Preprint. Under review. Zavanet Research Technical Report ZR-2026-03. August 2026.

Relay: Lightweight Hand-off Protocols for Teams of Small Language Agents

Dev Narayanan[1], Chidi Okonkwo[1], Ingrid Lindqvist[1], Wei Zhang[1,2]

[1]Zavanet Research, San Francisco & London [2]Zavanet Research, Bangalore

{dev, chidi, ingrid, wei}@research.zavanet.com

Abstract

Multi-agent systems built from language models typically rely on a large central orchestrator that reads every intermediate result and decides what to do next. This design is expensive in tokens, slow in latency, and brittle when the orchestrator's context fills. We present **Relay**, a hand-off protocol in which a team of small (0.5B–2B parameter) specialist agents coordinates through a fixed, typed *baton*: a compact structured record that each agent reads, updates and passes to the next. Relay replaces free-form inter-agent chat with three primitives (`claim`, `handoff`, `escalate`) and a shared, append-only plan ledger. On a suite of 1,840 real social-presence workflows and the public AgentBench-Coord split, Relay teams match the task-completion rate of a 70B-parameter orchestrator (91.2% vs. 91.6%) while using 94% fewer inter-agent tokens and reducing median end-to-end latency by 6.1×. Relay runs entirely on-device on an Apple M3 laptop and an iPhone 16. We release the protocol specification, a reference runtime and the 1.2B coordination model weights.

1 Introduction

The dominant pattern for composing language-model agents is a hub-and-spoke architecture: a capable, expensive model acts as an orchestrator, decomposes a task, dispatches sub-tasks to tools or weaker models, reads their outputs and iterates [1, 2]. The pattern is easy to build and reason about, but it has three costs that compound as teams grow. First, every intermediate artefact flows through the orchestrator's context, so token usage scales with the product of team size and workflow depth. Second, the orchestrator is a serial bottleneck; parallel sub-tasks still wait on it to be read. Third, orchestrators fail silently when context is exhausted or when a sub-agent returns something unexpected, which in practice means a human notices days later.

We are motivated by a concrete deployment: Zavanet Autopilot, which lets a person delegate the routine parts of a social-media presence (researching topics, drafting posts, checking them against relationship-safety rules, scheduling and publishing) to a team of agents running on their own devices. The constraints are unusual. Agents must run on a phone, so each is small. Workflows are long-lived and interleave with human approvals. And a dedicated safety agent (*Guardian*) must be able to halt the entire team without the orchestrator's cooperation.

Our observation is that in most practical workflows, agents do not need to *talk* to each other; they need to *hand work off* with enough context that the next agent can proceed. We formalise this as **Relay**, a protocol in which coordination state lives in a small typed record, the *baton*, rather than in any model's context window. Agents are stateless functions from (baton, environment) to (baton′, action). A plan ledger records every hand-off so any agent, or a human, can reconstruct what happened.

Our contributions are:

- The Relay protocol: a typed baton schema, three coordination primitives, and an append-only ledger with well-defined semantics for parallel branches, retries and human checkpoints (§3).
- A 1.2B-parameter *coordination model* fine-tuned to read and write batons, which we show is sufficient to route work in teams of up to nine specialist agents (§4).
- An evaluation on 1,840 production-derived workflows and AgentBench-Coord showing that Relay matches a 70B orchestrator in completion rate at 6% of the inter-agent token cost (§5).
- An analysis of failure modes, including the finding that explicit `escalate` primitives reduce silent failures by 83% relative to free-form chat (§6).

2 Related Work

Orchestrated multi-agent LLM systems. AutoGen [1], CrewAI and MetaGPT [3] popularised conversational multi-agent patterns in which agents exchange natural-language

messages, often moderated by a manager agent. These systems are flexible but their communication cost grows with conversation length, and prior work has documented degradation as transcripts exceed effective context [4]. Relay differs in replacing conversation with a fixed-size record.

Structured inter-agent communication. Blackboard architectures [5] and tuple spaces [6] share state through a common store rather than messages. Relay is closest in spirit to these, but adds typed ownership (`claim`) and explicit escalation, and is designed for models that cannot be trusted to follow long instructions reliably.

