NeuralFSM: Adaptive Multi-Agent Coordination via Learning Finite-State Execution Policy

Anonymous ACL submission

Abstract

LLM-powered multi-agent systems (MAS) have demonstrated strong performance on complex tasks. However, most existing approaches still rely on hand-crafted communication protocols or automatically designed communication topologies, which generalize poorly across tasks. We introduce **NeuralFSM**, a state-driven framework that formulates multi-agent problem solving as **a finite-state execution process**. NeuralFSM learns both the state transition distribution and inter-agent communication weights from interaction traces using a Temporal Coordination Controller. Rather than prioritizing explicit structure generation, the proposed framework uses task context to modulate transition and routing decisions, enabling flexible coordination without manual protocol design. To improve robustness against noisy or adversarial agents, we incorporate graph regularization during training and apply trust-aware message attenuation at runtime. Experiments on diverse benchmarks show that NeuralFSM consistently outperforms prior baselines by an average margin of $6.74\% \sim 19.39\%$, while substantially reducing token consumption. Moreover, NeuralFSM exhibits strong inherent robustness, which is further enhanced by the protection layer, resulting in only a 1.82% performance drop under attack[1].

1 Introduction

Large language models (LLMs) have enabled strong reasoning and generation capabilities, motivating LLM-based multi-agent systems (MAS) that decompose complex tasks into specialized roles and collaborative interactions (Wu et al., 2024a; Hong et al., 2023; Qian et al., 2024a). In many domains, such role specialization and coordination can outperform single-agent prompting, while also improving interpretability through explicit intermediate artifacts (Guo et al., 2024; Li et al., 2024; Tran et al., 2025).

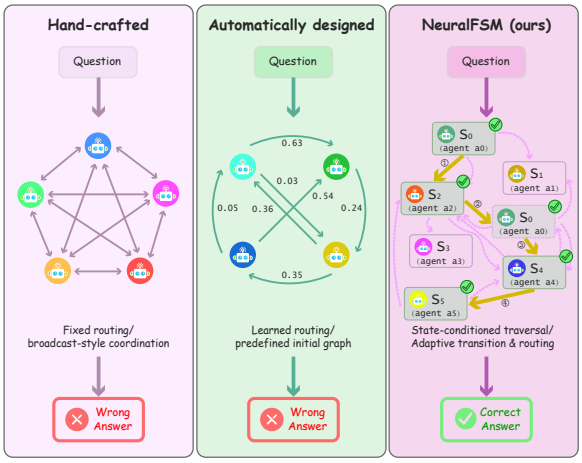

Figure 1: Comparison of multi-agent system design paradigms. Hand-crafted approaches rely on fixed inter-agent communication routing, lacking cross-domain flexibility, while broadcast-style collaboration incurs significant communication overhead. Automatically designed methods can learn problem-specific communication routing but still require partial specifying the initial agent interaction graph. In contrast, our approach dynamically and adaptively performs both state-conditioned transitions and communication routing within an FSM for each problem, guided by a temporal coordination controller.

Despite this progress, existing MAS are often limited by (i) fixed or hand-crafted communication topologies that fail to adapt to task context (Park et al., 2023; Zhang et al., 2024a), (ii) manually specified state-machine rules that require extensive domain engineering (Zhang et al., 2025c; Wu et al., 2024b), (iii) inefficient information sharing via broadcast or ad-hoc routing (Chen et al., 2025; Wu et al., 2024a), and (iv) fragility under noisy or adversarial agents as both coordination scale and domain complexity increase (Wang et al., 2025; Li et al., 2025c).

[1] Our anonymous code is available at https://anonymous.4open.science/r/neuralfsm-anonymous-F833/

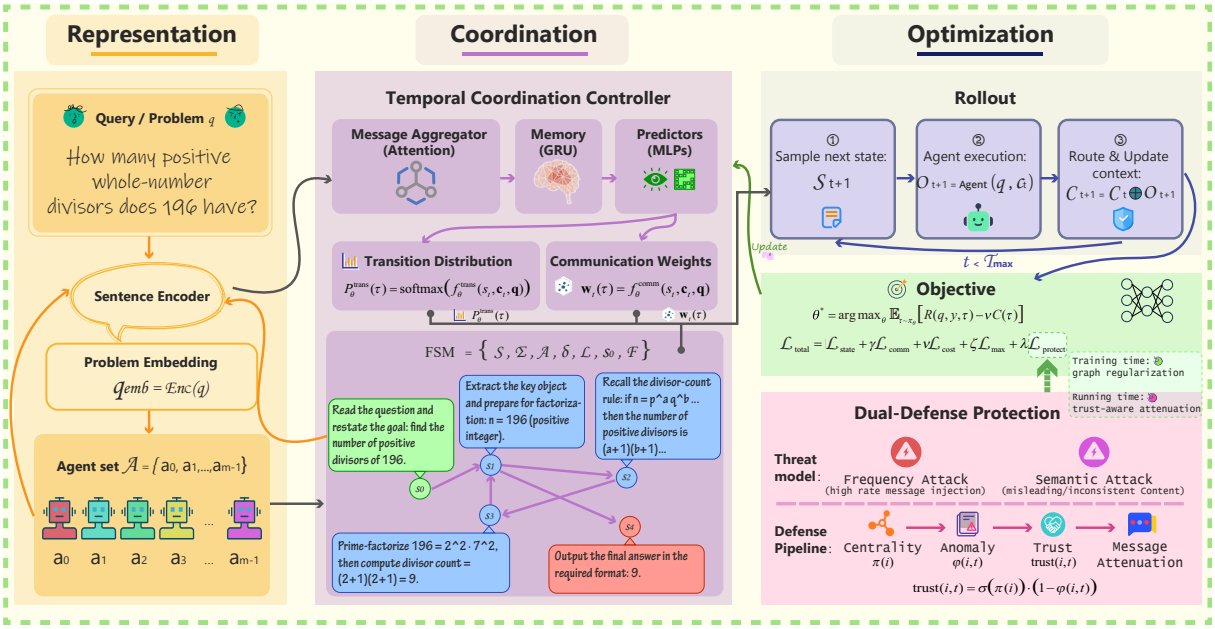

Figure 2: Overview of the NeuralFSM Framework. ***(Left)*** Initialization of the FSM by embedding the problem instance, agents, and states into a shared vector space. ***(Middle)*** A temporal Coordination Controller processes interaction trajectories to produce state transition distribution and communication routing weights. ***(Right)*** Joint optimization of the controller parameters and FSM execution by maximizing the effectiveness-efficiency objective under a fixed horizon, with robustness further enhanced by a dual-defense protection mechanism.

Recent work has explored automated MAS construction and workflow search (Zhuge et al., 2024; Zhang et al., 2024b; Yuan et al., 2025; Hu et al., 2024; Zhang et al., 2025b,a; Ke et al., 2025; Yang et al., 2025), topology learning and communication optimization (Li et al., 2025b; Yue et al., 2025; Li et al., 2025a; Zhou et al., 2025; Feng et al., 2025; Chen et al., 2025), FSM-driven system synthesis (Zhang et al., 2025c; Wu et al., 2024b; Liu et al., 2023; Xinjie et al., 2025), and robustness-oriented defenses (Wang et al., 2025; Li et al., 2025c). However, these directions are typically studied in isolation, lacking a unified learning-based coordination framework.

To address these challenges, we propose NeuralFSM: a framework that uses Temporal Graph Networks (TGN) (Rossi et al., 2020) as a task-adaptive finite-state temporal coordination controller to learn multi-agent coordination. TGNs extend graph neural networks to dynamic graphs evolving by updating node memories through temporal message passing. Our key insight is that inter-agent coordination naturally forms a dynamic interaction graph, making TGNs well suited for learning finite-state execution policies.

NeuralFSM performs task-conditioned coordination modeling. The controller conditions TGN-based predictors on task representations and execution context, enabling adaptive state transitions and communication routing without manually designed protocols or task-specific rules. The controller learns probabilistic distributions governing both state transitions and inter-agent communications from interaction traces within a finite state machine (FSM), thereby enabling task-adaptive coordination policies that generate structured and controllable execution trajectories.

NeuralFSM also includes a dual-defense protection layer, which regularizes anomalous communications during training and attenuates low-trust messages at runtime, improving robustness without altering the normal coordination mechanism.

Our main contributions are summarized as follows:

Adaptive FSM traversal. We cast coordination as *traversing* a Finite State Machine and learn a history-aware temporal coordination controller that jointly predicts state transitions and sparse communication routing from interaction traces.

Dual-Defense Protection. We introduce a protection layer combining training-time graph regularization with runtime trust-aware message attenuation to mitigate frequency and semantic attacks.

Empirical evaluation of effectiveness, efficiency, and robustness. Across six benchmarks, NeuralFSM achieves the best average performance, surpassing prior baselines by $6.74\% \sim 19.39\%$,

while reducing token cost via sparse routing and improving robustness under attack.

2 Related Work

Automated MAS Design and Topology Learning. Prior work constructs LLM-based multi-agent systems through workflow-oriented protocols and predefined templates (e.g., CAMEL, AutoGen, MetaGPT, and ChatDev) (Li et al., 2023; Wu et al., 2024a; Hong et al., 2023; Qian et al., 2024a), as well as automated workflow search and composition methods (Zhang et al., 2025c; Hu et al., 2024; Yuan et al., 2025; Zhang et al., 2025a, 2024b, 2025b; Ke et al., 2025; Yang et al., 2025). A complementary line of work focuses on learning or optimizing inter-agent communication topologies (Zhang et al., 2024a; Li et al., 2025b; Yue et al., 2025; Zhou et al., 2025; Li et al., 2025a; Chen et al., 2025; Feng et al., 2025; Xinjie et al., 2025). In contrast, NeuralFSM does not produce a fixed workflow or topology; instead, it learns a history-aware controller that *traverses* a reusable FSM via task-conditioned state transitions and sparse routing decisions.

Robustness and Security. Prior work studies the safety and robustness of LLM-based MAS from multiple perspectives, including topology-aware safety analyses (Yu et al., 2025; Wang et al., 2025), defenses against unknown or evolving attacks (Miao et al., 2025; Zeng et al., 2024), trust management (He et al., 2025; Amayuelas et al., 2024), and anomaly-aware mitigation for unreliable communications (Li et al., 2025c).

