Accelerating Multi-Agent Orchestration with Speculative Dispatching

Gnosis Research Center @ ILLINOIS TECH

Jie Ye Shazzadul Islam Jaime Cernuda Xian-He Sun Anthony Kougkas
jye20@hawk.illinoistech.edu, sislam6@hawk.illinoistech.edu,
jcernudagarcia@illinoistech.edu, sun@illinoistech.edu, and akougkas@illinoistech.edu

pythia MULTI-AGENT ORCHESTRATION

Introduction

- **AI agents are reshaping scientific computing** — HPC code generation, data-driven discovery, computational chemistry, and self-driving laboratories — and **workflows are restructuring around multi-agent coordination across heterogeneous resources and tools:**

ORNL Cross-Facility Workflow	ChemGraph (Argonne)	SWARM	IOWarp
CrewAI agents bridging instruments and HPC across facilities	LangGraph agents dispatching across chemistry simulation backends	Decentralized agentic workflow management resilient to failures	MCP servers exposing HPC-native I/O and persistent context to agents

- Existing orchestration frameworks fall into two camps:
  - **Static execution graphs** (LangGraph, CrewAI) - agents and their execution order are predefined before running.
  - **LLM-greedy routing** (AutoGen, LangChain) - lets the LLM pick the next agent step-by-step.
- **However, both have three limitations:**
  - Neither considers available resources (e.g., GPU, memory, and token budget) when deciding *which model or agent runs a subtask*.
  - As **agents/tools grow**, orchestration decisions face more candidates: getting slower and harder to make a plan.
  - Most rely on a **sequential plan-then-execute** pattern.
- **Existing agentic speculation** (e.g., ISP, DSP, SPAgent, PASTE, Sherlock) speculates *within a single agent's trace* — none target the **orchestration and dispatch decision** itself.
- We are the first to **apply the speculation paradigm to multi-agent orchestration** — generating and pre-executing the predicted plan while the plan is still being computed.

Background: Speculation Paradigm

Speculation in CPU Pipeline

- **Predict:** branch predictor guesses the outcome.
- **Execute ahead:** run instructions on the predicted path.
- **Verify:** commit on hit; flush (10–20 cycles) on miss.
- **Outcome:** latency hidden; TAGE **>95%** accurate.

Speculative Decoding in LLM Inference

- **Predict:** *draft* model emits γ candidate tokens.
- **Execute ahead:** *target* model scores all γ in parallel.
- **Verify:** commit matched tokens; decode from the first miss.
- **Outcome:** **2–3× speedup** at 60–85% acceptance.

Key Insight

The same paradigm (i.e., predict → execute ahead → verify) can be generalized from instructions and tokens to multi-agent orchestration, using a draft-model planner and a progressive speculation modes mechanism (i.e., commit/partial commit/flush) for verification.

