote strings, settling trust weights, drafts passing a similarity threshold, measures whether the agents sound alike, not whether they are right. Bidding checks evidence, but only at the door: citations decide who may speak, and then stop mattering while the answer is negotiated. Second, nothing persists. Trust and reputation are run state; whatever an agent earned is discarded when the run ends, so every new question pays the cold-start cost again, and a proven specialist enters the next run as a stranger. Third, the runs do not know when they are done. The stability signals fire late or never on open-ended tasks, and the bidding board is frozen after two passes regardless of what the task needs, so in practice a round cap does the stopping.

A system that keeps these gains without the trade-offs would need them as one design rather than four options: self-selection at the door, a shared board for deliberation, evidence that keeps its weight all the way into the answer, trust that outlives the run, and termination that emerges from the state of the work rather than from a counter. That is the agenda of the Market-Hive engine.

# 6. The Market-Hive

The Market-Hive engine is the final stage of the project. It drops the central component the earlier stages relied on to route work, judge quality, and stop the run. Each of those jobs is instead done by the swarm itself: an *auction* allocates work, a shared *blackboard* hosts the deliberation, *citations* decide what counts as a finding, and the run ends when the discussion itself winds down. The mechanisms behind this (contract-net bidding, blackboard coordination, stigmergy) were introduced in the methodology chapter, here we focus on how each one is actually built and how they fit together.

A run moves through four stages (Figure 6.1): the *Market Maker* turns the prompt into a typed *contract* and, for multi-step work, splits it into *threads*, an auction lets agents self-select into that contract, bounded deliberation rounds build and challenge findings on a two-layer board and a termination check ends the run, after which a trust vector settles the market for future runs.

No component holds authority over another. Every coordination act, from the opening contract to the last citation, is a validated tool call that writes a database row, so any run can be replayed from its trace.

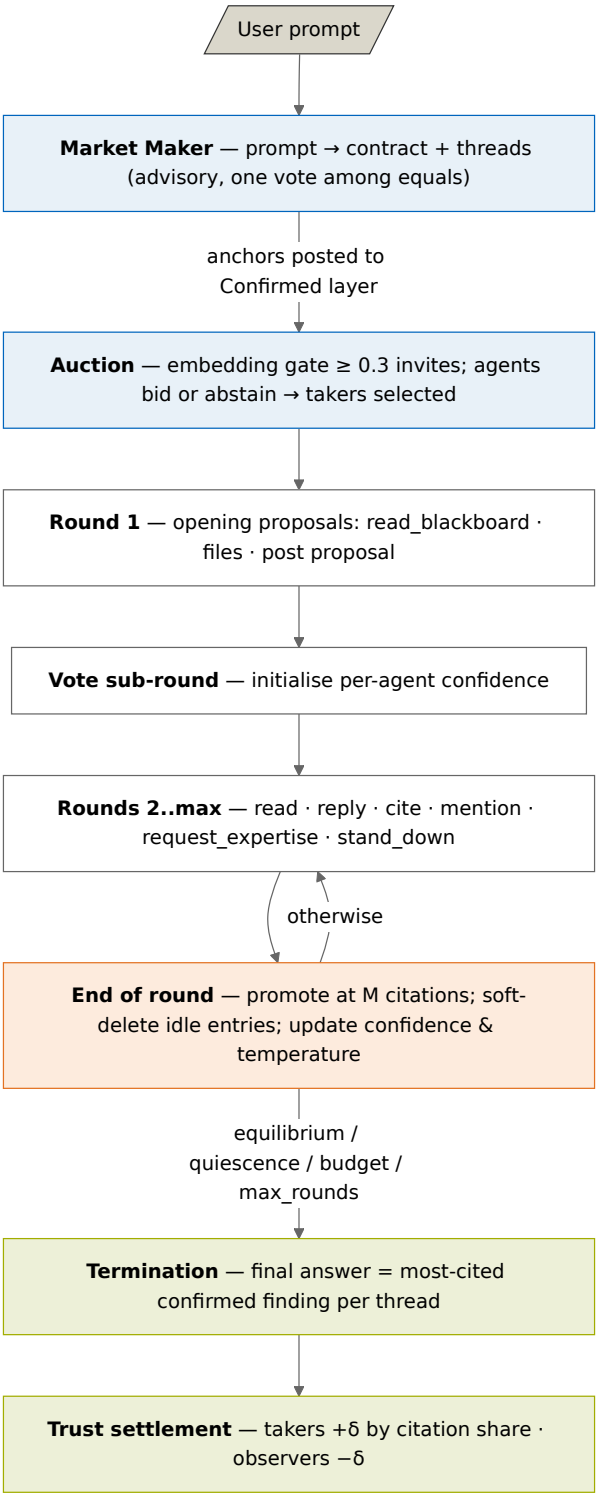



Figure 6.1.: The Market-Hive run lifecycle. The Market Maker advises, the auction allocates by self-selection, end-of-round bookkeeping is mechanical, and termination is read off the swarm's own behaviour.

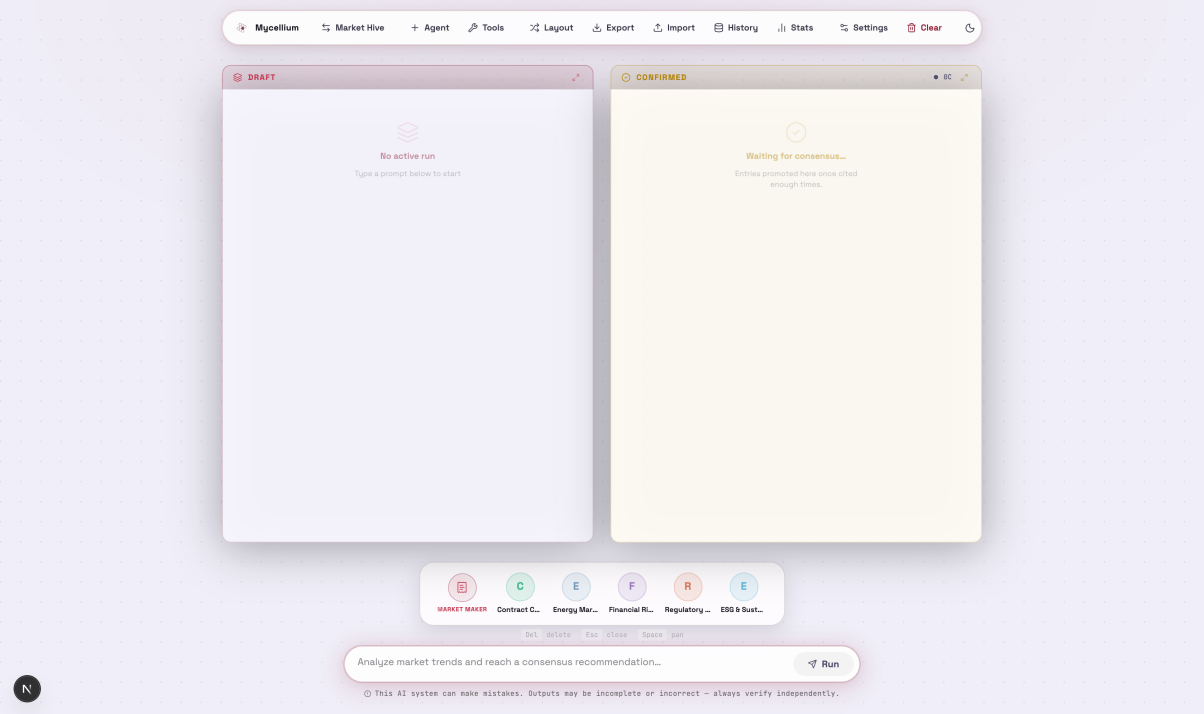



Figure 6.2.: The Market-Hive canvas after importing the energy-contract demo: the Draft and Confirmed board panels, the agent dock with the Market Maker and five specialist personas, and the prompt bar that launches a run.

## 6.1. The Market Maker

The Market Maker converts the raw user prompt into a *contract*: a structured record carrying the task type, single- or multi-step nature, required expertise, complexity, and the Market Maker's own confidence in its interpretation. Multi-step tasks are decomposed into threads, each scoping one sub-problem with its own required expertise and embedding. Contract and threads are posted to the board's permanent layer as *anchors*, the fixed points every later deliberation refers back to. Figure 6.3 shows a contract as the user sees it.

The Market Maker looks like an orchestrator and deliberately is not one. It never routes work to agents (the auction does), never evaluates their output (citations do), never terminates the run (equilibrium and quiescence do). Its decomposition can be contested mid-run through the amendment mechanism (Section 6.3.3), where it casts exactly one vote among equals.

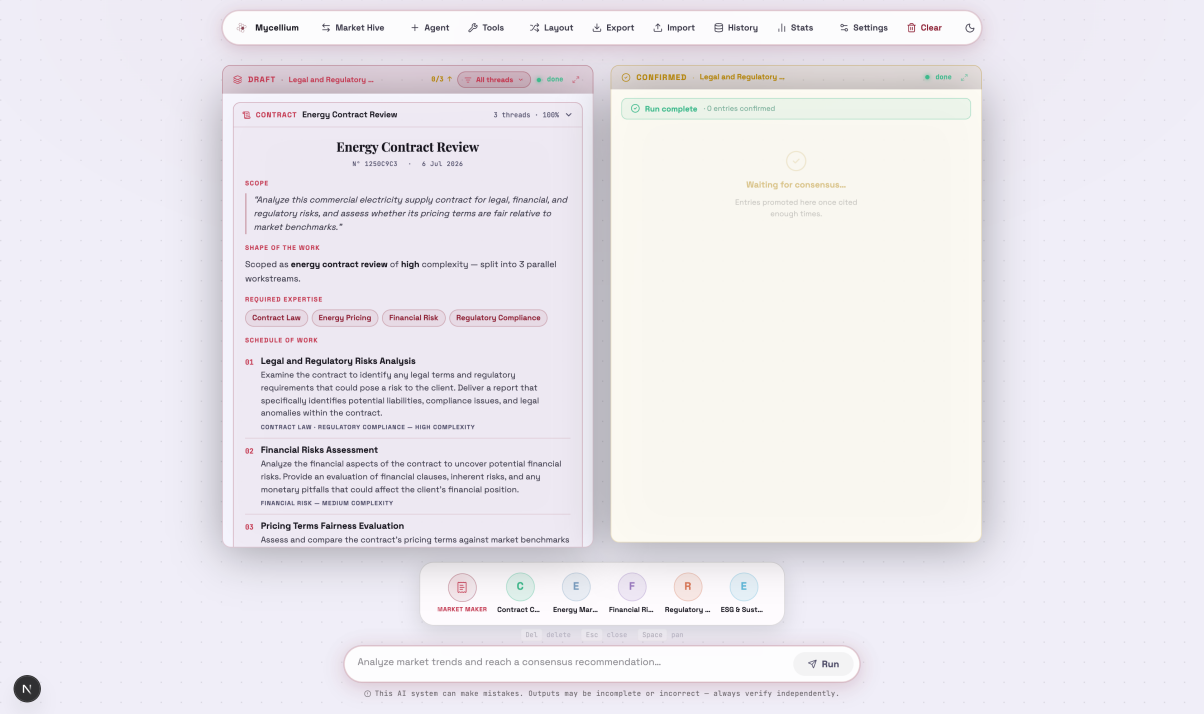



Figure 6.3.: A Market Maker contract for the energy-contract prompt: scoped as a high-complexity review and decomposed into three threads (legal/regulatory, financial, pricing fairness), each tagged with its required expertise.

## 6.2. The Auction

Task allocation follows the contract-net pattern in three steps:

1. **Invitation (no LLM).** Every agent's capability embedding, derived from its persona and knowledge files, is compared against the contract embedding. Agents above the invitation floor are invited. This is purely a cost gate: it prunes the pool of agents that will spend any tokens at all, using nothing but vector arithmetic. If nobody clears the floor, everyone is invited instead and self-selection does the filtering.

2. **Self-selection (one LLM call per invitee).** Each invited agent sees the contract, the thread list, its own similarity score, and its persistent per-domain trust score, and decides for itself: `submit_bid(justification)` or `abstain(reason)`. Abstention is a first-class outcome; computation is never forced on an unwilling agent, and the justification is persisted for audit. This step is the point of the whole design: the deciding information, "am I actually competent for this?", lives in the agent's own persona and files where no router can see it.

3. **Taker selection.** Bidders are ranked by a composite score

$$bid \;=\; \alpha \cdot \cos\left(\mathbf{e}_{\text{agent}}, \mathbf{e}_{\text{contract}}\right) \;+\; \beta \cdot trust[task\ type] \;+\; \gamma \cdot conf_{\text{MM}}, \tag{6.1}$$

with defaults $\alpha = 0.5$, $\beta = 0.4$, $\gamma = 0.1$. Up to `max_takers` become *takers* with full board access. In multi-step runs each taker additionally bids per thread. A thread left without

takers triggers a mini-auction or is dissolved into its parent. Agents that bid and lost become *observers* with no board access. The cost that keeps self-selection honest is the one LLM call each bid spends, not a trust penalty: a lost bid leaves the agent's cross-run trust untouched, so honest bidding is never punished.

## 6.3. Promotion

Once takers are selected, deliberation happens on the blackboard. Agents post, cite one another, and entries that collect enough endorsement are *promoted* from the live Draft layer to the permanent Confirmed layer. Promotion, not any judge, is how the swarm decides what counts as a finding.

### 6.3.1. The Blackboard

Each thread has a two-layer board structured like a discussion forum: top-level *proposals* with nested *replies*, plus *mentions* that pin an entry to the top of a named agent's next read (Figure 6.4).

