Research Proposal

AgentHyperEvo

Learning Higher-Order Structural Co-evolution of Dynamic Heterogeneous Hypergraphs in Long-Horizon LLM Agents

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

Prof. Amin Beheshti
Head of Data Science Research Lab
School of Computing
Faculty of Science and Engineering

Macquarie University

November 27, 2026

Abstract

Large language model (LLM) agents increasingly solve long-horizon tasks that require dynamic task planning, computational resource allocation, external tool invocation, and cache reuse. Each of these processes induces a distinct dynamic graph with its own runtime semantics: a task dependency graph, a compute-resource graph, a tool-invocation graph, and a cache-reuse graph. These graphs do not evolve independently. A local topological change in one layer can propagate across layers, trigger feedback, and accumulate over many execution rounds; a resource shortage, an anomalous tool result, or a cache eviction can equally force the agent to replan and thereby reshape the task graph. Modelling any single graph in isolation therefore cannot explain the long-term structural behaviour of the whole execution system.

This proposal introduces AgentHyperEvo, a framework that represents the execution state of a long-horizon LLM agent as a dynamic heterogeneous hypergraph with four runtime-semantic layers and explicit cross-layer hyperedges. The framework will learn a higher-order structural co-evolution model that predicts the joint next-step structure from the current hypergraph and its evolution history; perform counterfactual structure prediction that estimates the future multi-layer trajectory induced by candidate local structural interventions; and exploit these predictions to select valid structural transformations that reduce overall execution cost while preserving task correctness and quality. Experiments on data analysis, document processing, and mathematical reasoning tasks will compare AgentHyperEvo with dynamic heterogeneous graphs, temporal hypergraphs, association bipartite graphs, and deterministic runtime models, evaluating structural prediction, higher-order relation prediction, feedback prediction, counterfactual estimation, generalisation, and execution optimisation.

1 Introduction

LLM agents can autonomously plan, reason, and invoke external tools to complete complex tasks. As the task horizon grows from a single query to data-analysis pipelines, document-processing workflows, and multi-step mathematical reasoning, agent execution becomes a runtime system in its own right: a planner decomposes goals, a scheduler allocates compute and API capacity, an executor invokes tools, and a cache manager stores and reuses intermediate results. Each of these processes leaves a trace, and each trace has the structure of a graph that changes continuously during execution.

Four graph layers are central to this runtime, summarised in Table 1. They are not independent views of the same data. They are coupled processes with different semantics, different time scales, and different notions of a structural event.

Table 1: The four runtime-semantic graph layers 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 computational resources
Tool invocation graph	Tool invocation instances, execution order, and data dependencies
Cache reuse graph	Cache generation, sharing, eviction, and reuse relations

The coupling is direct and consequential. Consider a task that is split into parallel subtasks: the split changes the task graph, but it also changes the number of concurrent tool calls, the pattern of resource demand, and the set of intermediate results eligible for caching. If two subtasks contend for the same GPU or the same rate-limited API, the scheduler may serialise them, which delays dependent tool calls and reduces cache-hit opportunities. Conversely, a tool timeout, an anomalous tool result, or a cache eviction can invalidate a planning assumption and force the agent to regenerate part of the task graph. A resource shortage may trigger task migration,

which changes data locality and therefore tool execution order and cache reuse. These effects are not one-off: they propagate through the layers, feed back to their origin, and accumulate across many rounds of a long-horizon run.

Existing agent-infrastructure research optimises individual components of this runtime. Task-dependency and tool-call structure are exploited for parallel function calling [1]; application-level dependencies are exposed for serving [2]; workflows are scheduled atomically on GPU clusters [3]; agentic inference is made program-aware [4]; and workflows are managed as graphs for serving efficiency [5]. These systems show that explicit execution structure yields substantial latency and cost benefits. What remains missing is a model of the execution system as a whole: in the surveyed work, we have not found a method that represents task, resource, tool, and cache as separate dynamic layers of a single heterogeneous hypergraph, learns the higher-order structural co-evolution among those layers, or predicts how a local structural intervention will reshape the other layers over time.