SCAN ME

Speculative Dispatch Architecture

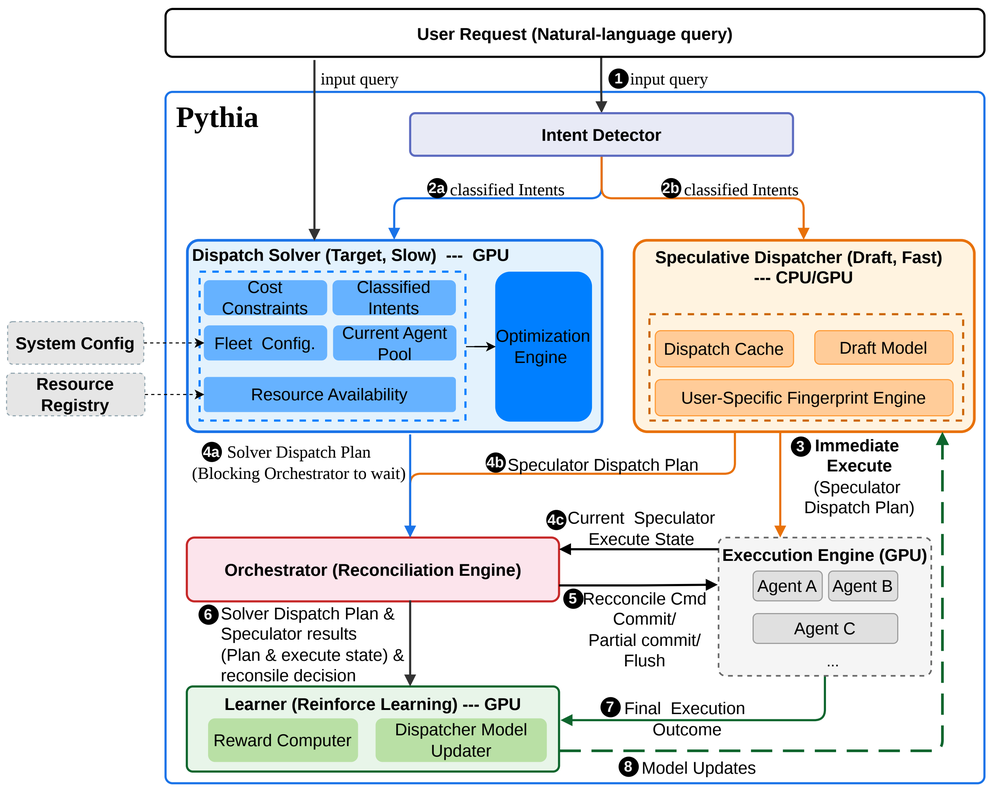

Figure 1: High-level Speculative Dispatch Architecture: Speculative Dispatcher (Draft) runs in parallel with the Dispatch Solver (Target); Orchestrator commits, partial-commits, or flushes the plan

Motivation Study: HPC Code Generation workload

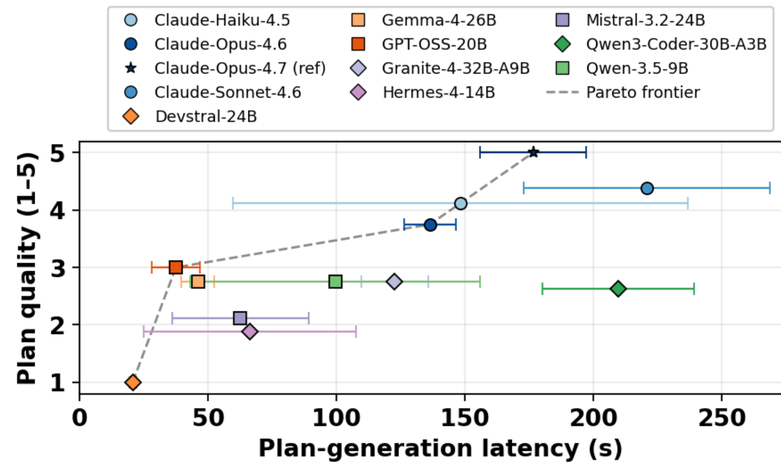

Figure 2: Plan-generation latency vs. plan quality on HPC Code Generation (mean over 5 tasks; Claude Opus 4.7 as quality reference).

- **Latency–quality trade-off:** Pareto spans ∼20 s @ Q1.0 to ∼175 s @ Q5.0, resulting in an 8.5× latency increase for peak quality.
- **Speculation Window**: 3–4× latency gap exists between fast draft models and high-quality targets

Motivation Study: Research Workflow Automation

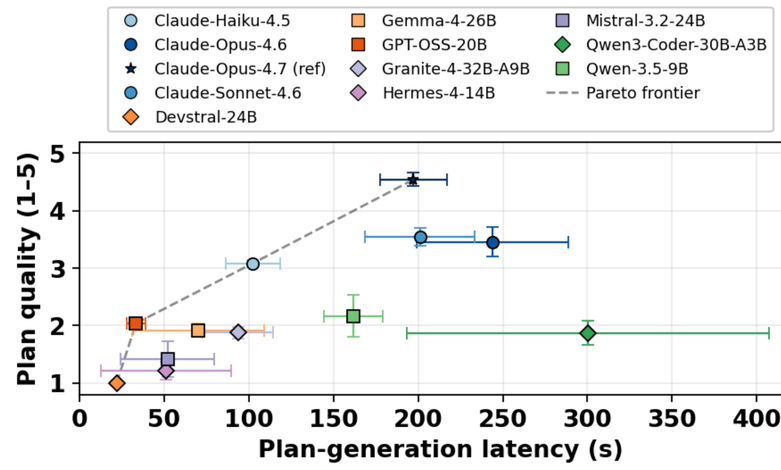

Figure 3: Plan-generation latency vs. plan quality on Research Workflow Automation (Claude Opus 4.7 as quality reference).

- **Larger speculation window:** latency reaches 200–250 s with claude models.
- **Haiku-4.5 can be a sweet-spot draft:** ∼80 s @ Q2.8 on Pareto, ∼2× faster than Opus 4.7.
- **most local models are unreliable:** either taking more time or generate the plan or the generated plan is useless (Q1.0). GPT-OSS-20 and gemma4 perform better among them (low latency, proper quality)