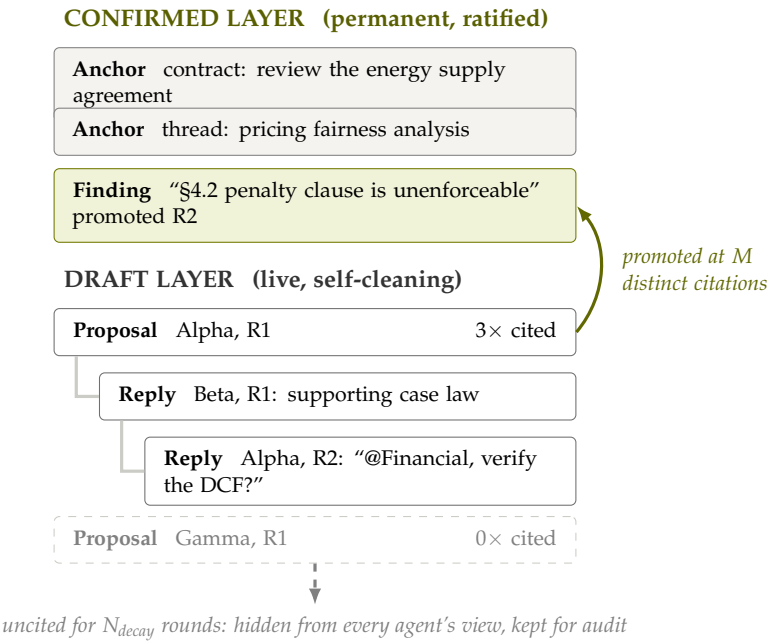



Figure 6.4.: The two-layer blackboard. The Draft layer holds the threaded live discussion. An entry that earns $M$ distinct citations moves up to the permanent Confirmed layer, and an entry nobody cites for $N_{\text{decay}}$ rounds (default 3) fades out of view. Anchors from the Market Maker seed the Confirmed layer.

**Draft layer.** The live discussion. Append-only; entries are never edited. An entry that attracts no citations for $N_{\text{decay}}$ consecutive rounds is *soft-deleted*: hidden from every agent's view but retained in the database for audit. The board agents actually read is therefore self-cleaning.

**Confirmed layer.** Permanent, ratified findings, seeded with the contract and thread anchors. Entries arrive here only through citation-based promotion; nothing is ever removed.

Agents read the board through an attention query, not a dump: `read_blackboard(query)` returns the top-$k$ visible entries ranked by relevance times endorsement, $\cos(\mathbf{e}_{\text{query}}, \mathbf{e}_{\text{entry}}) \cdot \log(1 + \textit{citations})$. Well-endorsed entries surface more readily, this is pheromone amplification implemented as nothing more than a ranking weight. Between the relevance ranking and the soft deletion, an agent's context each round contains what the colony currently considers alive and pertinent, not the full transcript, which is how the engine escapes the baseline's quadratic re-reading (Chapter 4).

### 6.3.2. Citation Mechanics and Convergence

The single most important design decision in the engine is separating content from endorsement. Writing is `post_to_blackboard`, agreeing is `cite(entry_ids)`, a distinct tool call meaning "I build on this". Citations are positive-only by convention: disagreement goes into reply content, never into a citation. Self-citations are filtered server-side, and one citation per agent per entry per round is enforced by a database uniqueness constraint.

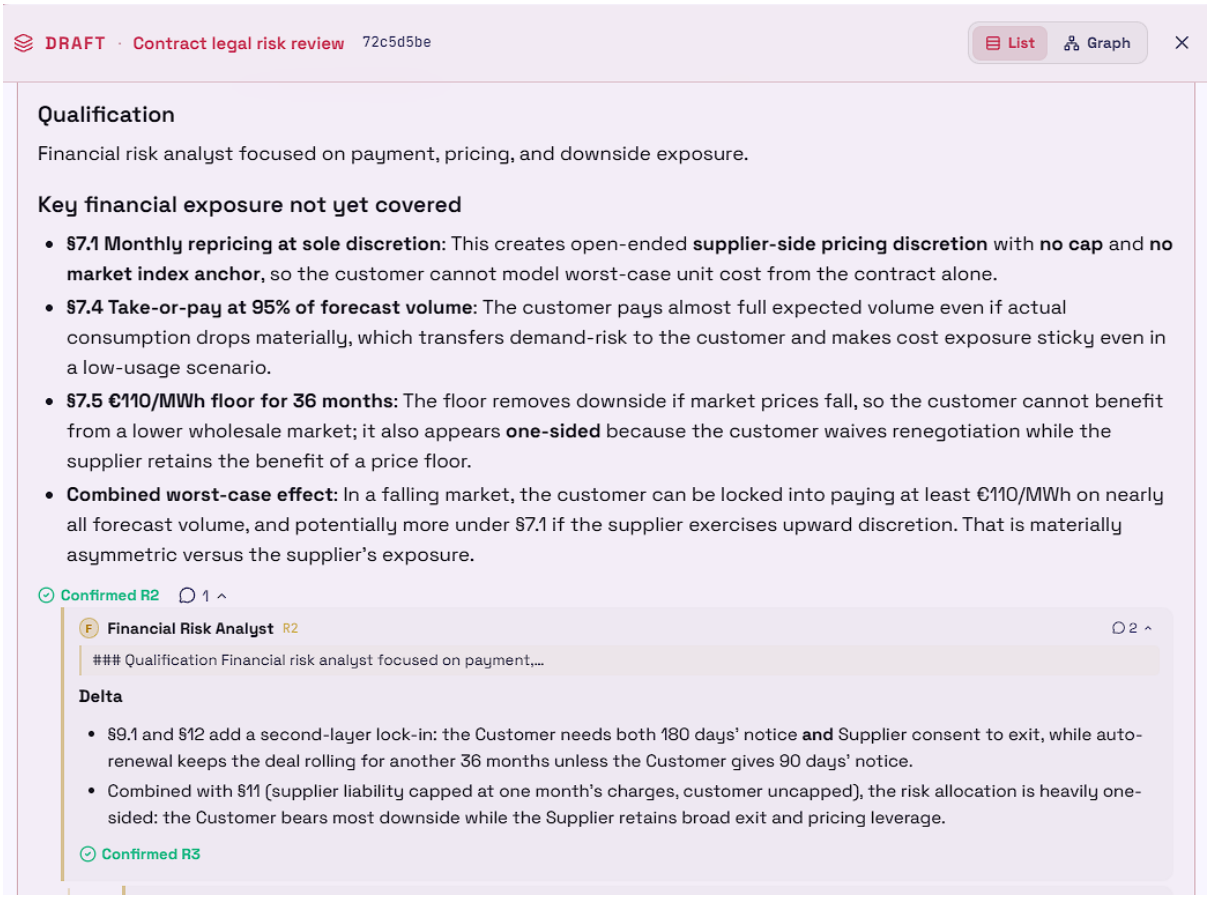



Figure 6.5.: A live deliberation thread on the Draft board. A financial-risk proposal that drew enough distinct endorsements now carries the green "Confirmed R2" badge; the threaded follow-up observes delta discipline (Section 6.5) and contributes only new points, the §9.1/§12 second-layer lock-in and the one-sided §11 liability cap. The accumulated citations are the swarm's pheromone: the signal that drives promotion, confidence, and cross-run trust.

**Promotion.** At the end of each round, a draft entry is promoted once it has been cited by roughly a third of the thread's takers. Concretely, a thread with $N$ takers needs $M$ distinct non-author citations, where $M$ grows with $N$ as shown in Table 6.1: at least 1, and never more than $N-1$, since an entry's own author cannot cite it and so at most $N-1$ agents could ever endorse any given post.

| Takers ($N$) | 2 | 3 | 4 | 6 | 9 |
|---|---|---|---|---|---|
| Citations needed ($M$) | 1 | 2 | 2 | 3 | 4 |

Table 6.1.: The promotion threshold $M$ for a few thread sizes $N$: roughly $N/3$, rounded up to a minimum of 1.

**Confidence and conceding temperature.** Each round, an agent's *confidence* is its share of that round's citations (a cold-start vote sub-round after the opening proposals initialises it). Confidence feeds a per-agent *conceding temperature* $T$, injected into the prompt as graduated natural language: near $T = 0$, "defend your analysis firmly", near $T = 1$, "prioritise consensus". $T$ is the larger of two independent pressures, whichever currently asks for more concession:

$$T_i(R) \;=\; \max\left(1 - \mathit{confidence}_i(R),\;\; T_{\text{floor}}(R)\right), \tag{6.2}$$

$$T_{\text{floor}}(R) \;=\; T_0 + (R-1)\,\Delta T. \tag{6.3}$$

Here $R$ is the round number, $\mathit{confidence}_i(R)$ is agent $i$'s share of that round's citations, and $T_0$ and $\Delta T$ are the floor's starting value and its per-round increase (defaults 0 and 0.1). Taking the *larger* of the two terms is what stops two well-cited, stubborn camps from stalemating forever: the floor is an annealing schedule that eventually forces concession regardless of citation count.

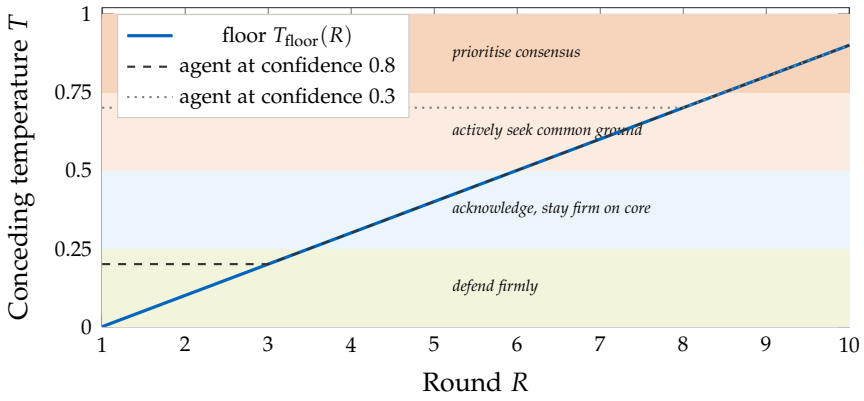



Figure 6.6.: The conceding-temperature schedule (Equation 6.2) with defaults $T_0 = 0$, $\Delta T = 0.1$. Shaded bands show the behaviour injected into the prompt at each temperature range. A consistently well-cited agent (confidence 0.8) stays defensive until the rising floor overtakes it; a poorly cited outlier (confidence 0.3) is pushed toward consensus-seeking immediately.

### 6.3.3. Amendments: Contesting the Decomposition

A taker who discovers mid-run that the contract lacks a needed competence calls `request_expertise(domain, reason)`. The request is posted publicly and all current takers plus the Market Maker vote, one vote each. A majority creates a new thread and runs a mini-auction over previously uninvolved agents, who join from the next round (Figure 6.7). Amendments are capped per run (default 2) to bound scope creep. This mechanism is what makes the Market Maker's fallibility acceptable: the swarm can repair a bad decomposition at runtime with the same democratic machinery it uses for everything else.

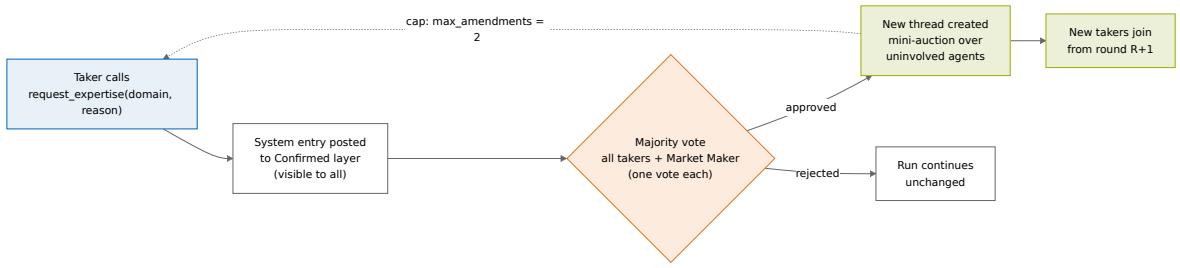



Figure 6.7.: The contract amendment flow. Scope changes are decided by majority vote among all takers plus the Market Maker, each holding exactly one vote. This is the symmetric-authority principle applied to the one place where a privileged decision would be most tempting.

## 6.4. Termination

Three terminators race, whichever fires first ends the run:

- **Equilibrium**: no new non-anchor confirmed entries for $k$ consecutive rounds (default 1, a single quiet promotion round) after a minimum round count.
- **Quiescence**: every active taker on every thread has either called `stand_down(reason)` or fallen silent for a full round.(Section 6.5).
- **Budget**: a hard token ceiling and a round cap (`max_rounds`) as backstops.

Turning the confirmed findings into a single reply is one further *synthesis* step, and it is not a privileged act of judgement either. A synthesizer drafts a brief answer grounded only in the confirmed findings, but the draft goes back to the board for a *ratification* sub-round: each taker calls `approve_synthesis` or `revise_synthesis(reason)`, and the draft stands only on a strict majority, silence counting as non-approval. Short of that majority, the objections are collected and the synthesizer produces one revision, which faces a second and final vote (Figure 6.8). Even the final answer is contestable, not handed down.

