Research Proposal

AgentInfraGraphWorld

Multi-Graph World Modeling for Structural Co-evolution in Long-Horizon LLM Agent Infrastructure

Degree	Doctor of Philosophy (PhD)
Area of the research proposal	Data Science and Engineering
Candidate	[Candidate Name]
Email	[Candidate Email]
Planned semester to start	[Planned Semester]

Advisor

[Advisor Name]
[Advisor Title / Lab]
School of Computing
Faculty of Science and Engineering

Macquarie University

November 28, 2026

Abstract

Long-horizon large language model (LLM) agents, and multi-agent systems built on top of them, increasingly run at infrastructure scale: they share GPUs, tool servers, API quotas, and caches, while their execution forms a system of coupled dynamic graphs: task dependencies, compute-resource allocation, tool invocation, and cache reuse. A local structural change in one graph can propagate to the others and feed back to its origin, so predicting a single graph in isolation cannot capture the long-term evolution of the execution system, and a static runtime heuristic cannot evaluate an execution decision before it is made. This proposal develops AgentInfraGraphWorld, a multi-graph world model for long-horizon agent infrastructure. Its core is a cross-graph structural transition mechanism, supported by layer-specific intra-graph evolution and closed-loop feedback. Given the current four graphs, their evolution history, and an execution action, the model directly predicts the future four graphs and their cross-graph relation changes, without generating natural-language descriptions and parsing them back into structure. Multi-step joint rollouts support prediction-based execution-structure optimization. Experiments will test not only prediction accuracy but also a scientific hypothesis: at a matched parameter and compute budget, does an explicit four-graph state with a cross-graph transition operator outperform a unified heterogeneous-graph world model? Comparisons include independent per-graph prediction, unified heterogeneous-graph world models, temporal graph neural networks, and deterministic runtime models, across multi-graph structure prediction, cross-graph propagation, closed-loop feedback, long-horizon rollout, generalization, and execution-cost reduction.

1 Introduction

As LLM agents take on increasingly complex tasks, their execution becomes a system of multiple dynamic graphs rather than a single plan. A planner generates and revises tasks, a scheduler allocates and migrates compute resources, an executor invokes external tools, and a cache manager stores, shares, and evicts intermediate results. Each process has its own runtime semantics, and each leaves a graph whose topology changes throughout a long-horizon run.

At scale, this runtime becomes the bottleneck. A long-horizon run can accumulate thousands of steps, and a multi-agent deployment multiplexes many runs over shared GPUs, tool servers, API quotas, and caches. Scheduling decisions create contention; a tool failure triggers retries; a retry invalidates a cache object; a cache miss forces recomputation and changes the task graph. Improvements in model capability do not remove these coupling effects; they make the execution system larger and more dynamic.

Four graph layers are central to this execution system, summarised in Table 1.

Table 1: The four dynamic execution graphs of a long-horizon LLM agent.

Layer	Structural semantics
Task dependency graph	Generation, dependency, branching, and merging of tasks
Compute resource graph	Allocation, sharing, contention, and migration of resources
Tool invocation graph	Tool invocation instances, execution order, and data dependencies
Cache reuse graph	Cache generation, sharing, eviction, and reuse relations

These graphs are coupled. Parallelising a task can change resource contention and the structure of tool calls, which in turn changes cache sharing. A cache eviction or a resource shortage can trigger replanning and alter the subsequent topology of the task graph. Such changes can propagate through several graphs and return to their origin, sometimes amplified and sometimes damped.

Because these effects accumulate over multiple rounds, predicting one graph in isolation cannot

describe the long-term behaviour of the whole execution system. A useful model must represent the joint state of the four graphs, the transition from one joint state to the next, the propagation of a local structural change across graphs, and the feedback that returns to the source graph.