The core scientific question of this proposal is therefore:

> **How can the higher-order structural co-evolution among the multiple dynamic execution graphs of a long-horizon LLM agent be learned, and how can the effect of a local structural transformation on the future overall execution structure be predicted?**

To answer this question, this proposal develops AgentHyperEvo. The framework represents the execution state as a four-layer dynamic heterogeneous hypergraph, learns a joint co-evolution model over its layers, conditions that model on candidate structural interventions to obtain counterfactual future trajectories, and uses the predicted trajectories to select execution-structure transformations that lower overall cost while preserving task correctness and quality. The unique contributions of this proposal are:

- **A dynamic heterogeneous hypergraph representation** that unifies the task, resource, tool, and cache layers of a long-horizon agent and treats intra-layer and cross-layer higher-order relations as first-class objects.
- **A higher-order co-evolution learning mechanism** that captures joint triggering, cross-layer propagation, and recursive feedback among heterogeneous layers.
- **A counterfactual structure prediction method** that relates local structural interventions to future multi-layer graph trajectories, providing the basis for adaptive execution-structure optimisation.

2 State of the Art

2.1 Agent Execution Graphs and Inference Infrastructure

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

Collectively, these systems demonstrate that making execution structure explicit produces significant latency and cost improvements. However, 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. To the best of our knowledge, no existing method explicitly models task, resource, tool, and cache as four dynamic graph layers and systematically learns the higher-order structural co-evolution among them.

2.2 Hypergraph Representation Learning

A graph describes pairwise relations, whereas a hypergraph allows a single hyperedge to connect an arbitrary number of nodes. This makes multi-entity joint participation a first-class learning object rather than a collection of pairwise links. Hypergraph Neural Networks (HGNN) [6] generalise graph convolution to hypergraphs through hyperedge-level aggregation, and AllSet [7] provides a multiset-function framework that subsumes a broad family of hypergraph neural networks.

These works supply the representational machinery for higher-order structure learning, but they are primarily designed for static, homogeneous hypergraphs. They do not provide layer-specific runtime semantics, heterogeneous cross-layer hyperedges whose members come from different layers, or mechanisms through which hyperedges and their membership are created, dissolved, and reorganised as an execution unfolds. This proposal builds on these formulations while extending them to a dynamic, multi-layer, heterogeneous setting.

2.3 Dynamic and Temporal Graph Learning

Temporal Graph Networks (TGN) [8] maintain per-node memories and update them from a stream of timestamped events, enabling continuous-time dynamic graph learning. Recent work [9] shows that higher-order structure improves link prediction on temporal graphs. These methods model the evolution of a single graph, or of a fixed higher-order pattern within it.

What they do not address is the joint evolution of several heterogeneous layers, in which an event in one layer changes the event distribution in another, nor do they support counterfactual queries about structural interventions. The co-evolution and counterfactual components of AgentHyperEvo therefore lie outside the scope of existing temporal graph models.

2.4 Synthesis and Research Gap

Three observations follow from the literature. First, agent-infrastructure research has established that task, resource, tool, and cache structure matter for performance, but treats these structures separately. Second, hypergraph learning provides the representational machinery for multi-entity relations, but not for dynamic heterogeneous layers. Third, temporal graph learning provides event-driven evolution, but not for coupled multi-layer systems or interventions.

Bridging these lines requires four capabilities:

(**R**1) a unified four-layer dynamic heterogeneous hypergraph with typed nodes, intra-layer and cross-layer hyperedges, and dynamic membership;

(**R**2) a learnable co-evolution model capturing cross-layer propagation, joint triggering, and recursive feedback;

(**R**3) counterfactual prediction of future multi-layer trajectories under local structural interventions;

(**R**4) an optimisation procedure that uses these predictions under task-correctness constraints.

AgentHyperEvo is designed around these four requirements.

3 Research Objectives and Approach

3.1 Dynamic Multilayer Heterogeneous Hypergraph Representation

The first objective is a unified representation of the execution state that preserves the distinct semantics of the four layers while making their interactions explicit. At time t, the overall execution hypergraph is

$$\mathcal{H}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathcal{X}_t), \tag{1}$$

where $\mathcal{V}_t$ is the set of heterogeneous nodes, $\mathcal{E}_t$ is the set of intra-layer and cross-layer hyperedges, and $\mathcal{X}_t$ collects the associated runtime attributes.

