ContextFlow: Physics-Aware Adaptive MCP Resource Placement for Multi-Robot LLM Coordination

---

One-Sentence Project Description

ContextFlow asks and answers a question that every multi-robot LLM system implicitly depends on but none has explicitly addressed: when the network is unstable and context is scattered across robot, edge, and cloud, how should a system decide — in real time — where each piece of a robot team's shared understanding of the world should live?

1. The Problem: LLMs Are Entering Robotics, But Nobody Asked Where Their Context Should Live

The past two years have seen a rapid convergence of large language models and physical robotics. Systems like COHERENT, CoMuRoS, and SayCan have demonstrated that LLMs can serve as high-level planners for robot teams — parsing task goals, decomposing them into sub-tasks, and coordinating heterogeneous agents. The results are impressive.

But every one of these systems shares a quiet assumption that almost nobody has questioned: **the context an LLM needs to make decisions is always available, always fresh, and always reachable.**

In a single-robot lab demo, this is fine. The robot's LLM reads local sensor data and acts. But the moment you deploy two or more robots in a real environment — a disaster site, a warehouse, a search-and-rescue scenario — the situation changes completely. Each robot's LLM suddenly needs information that lives outside its own sensors: where has the other robot already searched? What obstacles did it find? What is the current global task state? This shared context is the cognitive glue that makes multi-robot coordination possible.

The question nobody has answered is: **where should that shared context live?**

Today's systems make one of two naïve choices. They push everything to a cloud server, which collapses the moment the network degrades. Or they hardcode data flows through ROS topics,

which works for specific robot configurations but breaks the moment you swap in a different robot model or add a third agent. Neither approach is principled. Neither adapts. Neither scales.

This is the problem ContextFlow is designed to solve.

---

2. The Insight: Context Placement Is a First-Class System Problem

The Model Context Protocol (MCP), introduced by Anthropic in late 2024, provides a standardized interface for LLMs to access external tools and data. In desktop applications like Claude Desktop or Cursor, MCP is already solving the fragmentation problem for software agents — instead of custom integrations for every data source, you get a single semantic interface: `resources/read`, `resources/list`, `tools/call`.

The robotics community has begun to notice. Early work has applied MCP to offline ROS bag analysis, single-robot reasoning loops, and planning benchmarks. But all of these treat MCP as a local tool — the server is on the same machine as the client, the network is irrelevant, and there is only one robot.

Our key insight is this:

> **In a multi-robot system, every MCP resource has a placement — it is being served from somewhere. That placement is not a configuration detail. It is a system-level decision that directly determines whether the LLM gets fresh, low-latency context or stale, delayed context. And the right placement changes dynamically as network conditions shift, tasks evolve, and robots move.**

We call this the **MCP Resource Placement Problem**. To our knowledge, no prior work has formally defined it, modeled its tradeoffs, or proposed a system to solve it. ContextFlow is that system.

---

3. System Overview

ContextFlow is a three-layer architecture that sits between a MuJoCo physical simulation and the LLM agents running on each robot. Its central component is the **Edge MCP Hub**, which hosts a **Placement Policy Engine** that continuously monitors network conditions, task urgency, and physical change rates to decide where each MCP resource should be served from.

The three layers are:

Robot Layer. Each robot runs a lightweight MCP client and a local MCP server that exposes only its own sensor data — IMU readings, lidar snapshots, current pose. The LLM agent on each robot communicates exclusively through the MCP interface. It never reads raw simulation state directly. This is a deliberate design constraint: it means the quality of the LLM's decisions is entirely determined by the quality of the MCP resources it can access, making the effect of placement decisions cleanly measurable.

Edge Layer. A single Edge Hub serves as the coordination point for the robot team. It maintains a Resource Registry — a live mapping from every resource URI to its current serving location — and runs the Placement Policy Engine. It also acts as a local cache, able to serve resources that have been pulled down from the cloud when network conditions to the cloud degrade.

Cloud Layer. The cloud hosts computationally expensive or globally-scoped resources: the fused map of the entire environment, full mission history, and reasoning outputs that require more context than edge hardware can handle. Under good network conditions, robots access these resources through the edge hub's transparent routing. Under poor conditions, the Placement Policy Engine migrates these resources down to the edge.

The critical architectural principle throughout is **URI stability**. When a resource migrates from cloud to edge, its URI does not change. A robot asking for `mcp://hub/tasks/current/context` gets the same logical resource regardless of whether it is currently being served from cloud infrastructure or a cached copy on the edge hub. The registry handles the indirection transparently. This clean separation between logical identity and physical location is what makes the system composable and extensible.

---

4. The Placement Policy Engine

This is the algorithmic core of ContextFlow and the primary technical contribution of the paper.

4.1 Resource Profiling

Every MCP resource in the system is described by a profile with four dimensions. Staleness tolerance captures how quickly a resource becomes useless — a lidar snapshot becomes stale in under 200 milliseconds, while a record of which rooms have already been searched remains valid for minutes. Generation cost captures the computational expense of producing the resource — raw sensor data is cheap, a globally fused map is expensive. Access frequency captures how often the LLM actually calls the resource per second. And size captures the bandwidth cost of migrating the resource between layers.