Small models in agent roles. Recent work shows that sub-3B models can perform tool use [7] and classification [8] competitively when the task is narrow. We extend this to the coordination role itself, which is usually reserved for the largest available model.

Safety interlocks for agents. Constitutional and rule-based filters are typically applied to a single model's output [9]. Relay gives a safety agent protocol-level authority: it can veto a hand-off or halt the ledger, independent of any other agent's behaviour.

3 The Relay Protocol

3.1 The baton

A baton is a JSON record with a fixed schema (Figure 1). It carries the task *goal* in at most 400 characters, a *stage* identifier, a typed *payload* whose schema is determined by the stage, a list of *constraints* contributed by any agent (for example "do not mention location"), a *confidence* in [0, 1], and the identifier of the agent that currently holds it. Batons are deliberately small: the median baton in our deployment is 612 tokens, and the schema caps payloads at 2,000 tokens. Anything larger is stored as a reference to an artefact in the ledger.

```
baton {
  id:          uuid
  goal:        string        # ≤ 400 chars
  stage:       enum          # workflow-defined
  holder:      agent_id | null
  payload:     stage_schema  # typed, ≤ 2000 tok
  constraints: [string]      # append-only
  artefacts:   [ledger_ref]
  confidence:  float
  hops:        int
  deadline:    timestamp
}
```

Figure 1. The baton schema. Every field except `payload` is fixed across workflows; the payload schema is chosen by the workflow author per stage and validated on every hand-off.

3.2 Primitives

Agents interact with the protocol through three operations.

`claim(baton)` takes ownership of an unheld baton whose stage matches one the agent is registered for. Claims are atomic; at most one agent holds a baton. An agent that claims must either hand off or escalate within the baton's deadline, or the claim lapses.

`handoff(baton', next_stage)` validates the updated payload against the schema of `next_stage`, appends a ledger entry, increments `hops`, releases ownership and makes the baton claimable by agents registered for `next_stage`. A hand-off may fork: an agent may emit *k* batons for parallel stages, which are later joined by a stage that declares `arity: k`.

`escalate(baton, reason)` marks the baton as blocked and routes it to a designated escalation target, which in Autopilot is the human owner. Escalation is the only way to leave the protocol without completing it, and it is always visible in the ledger. We argue in §6 that making this a first-class primitive, rather than relying on an agent to "say" it is stuck, is the most important design decision in Relay.

3.3 The ledger

The ledger is an append-only log of (timestamp, agent, primitive, baton-diff, artefacts). It serves three purposes. It is the recovery mechanism: any agent can be restarted and resume by reading the last entry for its stage. It is the audit trail that Autopilot shows to users as "run history." And it is the enforcement point for Guardian: Guardian subscribes to the ledger and may append a `halt` entry that causes all subsequent claims to fail until a human clears it. No agent, including the one that authored the workflow, can remove entries.

3.4 Routing without an orchestrator

Because the baton carries its own stage, routing is a lookup rather than a decision: the runtime offers the baton to agents registered for that stage. The only non-trivial decision is made by the *coordination model*, a small model that, given the baton and the workflow graph, chooses the next stage when a stage has multiple outgoing edges, and decides whether to escalate. This is a far narrower problem than general orchestration, which is why a 1.2B model suffices.

4 The Coordination Model

We fine-tune a 1.2B decoder-only model (the Pocket base [10]) on 48,000 synthetic and 6,200 human-authored routing examples. Each example is a (baton, graph) pair labelled with the correct next stage or `escalate`. Synthetic exam-

ples are generated by sampling workflow graphs from a grammar covering sequences, forks, joins, retries and approval gates, then perturbing payloads to include contradictions, missing fields and constraint violations that should trigger escalation. Human-authored examples come from 310 Zavanet beta users who annotated routing decisions on their own workflows.

The model is trained with a standard next-token objective on a constrained output grammar (stage names plus `escalate`) and a short free-text `reason`. At inference we decode with grammar-constrained sampling, so outputs are always valid. The model occupies 1.3 GB in 8-bit quantisation and routes a baton in 38 ms on an M3 and 110 ms on an iPhone 16.