3 Problem Formulation

Given a domain problem $q \in \Sigma$, a finite state machine autonomously generates a set of states $\mathcal{S} = \{s_0, s_1, \ldots, s_{n-1}\}$ and a corresponding set of agents $\mathcal{A} = \{a_0, a_1, \ldots, a_{m-1}\}$ from the problem description. Our goal is to learn a coordination policy within the FSM that reliably solves q based on generated states, while minimizing unnecessary communication and tool or LLM calls.

3.1 Coordination policy

NeuralFSM executes in discrete steps $t \in \{0, \ldots, T_{\max} - 1\}$ with state $s_t \in \mathcal{S}$. At each step, the controller makes two coupled decisions: **state transition**, sample the next state s_{t+1}; and **communication routing**, select a sparse receiver set $\{a_r\}_t \subseteq \mathcal{A}$ with $|\{a_r\}_t| \leq k$, which obtains the output of sate s_t.

Let $\mathbf{q} = \mathrm{Enc}(q)$ denote the problem embedding and $\mathbf{c}_t$ denote the accumulated execution context at step t. The coordination policy is parameterized by θ and factorized as: $\pi_\theta(s_{t+1}, \{a_r\}_t \mid s_t, \mathbf{c}_t, \mathbf{q}) = P_\theta^{\text{trans}}(s_{t+1} \mid s_t, \mathbf{c}_t, \mathbf{q}) \cdot P_\theta^{\text{route}}(\{a_r\}_t \mid s_t, \mathbf{c}_t, \mathbf{q})$, where P_θ^{trans} and P_θ^{route} are distinct conditional distributions that share parameters θ through a common encoder, corresponding to state transition control and sparse communication routing.

Given s_{t+1}, the responsible agent $\dot{a}_{t+1} = \text{responsible}(s_{t+1})$, which may be reused across multiple states, is executed and produces an output o_{t+1}, which is routed to $\{a_r\}_{t+1}$ and used to update the context $\mathbf{c}_{t+1}$. A rollout yields a trajectory τ comprising accessed states, agent executions, routing decisions, and context updates, whose length is bounded by a fixed horizon $T_{\max}$.

3.2 Objective

We optimize a coordination policy that explicitly trades off performance and cost. Let $R(q, y, \tau)$ denote the execution reward of an FSM rollout τ, provided by a task-specific evaluator $u(q, \tau)$ that compares the prediction $\hat{y}$ with the ground-truth y. Let $C(\tau)$ denote the execution cost incurred during the rollout. The learning objective is formulated as the maximization of an expected utility:

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau \sim \pi_\theta} \big[R(q, y, \tau) - \nu C(\tau) \big], \quad (1)$$

where $\nu > 0$ controls the trade-off between performance and cost. In Section 4, we instantiate π_θ as a temporal coordination controller and optimize a differentiable multi-objective loss that jointly captures execution performance, communication sparsity, cost awareness, and robustness.

4 NeuralFSM

As shown in Fig. 2, this section presents NeuralFSM as a task-adaptive coordination mechanism formalized as a finite-state process. It consists of three components: (i) an FSM backbone that defines the coordination search space, (ii) a temporal coordination controller that predicts state transitions and communication routing conditioned on the task and execution context, and (iii) a specific protection layer designed for adversarial settings.

4.1 FSM Backbone

Following the classical formulation of finite state machines (Moore et al., 1956), we define an FSM

tuple extended with an explicit agent set and a communication mapping:

$$\text{FSM} = (\mathcal{S}, \Sigma, \mathcal{A}, \delta, \mathcal{L}, s_0, \mathcal{F}), \tag{2}$$

where $\mathcal{S}$ represents the set of states, corresponding to distinct phases in the problem-solving process, Σ is the input alphabet representing the set of domain problems, $\delta : \mathcal{S} \times \mathcal{X} \to \mathcal{S}$ is the state transition function determining the next state given the current state and contextual information $\mathcal{X}$ accumulated during execution, $\mathcal{L} : \mathcal{S} \to 2^{\mathcal{A}}$ is the communication mapping enabling selective information flow based on relevance, $s_0 \in \mathcal{S}$ is the initial state and $\mathcal{F} \subseteq \mathcal{S}$ is the set of final states, and the rollout returns an final answer by executing a final state.

Neural Finite State Machine dynamically traverses a task-adaptive sequence of states $s_0 \to s_2 \to \cdots \to s_n$, with the ability to backtrack from the current state to a previously visited state. Both forward transitions and backtracking behaviors are determined by the controller π_θ, as shown in Fig. 1.

4.2 Temporal Coordination Controller

Section 3 formulates coordination as state transitions with sparse routing. At step t, the controller produces (i) a transition distribution over $\mathcal{S}$ and (ii) communication weights over $\mathcal{A}$, both conditioned on feature embeddings and execution context:

$$P_\theta^{\text{trans}}(\tau) = \text{softmax}\big(f_\theta^{\text{trans}}(s_t, \mathbf{c}_t, \mathbf{q})\big), \tag{3}$$

$$\mathbf{w}_t(\tau) = f_\theta^{\text{comm}}(s_t, \mathbf{c}_t, \mathbf{q}) \in [0,1]^m, \tag{4}$$

where $\mathbf{w}_t(\tau)$ parameterizes the routing distribution P_θ^{route} induced by sparse sampling: $P_\theta^{\text{route}}(\{a_r\}_t \mid s_t, \mathbf{c}_t, \mathbf{q}) := \Pr\big(\text{Sample}_k(\mathbf{w}_t(\tau), k) = \{a_r\}_t\big)$. The receiver budget k enforces communication sparsity and controls context growth.

The two heads f_θ^{trans} and f_θ^{comm} share a temporal interaction encoder that encodes how information propagates across agents over time. Static graph designs compress the entire interaction history into a single fixed topology, thereby discarding temporal credit assignment information and preventing the collaboration pattern from being adjusted according to task-solving progress, i.e. state. Consequently, NeuralFSM models the multi-agent collaboration pattern within each rollout τ as a time-stamped directed communication graph $G = (V, E_\tau)$, and learns a history encoder ϕ_θ that maps past interactions on G into per-agent memories, which are then used as representations for predicting the next state and communication routing decisions.

In a graph G, the node set V corresponds to agents, and the time-stamped directed edge set E_τ represents routed communications. The controller maintains a memory state $\mathbf{m}_v(t)$ for each node v and updates it through message passing over observed interactions. The resulting node representation is denoted as $\mathbf{h}_v(t) = g_\theta(\mathbf{m}_v(t), \mathbf{e}_v)$, where $\mathbf{e}_v$ is the embedding of node v and g_θ is a learnable decoder.

Temporal message. For an interaction $(u \to v)$ at time t with edge features $\mathbf{e}_{uv}$ and time gap Δt_{uv}, we compute

$$\mathbf{m}(u \to v, t) = \phi_\theta\big([\mathbf{h}_u(t), \mathbf{h}_v(t), \mathbf{e}_{uv}, \psi(\Delta t_{uv})]\big), \tag{5}$$

where $\psi(\Delta t)$ is a temporal encoding (Fourier features with learnable frequencies $\{\omega_\ell\}_{\ell=1}^{d_\psi}$):

$$\psi(\Delta t) = \big[\cos(\omega_1 \Delta t), \sin(\omega_1 \Delta t), \ldots, \cos(\omega_{d_\psi} \Delta t), \sin(\omega_{d_\psi} \Delta t)\big]. \tag{6}$$

Aggregation and update. Incoming messages are aggregated to update memory:

$$\begin{aligned} \mathbf{M}_v^t &= \text{Agg}\big(\{\mathbf{m}(u \to v, t) : u \in \mathcal{N}(v)\}\big), \\ \mathbf{m}_v(t) &= \text{Upd}_\theta(\mathbf{m}_v(t^-), \mathbf{M}_v^t), \end{aligned} \tag{7}$$

where t^- denotes the previous time step. We then derive per-agent features and a global interaction summary:

$$\begin{aligned} \mathbf{z}_{a_j}(t) &= p_\theta^a\big(\mathbf{h}_{a_j}(t)\big), \\ \mathbf{g}_t &= \text{Pool}\big(\{\mathbf{h}_{a_j}(t)\}_{j=0}^{m-1}\big), \end{aligned} \tag{8}$$

where $p_\theta^*(\cdot)$ is a projection and $\text{Pool}(\cdot)$ denotes attention pooling. We augment the execution context as

$$\widetilde{\mathbf{c}}_t = \big[\mathbf{c}_t, \mathbf{g}_t\big], \tag{9}$$

so the controller conditions on both $\mathbf{c}_t$ and interaction history.

Transition head. Let $\mathbf{e}_{s_t}$ be the embedding of state s_t. We compute transition logits by fusing state, task, and context features:

$$\begin{aligned} &\mathbf{z}_s = p_\theta^s(\mathbf{e}_{s_t}), \mathbf{z}_q = p_\theta^q(\mathbf{q}), \mathbf{z}_c = p_\theta^c(\widetilde{\mathbf{c}}_t), \\ &f_\theta^{\text{trans}}(s_t, \mathbf{c}_t, \mathbf{q}) = r_\theta^{\text{trans}}\big([\mathbf{z}_s, \mathbf{z}_q, \mathbf{z}_c]^T\big), \end{aligned} \tag{10}$$

which instantiates Eq. (3). Here, $r_\theta^{\text{trans}}(\cdot)$ is a learnable readout mapping the fused representation $[\mathbf{z}_s, \mathbf{z}_c, \mathbf{z}_q]$ to a $|\mathcal{S}|$-dimensional logit vector over

next states. Similarly, $r_\theta^Q(\cdot)$ and $r_\theta^K(\cdot)$ are learnable readouts that produce query and key vectors for computing routing scores. In our implementation, these readouts are lightweight feed-forward networks whose parameters are included in θ.