4.2 The Placement Cost Model

For each resource and each possible placement location, the engine computes a cost:

The cost of serving a resource from the robot itself is the generation cost on robot hardware plus zero latency, since the data never leaves the device. The cost of serving from the edge is the generation or retrieval cost at the edge plus the round-trip latency between robot and edge, multiplied by access frequency. The cost of serving from cloud is the generation or retrieval cost at cloud plus the full round-trip latency through edge to cloud, plus a staleness risk term that grows with network latency variance — because a high-variance connection means the LLM will sometimes receive context that is much older than expected.

The optimal placement at any moment is the location that minimizes total cost given the current network state.

4.3 Dynamic Triggers

The engine does not continuously re-solve the full optimization. Instead, it maintains current placements and listens for trigger events that may shift the optimal solution. Three classes of triggers are defined.

Network triggers fire when measured latency crosses thresholds — if edge-to-cloud latency exceeds 150 milliseconds and is rising, resources currently served from cloud are candidates for migration to edge. If the robot-to-edge link degrades severely, high-frequency resources are candidates for caching directly on the robot.

Task urgency triggers fire when the task state changes qualitatively — for instance, when a robot detects a potential survivor signal, the urgency level escalates, which increases the weight on the staleness cost term and may promote certain resources to more local placement to reduce access latency.

Physical change rate triggers are the MuJoCo-specific contribution described in the next section.

4.4 Migration Execution

When the engine decides to migrate a resource, it executes a two-phase protocol. In the first phase, the destination node pulls the current version of the resource and begins serving it locally, while the registry still routes requests to the original location. In the second phase, once the destination confirms it has a valid copy, the registry atomically updates the routing entry. This ensures LLM agents never receive a failed resource request during migration — they may briefly get a slightly stale copy, but the system remains available throughout.

---

5. Physics-Aware Resource Updates: The MuJoCo Contribution

Standard distributed systems use time-based polling: update a resource every N milliseconds. This is wasteful when the robot is stationary and dangerous when the robot is moving fast — in both cases the update rate is wrong.

MuJoCo gives us something real hardware cannot easily provide: **precise, ground-truth physical state at every simulation timestep**. ContextFlow exploits this to drive resource updates based on the rate of physical change rather than the passage of time.

The physical change rate for each robot is computed as a weighted combination of three quantities read directly from the MuJoCo simulation: the magnitude of the robot's velocity vector, the magnitude of net contact forces (which spike when the robot collides with debris or navigates over obstacles), and the magnitude of pose delta since the last resource update. These three quantities together serve as a proxy for the information-theoretic entropy of the robot's current situation — a fast-moving robot encountering obstacles is generating new information rapidly, while a stationary robot in a cleared room is not.

The update interval for any resource is then computed as the resource's staleness tolerance divided by a factor that grows with physical change rate. When a robot is stationary, updates slow to a fraction of their nominal rate. When a robot is actively navigating through complex terrain, updates accelerate. The result is a system that uses bandwidth proportional to how much the world is actually changing, not proportional to elapsed time.

This mechanism is difficult to implement precisely on real hardware — ground truth velocity and contact forces require careful sensor calibration and filtering. In MuJoCo, they are exact. This is why using a physics simulator is not a limitation of this project. It is what makes this particular contribution possible and cleanly evaluable.

---

6. MCP Resource Taxonomy

ContextFlow defines three categories of MCP resources, each with a standardized schema.

Sensor Snapshot Resources are served from the robot layer and represent point-in-time readings from the robot's own sensors. They have tight staleness tolerances, small sizes, and are generated cheaply. The lidar snapshot resource includes an occupancy grid, the robot's current pose in the environment coordinate frame, and a timestamp. The IMU resource includes velocity, acceleration, and orientation. These resources never migrate — they are always local.

Shared State Resources are the resources that make multi-robot coordination possible and are the primary subject of the placement problem. The merged floor map resource is a fused occupancy grid built from the contributions of all robots currently operating on a given floor. The task allocation resource records which robot is currently responsible for which zone, preventing redundant coverage. The task history resource records all completed, failed, and abandoned

search actions across the entire mission. These resources are candidates for placement at any of the three layers and are the ones the Placement Policy Engine manages most actively.

Computed Reasoning Resources are generated by cloud-side inference and represent higher-level outputs: the next recommended exploration frontier given current coverage, a priority ranking of unsearched zones based on structural analysis of the environment, and cross-mission queries that draw on historical data from previous deployments. These resources are expensive to generate, moderately tolerant of staleness, and are candidates for downward migration when cloud connectivity degrades.

---

7. Experimental Design

The experimental evaluation is structured to answer three questions cleanly.

Question 1: Does adaptive placement outperform static placement strategies under dynamic network conditions?

Four conditions are compared. The Cloud-Only baseline serves all shared state and computed resources from the cloud. The Edge-Static baseline serves all shared state resources from the edge hub with no adaptation. The No-MCP baseline gives robots direct access to each other's state through simulated ROS topic subscriptions, bypassing the MCP semantic layer entirely. ContextFlow uses the full adaptive placement system.