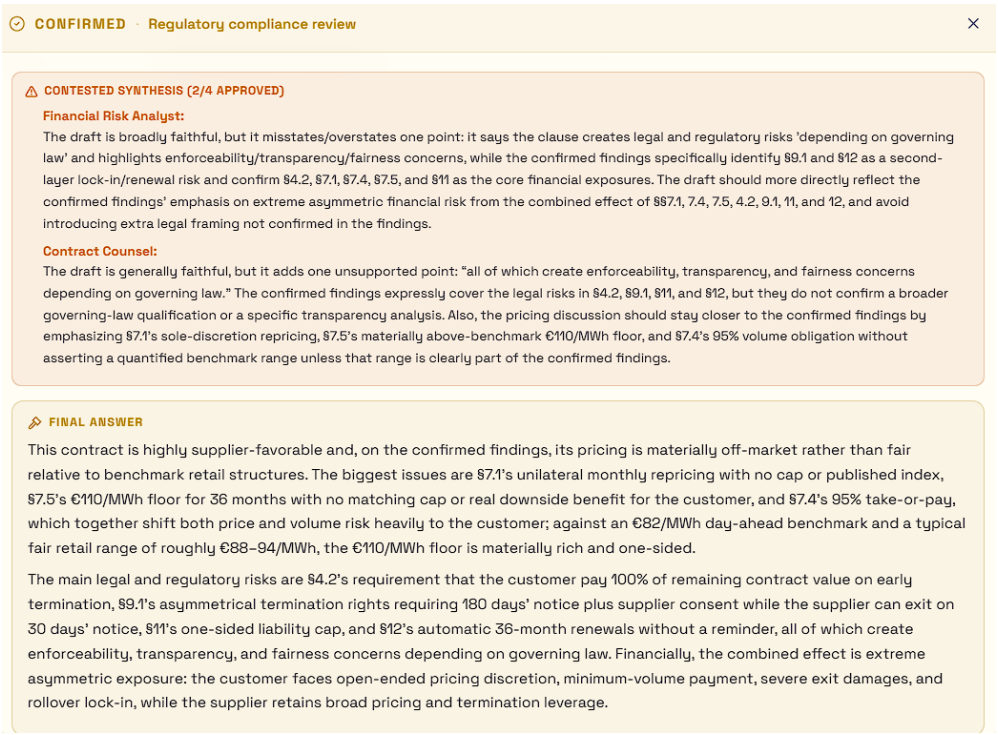



Figure 6.8.: The delivered answer for the energy-contract run, with its ratification trail above it. The synthesized draft did not reach a strict majority ("Contested Synthesis, 2/4 approved"). The Financial Risk Analyst and Contract Counsel each flagged legal framing not grounded in the confirmed findings, and their objections are shown alongside the revised Final Answer. The synthesis is drafted by the engine but ratified by the swarm.

Finally, the trust registry settles the market: each taker earns append-only, performance-only credit for confirmed findings and citations, and debits for entries that evaporated or were retracted, independent of any other agent's outcome. That ledger is collapsed into one trust score per agent per domain, weighting recent runs more than old ones so a single lucky or unlucky run cannot swing it. This score is the *trust* term of the bid score (Equation 6.1) in every future auction, so competence demonstrated in past runs compounds into allocation advantage. Observers and abstainers did no work, so they carry no penalty.

Every coordination act, from the opening contract to the last citation, is a validated tool call writing a database row (the full catalogue is in Appendix A), so a completed run can be reconstructed and replayed message by message from its persisted trace, including soft-deleted entries and the citation graph (Figure 6.9).

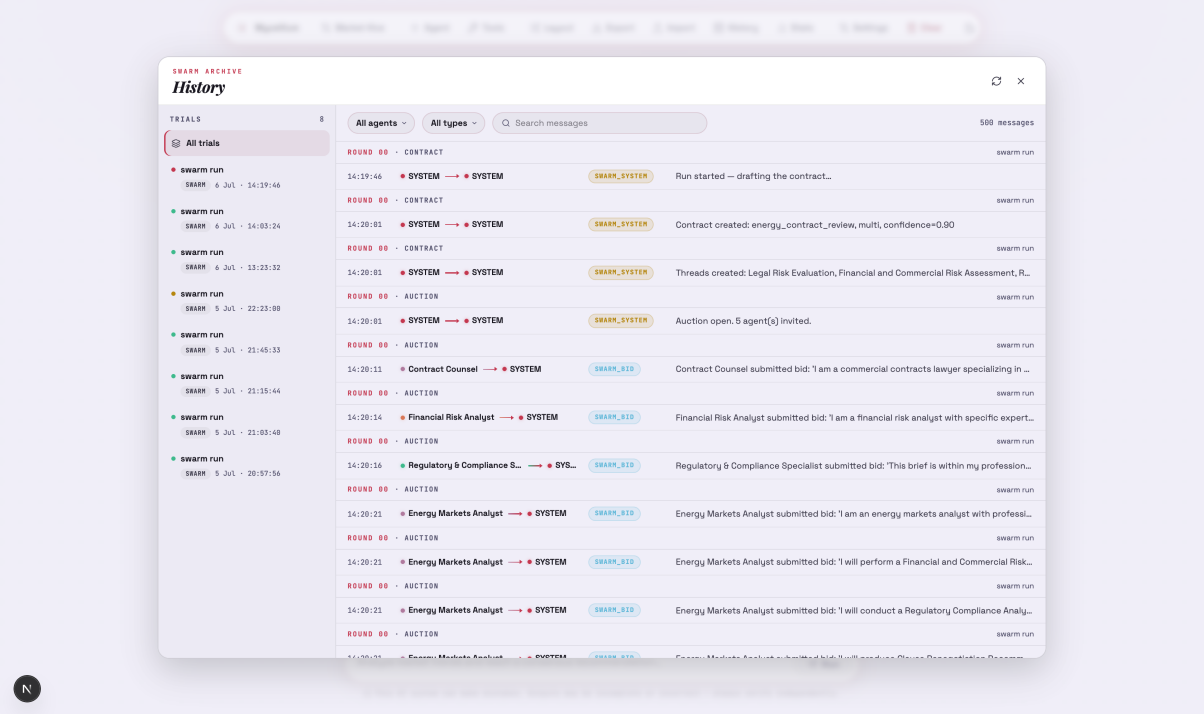



Figure 6.9.: The run archive. A completed swarm run replayed from its persisted trace: contract creation, thread decomposition, auction opening, and the takers' bids, each row a database record written by a tool call at execution time.

## 6.5. Design Iteration: What a Live Run Taught Us

A live run on a 36-month energy supply agreement, testing the mechanisms above for the first time, correctly flagged six real issues (an unenforceable penalty clause, illusory pricing, a 95% take-or-pay commitment, and more) but exposed four *conversation economics* problems: rounds two through four mostly restated round one at full token cost. Every post re-opened with the same qualification paragraph, causing *persona bleed* (a financial-risk agent opening "I am a commercial contracts lawyer…"). Equilibrium never fired, since agents kept posting and citing so every round confirmed something and output rendered as run-on paragraphs.

The fixes are small individually and decisive together:

1. **Stand-down.** A new tool, `stand_down(reason)`, lets an agent declare it has nothing material to add on a thread. It is then skipped on that thread while remaining active elsewhere. Quiescence, meaning all takers stood down or silent, became a termination reason alongside equilibrium. Termination authority stays distributed: the run ends when every participant individually decides it is done.

2. **Delta discipline.** From round two on, each post must be exactly one of: a new point, a disagreement, or new evidence, or the agent stands down.

3. **Shorter defaults.** `max_rounds` dropped from 10 to 5. With quiescence active, the cap is a backstop rather than the common case.

4. **Rendering safety net.** Agents are prompted to emit real Markdown, and the client renderer tolerates single-newline formatting, so board content stays legible however loosely a model formats it.

The general lesson travels beyond this system. In orchestrated pipelines the orchestrator implicitly rations speech, and removing the orchestrator removes the rationing. Emergent consensus needs an emergent way to stop talking, and it has to be engineered with the same care as the consensus mechanism itself.

## 6.6. Limitations

The invitation gate is only as good as the capability embeddings behind it, so an agent whose true competence is not reflected in its persona and files may never be invited to bid. Each invitee costs one LLM call, so auction cost grows with the invited pool before deliberation even begins. Promotion trusts the positive-only citation convention: it presumes agents cite honestly and cannot express disagreement by withholding endorsement alone. And the promotion threshold, the decay window, and the temperature schedule are tuned defaults rather than learned quantities, so a poorly matched task may converge too eagerly, or not at all, until they are re-tuned.

# 7. Evaluation

The preceding chapters introduced an application for configuring different agent setups, including agent providers, communication topologies, consensus protocols, and tool plug-ins. Building these modular components provided direct insight into the requirements of self-organising agent systems and thereby contributed to RQ2. This chapter presents how the resulting configurations were tested. Early in the project, testing consisted of smaller, development-driven experiments that were evaluated qualitatively; these were later replaced by a third-party benchmark with real statistics.

## 7.1. Benchmarking Up to the Midterm

### 7.1.1. A Needle In a Haystack

During the earliest stages of development, the focus lay on static topologies and consensus algorithms based on tallied votes, as described in Chapter 5. The first proposed benchmark was therefore a Needle-in-a-Haystack (NIAH) task. NIAH benchmarks are commonly used to evaluate whether long-context models can extract a key piece of information from a body of distracting text [36]. Two properties made this format a good fit for testing plurality voting and weighted opinion pooling: answers reduce to a single word or short phrase, and the task extends naturally to a multi-agent setting by distributing knowledge files among the agents.

We exposed the benchmark logic through a REST endpoint (see *server/topology/benchmarks/needlehaystack.py*). Given the number of agents and the number of filler sentences per agent, it generates one synthetic knowledge file per agent from combinations of names, organisations, responsibilities, and attributes such as access codes. A strict majority of the agents receives the correct needle sentence, inserted at a random position in their file, while the remaining agents receive a plausible-but-wrong variant naming a different person, so that plurality voting can demonstrably beat the minority. A knowledge file containing the needle looks as follows:

> *Organisation Meridian lists Mira Novak as the contact for internal reporting.* ***The person in Organisation Delta who has project access code Z is Clara Jensen.*** *Sofia Lind works in Organisation Cygnus and is responsible for archive access.*

The endpoint returns the generated files together with a question that rephrases the needle, here: *"Which person in Organisation Delta has project access code Z?"* For plurality voting, we would expect the response with the highest number of votes to be "Clara Jensen." The task proved straightforward for both plurality voting and weighted opinion pooling, and this fast generate-and-test loop allowed rapid progress during early development.

The benchmark also directly informed the design of the protocols. In the first iteration of plurality voting, agents voted solely on the basis of their own knowledge files. After testing

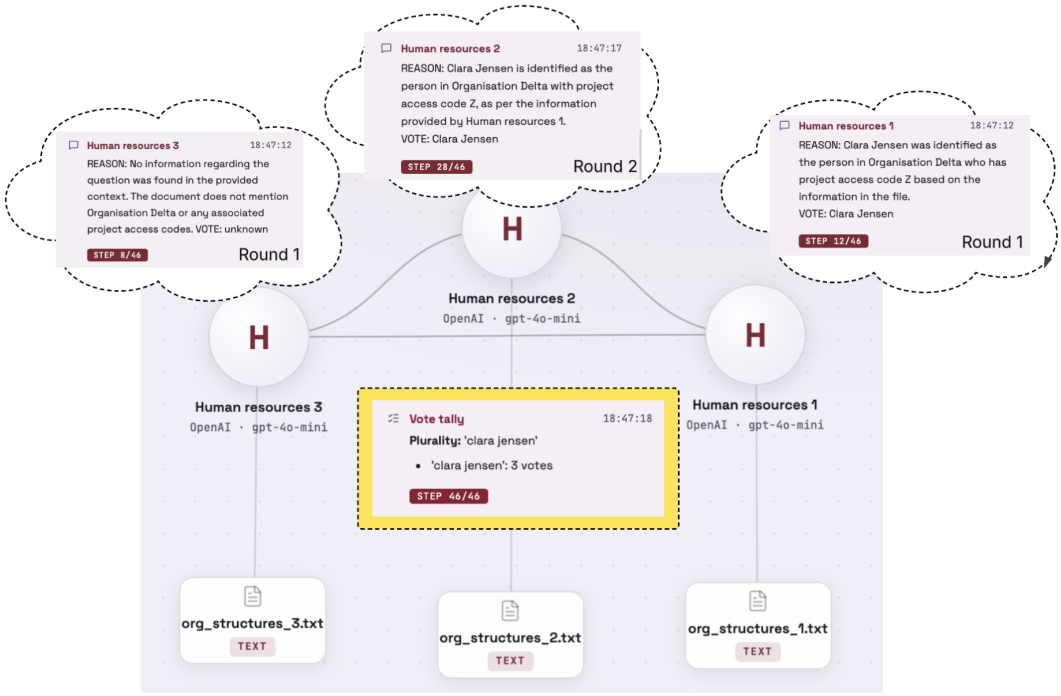



Figure 7.1.: Illustration showing plurality voting consensus applied to NIAH

this implementation on NIAH, we modified the protocol so that agents had to justify their votes, with the justification shared with their peers through the exchanged messages. Figure 7.1 shows the resulting reasoning flow on a three-agent demo variant in which exactly one agent holds the needle: in the first round, agents without relevant knowledge, such as "Human resources 3," abstain, whereas "Human resources 1" votes. In the following round, "Human resources 2" is persuaded by the evidence provided by the other agent. The system thus obtains three votes rather than one, leading to a greater degree of consensus.

### 7.1.2. An Enterprise Resource Planning Task

Following this first iteration, we developed the "bidding" consensus algorithm, which addressed several limitations of the voting-based protocols: agents could choose whether or not to contribute, and a shared blackboard allowed them to post both contributions and gaps in their knowledge. This global view enabled the system to answer more advanced, multi-step questions, which in turn called for a more challenging and more realistic evaluation task.

We selected the Companies dataset, a synthetic enterprise-style dataset derived from Northwind, which Microsoft originally created to demonstrate relational databases [37]. On Hugging Face, it is available as a single CSV file containing 2,680 rows [38], each labelled with a type: invoices (831), shipping orders (810), purchase orders (831), and reports (208). These categories can be distributed approximately evenly among the agents and give each agent a natural area of specialisation.

In the dataset, a field called "order_id" links purchase orders to invoices and shipping orders. Based on this relationship, we defined the task of determining whether a given invoice can be approved for payment. Each agent's role was specified in its system prompt; the "Finance" agent was additionally told that an invoice may only be approved if it has both a corresponding shipping order and a corresponding purchase order. The expected behaviour was therefore:

- The "Finance" agent posts a finding based on its knowledge of the problem.
- The "Finance" agent identifies a knowledge gap and requests confirmation from the "Logistics" and "Sourcing" agents.
- The "Sourcing" and "Logistics" agents post contributions containing entries with the corresponding `order_id`.