Routing head. We compute per-agent routing scores $w_{t,j} \in [0,1]$ via query-key compatibility between the fused representation and per-agent features:

$$\mathbf{Q}_t = r_\theta^Q\big([\mathbf{z}_s, \mathbf{z}_q, \mathbf{z}_c]\big), \mathbf{K}_j(t) = r_\theta^K\big(\mathbf{z}_{a_j}(t)\big).$$

$$u_{t,j} = \frac{\mathbf{Q}_t^\top \mathbf{K}_j(t)}{\sqrt{d_k}}, w_{t,j} = \sigma\big(u_{t,j}\big), \qquad (11)$$

where $\sigma(\cdot)$ is the sigmoid and d_k is the query/key dimension. The vector $\mathbf{w}_t(\tau) = [w_{t,0}, ..., w_{t,m-1}]$ instantiates Eq. (4).

To optimize Eq. (1) with discrete state traversal, we use a Monte Carlo policy-gradient estimator (REINFORCE; (Williams, 1992)) with trajectory-level regularizers, yielding the multi-objective loss:

$$\begin{aligned}\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{state}} + \gamma\mathcal{L}_{\text{comm}} + \nu\mathcal{L}_{\text{cost}} \\ + \zeta\mathcal{L}_{\text{max}} + \lambda\mathcal{L}_{\text{protect}},\end{aligned} \qquad (12)$$

where each term corresponds to a specific coordination requirement and is computed on the sampled trajectory τ.

Unified state-transition learning signal. We merge an outcome-driven policy gradient signal with a transition-likelihood regularizer into a single state-transition loss:

$$\mathcal{L}_{\text{state}} = -\sum_{t=0}^{T_{\max}-1} \log P_\theta^{\text{trans}}(\tau) \cdot \big(\alpha(R-b)+\beta\big), \qquad (13)$$

where b is the average reward based on historical batches. The term $\alpha(R-b)$ provides outcome-driven credit assignment, while β stabilizes learning under sparse rewards.

Routing likelihood. Let $y_{t,j} \in \{0,1\}$ indicate whether agent a_j is selected as a receiver at step t. Given routing scores $w_{t,j}$, we use a Bernoulli cross-entropy objective:

$$\begin{aligned}\mathcal{L}_{\text{comm}} = -\sum_{t=0}^{T_{\max}-1}\sum_{j=0}^{m-1}\Big(y_{t,j}\log(w_{t,j}+\epsilon) \\ + (1-y_{t,j})\log(1-w_{t,j}+\epsilon)\Big).\end{aligned} \qquad (14)$$

Cost and horizon regularization. We set $\mathcal{L}_{\text{cost}} = (C(\tau) - C_{\text{baseline}})^2$ with a running baseline C_{baseline}, and use $\mathcal{L}_{\text{max}} = \mathbb{1}[|\tau| = T_{\max}]$ to discourage degenerate long traversals.

$\mathcal{L}_{\text{protect}}$ is enabled only when the protection module is active. The complete optimization and execution procedures in this section are provided in Appendix B (Algorithms 1–2).

4.3 Protection Layer

To address robustness issues arising from adversarial agents, we augment NeuralFSM with a protection layer.

Threat model. A subset of agents may be compromised and can affect coordination via the communication channel of the induced graph G. We consider two representative attacks: (i) frequency attacks, where compromised agents inject messages at an abnormally high rate, and (ii) semantic attacks, where compromised agents inject misleading content into the execution context $\mathbf{c}_t$. We parameterize attacks by an attacked-agent ratio $\rho \in (0,1]$ and an attack strength $\eta > 0$.

The protection layer computes (i) a structural priority $\pi(i)$ on G and (ii) an online anomaly score $\varphi(i,t)$, then derives a bounded trust score $\text{trust}(i,t) \in [0,1]$ used for runtime attenuation and training-time regularization.

Centrality analysis. On the induced directed communication graph $G(V, E_\tau)$, we compute a priority score from two centrality measures, betweenness centrality $\text{BC}(i)$ and PageRank $\text{PR}(i)$, whose formal definitions are provided in Appendix D.1:

$$\pi(i) = w_{\text{BC}}\,\text{BC}(i) + w_{\text{PR}}\,\text{PR}(i), \qquad (15)$$

where $w_{\text{BC}}, w_{\text{PR}} \geq 0$ and $w_{\text{BC}} + w_{\text{PR}} = 1$. The score $\pi(i)$ serves as a graph-centric prior on structural influence.

Anomaly detection. We compute an online anomaly score from two complementary observable anomalous signals, frequency anomaly φ_{freq} and semantic anomaly φ_{sem}, whose computations are described in Appendix D.2:

$$\varphi(i,t) = \sigma\Big(\omega_{\text{freq}}\varphi_{\text{freq}}(i,t) + \omega_{\text{sem}}\varphi_{\text{sem}}(i,t)\Big), \qquad (16)$$

where $\omega_{\text{freq}}, \omega_{\text{sem}} \geq 0$ are fusion weights, and $\omega_{\text{freq}} + \omega_{\text{sem}} = 1$.

We convert $\pi(i)$ and $\varphi(i,t)$ into a bounded trust score that penalizes anomalous senders while accounting for graph structure:

$$\text{trust}(i,t) = \sigma\big(\pi(i)\big) \cdot \big(1 - \varphi(i,t)\big). \qquad (17)$$

Dual-defense strategy. We deploy the protection layer in two complementary ways. (i) **Run-**

time defense. The protection layer attenuates message representations produced by the temporal coordination controller using sender trust:

$$\widetilde{\mathbf{m}}(u \to v, t) = \text{trust}(u, t) \cdot \mathbf{m}(u \to v, t), \quad (18)$$

which reduces the influence of low-trust senders while preserving useful signals in clean-setting. (ii) **Training-time defense**. During training, a protection regularizer $\mathcal{L}_{\text{protect}}$ (see Appendix D.3) discourages strong messages from low-trust sources.

5 Experiments

5.1 Experiment Setup

Tasks and Benchmarks. We evaluate NeuralFSM on a diverse set of domains spanning mathematical reasoning, GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021); code synthesis, HumanEval (Chen, 2021), MBPP (Austin et al., 2021); knowledge-intensive QA, GPQA (Rein et al., 2024); multi-hop QA, HotpotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020), and MuSiQue (Trivedi et al., 2022); embodied decisions, ALFWorld (Shridhar et al., 2020); and tool using, GAIA (Mialon et al., 2023).

Baselines. We compare against three baseline families: Single-agent prompting: IO (OpenAI, 2025), CoT (Wei et al., 2022), and self-consistency (CoT×5) (Wang et al., 2022). Hand-crafted MAS: DyLAN (Liu et al., 2024), LLM-Debate (Du et al., 2023), AgentVerse (Chen et al., 2024), and MacNet (Qian et al., 2024b). Automatically designed MAS: GPTSwarm (Zhuge et al., 2024), AutoAgents (Chen et al., 2023), AFlow (Zhang et al., 2024b), G-Designer (Zhang et al., 2024a), and MaAS (Zhang et al., 2025b). For multi-hop QA, we additionally include strong reasoning baselines KAG (Liang et al., 2025), CoA (Zhang et al., 2024c), and ReAgent (Xinjie et al., 2025).

LLM Backbones. We use four closed-source models in our paper: `gpt-5-nano`, `gpt-4o-mini`, `gpt-4.1-nano`, and `grok-3-mini`. All LLMs are accessed via APIs, with the temperature set to 1.

Parameter Configuration. Unless otherwise stated, we use a rollout horizon $T_{\max} = 8$ and a sparse receiver budget $k = 3$. The embbeding function $\text{Enc}(\cdot)$ adopts all-MiniLM-L6-v2 (Wang et al., 2020). Loss weights follow Section 4.2 with $(\alpha, \beta, \gamma, \nu, \zeta, \lambda) = (1.0, 0.3, 0.2, 0.1, 0.5, 0.3)$. For attack settings, we set $(\rho, \eta) = (0.5, 3.0)$ by default. For the protection layer, the following parameters are fixed across all robustness runs: $(w_{\text{BC}}, w_{\text{PR}}) = (0.6, 0.4)$, $d_{\text{PR}} = 0.85$, $(\omega_{\text{freq}}, \omega_{\text{sem}}) = (0.3, 0.7)$.

5.2 Performance Analysis

We compare NeuralFSM with 13 baselines on the GSM8K, MATH, GPQA, HumanEval, MBPP and ALFWorld benchmarks in Table 1, and with 7 baselines on multi-hop QA benchmarks in Table 2. In addition, we analyze the model transferability of NeuralFSM in Table 6. The experimental results are analyzed as follows:

NeuralFSM achieves the best overall performance across six benchmarks. Compared to state-of-the-art (SOTA) designed MAS, NeuralFSM produces an average improvement of 6.74%; relative to the single-agent IO baseline, the average gain increases to 19.39%. The improvements are most pronounced in benchmarks with high interaction demands. For example, on MATH, NeuralFSM outperforms IO by 21.69% and the SOTA baseline by 12.75%. On GPQA and ALFWorld, it further surpasses the SOTA by 13.28% and 5.63%, respectively. These results are consistent with the advantages conferred by task-conditioned finite-state traversal.

On three multi-hop QA benchmarks, NeuralFSM achieves the best overall performance. Compared to the strongest multi-hop baseline ReAgent, NeuralFSM achieves an average improvement of 15.21%, with consistent gains of 6.29% on HotpotQA, 13.88% on 2Wiki, and 32.98% on MuSiQue. Notably, the performance of single-model prompting varies substantially with the choice of backbone (avg. 36.49–46.72), whereas the coordinated execution enabled by NeuralFSM yields a markedly larger performance improvement (45.96% over CoA) than simply switching to a stronger backbone.