Nodes are typed by layer. Let $\mathcal{L} = \{\mathsf{T}, \mathsf{R}, \mathsf{C}, \mathsf{K}\}$ denote the task, resource, tool, and cache layers, and let $\ell(v) \in \mathcal{L}$ be the layer of node v. The node set is partitioned accordingly:

$$\mathcal{V}_t = \bigcup_{L \in \mathcal{L}} \mathcal{V}_t^L, \qquad \mathcal{V}_t^L \cap \mathcal{V}_t^{L'} = \emptyset \quad (L \neq L'). \tag{2}$$

A hyperedge is a tuple

$$e = (\tau_e, M_e, t_e, \mathbf{x}_e), \qquad M_e \subseteq \mathcal{V}_t, \qquad |M_e| \geq 2, \tag{3}$$

where τ_e is its type, M_e its member set, t_e its timestamp, and $\mathbf{x}_e$ its runtime attributes. Intra-layer hyperedges have all members in one layer; cross-layer hyperedges have members in two or more layers. The type τ_e records this layer signature together with the relation semantics, so that, for example, a tool-call hyperedge joining a task, a tool instance, and a resource is distinct from a cache-reuse hyperedge joining a tool instance and several cached objects.

Definition 1 (Dynamic multilayer heterogeneous execution hypergraph)**.** *A dynamic multilayer heterogeneous execution hypergraph is a time-indexed sequence $\{\mathcal{H}_t\}_{t \geq 0}$ with typed nodes and hyperedges as above, in which nodes, hyperedges, and hyperedge membership may be created, removed, or reorganised at each step.*

The dynamics are expressed as structural edits. At each step,

$$\mathcal{H}_{t+1} = \mathcal{H}_t \oplus \Delta_t, \qquad \Delta_t = (\Delta\mathcal{V}_t^+, \Delta\mathcal{V}_t^-, \Delta\mathcal{E}_t^+, \Delta\mathcal{E}_t^-, \Delta\mathcal{M}_t), \tag{4}$$

where $\Delta\mathcal{V}_t^{\pm}$ are node births and deaths, $\Delta\mathcal{E}_t^{\pm}$ are hyperedge births and deaths, and $\Delta\mathcal{M}_t$ records membership replacements. Requiring the model to score structural edits, rather than to predict a fixed-size adjacency matrix, is essential: the number of nodes and hyperedges changes throughout a long-horizon run, and the same node may change its structural role without being removed.

Figure 1 illustrates the representation. The shaded regions are intra-layer hyperedges; the dashed links denote cross-layer higher-order dependencies, including relations that skip intermediate layers; and the long arrow denotes recursive feedback from lower layers back to the task layer.

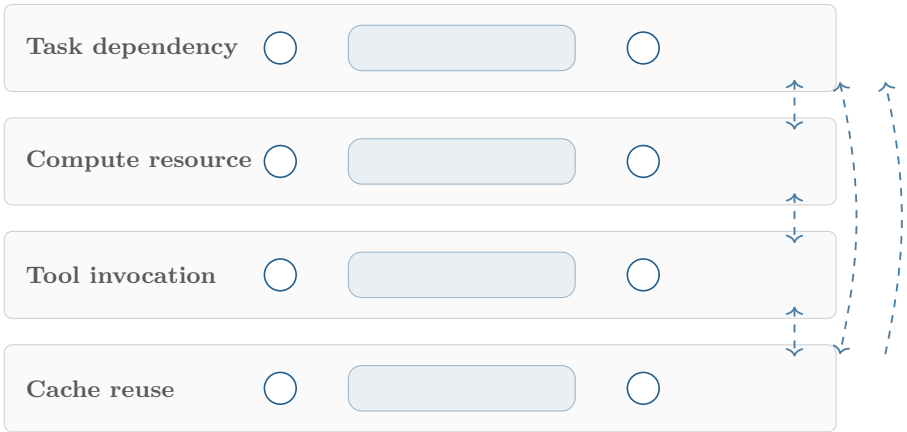


Figure 1: Illustrative four-layer execution hypergraph. Shaded boxes are intra-layer hyperedges, short dashed arrows are cross-layer dependencies between adjacent layers, and the two outer curves denote a cross-layer hyperedge spanning non-adjacent layers and recursive feedback.

