Position: Multi-Agent Systems Should Prioritize Concurrency Control

Xin Yang, Letian Li, Zimo Ji, Terry Jingchen Zhang, Wenyuan Jiang

Zhejiang University · Tsinghua University · HKUST · ETH Zurich

ICML 2026 · Seoul, South Korea

The Promise and the Reality

LLM-based multi-agent systems **promise scalable collaboration** — multiple agents working concurrently toward shared goals.

THE REALITY

41–87%

Failure rates across popular MAS benchmarks

37.2%

Premature submission due to missing sync barriers

29.9%

Consensus failure from concurrent conflicting states

Key Question: Why does scaling agents break things? Coordination failures and inter-agent misalignment dominate — but *what is the root cause?*

Sources: Cemri et al. (2025); Kim et al. (2025); Silo-Bench (Zhang et al., 2026)

A Hidden Pattern Behind "Coordination Failures"

Many MAS "coordination failures" are fundamentally concurrency control problems

THE MECHANISM

Agents concurrently **read and write shared mutable state**:

- Shared code repositories
- Blackboard memories
- Message buffers
- Embodied world states

THE TEMPORAL ASYMMETRY

Seconds → Minutes

LLM inference window

Milliseconds

Tool execution time

While an agent reasons for seconds or minutes, other agents may modify the shared environment **multiple times**, invalidating the assumptions underlying the first agent's decisions. This dramatically expands the conflict window.

Scope: This claim targets MAS where agents read/modify shared state during long inference — including coding, collaborative writing, embodied agents, and message-passing systems.

Running Example: Stale Read in Coding Agents

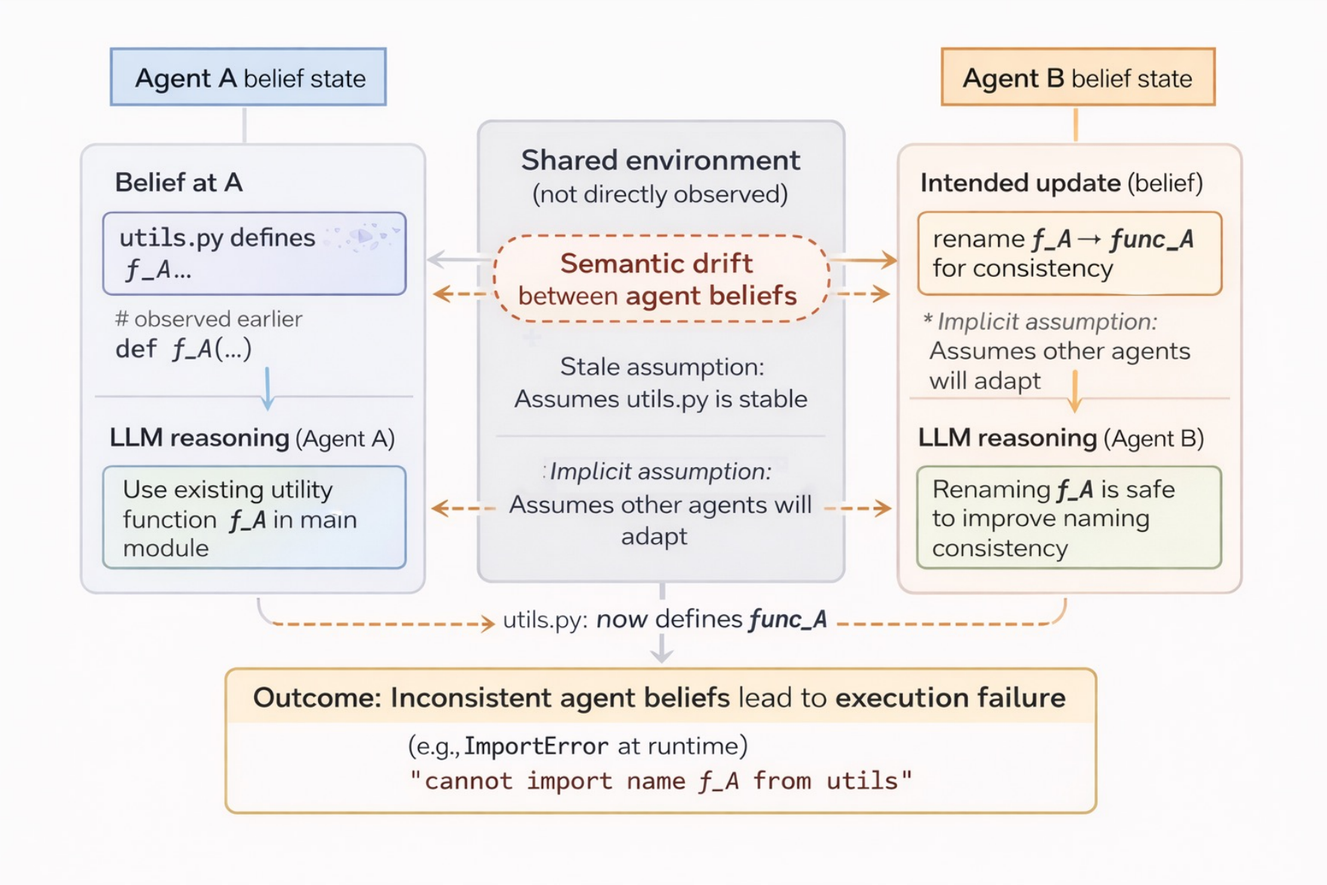

Figure 1: Semantic drift between agent beliefs leads to execution failure

THE SCENARIO

Agent A reads *utils.py* and enters long LLM inference to implement *main.py* using function *f_A*.