On GAIA, NeuralFSM benefits from stronger LLM backbones while effectively controlling cost. The average score increases from 4.18 (`gpt-4o-mini`) to 12.75 (`gpt-5-nano`), corresponding to a 205.0% relative improvement, while the cost increases by only 135.0%. Switching to `grok-3-mini` yields a modest additional performance gain of 17.5% over `gpt-5-nano`, but nearly doubles the cost. Especially at Level 2, performance improves by only 10.0% while the cost increases by 80.5%. At Level 3, performance even drops by 49.9%, despite a 147.6% increase in cost. These results indicate that Levels 2–3 remain challenging across different backbones, suggesting that

Methods	GSM8K	MATH	GPQA	HumanEval	MBPP	ALFWorld	Avg.
IO(OpenAI, 2025)	92.19	76.31	48.99	90.15	78.73	41.63	71.33
CoT(Wei et al., 2022)	91.95$_{-0.24}$	76.12$_{-0.19}$	49.08$_{+0.09}$	91.26$_{+1.11}$	78.52$_{-0.21}$	42.81$_{+1.18}$	71.62$_{+0.29}$
SC (CoT×5)(Wang et al., 2022)	92.76$_{+0.57}$	77.85$_{+1.54}$	49.17$_{+0.18}$	91.49$_{+1.34}$	80.24$_{+1.51}$	43.42$_{+1.79}$	72.49$_{+1.16}$
DyLAN(Liu et al., 2024)	94.72$_{+2.53}$	78.41$_{+2.10}$	50.36$_{+1.37}$	91.70$_{+1.55}$	83.90$_{+5.17}$	55.10$_{+13.47}$	75.70$_{+4.37}$
LLM-Debate(Du et al., 2023)	94.55$_{+2.36}$	77.94$_{+1.63}$	50.24$_{+1.25}$	91.38$_{+1.23}$	84.11$_{+5.38}$	48.67$_{+7.04}$	74.48$_{+3.15}$
AgentVerse(Chen et al., 2024)	94.85$_{+2.66}$	77.56$_{+1.25}$	51.53$_{+2.54}$	92.13$_{+1.98}$	82.45$_{+3.72}$	47.39$_{+5.76}$	74.32$_{+2.99}$
MacNet(Qian et al., 2024b)	93.03$_{+0.84}$	75.29$_{-1.02}$	50.75$_{+1.76}$	89.66$_{-0.49}$	74.27$_{-4.46}$	47.01$_{+5.38}$	71.67$_{+0.34}$
GPTSwarm(Zhuge et al., 2024)	93.74$_{+1.55}$	78.09$_{+1.78}$	52.65$_{+3.66}$	91.81$_{+1.66}$	84.39$_{+5.66}$	55.88$_{+14.25}$	76.09$_{+4.76}$
AutoAgents(Chen et al., 2023)	92.68$_{+0.49}$	74.63$_{-1.68}$	52.73$_{+3.74}$	90.30$_{+0.15}$	78.53$_{-0.20}$	49.75$_{+8.12}$	73.10$_{+1.77}$
AFlow(Zhang et al., 2024b)	95.50$_{+3.31}$	80.96$_{+4.65}$	53.83$_{+4.84}$	93.10$_{+2.95}$	88.31$_{+9.58}$	61.94$_{+20.31}$	78.94$_{+7.61}$
G-Designer(Zhang et al., 2024a)	96.17$_{+3.98}$	80.78$_{+4.47}$	53.06$_{+4.07}$	91.47$_{+1.32}$	87.87$_{+9.14}$	54.32$_{+12.69}$	77.28$_{+5.95}$
MaAS(Zhang et al., 2025b)	97.09$_{+4.90}$	81.22$_{+4.91}$	54.91$_{+5.92}$	94.72$_{+4.57}$	88.16$_{+9.43}$	62.55$_{+20.92}$	79.78$_{+8.45}$
MasHost(Yang et al., 2025)	98.41$_{+6.22}$	82.36$_{+6.05}$	55.28$_{+6.29}$	93.15$_{+3.00}$	87.93$_{+9.20}$	62.27$_{+20.64}$	79.90$_{+8.57}$
NeuralFSM (Ours)	**98.81**$_{+6.62}$	**92.86**$_{+16.55}$	**63.14**$_{+14.15}$	**98.73**$_{+8.58}$	**91.36**$_{+12.63}$	**66.07**$_{+24.44}$	**85.16**$_{+13.83}$

Table 1: Performance comparison across benchmarks with single-agent systems, hand-craft multi-agent systems, and automatically designed multi-agent systems. `gpt-5-nano` is used as LLM backbone for all baselines. We **bold** the best results and underline the runner-ups.

Models / Methods	HotpotQA	2Wiki	Musique	Avg.
GPT-4o-mini	35.97	50.43	23.06	36.49
GPT-4.1-nano	37.92	51.25	24.54	37.90
GPT-5-nano	43.69	57.11	29.68	43.49
Grok-3-mini	49.71	59.39	31.05	46.72
CoA(Zhang et al., 2024c)	44.54	58.19	30.82	44.52
KAG(Liang et al., 2025)	58.70	65.32	32.25	52.09
ReAgent(Xinjie et al., 2025)	62.16	70.40	36.63	56.40
NeuralFSM (Ours)	66.07	80.17	48.71	64.98

Table 2: Performance of different models and methods across three multi-hop QA datasets. All four LLM-based methods use `gpt-5-nano` as the backbone.

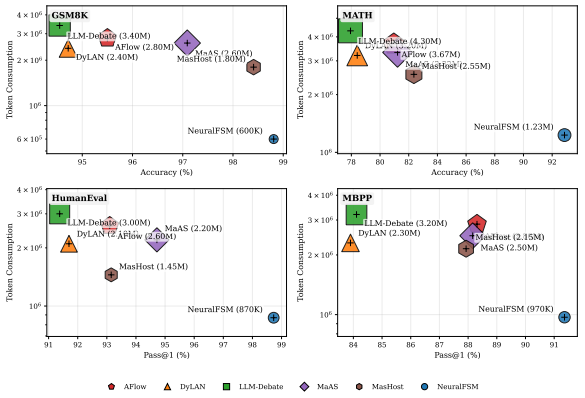

Figure 3: Performance-cost evaluation. Accuracy/pass@1 versus token consumption compared to representative baselines.

the remaining failures are primarily attributable to limitations of the underlying base models rather than coordination collapse.

5.3 Cost Analysis

NeuralFSM improves the effectiveness–efficiency frontier across representative benchmarks in Fig. 3. On GSM8K, it achieves comparable accuracy while using 66.67% fewer tokens than MasHost. On MATH, NeuralFSM improves accuracy by 12.77% while using 51.76% fewer tokens than MasHost. Similar Pareto-dominant behavior is observed on HumanEval, where NeuralFSM reduces token usage by approximately 60.45% compared to MaAS while improving pass@1 by 4.23%, as well as on MBPP, where it uses about 66.08% fewer tokens than AFlow and improves pass@1 by 3.45%. These results support our claim that task-conditioned traversal combined with sparse routing effectively reduces redundant deliberation and uncontrolled context growth.

5.4 Framework Analysis

Sensitivity Analysis. As shown in Fig. 4, increasing the FSM size and rollout horizon reveals clear diminishing returns. Increasing the FSM size from

Datasets	GSM8K		MATH	
	Perf.	Cost ($)	Perf.	Cost ($)
NeuralFSM	98.81	0.164	92.86	0.395
w/o Protection	98.67	0.159	92.78	0.389
w/o Trans	96.13	0.197	87.64	0.623
w/o Comm	96.03	0.192	88.61	0.432

Table 3: Ablation study of NeuralFSM.

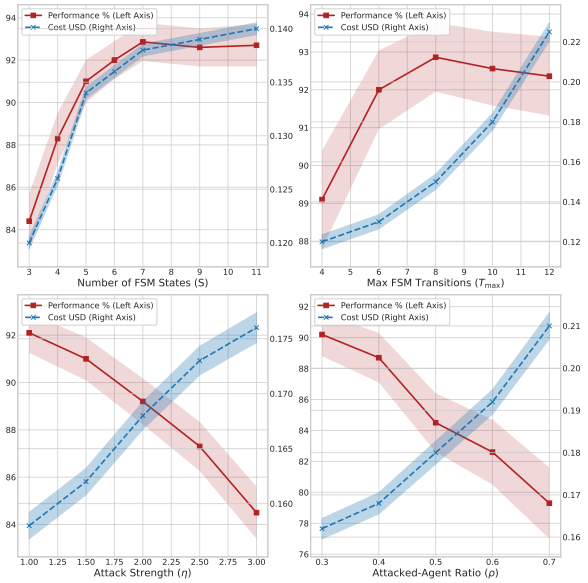

Figure 4: Parameter sensitivity of NeuralFSM. We vary FSM size S, rollout horizon $T_{\max}$, attack strength η, and attacked-agent ratio ρ, reporting performance and cost.

$S = 3$ to $S = 7$ yields a 10.02% relative performance improvement at the expense of a 15.21% increase in cost, whereas further enlarging the state space provides negligible additional benefit. Similarly, extending the rollout horizon from $T_{\max} = 4$ to $T_{\max} = 8$ improves performance by 4.22%, while increasing $T_{\max}$ beyond 8 leads to a 0.54% performance drop accompanied by a sharp 50.45% increase in cost, suggesting that adopting $T_{\max} = 8$ as the default setting in this paper is reasonable. Under attack settings without protection, increasing the attack strength η and the attacked-agent ratio ρ monotonically degrades performance while increasing cost. Specifically, raising η from 1.0 to 3.0 results in a 8.25% performance decrease and an 11.39% increase in cost, while increasing ρ from 0.3 to 0.7 causes performance to drop by 12.08% and cost to rise by 29.63%.

Ablation Study. Table 3 shows that removing the protection layer under no-attack settings leads to almost no change in either performance or cost for NeuralFSM, suggesting that the protection mechanism does not interfere with the normal execution of NeuralFSM. In contrast, disabling transition prediction significantly degrades performance, resulting in a 5.62% drop in accuracy and a 57.72% increase in cost on MATH, which indicates that learned traversal is crucial for hard multi-step reasoning. Disabling communication sampling also reduces performance, with a 2.82% decrease on GSM8K and a 17.07% increase in cost.

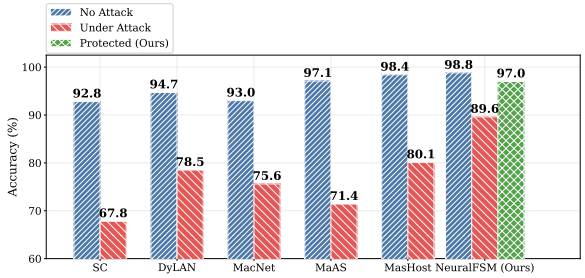

Figure 5: Robustness of NeuralFSM and other baselines under attack on GSM8K: accuracy in clean-setting, under attack, and with our proposed protection layer enabled.