A static runtime heuristic or a single-graph forecast is insufficient for planning. Dynamic graph prediction typically learns $P(G_{t+1} \mid G_{\leq t})$ from observed structure; it is not action-conditioned and cannot answer what happens if the agent splits a task, migrates a job, or evicts a cache object. A graph world model learns $P(\mathbf{G}_{t+1} \mid \mathbf{G}_t, a_t)$ and can roll forward, so it can evaluate candidate execution decisions before they are taken. The difference is between forecasting a trace and simulating the effect of an action.

Graph world models provide a starting point. Graph World Model [6] unifies graph structure and actions in a world-modeling framework; WorldGraph [7] treats graph evolution as world dynamics; and recent work studies dynamic graph world modelling [9], evolving topologies [10], and structural dynamics with constrained rollouts [11]. These methods model a single graph or a single structural system. Agent world models [12, 13] instead predict observations, task progress, or language-level environment behaviour. Neither line outputs the future structure of an agent's four execution graphs, and neither models cross-graph structural state transitions with closed-loop feedback.

The core scientific question of this proposal is therefore:

> **How can a multi-graph world model learn intra-graph evolution, cross-graph structural propagation, and closed-loop feedback among the multiple dynamic execution graphs of a long-horizon LLM agent, and predict the future four-graph structure under execution actions?**

To answer this question, this proposal develops AgentInfraGraphWorld. The framework keeps the four graphs separate, maintains an explicit cross-graph relation state, and learns a world model whose core is a cross-graph structural transition operator, supported by layer-specific intra-graph evolution and closed-loop feedback. Given an action, it directly outputs the future four graphs and their relation changes as structured objects. Multi-step rollouts support the selection of execution-structure transformations that reduce overall cost while preserving task correctness and quality. The unique contributions are:

- **A cross-graph structural state transition mechanism (core contribution)** that learns how a change in one graph propagates to the others and returns to its source, and that directly outputs future graph structure and relation changes without text generation or parsing.
- **A multi-graph world model architecture** that composes intra-graph evolution, cross-graph structural propagation, and closed-loop feedback over four independent execution graphs.
- **Prediction-based execution optimization** that uses multi-step joint rollouts to select valid structural transformations under task-correctness constraints.

2 State of the Art

2.1 Agent Execution Infrastructure

LLMCompiler [1] compiles a task plan into a directed acyclic graph of function calls and executes independent calls in parallel. Parrot [2] exposes dependencies between LLM requests through semantic variables and optimises application-level serving. SAGA [3] schedules agent workflows atomically on GPU clusters, ThunderAgent [4] provides program-aware agentic inference, and GraphFlow [5] manages agent workflows as graphs for serving efficiency.

These systems show that explicit execution structure lowers latency and cost. Each addresses a subset of the four runtime processes—typically task dependencies and tool calls, or scheduling and resource allocation—and treats the relevant structure as given by the runtime rather than as an object to be learned and predicted. They do not provide a world model of the joint evolution of the four graphs.

2.2 Graph World Models

Graph World Model [6] incorporates graph structure and actions into a unified world-modeling framework. WorldGraph [7] treats the evolution of a dynamic graph as world dynamics at the node, edge, and graph levels. A recent survey [8] formalises graph world models and organises them by relational inductive biases. DyG-WM [9] predicts future node and relation representations from temporal context; GDM [10] handles evolving topologies in stochastic, partially observable environments; and SD-GWM [11] models structural dynamics with constrained rollout and calibration.

These works establish that graph structure is a useful substrate for world modeling and that topology can change over time. They model one graph or one structural system. They do not represent several execution graphs with distinct transition rules, nor do they learn how a structural edit in one graph changes the transition distribution of another and returns as feedback.

2.3 Agent World Models

Agent world models target the agent's interaction with an environment. AEWM [12] models how reasoning and actions shape future task progress rather than reconstructing tool responses. Qwen-AgentWorld [13] builds language world models that simulate agentic environments across domains. These models operate on observations, task progress, or language-level state.