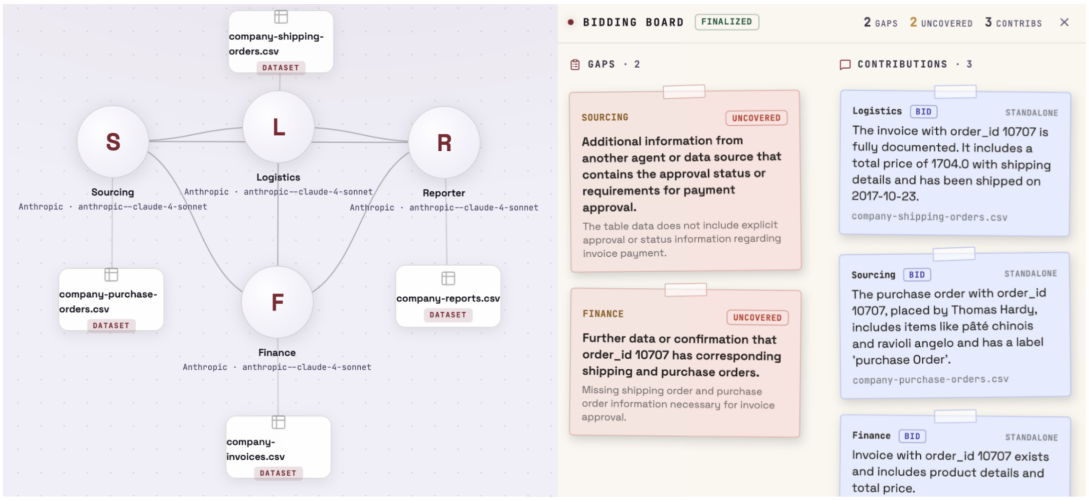



Figure 7.2.: left: Agent setup for bidding, right: bidding-board after consensus

Figure 7.2 shows this behaviour on the question *"Can the invoice with order_id 10707 be approved for payment?"*: the "Finance" agent posts a knowledge gap requesting confirmation of a shipping order and a purchase order, then posts a finding stating that it has a record of the invoice. From the shared information, each agent constructs the correct answer.

Across ten runs of this question, the system produced the correct answer every time, although the contents of the bidding board changed between runs and across LLM providers. On questions that should *not* result in approval, the system often failed; inspection of the board showed that the failures stemmed from agents hallucinating confirmations rather than from the consensus mechanism itself, a behaviour that also varied across models and providers. We did not spend significant time tuning the system prompts to suppress this.

To compare the consensus protocols directly, we ran plurality voting and weighted opinion pooling on the same question ten times each, changing only the consensus algorithm. The results in Table 7.1 support the qualitative picture: bidding is better suited to this kind of task, as it supports open-ended answers and gives agents a global view of the available information.

Table 7.1.: Performance of the consensus approaches on the invoice-approval task across ten repeated trials.

| Consensus approach | Correct responses |
|---|---|
| Bidding | 10/10 |
| Plurality voting | 6/10 |
| Weighted opinion pooling | 2/10 |

## 7.2. Evaluation on External Benchmarks

**Benchmark selection.** To evaluate the system more formally, we searched for a third-party benchmark situated in an enterprise-like environment with business-related tasks. Finding one that met our requirements proved challenging. SILO-Bench [39] evaluates multi-agent coordination over information silos, which matches department-based information splits, but its tasks (distributed top-$k$, global mean, prefix sums, distributed sorting) involve neither consensus nor business activities. MultiAgentBench [40] is closer to business processes (collaborative coding, research collaboration, negotiation) but would have required considerable refactoring, including specialised tool calls for web access and code execution. Deep Search over Heterogeneous Enterprise Data [8] was highly relevant to our setting but designed for a single-agent system. Workspace-Bench [41] targets heterogeneous, company-style documents whose file types allow clear specialisation among agents; however, its knowledge base exceeds 20 GB, which is not feasible in our architecture of in-memory data objects backed by a PostgreSQL container, and some of its tasks require artefacts such as updated files or generated reports rather than textual answers.

We therefore chose FanOutQA in the evidence-provided setting [42]. FanOutQA contains multi-hop questions that require information from multiple documents to be consolidated into a single answer. Although originally designed for a single-agent setting, it fits a conversational multi-agent system well: the tasks are pure natural-language question answering and require no additional tooling, the context files per task are small, and each question is associated with an average of seven knowledge files, which distribute easily across a small number of agents.

**Benchmark setup.** The benchmark uses individual Wikipedia pages as evidence, so we converted each provided page into a plain-text knowledge file. The number of agents assigned to a task was derived from the number of pages:

$$n_{\text{agents}} = \max\left(\texttt{MIN\_AGENTS}, \min\left(\texttt{MAX\_AGENTS}, \left\lceil \frac{n_{\text{pages}}}{\text{max\_pages\_per\_agent}} \right\rceil\right)\right)$$

The hyperparameter values used for the market hive are listed in Table 7.3. To ensure a fair comparison, we ran the same setup with the bidding algorithm, using a maximum of three bidding rounds, and additionally with a single agent. The single-agent setup serves as a reference point: a frontier LLM provided with all the necessary context upfront should produce a near-perfect answer, whereas the agents in the multi-agent systems only see their own data silos. For the single agent, we loaded all relevant Wikipedia pages into the context window together with an instruction prompt and used the zero-shot response as the final

answer. All agents, in every setup, used Anthropic's Claude Sonnet 4.6. Across the three configurations we used three semantically equivalent system prompts, each adapted to the specific setup (Table 7.2).

Table 7.2.: System prompts used for the benchmark experiments.

| Agent setup | System prompt |
|---|---|
| `Market hive` | You are a research assistant working in a team to answer complex questions. You have access to a set of Wikipedia articles as your knowledge base. Read all your files with the `files` tool before contributing to the discussion. Post your findings and partial answers to the blackboard, and cite other agents' entries when they support your reasoning. Your goal is to help the team reach a well-supported consensus answer. Do not answer using world knowledge. All information you contribute must be cited by a source. |
| `Single agent` | You are a research assistant answering complex questions on your own. You have access to a set of Wikipedia articles as your knowledge base. Read all your files with the `files` tool before answering. Your goal is to provide a well-supported, concise answer to the question. Do not answer using world knowledge. All information you provide must be cited inline, by the name of the source file. |
| `Bidding` | You are a research assistant working as part of a coordinated group to answer complex questions. You have access to a set of Wikipedia articles as your knowledge base. Read all your files with the `files` tool before declaring your findings. The system will collect your findings—you do not post to the blackboard directly. Declare only evidence you have actually read this turn; every citation is verified. In the final phase you will see other agents' draft answers and should revise toward a shared consensus. Do not answer using world knowledge. All information you contribute must be cited inline by the name of the source file. |

**Metrics.** FanOutQA scores the generated output against a provided reference string. As an illustration, the first question in the benchmark is:

*"What is the batting hand of each of the first five picks in the 1998 MLB draft?"*

The benchmark provides links to six Wikipedia pages as evidence: 1998_Major_League_Baseball_draft, Pat_Burrell, Mark_Mulder, Corey_Patterson, Jeff_Austin and J. D._Drew, which we fetch via the Wikipedia API and store as one .txt file per page, named after the page title with spaces replaced by underscores. The reference answer for this question is:

*"Pat Burrell - Right, Mark Mulder - Left, Corey Patterson - Left, Jeff Austin - Right, JD Drew - Left"*

Based on this, we compute:

- loose accuracy: number of sub-questions answered

Table 7.3.: Hyperparameter settings used in the market hive.

| Hyperparameter Name | Value | Explanation |
|---|---|---|
| Maximum rounds | 3 | Number of blackboard discussion rounds |
| `MAX_AGENTS` | 8 | Maximum number of agents |
| `MIN_AGENTS` | 4 | Minimum number of agents |
| `swarm_min_takers` | `n_agents` | Number of takers is equal to number of agents |
| `swarm_max_takers` | `n_agents` | Number of takers is equal to number of agents |
| `swarm_invite_floor` | 0.0 | All agents get to participate |
| `swarm_skip_auction` | `true` | All agents get to participate |
| `max_pages_per_agent` | 5 | Maximum number of pages allocated to agent |

- strict accuracy: number of perfect loose scores
- ROUGE-S recall: number of overlapping skip-bigrams per reference answer
- ROUGE-1 recall: number of overlapping unigrams per reference answer

This deviates from the original benchmark in two ways. First, the original reports ROUGE precision and F1 scores as well, but we found that these penalised verbose answers too heavily and therefore measure recall only. Second, we replaced ROUGE-2 with ROUGE-S: bigram overlap unfairly penalises differences in phrasing, whereas skip-bigrams (pairs of words in the same order but not necessarily adjacent) do not.

In addition, we created two supplementary reference strings. The first consists of the file names of the stored Wikipedia pages, so that its ROUGE-1 recall measures whether the relevant citations appear in the generated output:

*"1998_Major_League_Baseball_draft Pat_Burrell Mark_Mulder Corey_Patterson Jeff_Austin__baseball_ J__D__Drew"*

The second lists the answer to each sub-question directly before the corresponding source name, under the assumption that citations follow the answers they support, and should therefore resemble the structure of a well-cited generated response. In the results, ROUGE-1 recall (citations) is computed against the first reference string and ROUGE-S recall (citations) against the second. Note that accuracy and ROUGE-1 count differently: ROUGE-1 counts individual unigram matches, so a player's first and last names are two separate matches, whereas the accuracy metric counts the complete name as a single correct sub-answer.

**Results.** Table 7.4 reports the mean scores over the first 50 questions of the benchmark. We restricted the evaluation to this subset for two reasons: time and cost. At an average of 190 seconds per question, a single pass of the market hive alone takes approximately 150 minutes, and that evaluation cost roughly €500. As the results were already sufficiently conclusive, we

Table 7.4.: Comparison of Market Hive, Single Agent, and Bidding approaches.

| **Metric** | **Market Hive** | **Single Agent** | **Bidding** |
|---|---|---|---|
| Accuracy (loose) | 76.1% | 82.6% | 81% |
| Accuracy (strict) | 36.0% | 46.9% | 46% |
| Average number of rounds | 2.98 | 1 (zero-shot) | 2 |
| Average response time | 190 seconds | 18.5 seconds | 76 seconds |
| Average agents per question | 4.2 | 1 | 4.2 |
| Average sources per question | 7.26 | 6.5 | 7.26 |
| ROUGE-S recall | 0.64 | 0.78 | 0.755 |
| ROUGE-S recall (citations) | 0.54 | 0.75 | 0.72 |
| ROUGE-1 recall (citations) | 0.89 | 0.97 | 0.94 |

considered further runs unnecessary.

The single-agent setup achieved the best overall performance, which is reasonable given that a frontier model was provided with all of the relevant context. The bidding system came second, followed by the market hive, with broadly comparable results overall; the smallest gap appears in ROUGE-1 recall for citations.

We were nevertheless surprised that the market hive performed noticeably worse than the bidding system, given that it represented the culmination of our development efforts: it was designed to reproduce the dynamics of a discussion platform such as Reddit, with dedicated threads, replies between agents, and citation counts. One likely explanation is that the benchmark was conducted while the market hive was still being tuned. As the more complex system, it has more parameters that require careful adjustment; for example, agents must cite one another's posts before a contribution can be promoted to the confirmed layer, which only works if the system prompts instruct them clearly to do so. The system should therefore be given sufficient tuning of these parameters and prompts before it can be tested at full capacity.

At the same time, the accuracy and ROUGE scores across all three setups support a broader conclusion: our self-organising multi-agent systems converge on correct answers effectively, even when faced with data silos, reaching results close to those of a single reasoning agent provided with all data upfront. This directly targets RQ1. The citation scores further show that the agents propagate their sources successfully, adding a layer of explainability to the system. Finally, by deliberately choosing tasks such as the ERP-style invoice approval, the evaluation partly contributes to RQ3, although this question requires further testing and validation before conclusions can be drawn.

## 7.3. Conclusion

Testing guided the development of the systems throughout the project: the informal, development-driven benchmarks shaped the consensus protocols directly, and the project concluded with an evaluation on a third-party benchmark, which should make it easier for future developers to test and compare their own solutions should the project be continued.

Based on our evaluation, we believe the market hive is a strong system that could become highly effective with further development. We therefore recommend additional evaluations, including stress tests with substantially larger knowledge bases, more difficult tasks, and extensive tuning of the system parameters and agent prompts, so that the system is evaluated under conditions that allow it to reach its full potential.

# 8. Future Outlook

This chapter takes stock of what the system, and the project's claims, cannot yet support: first the genuine limitations of the current design, then the work that was consciously scoped out, and finally the directions the platform makes easy to explore next.

## 8.1. Limitations

**Depth of evaluation.** This project is a research prototype, not a full evaluation study, and the benchmarks were not built to specifically showcase the market hive's strengths. Designing tests that do that is left for future work.

**Good-faith citation is assumed.** The citation mechanism has no defence against agents gaming it: one agent citing everything indiscriminately, or two agents citing each other, would distort which entries look well-supported. The same problem applies to bidding: nothing stops an agent from misrepresenting its bid to win a task it isn't actually suited for. A useful stress test would be deliberately planting a wrong, over-citing, or over-bidding agent in a swarm and seeing what happens, connecting to work on debate-based AI oversight [29] and cooperative AI [35].

**Inter-agent security is thinner than user-facing security.** Because all agents write to the same shared blackboard, a single source document containing a prompt injection could, in principle, spread to every agent that reads it. The sanitizer blocks obvious cases, but a stronger safeguard, tracking where each entry's information originally came from and quarantining anything traced back to an untrusted source, has been designed but not yet built.

**Termination quality depends on model compliance.** The swarm stops on its own because agents call `stand_down` when they honestly have nothing left to add. If a model never admits it's done, the swarm just keeps going until it hits `max_rounds`. The only hard limit is a server-side cap on run length; the termination-reason metric in Chapter 7 at least keeps this failure visible when it happens.

## 8.2. Out of Scope

Several items were consciously deferred, recorded here so the boundary of the delivered system is explicit.

- **Priced bids.** Agents bid competence, not cost. Bidding a token-budget estimate would let taker selection trade quality against cost, the natural next step from market-based control [19, 20].

- **Endogenous specialisation.** Trust scores already decay (Bayesian shrinkage, recency-weighted, per domain); an agent's capability embedding does not, staying fixed at creation instead of updating from confirmed work.
- **Citation-hash resolution.** Truncated entry ids resolve inside tool calls but not when quoted in agent prose ("see `fddcbffe`"); no first-class resolvable reference exists yet.
- **Vector-index migration.** In-application cosine scans need to move to pgvector before large agent pools are practical.
- **Multi-turn benchmark coverage.** Follow-up runs (`parent_run_id`, ratified findings carried forward) work end-to-end; the benchmark's `multi-turn` task tier exists but no adapter yet chains a run against it.