5.5 Robustness Analysis

As shown in Fig. 5, the protection layer provides nearly complete isolation of NeuralFSM from communication channel attacks, and even without a protection layer, NeuralFSM demonstrates strong inherent robustness to attacks. Under attack, all baselines suffer substantial relative performance degradations: DyLAN drops by 17.11%, MaAS by 26.47% and MasHost by 18.60%. In contrast, NeuralFSM exhibits markedly greater resilience without protection, with a relative performance drop of 9.31%. Enabling the proposed protection layer further improves accuracy by 8.26% relative to the unprotected, reducing the remaining gap to clean performance to just 1.82%. These results are consistent with our dual-defense design, indicating that the protection layer effectively mitigates the influence of abnormal messages without disrupting the coordination mechanism. Additional robustness experiment is shown in Appendix E.2.

6 Conclusion

In this paper, we present NeuralFSM, a task-adaptive coordination framework that executes a finite-state process per problem and learns both state transitions and sparse communication routing via a temporal coordination controller. Unlike approaches that rely on a fixed collaboration topology, NeuralFSM generates time-step execution trajectories specific to the internal state of a reusable FSM, enabling adaptive execution policies that balance effectiveness and efficiency. Experiments on six benchmarks demonstrate that NeuralFSM achieves superior performance at markedly lower token cost. A specific protection layer further enhances robustness during coordination in adversarial settings.

Limitations

Although NeuralFSM outperforms fixed-protocol orchestration and other parameter optimization methods in both performance and cost by training a task-conditional coordinator to learn finite-state execution policies, ensuring policy stability and transferability still depends on careful hyperparameter tuning and sufficient interaction traces. Especially under varying task distributions, further investigation is required to enable automatic adaptation of key hyperparameters–such as the horizon $T_{\max}$–based on interaction traces across different task domains, achieving stronger cross-domain generalization capabilities. This direction constitutes an important avenue for future work. Moreover, the benchmarks adopted in our experiments are not particularly challenging for contemporary LLM backbones and are primarily selected to align with prior baselines. Finally, robustness remains an open and evolving challenge: as attack strategies continue to develop, corresponding defense mechanisms may require continual refinement to maintain reliable deployment in open environments.

References

Alfonso Amayuelas, Xianjun Yang, Antonis Antoniades, Wenyue Hua, Liangming Pan, and William Yang Wang. 2024. Multiagent collaboration attack: Investigating adversarial attacks in large language model collaborations via debate. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6929–6948.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F Karlsson, Jie Fu, and Yemin Shi. 2023. Autoagents: A framework for automatic agent generation. *arXiv preprint arXiv:2309.17288*.

Mark Chen. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, and 1 others. 2024. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *ICLR*.

Weize Chen, Jiarui Yuan, Chen Qian, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2025. Optima: Optimizing effectiveness and efficiency for llm-based multi-agent system. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11534–11557.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*.

Shangbin Feng, Zifeng Wang, Palash Goyal, Yike Wang, Weijia Shi, Huang Xia, Hamid Palangi, Luke Zettlemoyer, Yulia Tsvetkov, Chen-Yu Lee, and 1 others. 2025. Heterogeneous swarms: Jointly optimizing model roles and weights for multi-llm systems. *arXiv preprint arXiv:2502.04510*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

Pengfei He, Zhenwei Dai, Xianfeng Tang, Yue Xing, Hui Liu, Jingying Zeng, Qiankun Peng, Shrivats Agrawal, Samarth Varshney, Suhang Wang, and 1 others. 2025. Attention knows whom to trust: Attention-based trust management for llm multi-agent systems. *arXiv preprint arXiv:2506.02546*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2023. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Shengran Hu, Cong Lu, and Jeff Clune. 2024. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*.

Zixuan Ke, Austin Xu, Yifei Ming, Xuan-Phi Nguyen, Caiming Xiong, and Shafiq Joty. 2025. Mas-zero: Designing multi-agent systems with zero supervision. *arXiv preprint arXiv:2505.14996*.

Boyi Li, Zhonghan Zhao, Der-Horng Lee, and Gaoang Wang. 2025a. Adaptive graph pruning for multi-agent communication. *arXiv preprint arXiv:2506.02951*.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008.

Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. 2025b. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. *arXiv preprint arXiv:2507.18224*.

Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. 2024. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9.

Zherui Li, Yan Mi, Zhenhong Zhou, Houcheng Jiang, Guibin Zhang, Kun Wang, and Junfeng Fang. 2025c. Goal-aware identification and rectification of misinformation in multi-agent systems. *arXiv preprint arXiv:2506.00509*.

Lei Liang, Zhongpu Bo, Zhengke Gui, Zhongshu Zhu, Ling Zhong, Peilong Zhao, Mengshu Sun, Zhiqiang Zhang, Jun Zhou, Wenguang Chen, and 1 others. 2025. Kag: Boosting llms in professional domains via knowledge augmented generation. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 334–343.

Jia Liu, Jie Shuai, and Xiyao Li. 2023. State machine of thoughts: Leveraging past reasoning trajectories for enhancing problem solving. *arXiv preprint arXiv:2312.17445*.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2024. A dynamic llm-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

Rui Miao, Yixin Liu, Yili Wang, Xu Shen, Yue Tan, Yiwei Dai, Shirui Pan, and Xin Wang. 2025. Blindguard: Safeguarding llm-based multi-agent systems under unknown attacks. *arXiv preprint arXiv:2508.08127*.

Edward F Moore and 1 others. 1956. Gedankenexperiments on sequential machines. *Automata studies*, 34:129–153.

OpenAI. 2025. Gpt-5 nano model documentation. Accessed: 2025-10-01.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, and 1 others. 2024a. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186.

Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, and 1 others. 2024b. Scaling large language model-based multi-agent collaboration. *arXiv preprint arXiv:2406.07155*.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. 2025. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Shilong Wang, Guibin Zhang, Miao Yu, Guancheng Wan, Fanci Meng, Chongye Guo, Kun Wang, and Yang Wang. 2025. G-safeguard: A topology-guided security lens and treatment on llm-based multi-agent systems. *arXiv preprint arXiv:2502.11127*.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2024a. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*.

Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. 2024b. Stateflow: Enhancing llm task-solving through state-driven workflows. *arXiv preprint arXiv:2403.11322*.

Zhao Xinjie, Fan Gao, Xingyu Song, Yingjian Chen, Rui Yang, Yanran Fu, Yuyang Wang, Yusuke Iwasawa, Yutaka Matsuo, and Irene Li. 2025. Reagent: Reversible multi-agent reasoning for knowledge-enhanced multi-hop qa. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 4067–4089.

Kuo Yang, Xingjie Yang, Linhui Yu, Qing Xu, Yan Fang, Xu Wang, Zhengyang Zhou, and Yang Wang. 2025. Mashost builds it all: Autonomous multi-agent system directed by reinforcement learning. *arXiv preprint arXiv:2506.08507*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380.

Miao Yu, Shilong Wang, Guibin Zhang, Junyuan Mao, Chenlong Yin, Qijiong Liu, Kun Wang, Qingsong Wen, and Yang Wang. 2025. Netsafe: Exploring the topological safety of multi-agent system. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2905–2938.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. 2025. Evoagent: Towards automatic multi-agent generation via evolutionary algorithms. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6192–6217.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. 2025. Masrouter: Learning to route llms for multi-agent systems. *arXiv preprint arXiv:2502.11133*.

Yifan Zeng, Yiran Wu, Xiao Zhang, Huazheng Wang, and Qingyun Wu. 2024. Autodefense: Multi-agent llm defense against jailbreak attacks. *arXiv preprint arXiv:2403.04783*.

Guibin Zhang, Kaijie Chen, Guancheng Wan, Heng Chang, Hong Cheng, Kun Wang, Shuyue Hu, and Lei Bai. 2025a. Evoflow: Evolving diverse agentic workflows on the fly. *arXiv preprint arXiv:2502.07373*.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. 2025b. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. 2024a. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, and 1 others. 2024b. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*.

Yaolun Zhang, Xiaogeng Liu, and Chaowei Xiao. 2025c. Metaagent: Automatically constructing multi-agent systems based on finite state machines. *arXiv preprint arXiv:2507.22606*.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Arik. 2024c. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Ö Arık. 2025. Multi-agent design: Optimizing agents with better prompts and topologies. *arXiv preprint arXiv:2502.02533*.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*.

A Notation

We present a comprehensive review of commonly used notation and their definitions in Table 4.

B Algorithm

We provide the complete pseudocode referenced in Section 4. The algorithmic steps are consistent with the coordination policy and the learning objective.

Algorithm 1 performs end-to-end learning of the task-conditioned coordination policy described in Section 3. Each episode rolls out a trajectory τ by repeatedly sampling the next state from $P_\theta^{\text{trans}}(\tau)$ and selecting a sparse receiver set from $\mathbf{w}_t(\tau)$, while executing the responsible agent of the visited state and updating the shared context $\mathbf{c}_t$. The parameters θ are optimized with the integrated objective $\mathcal{L}_{\text{total}}$ in Section 4.2, combining state-control learning, routing regularization, cost-aware shaping, and robustness regularization.

Algorithm 2 is the deployment-time executor. Given an input q, it traverses the master FSM for up to $T_{\max}$ steps, producing a sequence of states and sparse communications that is conditioned on the evolving context $\mathbf{c}_t$. The resulting execution log exposes which state was chosen, which agent acted, and which receivers were routed at each step, enabling both interpretability and controllable decoding.

Algorithm 3 describes the protection layer in Section 4.3. It computes a graph-centric prior $\pi(i)$ from centrality on the induced communication graph and an online anomaly score $\varphi(i,t)$ from frequency and semantic signals, then converts them into a bounded trust score $\text{trust}(i,t)$. At runtime, the controller attenuates message representations through $\widetilde{\mathbf{m}}(\cdot,t)$ *before* producing state-transition and routing predictions, reducing the influence of suspicious senders while preserving the clean-setting coordination mechanism. The protection loss $\mathcal{L}_{\text{protect}}$ is used only during training as a regularizer and is not computed in this forward pass routine.