They do not expose the runtime infrastructure of the agent as a structured multi-graph state, and their outputs are not the future four execution graphs and cross-graph relations. Predicting execution structure from text would also introduce an unnecessary generation-and-parsing step; a world model can output structure directly.

2.4 Multi-Graph and Cross-Graph Learning

Multilayer network analysis [14] studies systems composed of multiple relation layers, and temporal graph networks [15] learn continuous-time dynamics from timestamped events. These lines of work show that multiple layers and dynamic topology can be represented and learned.

What remains open is an action-conditioned joint structural predictor for multiple dynamic graphs: one that keeps layer-specific transition rules, models cross-graph propagation, closes the loop back to the source graph, and outputs the future structure of all graphs directly. Existing methods do not provide this combination.

2.5 Research Gap and Innovation Space

A natural alternative is to flatten the four graphs into one heterogeneous graph and learn a single world model over it. This is a strong baseline, not a straw man: a heterogeneous graph can represent all node and edge types, and a shared transition may exploit cross-type regularities that separate graphs miss. Whether factorisation helps is therefore an empirical question. We state it as hypothesis H1: at a matched parameter and compute budget, an explicit four-graph state with a cross-graph transition operator yields lower long-horizon prediction error and better execution-cost reduction than a unified heterogeneous-graph world model. The expected difference is that a unified model collapses layer-specific transition rules into one parameterised transition, so a task

split and a cache eviction become the same kind of structural event, and cross-graph relations become ordinary edges among many others, which makes propagation and feedback difficult to attribute and control. The experiments are designed to make a rejection of H1 visible rather than to assume the answer.

The proposed model keeps the four graphs separate, maintains an explicit cross-graph relation state, and learns the three mechanisms as distinct components. This preserves layer identity, makes propagation and feedback traceable, and allows the decoder to emit layer-specific structural edits directly. The innovation space is therefore the architecture and learning of multi-graph structural state transitions, not a new neural-network backbone.

3 Research Objectives and Approach

3.1 Multi-Graph Execution System Representation

At time t, the agent execution system is represented as

$$\mathbf{G}_t = \left(G_t^T, G_t^R, G_t^O, G_t^C, \mathcal{R}_t\right), \tag{1}$$

where G_t^T, G_t^R, G_t^O, and G_t^C are the task, resource, tool, and cache graphs, and $\mathcal{R}_t$ denotes the dynamic relations among them.

Each graph retains its own topology and runtime attributes:

$$G_t^L = (V_t^L, E_t^L, X_t^L), \qquad L \in \{T, R, O, C\}, \tag{2}$$

where V_t^L is its node set, E_t^L its edge set, and X_t^L its node, edge, and graph-level attributes. The cross-graph relation set $\mathcal{R}_t$ records typed dependencies between elements of different graphs, such as a task requiring a resource, a tool call consuming a cached object, or a resource constraint changing the executable order of tool calls.

The representation does not fix whether an individual layer is a homogeneous graph, a heterogeneous graph, or a hypergraph. It fixes two requirements: each layer preserves its own topology, and cross-graph dependencies are represented explicitly rather than collapsed into a single graph.

The system evolves through structural edits:

$$\mathbf{G}_{t+1} = \mathbf{G}_t \oplus \Delta_t, \qquad \Delta_t = \left(\Delta G_t^T, \Delta G_t^R, \Delta G_t^O, \Delta G_t^C, \Delta \mathcal{R}_t\right), \tag{3}$$

where each ΔG_t^L contains node and edge births, deaths, and reorganisations, and $\Delta \mathcal{R}_t$ records changes in cross-graph relations. Figure 1 illustrates the representation: a cross-graph relation is an edge between specific entities in two layers, not a central hub, and a feedback relation can return from the layer where the effect appears to the layer where the action originated.