## 8.3. Directions

Three lines of work follow naturally from the platform as it stands.

**Model-agnostic swarms.** Mixing model providers is an interesting extension to the evaluation campaign: it directly tests the assumption that collective-decision theory says matters most regardless of the models used.

**Enterprise integration.** For business AI platforms of the kind SAP builds, the swarm can sit behind one API endpoint: send in a contract, get back a ratified answer with a full audit trail. Adding a human reviewer does not require redesigning the system: they simply join the amendment vote as one more equal voter, so they can block a scope change without becoming a privileged controller who overrides everyone else. The confirmed findings, small in number, backed by evidence, and produced as the run goes rather than all at once, give that reviewer something practical to check. The best way to show this is useful in practice, not just in theory, is to pilot it on workflows that are currently reviewed by hand across several domains, procurement intake, audit preparation, compliance screening. The run's full history, already recorded for research purposes, doubles as the audit record these deployments need.

**Topology as an observable.** The platform already records the full history of who cited whom. Studying that citation graph itself, not just the answers it produces, would connect this work back to the self-organisation research it draws on [32, 33]. Likewise, checking how answer quality changes as the number of agents grows would place this system alongside existing results on scaling up sampling [43, 44].

# 9. Reflection

This chapter records what the team would tell a group starting the same project, about the system, and about building it.

## 9.1. On the Architecture

**Judgment decentralises easily, restraint does not.** The energy-contract trace captures the central finding: decentralising *judgment* worked almost immediately, agents found the right defects and corroborated each other without a judge, while decentralising *restraint* did not. Every function an orchestrator quietly performs, rationing speech, deciding when to stop, keeping speakers on topic, reappeared as a failure mode once the orchestrator was removed, and each had to be rebuilt by hand: temperature, delta discipline, stand-down. This is the blackboard literature's control problem [22]. We should have budgeted for it from the start.

**Determinism at the edges made the middle debuggable.** The best early decision was routing all coordination through typed tool calls writing database rows. Stochastic agents on a deterministic substrate meant that strange behaviour, persona bleed, citation stuffing, non-converging equilibria, was diagnosable from a replayable trace, not anecdote. The worst decisions were the places we broke that rule (parsing votes from prose) and later had to repair.

**Baselines earned their keep.** Preserving V0 unchanged as a benchmark mode, and building the deliberately honest minimal supervisor, kept the project falsifiable. The strongest architectural argument here is not that the swarm is clever, but that its comparison against previous versions is mechanical, blind, and reproducible by anyone with the repository.

## 9.2. On the Process

The team ran Scrum with the course's milestone structure, with roles (product owner, scrum master, research, DevOps, documentation, validation) held alongside full development work. Three process choices had substantial returns. *Convention enforcement in tooling*: conventional commits, pre-commit formatting hooks, and CI checks on every pull request meant code review argued about design, not style. *Design documents before contested code*: the larger mechanisms (promotion, quiescence) were specified in reviewed design documents with explicit non-goals, which repeatedly prevented scope creep in the weeks the swarm was misbehaving and the temptation to patch everything at once was strongest. *Demo-driven debugging*: the importable demo configurations doubled as reproduction cases, one of the single most productive artefacts of the semester was a saved canvas whose run reliably exhibited all four conversation-economics failures at once.

## 9.3. Personal Perspectives

Each team member carried a course role alongside full development work, and each role saw the project from a different angle; the perspectives below are the ones we would pass on.

**Product ownership.** The hardest product decision was repeatedly saying no to features the swarm *could* have and asking instead what the demo run *proved*. The harder part was not the deciding but the believing: committing to a project that, at the outset, was little more than a handful of sentences, with no working system yet and no proof the direction was right. Much of the role sat outside the codebase entirely, translating an abstract, pre-enterprise idea into scrum slides that stakeholders and teammates could actually follow, and mediating between a technical team and stakeholders evaluating a research direction rather than a finished product; with nothing familiar to point at, finding the right way to present it took real trial and error. Carrying that alongside the backlog and writing code as one more team member left little room to spare, but it is where the real learning happened: steering a project through genuine uncertainty and keeping a team engaged for the whole of it.

**Research.** The research itself turned out to be challenging in two different ways. On the one hand, designing an orchestrator-less architecture required identifying coordination mechanisms that were both theoretically grounded and practically implementable. While the literature provided valuable concepts and inspiration, it rarely translated directly into implementation details. As a result, many architectural decisions ultimately emerged through iterative prototyping and refinement of our own ideas rather than by following an existing design. On the other hand, finding a suitable benchmark proved even more difficult. Because our project addresses a very specific class of multi-agent systems, existing benchmarks were either too narrow or tailored towards single-agent evaluation. We therefore adapted an established single-agent benchmark to a collaborative multi-agent setting (see Chapter 7).

**Process and coordination.** Running Scrum with six developer-researchers taught us that research tasks do not decompose like feature tickets: on a typical sprint, half the board was questions rather than deliverables. “Does stigmergic decay actually change allocation outcomes?” sits awkwardly next to “add a decay parameter to the config.” A question has no natural estimate and no obvious “done,” so ordinary velocity tracking quietly breaks, and sprint planning drifts toward the tickets that are easy to size rather than the ones that matter. What kept velocity honest was making “a traced run demonstrating X” the definition of done wherever possible: an item closed only when it prod and inspectable run, that let the whole team *see* the behaviour rather than just read a merged diff. Phras system behaviour rather than merged code did three things at once. It gave open-ended research questions a concr could be planned and closed like any other item. It made progress legible in sprint reviews to teammates and stakehoy pull request but could watch a run. And it exposed disagreement early, because two people rarely realise they mea “it works” until they are staring at the same trace. The cost was that some sprints delivered understanding process had to make room for that: an increment of knowledge is still an increment, provided it leaves behind a run someone else can reproduce.

**Infrastructure.** The hardest infrastructure decision was how much structure to put in place when the system itself hadn't settled on what it was yet. Branch protection, precommit linting, a CI pipeline, a database schema, all of it had to get built while the system's architecture, its agents, its seams, its market logic, was still being reshuffled week to week. Doing that setup early, before the codebase grew large enough to make it painful, paid off for the rest of the project: once CI, linting, and branch rules were in place, every later merge stayed clean almost for free. The trickier judgment call was schema: lock it down too early and it has to be ripped out the moment the swarm model changes again, but skip it and everyone else builds on sand. Deployment carried the same tension, keeping a Railway-hosted service usable for a team pushing changes daily, without the checks meant to keep things safe becoming the thing that slows everyone down. Most of this work only gets noticed when it fails; it succeeded when nobody had to think about the plumbing behind a demo. The takeaway wasn't technical so much as a rule of thumb: build just enough to support the next step, early enough that it's cheap to build, for a project where the only sure thing was that it wouldn't stay still.

**Documentation.** Documenting the different topologies, consensus mechanisms, and readings throughout the project proved to be useful compared to documenting them at the end, as it enabled us to correctly identify and cite the strengths and weaknesses of our solutions when planning the next iteration. Documenting previous meeting conclusions is also good practice to prevent any missing and unimplemented requirements from our stakeholders.
What could be improved is better coordination between the documentation within the repository and within the external tool that we used, which was Confluence. Confluence data can be sometimes outdated during the sprint and vice-versa. A possible fix could be to add an additional task for the documenter to update the Confluence to reflect the repository. One other fix is to use the repository as the central documentation place by utilizing Markdown files.

**Validation and benchmarking.** Evaluating a system which uses LLMs at its core is challenging, because initial perceptions of the systems may give a false sense of ability. Attempting to come up with tasks for a system to carry out ad-hoc, makes it difficult to verify whether it is performant or not. What we would recommend instead, is to select a task or domain first, such that it is clear what a final evaluation should include. This also makes it easier to plan out what a potential benchmark might require from the systems in terms of ability, and allows for more time to be spent on fine tuning a system to the task, in order to make the evaluation more fair.
For systems that compare multiple coordination mechanisms, benchmark design becomes an additional challenge. The benchmark must be sufficiently complex to require meaningful coordination between agents, while remaining feasible for all evaluated modes, from a single-agent baseline to fully orchestrator-less swarms. If the tasks are too simple, architectural differences become negligible; if they are too difficult, every configuration performs poorly, making it difficult to distinguish weaknesses in the coordination mechanism from the intrinsic difficulty of the task. Selecting benchmarks that strike this balance proved to be one of the most challenging aspects of the evaluation.

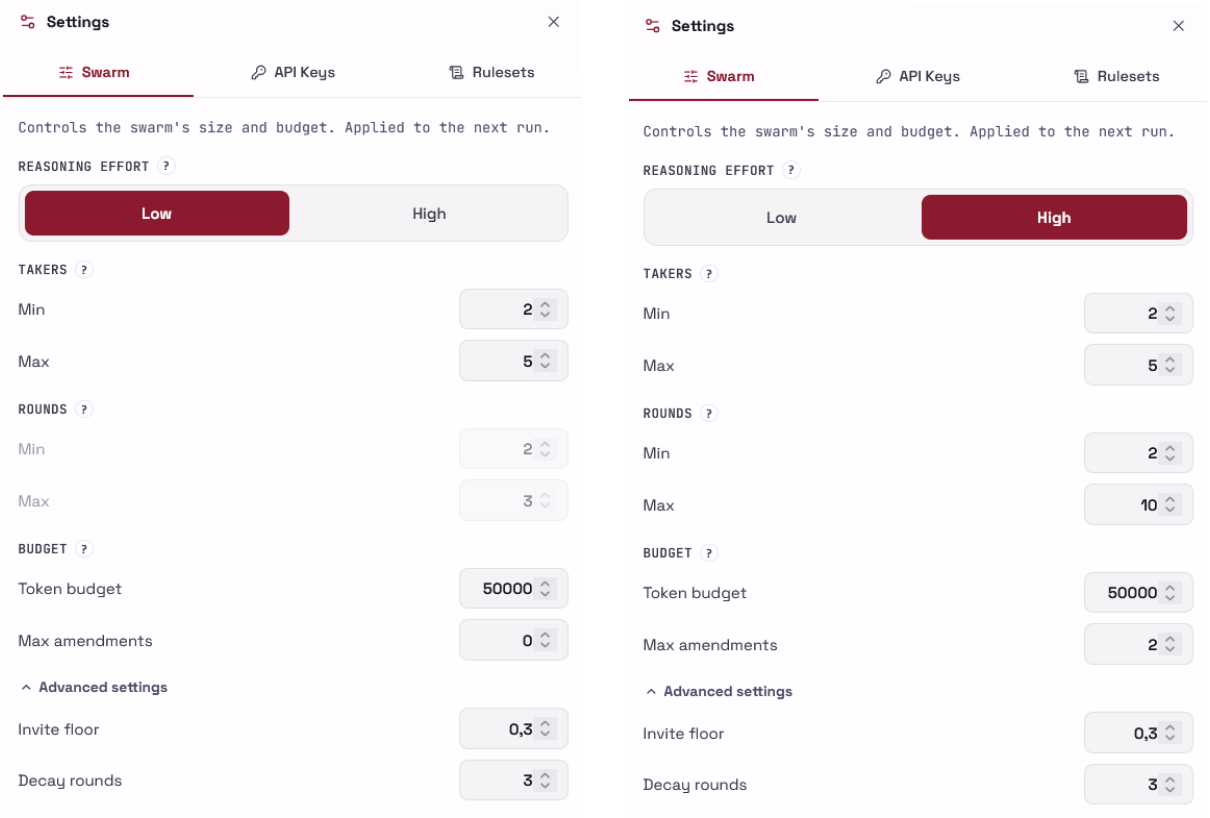
## 9.4. Closing

The question this project set out to answer, can LLM agents coordinate on real business tasks without anyone in charge, has a qualified yes as its answer. Allocation by market, deliberation by blackboard, and ratification by citation all worked as designed. But agents organising themselves well does not mean they naturally know when to stop talking or when the discussion is actually over, that had to be built in just as carefully as getting them to agree with each other. The platform is now built to measure that qualification, and every run it produces leaves a full trace that anyone can replay and check for themselves.

# A. Appendix

## A.1. Configuration Constants

Figure A.1 illustrates the Swarm settings interface, comparing the configuration parameters between the Low and High reasoning effort modes. Notably, increasing the reasoning effort automatically adjusts the maximum allowed rounds and amendments for the swarm.




(a) Low Reasoning Effort

(b) High Reasoning Effort

Figure A.1.: A side-by-side comparison of the Swarm settings.

## A.2. API Surface

This appendix lists the complete REST API surface of the Mycelium platform, grouped by resource.