C Dataset Statistics and Splits

Table 5 summarizes the dataset statistics used in our pipeline, where all datasets are initially obtained from authenticated Hugging Face repositories, and we have not used this data for commercial purposes. Unless otherwise specified, we construct a lightweight benchmark subset by stratified sampling based on the difficulty of different benchmarks and split them into train/test with a fixed ratio of $4{:}1$ to standardize training signals and evaluation difficulty across domains. For train-only benchmarks (Humaneval, GPQA, and GAIA), the test size is marked as "–", with GAIA containing 165 questions ($53/86/26$ across three levels). For multi-hop QA tasks, EM denotes *Exact Match*, i.e., the percentage of predictions that match the ground-truth answer string exactly after standard normalization.

D Protection Mechanism Implementation Methods

D.1 Centrality Measures

The definitions of the two centrality measures used in the paper are as follows: First, Betweenness centrality:

$$\text{BC}(i) = \sum_{\substack{s,t \in V \\ s \neq i \neq t,\ s \neq t}} \frac{\sigma_{st}(i)}{\sigma_{st}}, \tag{19}$$

where σ_{st} is the number of shortest paths from s to t and $\sigma_{st}(i)$ counts those paths that pass through node i. Second, PageRank with damping factor $d_{\text{PR}} \in (0,1)$:

$$\text{PR}(i) = \frac{1 - d_{\text{PR}}}{|V|} + d_{\text{PR}} \sum_{j \in \mathcal{N}_{\text{in}}(i)} \frac{\text{PR}(j)}{\max(1, \deg_{\text{out}}(j))} \tag{20}$$

, where $\mathcal{N}_{\text{in}}(i)$ denotes in-neighbors and $\deg_{\text{out}}(j)$ is out-degree.

D.2 Anomalous Signals

The definitions and computations for the two anomalous signals are as follows: (i) Frequency anomaly. We define the behavior of an agent injecting messages into the system at a high frequency as frequency anomaly. Let $n_i(t;W)$ be the number of messages sent by agent i within a window of size W ending at t, and let $\mu_W(t)$, $\sigma_W(t)$ be the mean and standard deviation of $\{n_j(t;W)\}_{j=0}^{m-1}$. We define

$$\varphi_{\text{freq}}(i,t) = \sigma\Big(\frac{n_i(t;W) - \mu_W(t)}{\sigma_W(t) + \epsilon}\Big), \tag{21}$$

where $\sigma(\cdot)$ is the sigmoid and $\epsilon > 0$ is a small constant. (ii) Semantic anomaly. We define the behavior of an agent's execution output is inconsistent with the problem and the current context as semantic anomaly. Let $\text{sim}(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\|\,\|\mathbf{y}\|}$ be

Algorithm 1 NeuralFSM Training

Require: Dataset $\mathcal{D} = \{(q, y)\}$; horizon $T_{\max}$; receiver budget k; loss weights $(\alpha, \beta, \gamma, \nu, \zeta, \lambda)$; learning rate η_{lr}; max epochs $E_{\max}$; batch size B

Ensure: Trained parameters θ

Initialize master FSM $= (\mathcal{S}, \Sigma, \mathcal{A}, \delta, \mathcal{L}, s_0, F)$
Initialize θ; initialize encoder $\mathrm{Enc}(\cdot)$; initialize optimizer with η_{lr}
Initialize baseline $b \leftarrow 0$; cost history cost_hist $\leftarrow [\,]$
for epoch = 1 to $E_{\max}$ **do**
  for minibatch $\{(q_i, y_i)\}_{i=1}^{B}$ **do**
    $L_{\text{batch}} \leftarrow 0$
    for $i = 1$ **to** B **do**
      $\mathbf{q} \leftarrow \mathrm{Enc}(q_i)$; $s_t \leftarrow s_0$; $\mathbf{c}_t \leftarrow \emptyset$; $t \leftarrow 0$
      trans_hist $\leftarrow [\,]$; comm_hist $\leftarrow [\,]$; $C_{\text{episode}} \leftarrow 0$
      while $t < T_{\max}$ **do**
        $(P_\theta^{\text{trans}}(\tau), \mathbf{w}_t(\tau), \text{aux}) \leftarrow \mathrm{Controller}_\theta(s_t, \mathbf{c}_t, \mathbf{q})$
        Sample next state $s_{t+1} \sim P_\theta^{\text{trans}}(\tau)$; append (t, s_t, s_{t+1}) to trans_hist
        Let executing agent $\dot{a}_{t+1} \leftarrow \mathrm{responsible}(s_{t+1})$; $(o_{t+1}, \text{cost}) \leftarrow \dot{a}_{t+1}.\mathrm{execute}(q_i, \mathbf{c}_t)$
        $C_{\text{episode}} \leftarrow C_{\text{episode}} + \text{cost}$
        Sample receivers $\{a_r\}_t \leftarrow \mathrm{Sample}_k(\mathbf{w}_t(\tau), k)$
        For each $a_j \in \mathcal{A}$, record $y_{t,j} \leftarrow 1$ if $a_j \in \{a_r\}_t$ else 0; append $(t, a_j, w_{t,j}, y_{t,j})$ to comm_hist
        $\mathbf{c}_{t+1} \leftarrow \mathrm{UpdateContext}(\mathbf{c}_t, \{a_r\}_t, o_{t+1})$; $s_t \leftarrow s_{t+1}$; $t \leftarrow t + 1$
      end while
      Compute episode reward $R \leftarrow R(q_i, y_i, \tau)$ from the final output and evaluator
      Compute $\mathcal{L}_{\text{state}}, \mathcal{L}_{\text{comm}}, \mathcal{L}_{\text{cost}}, \mathcal{L}_{\max}, \mathcal{L}_{\text{protect}}$ following Section 4
      $\mathcal{L}_i \leftarrow \mathcal{L}_{\text{state}} + \gamma\mathcal{L}_{\text{comm}} + \nu\mathcal{L}_{\text{cost}} + \zeta\mathcal{L}_{\max} + \lambda\mathcal{L}_{\text{protect}}$
      $L_{\text{batch}} \leftarrow L_{\text{batch}} + \mathcal{L}_i$; append C_{episode} to cost_hist; update baseline $b \leftarrow \mathrm{EMA}(b, R)$
    end for
    $L_{\text{batch}} \leftarrow L_{\text{batch}}/B$; update θ by one optimizer step using $\nabla_\theta L_{\text{batch}}$
  end for
end for
return θ

cosine similarity. For a sender message $M_{i,t}$, we compute

$$\varphi_{\text{sem}}(i, t) = 1 - \mathrm{sim}\Big(\mathrm{Enc}(M_{i,t}), \mathrm{Enc}([q, \mathbf{c}_t])\Big). \tag{22}$$

D.3 Protection Loss

A protection loss function is used to regularize the communication graph during training by reducing anomalous communication routing.

$$\mathcal{L}_{\text{protect}} = \sum_{t=0}^{T_{\max}-1} \sum_{(u,v)\in E_\tau} \Big(\big(1 - \text{trust}(u, t)\big) \cdot ||\mathbf{m}(u \to v, t)||^2\Big). \tag{23}$$

E Supplementary Results

The performance is measured using accuracy or pass@1, both of which are reported in percentage (%). All experimental results of the methods evaluated on the benchmarks are averaged over three runs. The parameter sensitivity analysis experiments reflect the variance in the experimental results.

E.1 Backbone sensitivity on GAIA

This section provides Table 6 of model transferability analysis for NeuralFSM from Section 5.2.

E.2 Additional robustness experiment

As shown in Table 7, the results on MBPP closely mirror the robustness trends observed in the main text. The protection layer attenuates suspicious

Algorithm 2 Task-Adaptive FSM Execution

Require: Problem q; trained parameters θ; master FSM $= (\mathcal{S}, \Sigma, \mathcal{A}, \delta, \mathcal{L}, s_0, F)$; horizon $T_{\max}$; receiver budget k

Ensure: Answer $\hat{y}$ and execution log

$\mathbf{q} \leftarrow \mathrm{Enc}(q)$; $s_t \leftarrow s_0$; $\mathbf{c}_t \leftarrow \emptyset$; log $\leftarrow [\,]$
for $t = 0$ to $T_{\max} - 1$ **do**
  $(P_\theta^{\text{trans}}(\tau), \mathbf{w}_t(\tau), \text{aux}) \leftarrow \mathrm{Controller}_\theta(s_t, \mathbf{c}_t, \mathbf{q}_{\text{emb}})$
  Sampling choose s_{t+1} from $P_\theta^{\text{trans}}(\tau)$; $\dot{a}_{t+1} \leftarrow \mathrm{responsible}(s_{t+1})$
  $o_{t+1} \leftarrow \dot{a}_{t+1}.\mathrm{execute}(q, \mathbf{c}_t)$
  $\{a_r\}_t \leftarrow \mathrm{Sample}_k(\mathbf{w}_t(\tau), k)$
  $\mathbf{c}_{t+1} \leftarrow \mathrm{UpdateContext}(\mathbf{c}_t, \{a_r\}_t, o_{t+1})$; append $(t, s_{t+1}, \dot{a}_{t+1}, \{a_r\}_t)$ to log
  $s_t \leftarrow s_{t+1}$
end for
$\hat{y} \leftarrow \mathrm{Extract}(o_t)$; **return** $(\hat{y}, \log)$

Algorithm 3 Protected Controller Forward Pass

Require: $s_t, \mathbf{c}_t, \mathbf{q}$; induced edges E_τ and messages; parameters θ

Ensure: $(P_\theta^{\text{trans}}(\tau), \mathbf{w}_t(\tau), \text{aux})$

Build induced graph $G = (V, E_\tau)$
Compute priority $\pi(i)$ from centrality on G (Eq. (15))
Compute anomalies $\varphi_{\text{freq}}(i,t), \varphi_{\text{sem}}(i,t)$ and fuse to $\varphi(i,t)$ (Eq. (16))
Compute trust$(i,t) = \sigma(\pi(i)) \cdot (1 - \varphi(i,t))$
Run base controller to compute raw messages $\mathbf{m}(u \to v, t)$ on E_τ
Apply trust-aware attenuation: $\widetilde{\mathbf{m}}(u \to v, t) = \text{trust}(u,t) \cdot \mathbf{m}(u \to v, t)$
Read out predictions from attenuated messages to obtain $(P_\theta^{\text{trans}}(\tau), \mathbf{w}_t(\tau))$; set aux $\leftarrow \{\pi, \varphi, \text{trust}\}$
return $(P_\theta^{\text{trans}}(\tau), \mathbf{w}_t(\tau), \text{aux})$

messages before they enter the shared context, leading to additional robustness improvements for NeuralFSM. Under attack, all baseline methods experience pronounced relative performance degradations: SC decreases by 33.67%, DyLAN by 25.15%, MacNet by 27.58%, MaAS by 19.95%, and MasHost by 20.93%. In contrast, NeuralFSM exhibits a markedly smaller relative drop of 11.05%, enabling the protection layer further improves pass@1 by 10.58% compared to the unprotected attacked setting and reducing the remaining gap in clean performance to only 1.64%.