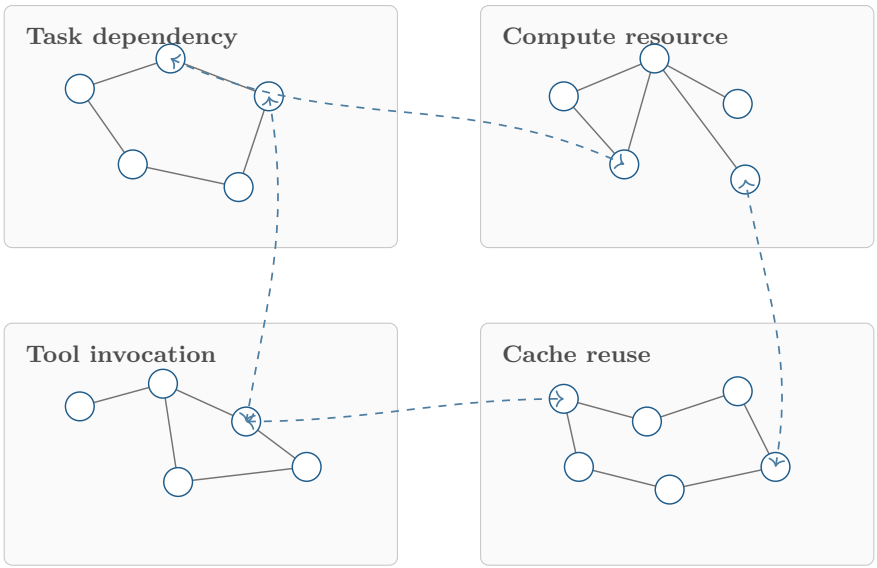

Figure 1: Four dynamic execution graphs coupled by direct cross-graph relations. Each layer keeps its own topology; dashed bidirectional links connect specific entities across layers, for example a task to the resource that runs it or a tool call to the cache object it writes. Feedback returns from the graph where the effect appears to the graph where the action originated.

3.2 Cross-Graph Structural Transition

The scientific core is a cross-graph structural transition operator: it takes a joint state and an action, applies layer-specific deltas, propagates them across graphs, and closes the loop. Layer-specific intra-graph evolution and closed-loop feedback are supporting components of this operator, not independent predictors. The operator can be instantiated with different neural backbones; the research contribution is the transition semantics and the architecture, not a particular encoder or decoder.

3.2.1 Intra-Graph Evolution

Each graph evolves under a layer-specific transition rule. For layer L, the structural delta is generated from

$$\Delta_t^L \sim p_\theta \left(\Delta_t^L \mid G_t^L, \mathcal{S}_t^L, a_t \right), \qquad L \in \{T, R, O, C\}, \tag{4}$$

where $\mathcal{S}_t^L$ is the layer's evolution history and a_t is the action. This mechanism captures task decomposition, resource allocation, tool-call ordering, and cache reuse without forcing them into a shared transition.

3.2.2 Cross-Graph Structural Propagation

A structural change in one graph can alter the transition distribution of another through the relation state. The propagation operator maps the set of intra-graph deltas and the relations to deltas on the other graphs:

$$\Delta_t^{\text{prop}} = \Pi_\theta \left(\{\Delta_t^L\}_{L \in \{T,R,O,C\}}, \mathcal{R}_t, a_t \right). \tag{5}$$

It learns which relations carry a change, how far the change travels, and where it stops. Propagation is not a fixed message-passing rule; it is learned from observed multi-graph transitions.

3.2.3 Closed-Loop Feedback

The propagated changes can return to the source graph. The feedback operator maps the propagated deltas and the relations back to the originating layer:

$$\Delta_t^{\text{fb}} = \Phi_\theta \left(\{\Delta_t^L\}_{L \in \{T,R,O,C\}}, \Delta_t^{\text{prop}}, \mathcal{R}_t \right). \tag{6}$$

Feedback can be immediate or delayed. The model learns the return paths and the conditions under which a local change is amplified, damped, or reversed. This mechanism distinguishes a multi-graph world model from a feed-forward multi-graph predictor.