## Agents

| Method | Endpoint | Description |
|---|---|---|
| POST | /api/agents | Create a new agent |
| GET | /api/agents | List all agents |
| DELETE | /api/agents | Clear the canvas: delete all agents and knowledge files |
| GET | /api/agents/{agent_id} | Get an agent by ID |
| PUT | /api/agents/{agent_id} | Replace an agent |
| PATCH | /api/agents/{agent_id} | Partially update an agent |
| DELETE | /api/agents/{agent_id} | Delete an agent |
| POST | /api/agents/{agent_id}/run | Run an agent |
| GET | /api/agents/{agent_id}/files | List files attached to an agent |
| POST | /api/agents/{agent_id}/files/{file_id} | Attach a file to an agent |
| DELETE | /api/agents/{agent_id}/files/{file_id} | Detach a file from an agent |
| GET | /api/agents/{agent_id}/trust | Get cross-run trust scores for an agent |

Table A.1.: Agent management endpoints

## Knowledge Files & Models

| Method | Endpoint | Description |
|---|---|---|
| POST | /api/files | Create a new knowledge file |
| GET | /api/files | List the caller's knowledge files |
| GET | /api/files/{file_id} | Get a knowledge file by ID |
| PUT | /api/files/{file_id} | Replace a knowledge file |
| PATCH | /api/files/{file_id} | Partially update a knowledge file |
| DELETE | /api/files/{file_id} | Delete a knowledge file |
| GET | /api/models | List available models |

Table A.2.: Knowledge file and model endpoints

## History

| Method | Endpoint | Description |
|---|---|---|
| GET | /api/history/trials | List trials |
| DELETE | /api/history/trials | Clear all run history |

| | | |
|---|---|---|
| GET | /api/history/trials/{trial_id} | Get trial |
| DELETE | /api/history/trials/{trial_id} | Delete a single trial and its history |
| GET | /api/history/trials/{trial_id}/rounds | List rounds |
| GET | /api/history/rounds/{round_id}/connections | List connections |
| GET | /api/history/messages | List messages |

Table A.3.: Run history endpoints

## Statistics

| Method | Endpoint | Description |
|---|---|---|
| GET | /api/statistics/trials/{trial_id}/summary | Trial summary |
| GET | /api/statistics/trials/{trial_id}/agents | Trial agents |
| GET | /api/statistics/trials/{trial_id}/langfuse | Trial Langfuse trace |
| GET | /api/statistics/trials/{trial_id}/edges | Trial edges |
| GET | /api/statistics/trials/{trial_id}/timeline | Trial timeline |
| GET | /api/statistics/experiments | List experiments |
| GET | /api/statistics/experiments/{exp_id}/summary | Experiment summary |
| GET | /api/statistics/experiments/{exp_id}/agents | Experiment agents |
| GET | /api/statistics/experiments/{exp_id}/edges | Experiment edges |
| GET | /api/statistics/experiments/{exp_id}/timeline | Experiment timeline |
| GET | /api/statistics/experiments/{exp_id}/langfuse | Experiment Langfuse trace |
| GET | /api/statistics/experiments/{exp_id}/consistency | Experiment consistency |

Table A.4.: Statistics and analytics endpoints

## Rules & Benchmarks

| Method | Endpoint | Description |
|---|---|---|
| GET | /api/rules | List available reasoning rules |

| | | |
|---|---|---|
| POST | /benchmark/needlehaystack/create | Create a majority-vote benchmark |
| POST | /benchmark/needlehaystack/demo | Create an importable three-agent needle-in-a-haystack demo |
| POST | /benchmark/needlehaystack/evaluate | Evaluate a generated answer for a majority-vote benchmark |

Table A.5.: Rules and benchmark endpoints

## Market-Hive / Simulation

| Method | Endpoint | Description |
|---|---|---|
| GET | /api/simulation/{run_id}/blackboard | Get blackboard state for a run |
| GET | /api/simulation/{run_id}/audit | Full audit trail for a run |
| GET | /api/simulation/{run_id}/status | Run status summary |
| GET | /api/simulation/{run_id}/metrics | Consensus quality metrics for a terminated run |
| POST | /api/simulation/seed | Seed test blackboard data (debug only) |
| POST | /api/simulation/seed-rich | Seed rich multi-agent blackboard scenario (debug only) |
| POST | /api/simulation/{run_id}/resume | Resume an interrupted swarm run |
| POST | /api/simulation/run | Run a topology simulation |

Table A.6.: Simulation and Market-Hive endpoints

## Testing

| Method | Endpoint | Description |
|---|---|---|
| POST | /api/test/openai | Test OpenAI connectivity |
| POST | /api/test/anthropic | Test Anthropic connectivity |
| POST | /api/test/groq | Test Groq connectivity |
| GET | /api/test/tools | List available tools |
| POST | /api/test/tools/files | Test files tool |
| GET | /api/test-stream | Test streaming endpoint |

Table A.7.: Internal testing and diagnostic endpoints

## A.3. Database Schema Model

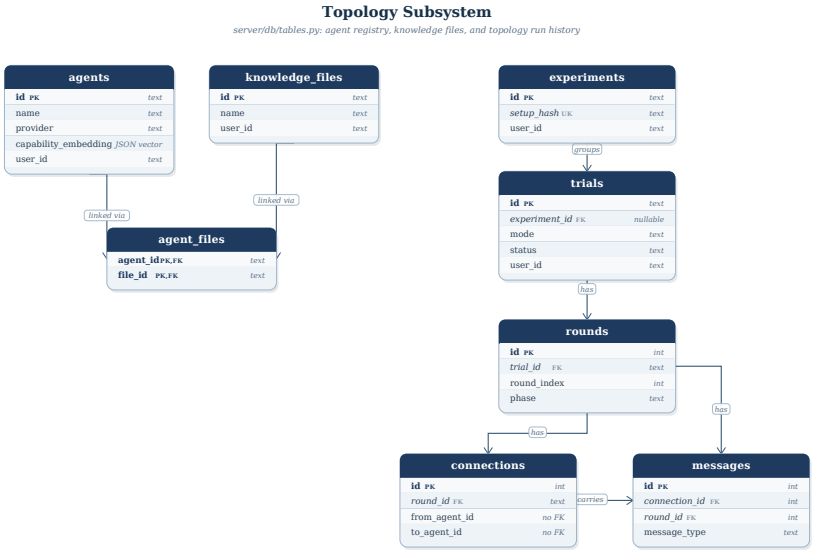



Figure A.2.: Entity-relationship diagram of the topology subsystem.

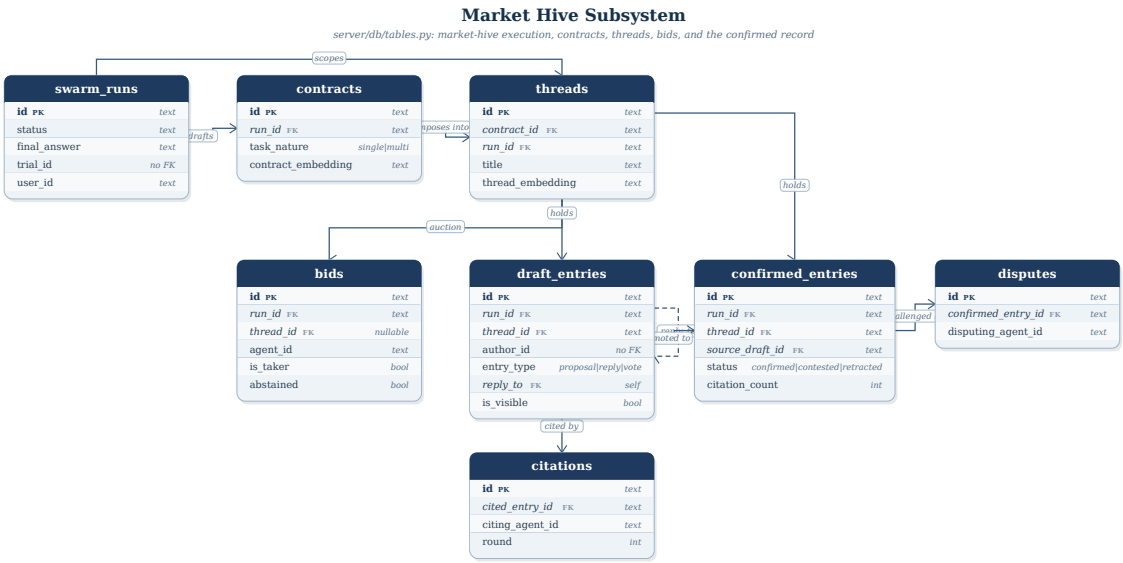



Figure A.3.: Entity-relationship diagram of the Market-Hive subsystem.

## A.4. User Tour: Mycellium

The walkthrough follows a single worked example using static topology: a five-agent swarm (*Contract Counsel*, *Energy Markets Analyst*, *Financial Risk Analyst*, *Regulatory & Compliance*

*Specialist*, and *ESG & Sustainability Advisor*) asked to jointly assess a commercial electricity supply contract.

### A.4.1. The Canvas

The canvas (Figure A.4) is the default view. Each circular node is an agent, labelled with its role and model. Solid lines are static topology edges, defining who may address whom during a run. Cards along the bottom are attached knowledge files, linked to the agents that can read them. The top bar provides the swarm's core actions: **Topology**, **Agent**, **Tools**, **Layout**, **Export**/**Import**, **History** (Appendix A.4.5), **Stats**, **Settings**, and **Clear**.

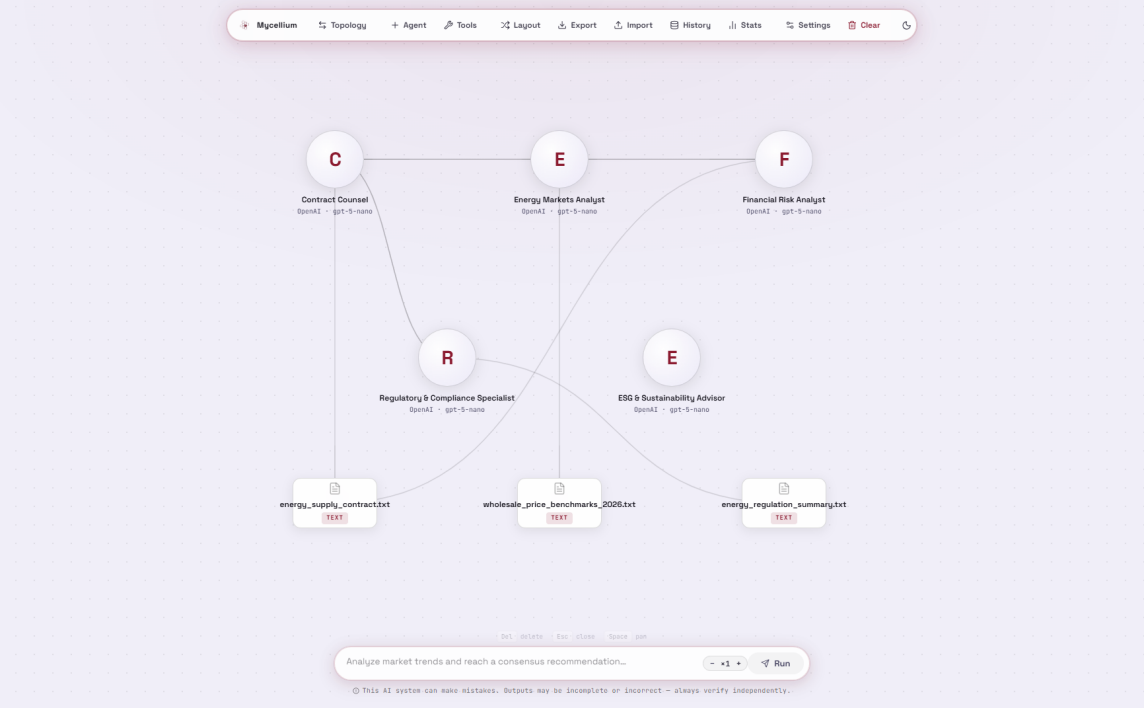



Figure A.4.: The topology canvas with five agents and three attached knowledge files.

### A.4.2. Configuring an Agent

Clicking a node opens **Configure Agent** (Figure A.5). It sets the agent's **Name**, **Description**, **Provider**/ **Model**, and **System Prompt**, and lists its attached **Tools** and **Connections**. **Add file**/**Add table** attach further knowledge sources; **Save Agent**, **Clear**, and **Delete** manage the change.

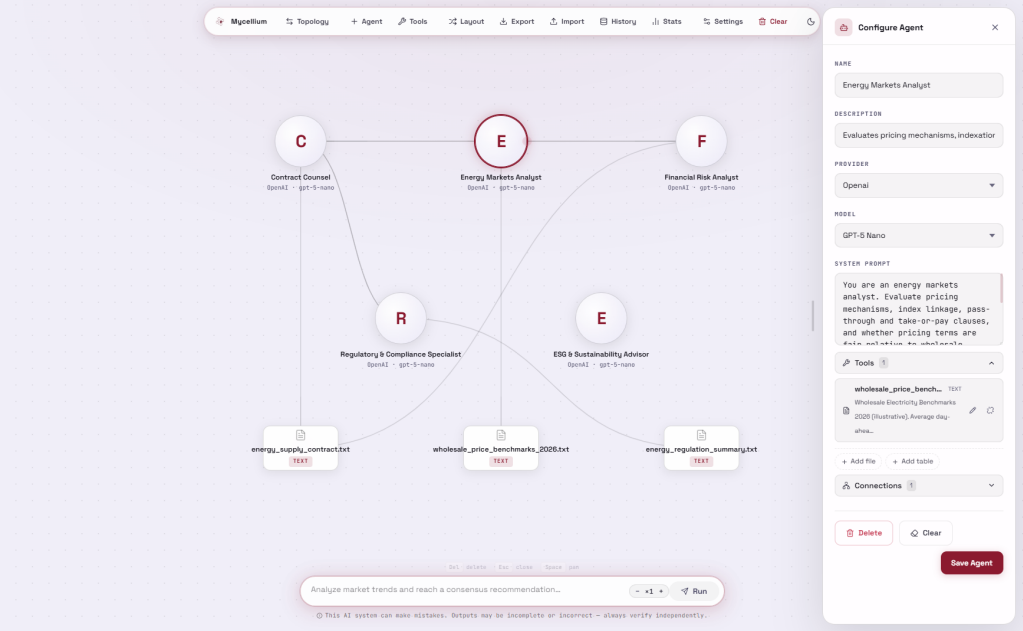



Figure A.5.: The agent configuration panel for the Energy Markets Analyst.

### A.4.3. Attaching Knowledge Files

Clicking a file card opens **Configure File** (Figure A.6), showing its **Type**, **Name**, and **Content**. The **Assign to Agents** chips control read access: red chips are attached agents, grey chips are available but unattached, and toggling one updates the canvas edge immediately.