3.2 Higher-Order Structural Co-evolution Learning

The second objective is to learn how the layers influence one another. Given the current hypergraph and a summary of its evolution history, the model predicts the joint next-step structure:

$$\hat{\mathcal{H}}_{t+1} = F_\theta(\mathcal{H}_t, \mathcal{S}_t), \tag{5}$$

where $\mathcal{S}_t$ denotes historical structural evolution information and F_θ is the co-evolution model to be learned. The prediction targets are:

- the birth, death, and reorganisation of future nodes and hyperedges;
- changes in hyperedge membership and in the structural roles of nodes;
- the formation and dissolution of cross-layer higher-order relations;
- the propagation paths and feedback relations of multi-layer structural change.

We will formulate the predictor over structural events. A structural delta is generated from a distribution conditioned on all layers,

$$\Delta_t \sim p_\theta(\Delta_t \mid \mathcal{H}_t, \mathcal{S}_t), \qquad \hat{\mathcal{H}}_{t+1} = \mathcal{H}_t \oplus \hat{\Delta}_t, \tag{6}$$

and multi-step predictions are obtained by autoregressive application,

$$\hat{\mathcal{H}}_{t+1:t+K} = F_\theta^{(K)}(\mathcal{H}_t, \mathcal{S}_t). \tag{7}$$

The model will combine four components. A *typed hyperedge encoder* maps nodes, hyperedges, and their memberships to layer-specific embeddings, with cross-layer hyperedges encoded as higher-order relations over heterogeneous members. A *temporal memory* maintains per-node, per-hyperedge, and per-layer states that summarise $\mathcal{S}_t$ and are updated by observed structural events. A *co-evolution predictor* conditions the next-event distribution on all layers and on the cross-layer context, so that a task event can change the event scores in the resource, tool, and cache layers, and vice versa. A *structured decoder* emits a valid structural edit—birth, death, or membership change—rather than independent pairwise links, which keeps the predicted hypergraph well formed.

Training will combine next-event likelihood on observed trajectories with a multi-step rollout loss that penalises compounding structural error, together with auxiliary objectives for membership prediction and cross-layer relation prediction. The rollout loss is important because the downstream use of the model is a K-step forecast, not a one-step reconstruction.

3.3 Counterfactual Structure Prediction

The third objective is to predict what happens under a structural intervention. For a candidate intervention a applied to any layer—for example, replacement of a local task subgraph, a change in a resource allocation relation, an adjustment of a tool invocation, or a change in a cache reuse relation—the model predicts the future multi-layer trajectory

$$\hat{\mathcal{H}}^{(a)}_{t+1:t+K} = F_\theta(\mathcal{H}_t, \mathcal{S}_t, a). \tag{8}$$

Operationally, the intervention is applied to a copy of the current state, $\mathcal{H}_t^{(a)} = \mathrm{do}(\mathcal{H}_t, a)$, and the predictor is rolled forward K steps. Conditioning on a explicitly, rather than only on the post-intervention state, is what makes the query counterfactual: the same observed state can be paired with different candidate interventions.

Training data will be obtained from paired rollouts under controlled initial states. The immediate, deterministic consequences of an intervention are known by construction—splitting a task creates

its child tasks, for instance—while the subsequent cross-layer response must be learned. The training objective will therefore separate these two components: the rule-determined part of the delta is applied exactly, and the model is trained on the residual evolution. A paired counterfactual loss will reward the model for correctly ordering interventions by their downstream structural effect, in addition to fitting each observed trajectory.

At inference time, comparing the predicted trajectory distributions of different interventions yields an estimate of their long-term structural impact. This is the signal used by the optimisation procedure in Section 3.4.