3.2.4 Composition and Architectural Advantage

The joint transition combines the three mechanisms:

$$\mathbf{G}_{t+1} = \mathbf{G}_t \oplus \left(\{\Delta_t^L\}_{L \in \{T,R,O,C\}} \cup \Delta_t^{\text{prop}} \cup \Delta_t^{\text{fb}} \right). \tag{7}$$

Intra-graph evolution preserves layer identity; propagation models cross-graph influence; feedback closes the loop. The core object is their composition—a transition operator on the joint multi-graph state—rather than any mechanism in isolation. A unified heterogeneous-graph world model would collapse the first into a shared transition and the second and third into ordinary edges. Table 2 summarises the difference.

Table 2: Multi-graph world model versus a unified heterogeneous-graph world model.

Aspect	**Unified heterogeneous graph**	**AgentInfraGraphWorld**
Layer identity	Collapsed into node and edge types	Four separate graphs with layer-specific state
Transition	One shared transition over all edges	Intra-graph evolution plus cross-graph propagation
Feedback	Implicit in message passing	Explicit closed-loop feedback operator
Output	Edges in one mixed graph	Direct four-graph structure and relation deltas
Attribution	Entangled across types	Layer-wise propagation and feedback traces

3.3 Action-Conditioned Multi-Step Joint Rollout

Given an action sequence, the world model rolls out the future joint structure:

$$\hat{\mathbf{G}}_{t+1:t+K} = F_\theta^{(K)} \left(\mathbf{G}_t, \mathcal{S}_t, a_{t:t+K-1} \right). \tag{8}$$

The output is a structured state, not text. For each graph, the decoder emits node and edge births, deaths, and attribute changes; for the relation state, it emits relation births, deaths, and updates. No natural-language description is generated and parsed back into structure.

Training combines three objectives: next-state likelihood on observed multi-graph transitions, multi-step rollout consistency between predicted and observed states at horizons $2, \ldots, K$, and auxiliary losses for cross-graph propagation and feedback. The multi-step term is essential because a one-step error in the task graph can change resource contention, tool ordering, and cache reuse in later steps.

3.4 Prediction-Based Execution Optimization

The world model is used for planning. Given a set of valid action sequences, the agent selects

$$a^*_{t:t+K-1} = \arg \min_{a_{t:t+K-1} \in \mathcal{A}_{\text{valid}}} \mathbb{E} \left[C \left(\hat{\mathbf{G}}_{t+1:t+K} \right) \right], \tag{9}$$

where $\mathcal{A}_{\text{valid}}$ is the set of legitimate structural transformations and C is the overall execution cost.

Validity is defined by task correctness and quality requirements, together with resource and tool-availability limits. The cost aggregates latency, token and compute consumption, failure and retry penalties, and cache misses. Candidate action sequences are generated from high-probability structural changes proposed by the model and evaluated by rollout. The procedure is receding-horizon: after executing an action, the actual graphs are updated and the model re-predicts the future structure.

3.5 Experimental Design and Evaluation

Data collection. We will instrument an agent runtime to emit timestamped events from its planner, scheduler, tool executor, and cache manager, and reconstruct the multi-graph execution trajectory. Long-horizon tasks will be drawn from data analysis, document processing, and mathematical reasoning.

Baselines and ablations. AgentInfraGraphWorld will be compared with independent per-graph prediction, a unified heterogeneous-graph world model, temporal graph neural networks, and a deterministic runtime model. Ablations will remove cross-graph propagation, closed-loop feedback, and action conditioning to isolate their contributions.

Evaluation dimensions. Table 3 summarises the evaluation.

Table 3: Evaluation dimensions and their research objectives.