Agent B concurrently refactors *utils.py*, renaming *f_A* → *func_A*.

Both agents act **correctly in isolation**.

THE OUTCOME

ImportError: cannot import name 'f_A' from 'utils'

A classic **stale-read anomaly**

Superficially a "coordination" problem — but examining through the lens of concurrent systems reveals a **classic concurrency hazard**.

Four Concurrency Hazards in Disguise

1. Stale Read

Agent reads a configuration file and implements code based on observed endpoints. Concurrently, another agent updates the configuration. The first agent relies on **outdated assumptions**, resulting in inconsistent state.

Coding MAS (Qian et al., 2024)

2. Lost Update

Two agents independently modify different parts of a shared file. Both read the same initial version and write back independently. The **later write overwrites the earlier one**, silently discarding one agent's contribution.

Collaborative editing

3. Stale Correction

Agent A broadcasts a plan, Agent B sends a correction, but Agent C begins executing the **original plan before receiving the update**. The correction is not applied atomically with respect to all agents.

Message-based MAS (Li et al., 2023)

4. Action–Message Desync

In embodied environments, agents coordinate via both messages and world-altering actions. An agent may act on a message describing an intended state that has **already been invalidated** by another agent's concurrent action.

Embodied MAS (Fan et al., 2022)

A Unified Framework: Consistency + Isolation

FORMALIZING MAS

A multi-agent system: *n* agents with internal states $\{A_1, A_2, ..., A_n\}$, operating within a **shared environment E**. Each agent executes a sequence of actions that interleave concurrently across agents. Actions are either **reads** (side-effect-free) or **writes** (side-effecting).

Key Observation: LLM-based reasoning is side-effect-free w.r.t. E, but requires substantial time to complete — creating a large conflict window.

TWO FUNDAMENTAL PROPERTIES

Consistency

Concurrent operations do not leave the system in a **conflicting or contradictory state**.

Global property: The combined effects of all agents' actions should be mutually compatible.

Isolation

Each agent's execution proceeds **"as if" it were the only agent** in the system.

Agent-centric property: No agent should observe another agent's incomplete operation sequence.

Implication: Consistency and isolation are complementary — consistency ensures global coherence, isolation ensures local predictability. Violations of *either* lead to the described anomalies. The DB and distributed systems communities have developed extensive techniques for precisely these problems.

The Design Space for MAS Concurrency Control

1. SYSTEM DESIGN

Isolation Levels

Weak (Read Committed) ↔️ Strong (Serializable)

Control Strategy

Pessimistic (locking) vs. Optimistic (validation)

Versioning

Single-version vs. MVCC

Transaction Granularity

Fine-grained (single action) vs. Coarse (subtask)

Lock Granularity

Coarse (files) vs. Fine (functions)

2. INFRASTRUCTURE

Backend Systems

DB/Git/FS with mature guarantees

Version Control

Branch-per-subtask vs. Validation-at-merge

Inference Optimization

Batching, speculation, quantization

Checkpointing

KV-cache checkpointing for efficient rollback

Semantic Feedback

Rich conflict details vs. opaque "retry"

3. MODEL CAPABILITIES

Benchmarking

Contention-aware evaluation metrics

Training

SFT/RL on conflict scenarios

Prompt Engineering

Concurrency-aware prompts

Task Decomposition

Disjoint partitioning to reduce conflicts

Failure Feedback

Semantic conflict details for adaptation

TRADE-OFF DIMENSIONS

Task Success (S) | **Compatibility (C)** | **Efficiency (E)** | **Inference Cost (I)**

Addressing Alternative Views

VIEW 1

"Foundation model capabilities alone suffice"

Conversation-centric frameworks (AutoGen) rely on model capability and dialogue structure. However, as agent count grows, **conflict probability approaches certainty** regardless of model quality. Natural-language negotiation wastes inference budget; coordination in weights is neither debuggable nor auditable.

VIEW 2

"Traditional systems techniques suffice"

Existing database transactions and locks cannot be directly applied. LLM agents are **semantic-driven, non-deterministic**, and incur inference costs orders of magnitude larger than system-level operations. These mismatches make direct transplantation ineffective — they call for **system-model co-design**.

VIEW 3

"Eventual consistency and convergence suffice"

This conflates convergence with semantic correctness. While replicas may eventually agree, **application invariants can be violated** when agents reason over long inference windows based on invalidated state. CRDTs suit independent operations, but MAS often relies on cross-resource invariants.

Call to Action: Two Communities, One Problem

For ML Researchers

1. Recognize concurrency as a distinct failure mode

Timing-dependent errors despite locally correct actions → root cause is often concurrent access to shared state.

2. Design benchmarks with contention in mind

Current evaluations cannot distinguish capability failures from concurrency failures. Vary contention levels systematically.

3. Train for concurrency awareness

Pretraining corpora lack concurrent coordination patterns. Explicit training via multi-agent RL is needed.

For Systems Researchers

1. Adapt, not transplant

Classical techniques need redesign for LLM agents — semantic-driven, non-deterministic, costly inference.

2. Design for latency asymmetry

New protocols should account for seconds-long inference — via interruptible inference, KV-cache rollback, or contention-aware scheduling.

3. Build agent-friendly concurrency infrastructure

Agents need versioning, conflict detection, and rollback semantics exposed through interfaces they can reliably use.

Neither perspective alone is sufficient. Addressing this gap requires joint effort across traditionally separate communities.

Concurrency Control Should Be a First-Class Design Concern in Multi-Agent Systems

The same anomalies that plague parallel programs and distributed databases reappear when agents share mutable state — amplified by long LLM inference windows and non-determinism.

wenyjiang@ethz.ch · ICML 2026, Seoul

Thank You