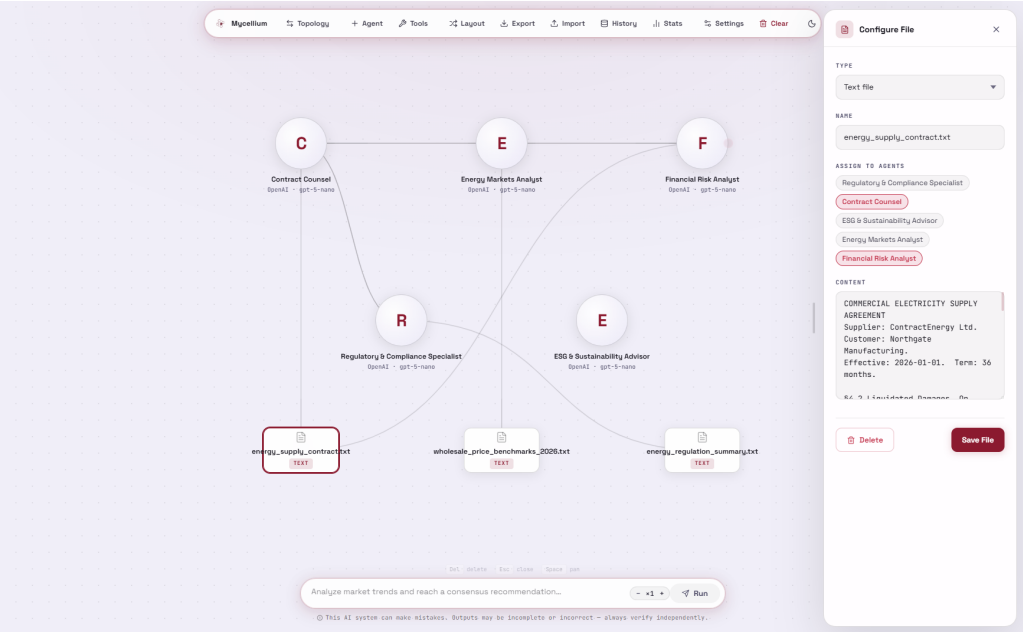



Figure A.6.: Configuring the `energy_supply_contract.txt` knowledge file and its agent assignments.

### A.4.4. Running the Swarm

Clicking **Run** starts execution. Figure A.7 shows a run mid-flight: **Run** becomes **Stop**, the active agent is highlighted on the canvas, and the **Activity Log** streams broadcasts, tool calls, and outputs live. The **Important** filter narrows the log to higher-signal events.

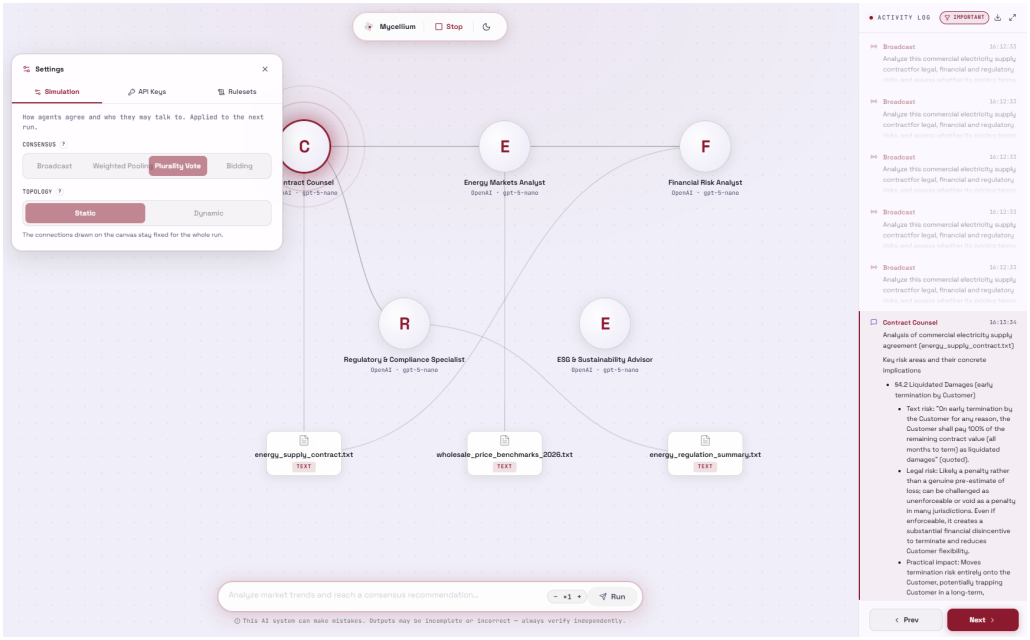



Figure A.7.: A run in progress: Contract Counsel is active, and its output is streaming into the activity log.

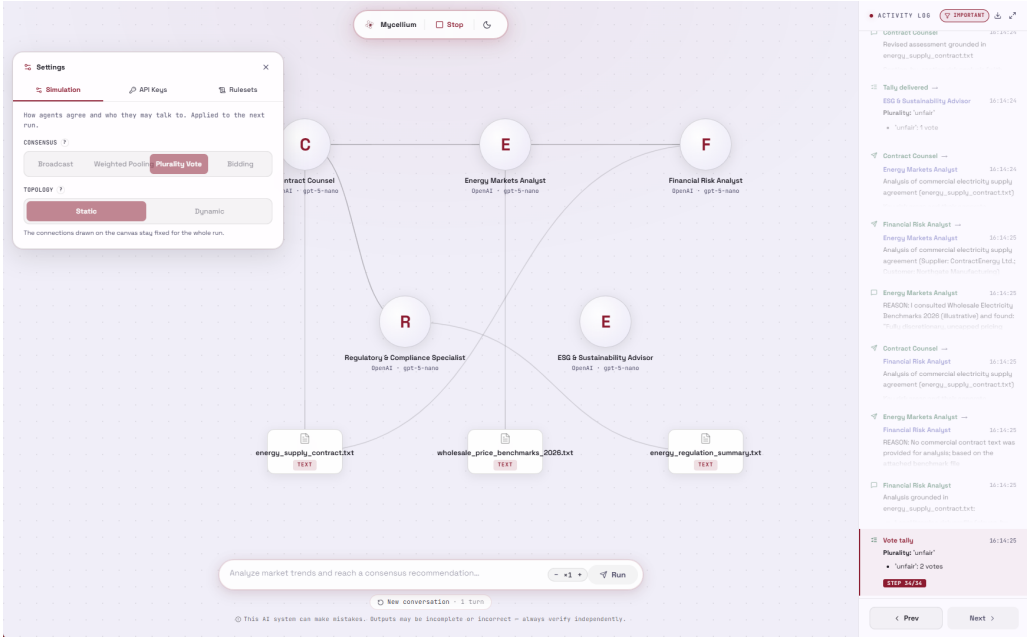



Figure A.8.: A completed run: the activity log shows the final plurality vote tally.

On completion, the log's final entries record the outcome (Figure A.8): a **Vote tally** for plurality runs, plus the terminal step counter (`STEP 34/34`). **Prev**/ **Next** step through the run's history.

### A.4.5. Reviewing Run History

**History** opens the **Swarm Archive** (Figure A.9), which persists every past trial. The **Trials** list on the left filters by run; the message stream on the right is grouped by round and taggable by type (`SYSTEM_BROADCAST`, `TOOL`, `AGENT_OUTPUT`) or searched directly — the primary tool for auditing a run after the fact.

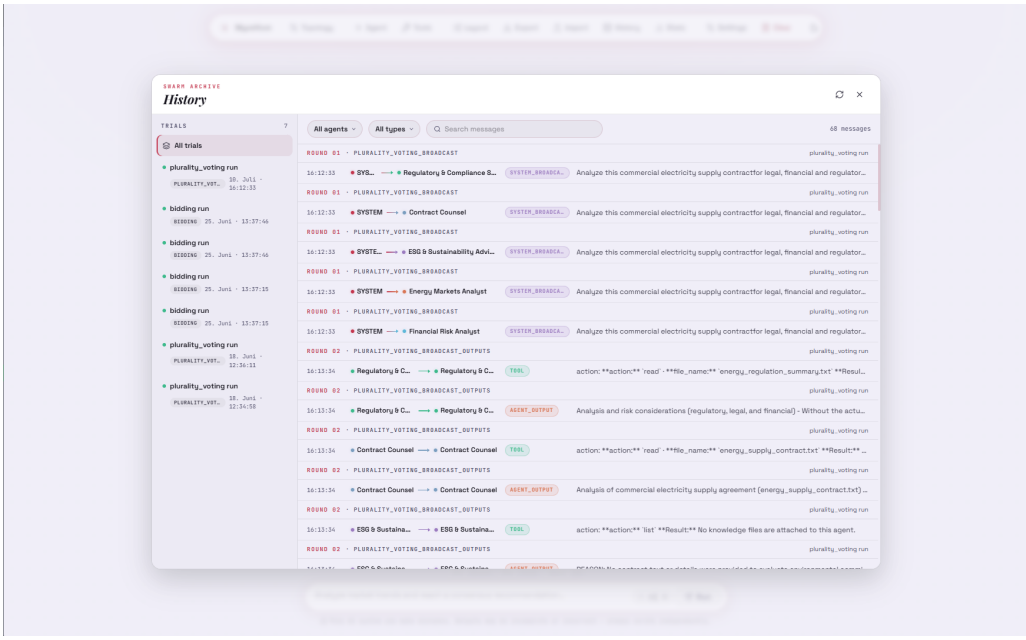



Figure A.9.: The Swarm Archive, showing round-by-round message history across trials.

### A.4.6. Statistics: Collaboration, LLM Usage, and Comparison

Beyond the canvas, the **Statistics** view offers three tabs: **Collaboration**, **LLM usage**, and **Comparison**, for analysing a run after the fact. Each tab is scoped to one selected experiment via the **Experiment** dropdown at the top of the page.

The **Collaboration** tab (Figures A.10 and A.11) shows the run's prompt, convergence mode, and headline averages (**Avg Agents**, **Avg Rounds**, **Avg Messages**, **Avg Edges**, **Runs**), along with a **Self-Organisation over Rounds** chart plotting message volume against active edges per round.

The **Comparison** tab (Figure A.13) places two experiments side by side, selected independently via **First experiment** and **Second experiment**, and mirrors their fixed statistics and message-type mix for direct comparison. This is the primary tool for evaluating engine choice, for instance contrasting a Topology `plurality_voting` run against a Market-Hive `swarm` run on the same underlying task.

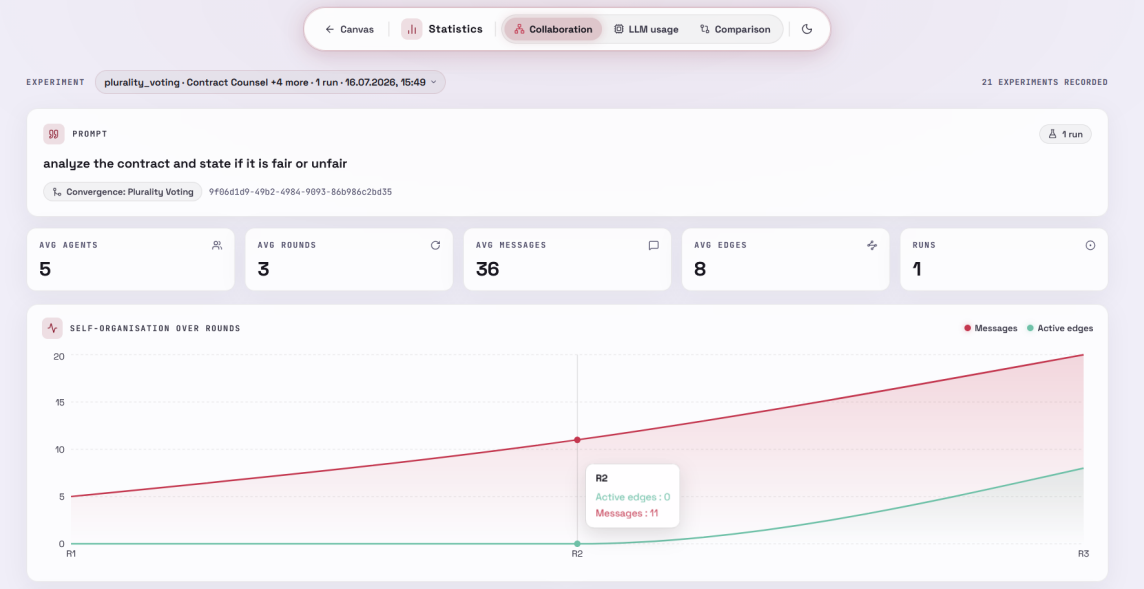



Figure A.10.: The Collaboration tab: prompt, run averages, and self-organisation over rounds.

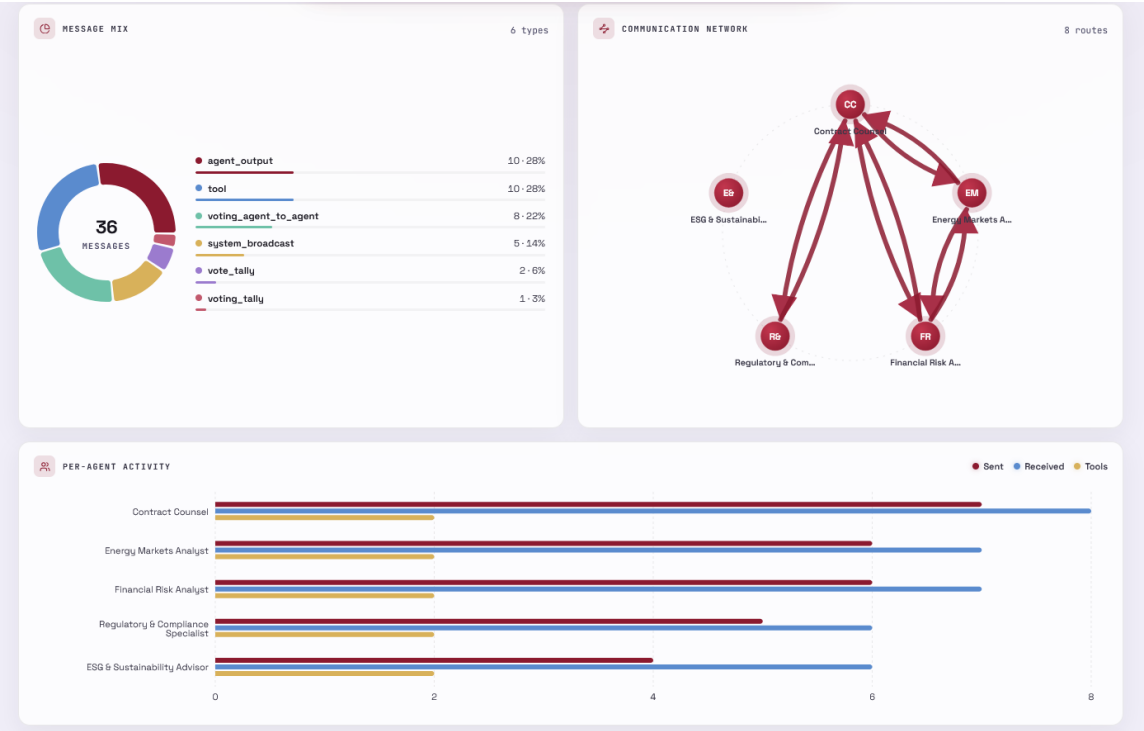



Figure A.11.: The Collaboration tab, continued: message mix, communication network, and per-agent activity.