Evaluation dimension	Research objective	Representative metric
Action-conditioned prediction	Predict the effect of a candidate action, not only the next observed state	Counterfactual rollout error, action-ranking accuracy
H1: multi-graph factorisation	Test explicit four-graph state against a unified heterogeneous graph at matched capacity	Prediction error and cost at matched parameters/compute
Multi-graph structure prediction	Predict the future topology of each graph	Birth/death F1, relation accuracy
Cross-graph propagation	Learn how a local change reaches other graphs	Propagation path F1, target-change accuracy
Closed-loop feedback	Capture how changes return to the source	Feedback path accuracy, return-effect error
Long-horizon rollout	Reduce error accumulation over K steps	Error growth over K steps
Generalization	Predict unseen cross-graph structure combinations	Held-out structure performance
Execution optimization	Reduce overall execution cost	Cost reduction at matched success

4 Expected Contributions and Conclusions

This proposal is expected to make three main scientific contributions.

1. **Cross-graph structural state transition learning (core).** A mechanism that learns how a structural change in one graph propagates to the others and returns to its source, with direct structured output of future graphs and relation changes rather than text generation and parsing.

2. **Multi-graph world model architecture.** A model that composes intra-graph evolution, cross-graph structural propagation, and closed-loop feedback over four independent execution graphs, preserving layer identity and explicit cross-graph relations.

3. **Prediction-based execution optimization.** A planning procedure that uses multi-step joint rollouts to select valid structural transformations under task-correctness constraints.

This research treats the runtime infrastructure of a long-horizon agent as a dynamic world composed of multiple interacting graphs. By learning intra-graph evolution, cross-graph propagation, and closed-loop feedback as components of one transition operator, it aims to provide a new architecture for multi-graph world modeling and a foundation for optimizing agentic execution infrastructure. The central empirical question is whether explicit multi-graph factorisation is worth its structural prior; the H1 comparison is designed to answer it rather than assume it.

References

[1] S. Kim et al., "An LLM Compiler for Parallel Function Calling," in *Proc. International Conference on Machine Learning (ICML)*, 2024.

[2] C. Lin et al., "Parrot: Efficient Serving of LLM-based Applications with Semantic Variable," in *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

[3] D. Guo et al., "SAGA: Workflow-Atomic Scheduling for AI Agent Inference on GPU Clusters," 2026.

[4] H. Kang et al., "ThunderAgent: A Simple, Fast and Program-Aware Agentic Inference System," 2026.

[5] A. Li et al., "GraphFlow: A Graph-Based Workflow Management for Efficient LLM-Agent Serving," in *Proc. International Conference on Machine Learning (ICML)*, 2026.

[6] T. Feng, Y. Wu, G. Lin, and J. You, "Graph World Model," in *Proc. International Conference on Machine Learning (ICML)*, 2025.

[7] Z. Ding, Y. Li, and X. Xie, "WorldGraph: Graph-Native World Modeling," arXiv:2609.34159, 2026.

[8] J. Liu, S. Yang, M. Wang, Y. Wang, and B. Yu, "Graph World Models: Concepts, Taxonomy, and Future Directions," arXiv:2604.27895, 2026.

[9] B. Li et al., "Dynamic Graph World Modelling: Learning Future Dynamics from Temporal Context," arXiv:2610.19364, 2026.

[10] A. Schutz, N. Hawes, and V.-A. Darvariu, "Beyond Static Graph World Models: Learning Stochastic Latent Dynamics over Evolving Topologies," arXiv:2609.28670, 2026.

[11] W. Wang et al., "A Structural Dynamics Graph World Model: Unified Modeling, Constrained Rollout, and Interpretable Calibration," arXiv:2608.08689, 2026.

[12] S. Sun et al., "Agent-Editing World Model: Rethinking World Modeling for LLM Agents," arXiv:2609.28416, 2026.

[13] Y. Zuo et al., "Qwen-AgentWorld: Language World Models for General Agents," arXiv:2606.24597, 2026.

[14] M. Kivelä et al., "Multilayer Networks," *Journal of Complex Networks*, vol. 2, no. 3, pp. 203–271, 2014.

[15] E. Rossi et al., "Temporal Graph Networks for Deep Learning on Dynamic Graphs," arXiv:2006.10637, 2020.