All four conditions run in the same MuJoCo environment — a multi-room indoor scenario with rubble obstacles — with the same two robots, the same task, and the same LLM. The only difference is how context is managed.

The network conditions are varied across four scenarios injected at the Python layer: a stable scenario with low latency throughout, a degrading scenario where robot-to-edge latency increases linearly over the mission duration, an intermittent scenario with random latency spikes simulating earthquake aftershock effects on radio communication, and a partitioned scenario where edge-to-cloud connectivity drops entirely for a period before recovering.

Question 2: Does physics-aware update triggering reduce bandwidth consumption without harming decision quality?

Two sub-conditions are compared within the ContextFlow system: one using fixed-interval polling at 500 milliseconds (a reasonable baseline) and one using the physics-aware adaptive trigger. Both run under the same network conditions and the same task. Metrics include total bytes transmitted between layers, average context freshness at the moment of each LLM decision, and task performance.

Question 3: Does the MCP semantic layer add value beyond raw data sharing?

This is answered by comparing ContextFlow against the No-MCP baseline on the robot's ability to correctly reason about inherited context — specifically, whether robot B correctly avoids re-searching areas that robot A has already cleared. This requires the LLM to interpret the task history resource semantically, not just numerically, which is only possible through the MCP schema.

Metrics. The primary metrics are context freshness at decision time (lower is better), LLM decision latency from resource request to action output (lower is better), task completion time, and redundant coverage rate — the fraction of already-searched area that gets searched again. Secondary metrics include edge hub bandwidth usage and migration overhead.

---

8. Related Work and Differentiation

The literature closest to this work falls into four clusters, none of which addresses the placement problem.

Multi-robot LLM coordination systems like COHERENT and CoMuRoS demonstrate that LLMs can effectively coordinate heterogeneous robot teams, but both assume stable centralized infrastructure. Network dynamics are out of scope for both papers, and neither uses a standardized semantic interface layer.

Early MCP-robot work including the ROSBag MCP server and CTM-MCP applies MCP to single-robot settings. The ROSBag work is offline; CTM-MCP focuses on single-robot reasoning loops. Neither considers multi-robot shared context or network-adaptive behavior.

CA-MCP is the closest prior work, proposing shared context storage for multi-agent MCP systems. However, CA-MCP targets software agents in stable network environments (travel planning, logistics), assumes centralized storage, and does not define or solve the placement problem.

Edge-cloud LLM offloading work like MoA-Off addresses the question of where to run inference computation, not where to store and serve context resources. The problem structure is different: compute offloading optimizes for FLOPs and model size, while context placement optimizes for latency, freshness, and bandwidth given semantic resource profiles.

ContextFlow is the first work to sit at the intersection of all four: multi-robot coordination, MCP as semantic interface, edge-cloud layering, and physics-driven adaptive behavior.

---

9. Three-Month Execution Plan

Month 1 — Infrastructure. Build the MuJoCo simulation environment: a multi-room indoor floor plan with debris obstacles, two robot models (differential drive and a simplified quadruped), and basic locomotion controllers. Implement the Edge MCP Hub using the official Python MCP SDK, exposing the resource registry and routing logic. Implement MCP clients on each robot and confirm end-to-end resource reads. Deploy a local LLM using Ollama with Phi-3 mini or Llama 3.2 3B. Validate the full loop: MuJoCo physics state → MCP resource → LLM decision → robot action.

Month 2 — Core System. Implement the Placement Policy Engine with the cost model and three trigger classes. Implement the physics-aware update trigger reading directly from MuJoCo's data structures. Implement the two-phase migration protocol with URI stability. Define and implement the full resource schema for all three resource categories. Run smoke tests across the four network condition scenarios to confirm the system behaves correctly under degraded connectivity.

Month 3 — Evaluation and Writing. Run all experimental conditions with sufficient repetitions for statistical validity (at least 20 runs per condition). Collect all metrics. Produce figures: latency CDFs under each network condition, redundant coverage rate bar charts, bandwidth consumption comparisons. Write the paper targeting ICRA 2026 system track or SEC 2026, aiming for 8 pages.

---

10. Contributions Summary

C1 — Problem Formalization. We formally define the MCP Resource Placement Problem for multi-robot LLM systems, characterizing the tradeoff space between access latency, context freshness, and computational cost across robot, edge, and cloud layers.

C2 — System Design. We design and implement ContextFlow, a three-layer adaptive MCP architecture featuring a Placement Policy Engine with network-aware, urgency-aware, and physics-aware placement decisions, and a transparent URI-stable migration protocol.

C3 — Physics-Aware Updates. We design a resource update trigger mechanism driven by MuJoCo ground-truth physical quantities, replacing fixed-interval polling with information-rate-proportional updates, reducing bandwidth usage while maintaining context freshness.

C4 — Empirical Evaluation. We provide the first quantitative comparison of static versus adaptive MCP resource placement strategies under realistic dynamic network conditions in a multi-robot coordination task.