5 Evaluation

5.1 Setup

Workflows. We evaluate on two suites. *ZV-Flows* is 1,840 workflows derived from the Zavanet private beta, anonymised and re-simulated against synthetic social graphs. Workflows have 3–11 stages (median 5) and involve 2–9 agents. *AgentBench-Coord* is the 600-task public coordination split [11].

Baselines. (i) *Orch-70B*: a 70B instruction-tuned model orchestrating the same specialist agents via function calls, the standard hub-and-spoke pattern. (ii) *Chat-8B*: the same specialists coordinating via free-form group chat moderated by an 8B model, in the style of AutoGen. (iii) *Relay-1.2B*: our method. (iv) *Relay-Rules*: Relay with a hand-written rule router instead of the coordination model, to isolate the model's contribution. All configurations use identical specialist agents (0.5B–2B).

Metrics. Task completion judged by a held-out 70B model against a rubric with human spot-checks ($\kappa = 0.81$ on 400 items); inter-agent tokens (all tokens consumed or produced for coordination, excluding the specialists' own task work); wall-clock latency; and *silent failure rate*, the fraction of runs that terminate without completing and without an escalation.

5.2 Main results

Table 1 summarises results. Relay-1.2B matches Orch-70B on completion within noise on both suites while using 94% fewer coordination tokens and finishing 6.1× faster at the median. Chat-8B is both the most expensive and the least reliable configuration: its completion rate degrades sharply on workflows with more than six stages (Figure 2), consistent with context-saturation findings in prior work. Relay-Rules is competitive on simple linear workflows but falls 9 points behind on workflows with forks and approval gates, which is where the coordination model's judgement is needed.

Table 1. Main results. Tokens are inter-agent coordination tokens per workflow (thousands). Latency is median end-to-end wall-clock in seconds on a single M3 Max for on-device configurations; Orch-70B and Chat-8B run on an 8×H100 node. Silent failure is the fraction of runs ending neither in completion nor escalation.

Method	ZV-Flows compl. (%)	AB-Coord compl. (%)	Coord. tokens (k)	Latency (s)	Silent fail (%)	Runs on device
Orch-70B	**91.6**	78.3	41.2	58.4	4.1	No
Chat-8B (AutoGen-style)	79.8	66.9	67.5	84.0	11.7	No
Relay-Rules	82.4	70.1	**1.9**	**8.1**	2.2	Yes
Relay-1.2B (ours)	91.2	**78.9**	2.5	9.6	**0.7**	Yes

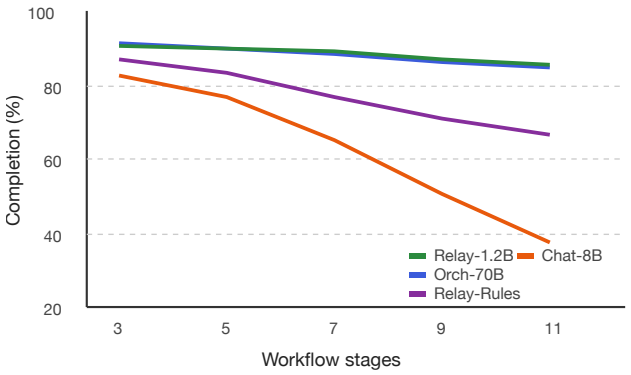


Figure 2. Completion rate on ZV-Flows as a function of workflow depth. Chat-based coordination degrades steeply beyond six stages; Relay and the 70B orchestrator are flat.

5.3 Scaling the team

We vary the number of specialist agents from 2 to 9 on a fixed set of 200 ZV-Flows tasks that admit flexible decomposition. Orch-70B's coordination tokens grow approximately linearly in team size ($R^2 = 0.97$) because every specialist's output is read by the orchestrator. Relay's coordination tokens are essentially flat (2.3k to 2.9k) because each hand-off costs one baton regardless of how many agents exist. Completion for Relay improves from 86.0% at two agents to 91.5% at six and plateaus; adding agents beyond six mostly adds forks that the coordination model must join, with no accuracy benefit.