F Prompt repository

This appendix enumerates the prompts used to generate a master FSM for each domain. In our implementation, the FSM generation process invokes a LLM twice: (i) Step 1 designs the workflow states, and (ii) Step 2 designs agents aligned with these states. The domain-specific FSM is then initialized. Notably, no directed edges are established in the initialized FSM, meaning that no message passing rules are defined. Both prompts are parameterized by a domain template.

Notation	Definition
$q \in \Sigma$	Input domain problem
$y, \hat{y}$	Ground-truth and prediction
$\mathcal{A} = \{a_1, \ldots, a_{m-1}\}$	Agent set
$\mathcal{S} = \{s_1, \ldots, s_{n-1}\}$	FSM state set
$\text{FSM} = (\mathcal{S}, \Sigma, \mathcal{A}, \delta, \mathcal{L}, s_0, \mathcal{F})$	FSM tuple
$s_0, \mathcal{F}$	Initial state; Final state set
$t, T_{\max}$	Time step index; Rollout horizon
$\mathbf{q} = \text{Enc}(q)$	Problem embedding
$\mathbf{c}_t$	Accumulated execution context
π_θ	Temporal coordination controller
θ	Learnable parameters of the temporal controller
$\mathcal{D}$	Training dataset
η_{lr}	Learning rate of the optimizer
$E_{\max}, B$	Max training epochs; Batch size
b	Reward baseline (exponential moving average)
$P_\theta^{\text{trans}}(\tau)$	Transition distribution
$\mathbf{w}_t(\tau)$	Routing score vector over agents
$\{a_r\}_t, k$	Receiver set; Receiver budget
$\mathbf{m}_v(t) \in \mathbb{R}^{d_m}$	Memory state of node v; d_m is memory dimension
$\mathbf{h}_v(t)$	Node representation of node v at time t
$\mathbf{e}_{uv}$	Edge features for interaction (u, v)
$\Delta t, \Delta t_{uv}$	Time difference and last-interaction time gap
$\psi(\Delta t)$	Temporal encoding function for time gaps
C_{episode}	Per-episode FSM execution cost
$\mathcal{L}_{\text{total}}$	Integrated training objective
$\alpha, \beta, \gamma, \nu, \zeta, \lambda$	Loss weights
$\pi(i)$	Protection priority score
$\varphi(i, t)$	Fused anomaly score
$\varphi_{\text{freq}}(i, t), \varphi_{\text{sem}}(i, t)$	Frequency/semantic anomaly indicators
$\sigma(\cdot)$	Sigmoid function
$\omega_{\text{freq}}, \omega_{\text{sem}}$	Anomaly fusion weights
$\text{trust}(i, t)$	Trust score derived from priority and anomaly
$w_{\text{BC}}, w_{\text{PR}}$	Centrality coefficients
d_{PR}	PageRank damping factor
η	Attack strength
ρ	Attacked-agent ratio
ϵ	Small constant for numerical stability

Table 4: Notation and Definitions

Domain	Dataset	#Train	#Test	Metric
Code Generation	HumanEval	164	–	pass@1
	MBPP	699	175	pass@1
Math Reasoning	GSM8K	640	160	Accuracy
	MATH	640	160	Accuracy
Knowledge QA	GPQA	198	–	Accuracy
Multi-hop QA	HotpotQA	720	180	EM
	2WikiMultiHopQA	720	180	EM
	MuSiQue	720	180	EM
Embodied	ALFWorld	219	55	Success rate
Tool-use	GAIA (L1/L2/L3)	165	–	Accuracy

Table 5: Dataset statistics for the benchmarks used in this paper.

GAIA	**Level 1**		**Level 2**		**Level 3**		**Avg.**	
NeuralFSM	Perf.	Token.	Perf.	Token.	Perf.	Token.	Perf.	Token.
- GPT-4.1-nano	13.21	0.25	6.98	0.39	0.00	0.16	6.73	0.27
- GPT-4o-mini	7.55	0.24	5.00	0.24	0.00	0.13	4.18	0.20
- GPT-5-nano	18.92	0.45	11.63	0.77	7.69	0.21	12.75	0.47
- Grok-3-mini	28.30	0.76	12.79	1.39	3.85	0.52	14.98	0.89

Table 6: Performance and token consumption of NeuralFSM with different LLM backbones on the GAIA dataset (Token units: 1M tokens).

Method	**No Attack**	**Under Attack**	**Protected (Ours)**
SC	80.2	53.2	–
DyLAN	83.9	62.8	–
MacNet	74.3	53.8	–
MaAS	88.2	70.6	–
MasHost	87.9	69.5	–
NeuralFSM (Ours)	91.4	81.3	89.9

Table 7: Robustness of NeuralFSM and other baselines under attack on MBPP: pass@1 in no attack, under attack, and with our proposed protection layer enabled.

GSM8K template

```
[task_description]
Grade School Math Word Problem Solving Task
Characteristics:
- Problems require multi-step reasoning
- Requires precise numerical calculations
- Answer is a specific numerical value
- Need to verify calculation logic

Requirements:
1. States should decompose problem-solving process
2. Each state needs clear completion criteria
3. Agents need mathematical reasoning and calculation capabilities
4. Support iterative improvement and error correction

FSM Design Requirements:
1. States should follow a clear progression: understanding -> planning -> calculation ->
verification -> submission
2. Include explicit verification states to catch calculation errors
3. Support iterative refinement loops for error correction
4. Keep states fine-grained (6-10) to enable TGN optimization

[agent_design_hints]
Agent Design Recommendations:
1. Problem_Parser (non-calculation): extracts key information, quantities, and relationships
from problem text
2. Strategy_Planner (non-calculation): formulates solution approach and calculation sequence
3. Calculator (calculation agent): performs numerical computations step-by-step
4. Verifier (non-calculation): checks answer reasonableness and identifies potential errors
5. Refiner (calculation agent): corrects identified errors and updates calculations
6. Finalizer (final state): formats and submits the final numerical answer

[state_granularity_guide]
State Decomposition Guide (6-10 states):
1. Initial Understanding Phase (1-2 states):
- Problem text parsing
- Key information extraction

2. Solution Planning Phase (1-2 states):
- Solution strategy formulation
- Calculation step planning

3. Calculation Execution Phase (2-3 states):
- Intermediate value calculation
- Final result calculation
- Unit conversion handling

4. Verification Phase (1-2 states):
- Answer reasonableness check
- Error correction iteration

5. Final Submission Phase (1 state):
- Answer formatting
- Result submission
```

GPQA template

```
[task_description]
GPQA (Graduate-Level Multiple-Choice QA) Task
Characteristics:
- High-difficulty, graduate-level science questions (often multi-step reasoning)
- Multiple-choice format (A/B/C/D)
- Requires careful option elimination and consistency checks

Global Output Contract (CRITICAL):
- FINAL answer MUST be a SINGLE LETTER: A / B / C / D
- NO explanations, NO extra text, NO markdown
- If the prompt contains both a)/b)/c)/d) and a mapping to A/B/C/D, FOLLOW the final A/B/C/D
mapping.
- If unsure, still choose the best option letter based on reasoning

FSM Design Requirements:
1. States should support multi-step reasoning for graduate-level questions
2. Include thorough option-by-option analysis to identify contradictions
3. Incorporate cross-checking and consistency verification states
4. Ensure final state outputs ONLY the option letter (A/B/C/D)

[agent_design_hints]
Agent Design Recommendations:
1. Question_Parser: extracts key constraints, units, and what is being asked
2. Concept_Activator: recalls relevant graduate-level domain concepts and formulas
3. Option_Analyzer: evaluates each option (A/B/C/D) individually, identifies contradictions and
distractors
4. Consistency_Checker: performs sanity checks, dimensional analysis, and cross-verification
with known facts
5. Decision_Maker (final state): outputs ONLY the final option letter (A/B/C/D), no explanations

[state_granularity_guide]
State Decomposition Guide (7-10 states):
1. Question Parsing (1 state):
- Extract what is being asked, key constraints, units

2. Concept Activation (1-2 states):
- Recall relevant domain concepts / formulas

3. Option-by-Option Analysis (2-4 states):
- Evaluate each option, identify contradictions/distractors

4. Cross-check & Elimination (1-2 states):
- Sanity check, dimensional check, consistency with known facts

5. Final Decision (1 state):
- Output ONLY the final option letter (A/B/C/D)
```