The **LLM usage** tab (Figure A.12) surfaces the underlying Langfuse traces recorded for each agent call: total **Traces**, **Generations**, **Total Tokens**, **Estimated Cost**, **Avg Latency**, and the **Slowest Model**, along with a cost-and-tokens-over-rounds chart, a token breakdown by model, and per-agent token usage and latency. This is the primary view for diagnosing which agent

or model drives cost or latency in a run.

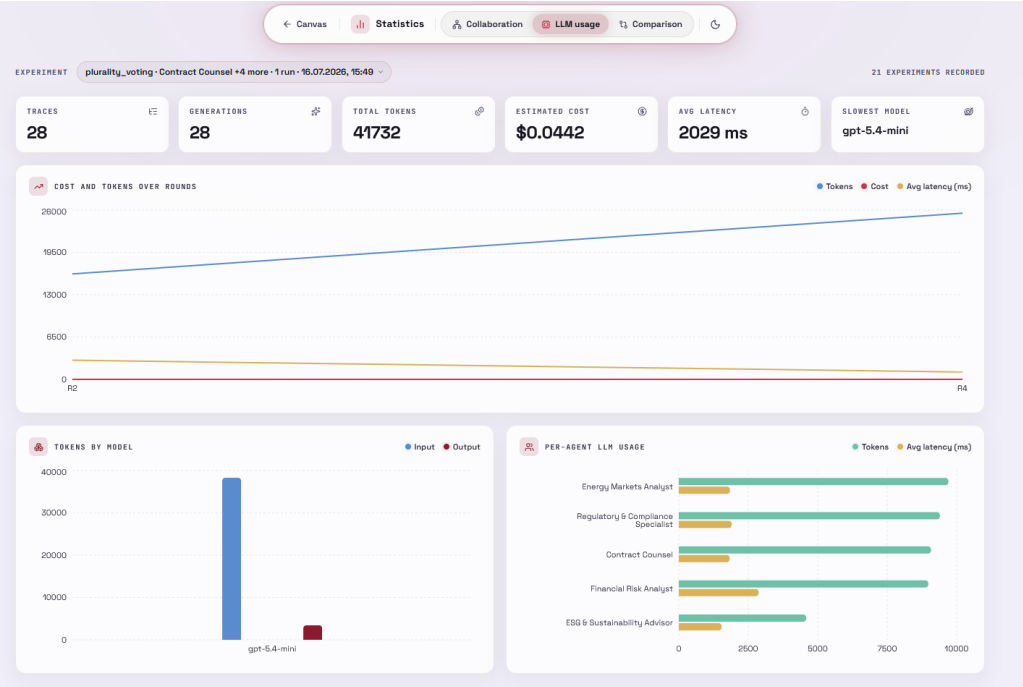



Figure A.12.: The LLM usage tab, sourced from Langfuse traces for the run.

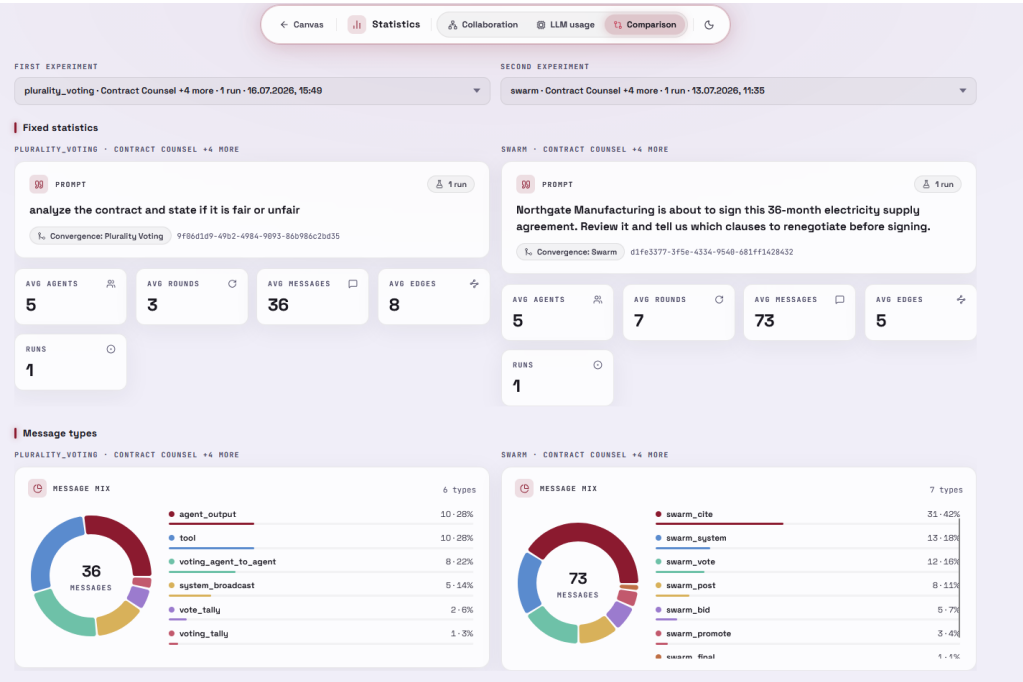



Figure A.13.: The Comparison tab, contrasting a plurality-voting run against a Market-Hive swarm run.

# Bibliography


[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*. 2017.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, et al. "Language Models Are Few-Shot Learners". In: *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*. 2020.

[3] OpenAI. "GPT-4 Technical Report". In: *arXiv preprint arXiv:2303.08774* (2023).

[4] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, et al. "Training Language Models to Follow Instructions with Human Feedback". In: *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*. 2022.

[5] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*. 2022.

[6] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*. 2020.

[7] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. "Toolformer: Language Models Can Teach Themselves to Use Tools". In: *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*. 2023.

[8] P. K. Choubey, X. Peng, S. Bhagavath, K.-H. Huang, C. Xiong, and C.-S. Wu. *Benchmarking Deep Search over Heterogeneous Enterprise Data*. 2025. DOI: 10.48550/arXiv.2506.23139. arXiv: 2506.23139 [cs.CL]. URL: https://arxiv.org/abs/2506.23139.

[9] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. "ReAct: Synergizing Reasoning and Acting in Language Models". In: *International Conference on Learning Representations (ICLR 2023)*. 2023.

[10] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. "Generative Agents: Interactive Simulacra of Human Behavior". In: *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST 2023)*. 2023. DOI: 10.1145/3586183.3606763.

[11] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang. "Large Language Model Based Multi-Agents: A Survey of Progress and Challenges". In: *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)*. 2024.

[12] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, et al. "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation". In: *arXiv preprint arXiv:2308.08155* (2023).

[13] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, et al. "MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework". In: *International Conference on Learning Representations (ICLR 2024)*. 2024.

[14] G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem. "CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society". In: *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*. 2023.

[15] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, et al. "ChatDev: Communicative Agents for Software Development". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*. 2024.

[16] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, et al. "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena". In: *Advances in Neural Information Processing Systems 36 (NeurIPS 2023), Datasets and Benchmarks Track*. 2023.

[17] F. A. Hayek. "The Use of Knowledge in Society". In: *The American Economic Review* 35.4 (1945), pp. 519–530.

[18] R. G. Smith. "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver". In: *IEEE Transactions on Computers* C-29.12 (1980), pp. 1104–1113. DOI: 10.1109/TC.1980.1675516.

[19] S. H. Clearwater, ed. *Market-Based Control: A Paradigm for Distributed Resource Allocation*. Singapore: World Scientific, 1996.

[20] M. P. Wellman. "A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems". In: *Journal of Artificial Intelligence Research* 1 (1993), pp. 1–23. DOI: 10.1613/jair.2.

[21] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty". In: *ACM Computing Surveys* 12.2 (1980), pp. 213–253. DOI: 10.1145/356810.356816.

[22] H. P. Nii. "The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures". In: *AI Magazine* 7.2 (1986), pp. 38–53.

[23] M. J. A. N. d. C. de Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. Paris: Imprimerie Royale, 1785.

[24] F. Galton. "Vox Populi". In: *Nature* 75 (1907), pp. 450–451. DOI: 10.1038/075450a0.

[25] J. Surowiecki. *The Wisdom of Crowds*. New York: Doubleday, 2004.

[26] L. Hong and S. E. Page. "Groups of Diverse Problem Solvers Can Outperform Groups of High-Ability Problem Solvers". In: *Proceedings of the National Academy of Sciences* 101.46 (2004), pp. 16385–16389. DOI: 10.1073/pnas.0403723101.

[27] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch. "Improving Factuality and Reasoning in Language Models through Multiagent Debate". In: *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*. 2024.

[28] T. Liang, Z. He, W. Jiao, X. Wang, Y. Wang, R. Wang, Y. Yang, Z. Tu, and S. Shi. "Encouraging Divergent Thinking in Large Language Models through Multi-Agent Debate". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP 2024)*. 2024.

[29] G. Irving, P. Christiano, and D. Amodei. "AI Safety via Debate". In: *arXiv preprint arXiv:1805.00899* (2018).

[30] P.-P. Grassé. "La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs". In: *Insectes Sociaux* 6 (1959), pp. 41–80. DOI: 10.1007/BF02223791.

[31] M. Dorigo, V. Maniezzo, and A. Colorni. "Ant System: Optimization by a Colony of Cooperating Agents". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26.1 (1996), pp. 29–41. DOI: 10.1109/3477.484436.

[32] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. New York: Oxford University Press, 1999.

[33] F. Heylighen. "Stigmergy as a Universal Coordination Mechanism I: Definition and Components". In: *Cognitive Systems Research* 38 (2016), pp. 4–13. DOI: 10.1016/j.cogsys.2015.12.002.

[34] W. Chen, Y. Su, J. Zuo, C. Yang, C. Yuan, C.-M. Chan, H. Yu, et al. "AgentVerse: Facilitating Multi-Agent Collaboration and Exploring Emergent Behaviors". In: *International Conference on Learning Representations (ICLR 2024)*. 2024.

[35] A. Dafoe, Y. Bachrach, G. Hadfield, E. Horvitz, K. Larson, and T. Graepel. "Cooperative AI: Machines Must Learn to Find Common Ground". In: *Nature* 593 (2021), pp. 33–36. DOI: 10.1038/d41586-021-01170-0.

[36] C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekesh, F. Jia, Y. Zhang, and B. Ginsburg. "RULER: What's the Real Context Size of Your Long-Context Language Models?" In: *arXiv preprint arXiv:2404.06654* (2024).

[37] Mythilyram. *The Northwind Database*. https://medium.com/@mythilyrm/the-northwind-database-f2fa1f59daa7. Medium article. Accessed: 2026-07-09. Apr. 2023.

[38] A. Cherguelaine. *Company Documents Dataset*. https://www.kaggle.com/datasets/ayoubcherguelaine/company-documents-dataset. Kaggle dataset. Accessed: 2026-07-09. 2024.

[39] Y. Zhang, F. Liu, Y. Shan, X. Huang, X. Yang, Y. Zhu, X. Cheng, C. Liu, K. Zeng, T. J. Zhang, and W. Jiang. *Silo-Bench: A Scalable Environment for Evaluating Distributed Coordination in Multi-Agent LLM Systems*. 2026. DOI: 10.48550/arXiv.2603.01045. arXiv: 2603.01045 [cs.MA]. URL: https://arxiv.org/abs/2603.01045.

[40] K. Zhu, H. Du, Z. Hong, X. Yang, S. Guo, Z. Wang, Z. Wang, C. Qian, X. Tang, H. Ji, and J. You. *MultiAgentBench: Evaluating the Collaboration and Competition of LLM agents*. 2025. DOI: 10.48550/arXiv.2503.01935. arXiv: 2503.01935 [cs.MA]. URL: https://arxiv.org/abs/2503.01935.

[41] Z. Tang, X. Zhou, Y. Liu, L. Li, Y. Wu, W. Wang, H. Huang, W. Zhou, J. Zhou, J. Song, S. Yu, J. Wang, Z. Zhou, H. Zhou, Y. Lv, J. Li, J. Liu, R. Chen, C. Liu, G. Li, J. Kang, and F. Wu. *Workspace-Bench 1.0: Benchmarking AI Agents on Workspace Tasks with Large-Scale File Dependencies*. 2026. DOI: 10.48550/arXiv.2605.03596. arXiv: 2605.03596 [cs.AI]. URL: https://arxiv.org/abs/2605.03596.

[42] A. Zhu, A. Hwang, L. Dugan, and C. Callison-Burch. "FanOutQA: A Multi-Hop, Multi-Document Question Answering Benchmark for Large Language Models". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Ed. by L.-W. Ku, A. Martins, and V. Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 18–37. DOI: 10.18653/v1/2024.acl-short.2. URL: https://aclanthology.org/2024.acl-short.2/.

[43] J. Li, Q. Zhang, Y. Yu, Q. Fu, and D. Ye. "More Agents Is All You Need". In: *arXiv preprint arXiv:2402.05120* (2024).

[44] J. Wang, J. Wang, B. Athiwaratkun, C. Zhang, and J. Zou. "Mixture-of-Agents Enhances Large Language Model Capabilities". In: *arXiv preprint arXiv:2406.04692* (2024).