3.4 Overall Execution Structure Optimisation

The fourth objective is to use counterfactual predictions to improve the execution structure itself. Given a set of valid candidate transformations, the agent selects

$$a^* = \arg \min_{a \in \mathcal{A}_{\text{valid}}} C\Big(\hat{\mathcal{H}}^{(a)}_{t+1:t+K}\Big), \tag{9}$$

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

Validity is defined by task correctness and quality requirements, together with resource and tool-availability limits; a transformation that lowers predicted cost but violates these requirements is excluded. The cost C aggregates the runtime quantities that the four layers expose, including latency, token and compute consumption, failure and retry penalties, and cache misses. Because the predicted trajectories are uncertain and the space of interventions is combinatorial, candidate interventions will be generated from high-probability structural events proposed by the model and evaluated by rollout, and selection will account for predictive uncertainty. The procedure is receding-horizon: after the selected transformation is executed, the hypergraph is updated with observed events and optimisation is repeated.

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 four-layer hypergraph and its evolution trajectory from these events. Long-horizon tasks will be drawn from data analysis, document processing, and mathematical reasoning, each of which exercises all four layers but with different structural patterns.

Baselines and ablations. AgentHyperEvo will be compared with (i) a dynamic heterogeneous graph without hyperedges, (ii) a temporal hypergraph model, (iii) an association bipartite graph representation, and (iv) a deterministic runtime model. Ablations will remove cross-layer hyperedges, evolution history, and counterfactual training, respectively, to isolate the contribution of each component.

Evaluation dimensions. Table 2 summarises the evaluation. Structural prediction will be measured by node and hyperedge birth–death F1 and membership accuracy; higher-order relation prediction by cross-layer hyperedge F1; feedback prediction by the direction and magnitude accuracy of cross-layer effects; counterfactual prediction by the error between predicted and observed intervention effects and by the rank correlation of interventions; generalisation by performance on structure combinations and task families unseen during training; and execution optimisation by cost reduction at matched task success and quality. All comparisons will be made at matched task correctness, so that a cost reduction cannot be obtained by degrading output quality.

Table 2: Evaluation dimensions and their research objectives.

Evaluation dimension	Research objective	Representative metric
Structural prediction	Accurately predict future multi-layer graph structure	Birth/death F1, membership accuracy
Higher-order relation prediction	Identify joint interactions among multiple layers	Cross-layer hyperedge F1
Feedback prediction	Capture bidirectional cross-layer structural evolution	Effect direction and magnitude accuracy
Counterfactual prediction	Estimate the impact of different structural interventions	Intervention-effect error, rank correlation
Generalisation	Predict unseen structural combinations	Held-out task and structure performance
Execution optimisation	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. **Dynamic heterogeneous hypergraph representation.** A unified multi-layer, higher-order representation of long-horizon agent execution, in which task, resource, tool, and cache structures are modelled as distinct yet coupled dynamic layers.
2. **Higher-order co-evolution learning.** A new graph-learning mechanism that learns joint interactions, cross-layer propagation, and recursive feedback among heterogeneous layers, going beyond the single-graph evolution assumed by existing temporal and hypergraph models.
3. **Counterfactual structure prediction.** A predictive relation between local structural interventions and future overall graph structure, providing the foundation for adaptive execution-structure optimisation.

AgentHyperEvo aims to reveal, from a graph-machine-learning perspective, the structural evolution laws of the complex runtime systems behind long-horizon agents, and to advance the intersection of dynamic heterogeneous graph learning and agentic execution infrastructure. The framework will be evaluated end to end: predicted structures will be compared against observed trajectories, predicted intervention effects against paired rollouts, and selected transformations against the execution cost of unmodified runs at equal task quality.

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] Y. Feng et al., "Hypergraph Neural Networks," in *Proc. AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[7] E. Chien et al., "You Are AllSet: A Multiset Function Framework for Hypergraph Neural Networks," in *Proc. International Conference on Learning Representations (ICLR)*, 2022.

[8] E. Rossi et al., "Temporal Graph Networks for Deep Learning on Dynamic Graphs," 2020.

[9] J. Liu et al., "Higher-order Structure Boosts Link Prediction on Temporal Graphs," 2025.