GAIA template

```
[task_description]
GAIA (General AI Assistant) Benchmark Task
Characteristics:
- Mixed tool-using / multimodal / document-grounded reasoning
- Questions may reference external files (e.g., .xlsx/.pdf/.png/.docx/.csv/.txt) via a file_path
- Requires step-by-step planning, extraction from the given file when present, and final short answers

Global Output Contract:
- Provide ONLY the final answer (short), no extra commentary unless explicitly requested.
- If a numerical answer is required, return the number in the requested format.
- If the task requires reading an attached file, treat the attachment as authoritative evidence.

FSM Design Requirements:
1. Separate planning from evidence extraction (especially for file-backed questions).
2. Include an explicit verification / sanity-check step.
3. Keep states compact (6-10) and robust to cases with/without file attachments.

[agent_design_hints]
Agent Design Recommendations:
1. Task_Parser: parses question, constraints, and required output format; detects file_path if present
2. Planner: decides solution approach (web lookup vs file reading vs pure reasoning); defines intermediate quantities
3. Evidence_Extractor: reads/interprets files (tables/images/PDFs/docs) or gathers facts from tools/reasoning
4. Reasoner/Computer: performs calculations, logic, or multi-step synthesis
5. Verifier: cross-checks units, constraints, and consistency with evidence
6. Finalizer (final state): emits final answer strictly in requested format (short, no extra commentary)

[state_granularity_guide]
State Decomposition Guide (6-10 states):
1. Task Parsing (1 state):
- Parse question, constraints, required output format
- Detect whether an attachment is required (file_path exists)

2. Plan & Tool Selection (1 state):
- Decide how to solve: web lookup vs file reading vs pure reasoning
- Define intermediate quantities to compute

3. Evidence Acquisition (1-2 states):
- If file_path exists: read/interpret the file (table/image/pdf/doc)
- Otherwise: gather key facts from internal reasoning or allowed tools

4. Reasoning / Computation (1-3 states):
- Perform calculations, logic, or multi-step synthesis

5. Verification (1 state):
- Cross-check units, constraints, and consistency with evidence

6. Final Answer (1 state):
- Emit final answer strictly in requested format
```

HumanEval template

```
[task_description]
Python Code Generation Task (HumanEval)
Characteristics:
- Implement a single Python function based on signature and docstring
- Must pass hidden test suite
- Primary objective: correct, executable function

Global Output Contract (CRITICAL):
- FINAL answer MUST be single, syntactically valid Python FUNCTION
- No explanations, no markdown, no extra code
- Function name/signature MUST match given problem
- Use proper newlines/indentation

FSM Design Requirements:
1. Most states (except first understanding and final submission) SHOULD be code-producing
2. Almost every non-final state should output FULL updated function
3. Minimize purely analytical states
4. Model iterative code-writing/refinement process

[agent_design_hints]
Agent Design Recommendations (code-centric):
1. Problem Reader (non-coding): summarize requirements/edge cases; MUST NOT output executable
code.
2. Code Writer (coding agent): output ONLY the function definition; in every non-final coding
state, output a COMPLETE function definition.
3. Code Refiner / Debugger (coding agent): ALWAYS outputs a FULL updated function definition.
4. Test Reasoner (non-coding): outputs ONLY natural-language analysis/instructions; NEVER emits
code.
5. Finalizer (coding agent, final state): MUST output the FINAL Python function definition ONLY
(code-only).

[state_granularity_guide]
State Decomposition Guide (code-heavy):
1. Initial Understanding (1 state, non-coding): Read signature + docstring; summarize
requirements/edge cases.
2. First Implementation Draft (1 state, coding): Code Writer produces FIRST full implementation.
3. Guided Refinement Loops (3-5 states, mostly coding): alternate Test Reasoner and Code Refiner.
4. Pre-Submission Check (optional, 1 state, non-coding).
5. Final Submission (1 state, coding): Finalizer outputs final polished function.
Total states: 6-9, majority code-producing states
```

MBPP template

```
[task_description]
Basic Python Programming Task (MBPP)
Characteristics:
- Small self-contained Python tasks
- Function signature + assertion-based test cases
- Goal: implement function passing all tests

Global Output Contract (CRITICAL):
- FINAL answer MUST be single, syntactically valid Python FUNCTION
- Function name/parameters MUST match signature exactly
- Function body MUST be executable; no input()/print()
- Output pure code (no explanations, no markdown)
- FORBID multiple statements on one line (e.g., def f(): a=0; b=1)

FSM Design Requirements:
1. Most states SHOULD be code-writing/refinement states
2. FSM represents iterative coding: draft -> refine -> fix -> finalize
3. Avoid long non-coding sequences
4. Each transition brings solution closer to passing tests

[agent_design_hints]
Agent Design Recommendations (code-focused):
1. CRITICAL - FUNCTION SIGNATURE IS PROVIDED IN THE PROMPT: use the EXACT function name/signature;
NEVER rename to solve/solution.
2.  Roles:  Requirement_Analyst (non-coding), Code_Writer (coding), Test_Reasoner (non-coding),
Code_Refiner (coding), Finalizer (coding, final state).

[state_granularity_guide]
State Decomposition Guide (code-centric, 5-8 states):
1. Understand_Problem (1 state, non-coding)
2. Initial_Implementation (1-2 states, coding)
3. Test_Design_and_Bug_Hunting (1-2 states, non-coding)
4. Refinement_Iterations (2-3 states, coding)
5. Final_Submission (1 state, coding)
Total states: 5-8, majority dedicated to executable code
```

MATH template

```
[task_description]
Competition Mathematics Task
Characteristics:
- Advanced mathematical reasoning
- Requires proof and derivation
- Symbolic manipulation and theorem application
- Multiple solution approaches possible

Requirements:
1. Deep problem analysis and strategy selection
2. Rigorous derivation and proof construction
3. Symbolic manipulation capabilities
4. Solution verification and alternative approaches

FSM Design Requirements:
1. States should support deep mathematical reasoning: analysis -> strategy -> derivation ->
verification
2. Include multiple derivation states for step-by-step symbolic manipulation
3. Support alternative solution approaches when initial strategy fails
4. Include verification states to check solution validity

[agent_design_hints]
Agent Design Recommendations:
1. Problem_Analyzer: identifies problem type and key concepts; extracts given conditions
2. Strategy_Selector: chooses solution approach; identifies applicable theorems and techniques
3. Deriver: performs step-by-step derivation, symbolic manipulation, and intermediate
calculations
4. Verifier: checks solution validity; suggests alternative approaches if needed
5. Alternative_Solver: attempts alternative solution approaches when initial strategy fails
6. Finalizer (final state): formats and submits the final answer

[state_granularity_guide]
State Decomposition Guide:
1. Problem Analysis Phase (1-2 states): identify problem type and key concepts; extract given
conditions
2. Strategy Selection Phase (1-2 states): choose solution approach; identify applicable theorems
3. Solution Derivation Phase (3-5 states): step-by-step derivation; symbolic manipulation;
intermediate results
4. Verification Phase (1-2 states): check validity; try alternative approaches
5. Final Submission Phase (1 state): format final answer
Total states: 8-12
```

ALFWorld template

```
[task_description (condensed excerpt)]
Embodied AI Interactive Tasks (ALFWorld)
Characteristics:
- Text-based environment simulation in household scenarios
- Sequential action planning with state changes
- Goal-oriented navigation and object manipulation
- Typical structure: locate -> pick up -> (transform) -> place/examine

Evaluation:
- Subgoals are regex patterns matched against action feedback
- All subgoals must be satisfied sequentially to complete the task

Requirements (excerpt):
1. Parse goal to identify target object(s), required transformation (heat/cool/clean/none), and
destination
2. Plan an action sequence based on task type
3. Execute actions and parse feedback for confirmation; handle failures with retries
4. Output the final action sequence as the answer

FSM Design Requirements:
1.  States should follow embodied AI workflow: goal parsing -> exploration -> acquisition ->
transformation (optional) -> placement -> verification
2. Include state for parsing action feedback and matching against subgoal regex patterns
3. Support optional transformation state for heat/cool/clean operations
4. Final state must output complete action sequence

[agent_design_hints]
Agent Design Recommendations:
1.      Goal_Parser:    parses   goal   to   extract   target   object(s),   transformation   type
(heat/cool/clean/none), and destination
2. Explorer: navigates environment to locate target objects; opens receptacles if needed
3. Object_Acquirer: executes "take [object] from [receptacle]" and verifies pickup feedback
4.  Transformer (optional):  navigates to appliance (microwave/stove/fridge/sink) and executes
transformation
5. Placer: navigates to destination and executes "put [object] in/on [receptacle]"
6. Subgoal_Verifier: matches action feedback against subgoal regex patterns
7. Action_Sequencer (final state): compiles and submits complete action sequence

[state_granularity_guide]
Recommended FSM Structure:
1. Goal_Parsing_and_Planning [INITIAL]
2. Environment_Exploration
3. Object_Acquisition
4. State_Transformation [OPTIONAL]
5. Object_Placement
6. Subgoal_Verification
7. Final_Action_Sequence_Submission [FINAL]
Total states: 6-7
```

HotpotQA template

```
[task_description]
Multi-hop Question Answering Task
Characteristics:
- Requires reasoning across multiple documents
- Need to identify and connect supporting facts
- Questions involve bridge or comparison reasoning
- Long context with multiple document paragraphs

Requirements:
1. States should decompose multi-hop reasoning process
2. Agents need document retrieval and cross-document reasoning
3. Support extraction and verification of supporting facts
4. Handle both bridge (A -> B -> C) and comparison (A vs B) questions

FSM Design Requirements:
1. States should support multi-hop reasoning: question analysis -> document retrieval ->
hop-by-hop connection -> synthesis
2. Include dedicated states for extracting relevant passages and identifying key entities
3. Support both bridge (A->B->C) and comparison (A vs B) question types
4. Include verification states to validate supporting facts

[agent_design_hints]
Agent Design Recommendations:
1. Question_Analyzer: parses question type (bridge/comparison); identifies required information
2. Document_Retriever: extracts relevant passages from multiple documents; identifies key entities
3. Multi_hop_Reasoner: connects facts hop-by-hop (first hop -> intermediate hops -> final hop)
4. Evidence_Synthesizer: combines evidence from multiple hops; verifies supporting facts
5. Finalizer (final state): formats and submits the final answer

[state_granularity_guide]
State Decomposition Guide:
1. Question Analysis Phase (1-2 states): parse question type (bridge/comparison); identify
required information
2. Document Retrieval Phase (2-3 states): extract relevant passages; identify key entities
3. Multi-hop Reasoning Phase (2-4 states): connect facts hop-by-hop
4. Answer Synthesis Phase (1-2 states): combine evidence; verify supporting facts
5. Final Submission Phase (1 state): format and submit answer
Total states: 7-11
```