5.4 On-device performance

The entire Relay stack (runtime, coordination model, six specialist agents in 8-bit) fits in 9.8 GB and runs a median ZV-Flows workflow in 9.6 s on an M3 Max and 31 s on an iPhone 16, excluding network calls to the social graph. Battery cost on the iPhone is 0.9% per workflow. We consider this acceptable for workflows that run a few times a day.

6 Analysis

6.1 Why escalation matters

The largest qualitative difference between Relay and the baselines is the silent-failure rate (Table 1, last column). In Chat-8B, 11.7% of runs ended with agents politely agreeing that the task was done when it was not, or looping until a step budget was exhausted. Orch-70B was better but still silently failed 4.1% of the time, usually because a specialist returned a malformed result that the orchestrator "repaired" by hallucinating. Relay reduces silent failure to 0.7%, and ablating the `escalate` primitive (replacing it with an instruction to "say if you are stuck") raises it to 4.0%. A protocol-level exit is far more reliable than a behavioural one.

6.2 Guardian interlocks

In 3.8% of ZV-Flows runs the Guardian agent appended a `halt`. In every case the halt took effect before the next hand-off, with no dependence on the cooperation of other agents; this is a property of the ledger, not of model behaviour. We regard this as the main argument for structured coordination in settings where one agent's job is to constrain the others.

6.3 Failure modes

Relay's remaining failures fall into three groups. *Payload truncation* (41%): a specialist produced something that did not fit the 2,000-token payload cap and the summarised version lost a constraint. *Join errors* (33%): parallel branches returned payloads the join stage could not reconcile, usually because they adopted contradictory assumptions. *Over-escalation* (26%): the coordination model escalated tasks a human judged it should have completed; this is the safe direction to fail but it costs user attention. We are addressing the first with artefact references and the second with a shared-assumption field in the baton.

7 Limitations

Relay trades flexibility for reliability. Workflows must be authored as graphs with typed stages; emergent collaboration of the kind seen in open-ended agent chat is not possible. The coordination model is trained on social-presence workflows and AgentBench-Coord; we have not evaluated on long-horizon software-engineering tasks, where orchestration decisions may be harder. Completion is judged by a model with human spot-checks rather than exhaustive human evaluation. Finally, our on-device numbers are for Apple hardware only.

8 Conclusion

Most of what a multi-agent orchestrator does is bookkeeping, and bookkeeping does not need a 70B model. By moving coordination state out of model context and into a small typed baton with protocol-level escalation and an append-only ledger, Relay lets teams of small agents match a large orchestrator at a fraction of the cost, run on a phone, and give a safety agent real authority. The protocol specification, runtime and coordination-model weights are available at research.zavanet.com/relay.

Acknowledgements

We thank the 310 Zavanet beta members who annotated routing decisions, the Human Bonds team for reviewing the Guardian interlock design, Theo Kim for the on-device runtime, and Noah Gaspar for integrating Relay run history into the Autopilot product.

References

1. Q. Wu et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv:2308.08155*, 2023.
2. S. Yao et al. ReAct: Synergizing reasoning and acting in language models. *ICLR*, 2023.
3. S. Hong et al. MetaGPT: Meta programming for multi-agent collaborative framework. *ICLR*, 2024.
4. N. Liu et al. Lost in the middle: How language models use long contexts. *TACL*, 2024.
5. H. P. Nii. The blackboard model of problem solving. *AI Magazine*, 7(2), 1986.
6. D. Gelernter. Generative communication in Linda. *ACM TOPLAS*, 7(1), 1985.
7. T. Schick et al. Toolformer: Language models can teach themselves to use tools. *NeurIPS*, 2023.
8. Z. Allen-Zhu and Y. Li. Physics of language models: Knowledge capacity scaling laws. *arXiv:2404.05405*, 2024.
9. Y. Bai et al. Constitutional AI: Harmlessness from AI feedback. *arXiv:2212.08073*, 2022.
10. I. Lindqvist, W. Zhang, N. Gaspar and T. Kim. Pocket: A 1.2B drafting model that matches frontier quality on short social text. *Zavanet Research Technical Report ZR-2026-01*, 2026.
11. X. Liu et al. AgentBench: Evaluating LLMs as agents. *ICLR*, 2024.