WHICH LLM MULTI-AGENT PROTOCOL TO CHOOSE?

Anonymous authors
Paper under double-blind review

ABSTRACT

As large-scale multi-agent systems evolve, the communication protocol layer has become a critical, yet understudied, component affecting system performance and reliability. Despite a range of protocols, such as JSON-RPC, A2A, ANP, and ACP, protocol selection remains ad hoc. To address this, we introduce *ProtocolBench*, a benchmark designed to evaluate agent communication protocols across task utility, communication overhead, system performance, and resilience under failure. *ProtocolBench* uses a three-layer architecture with protocol adapters for fair comparison, diverse scenarios (e.g., document aggregation, collaborative coding), and detailed telemetry. Our results show protocol choice can impact task completion time by up to *36%*, communication overhead by *3.5 seconds*, and resilience with statistically observable differences. We also propose ProtocolRouter, a learnable protocol routing system that dynamically selects protocols based on runtime conditions, improving performance by up to *18%* compared to individual protocols. Our findings highlight that hybrid protocol deployments outperform homogeneous ones by approximately *6.6%*, with negligible protocol translation overhead. We release ProtocolBench as an open-source framework to standardize protocol evaluation and improve multi-agent system reliability at scale.

1 INTRODUCTION

Large Language Model (LLM) based multi-agent systems evolve from research prototypes to production deployments. These systems rely on effective protocols to coordinate communications among agents, from lightweight JSON-RPC to sophisticated frameworks like MCP for standardized tool invocation (Anthropic, 2024), A2A for enterprise-scale agent communication (Google Cloud, 2025), IBM's ACP for cross-framework collaboration (IBM BeeAI, 2025), and IoA for dynamic agent discovery and orchestration (Chen et al., 2024). Despite this proliferation of protocols, there is a lack of fundamental understanding of their trade-offs. While existing benchmarks (Zhu et al., 2025; Hyun et al., 2025) typically assume fixed protocols and focus on task-level metrics, and surveys (Yang et al., 2025; Ehtesham et al., 2025) highlight the need for systematic evaluation across dimensions such as efficiency, scalability, and security, practitioners still lack systematic guidance for protocol selection and environments to compare different protocols' roles in multi-agent systems. In this paper, we aim to answer the key research question: can we understand the tradeoffs among multi-agent protocols, and further help users systematically choose optimal protocols?

The technical challenges in building such a benchmark are substantial. Multi-agent Protocols exhibit tightly coupled dimensions where task accuracy, end-to-end latency, and communication costs interact in complex ways. Ensuring fair comparison requires isolating protocol-specific factors from hardware, model, and scheduling variations. The combinatorial explosion of protocol-topology-scale configurations, combined with dynamic events like failures and scaling, necessitates precise, low-overhead monitoring and analysis infrastructure. Previous approaches that only measure final task accuracy miss critical insights about communication efficiency and system stability. Building a router over protocols is highly nontrivial. Real-world constraints like budget limits, latency requirements, and security policies often conflict, requiring dynamic trade-offs that static protocol selection cannot address. Make protocol interoperable is challenging, where agents using different protocols must collaborate, demands a unified framework for translation and routing.

To address these gaps, we first introduce *ProtocolBench*, the first benchmark explicitly designed to evaluate communication protocols in multi-agent systems. ProtocolBench provides comprehensive evaluation across multiple dimensions including task utility, communication overhead, system performance, and resilience. Through a three-layer architecture—Protocol Adapter Layer for uni-

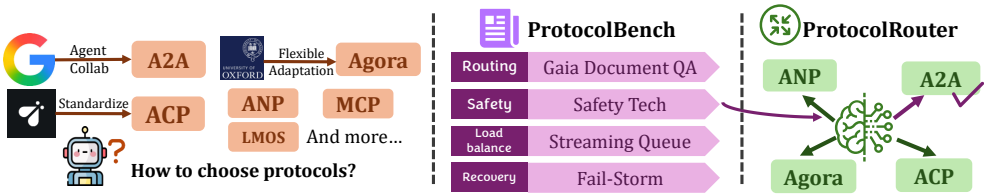

Figure 1: **Overview of ProtocolBench and ProtocolRouter**. To understand the tradeoff across existing LLM multi-agent protocols, we first design ProtocolBench that covers four core evaluation dimensions, then propose ProtocolRouter to help users select the optimal protocol.

fied protocol interfaces, Scenario Harness with five representative workloads, and Metric and Logger Stack for fine-grained telemetry—the benchmark enables fair comparison across heterogeneous protocol implementations while addressing key evaluation challenges such as protocol isolation and dynamic event handling.

Our evaluation reveals several surprising findings. In gaia document QA tasks, A2A's distinguished long-context communication directly translates into a 25.8% higher task success rate and 7.7% higher quality compared to the next-best protocol. For streaming workloads, A2A's offset-replay mechanism maintains 99.5% reliability under 30% node failures while JSON-based protocols drop to 67%.

Building on insights from ProtocolBench, we further present *ProtocolRouter*, a learnable protocol routing system that dynamically selects and composes protocols based on scenario characteristics and runtime conditions. To evaluate protocol routing capabilities, we construct an annotated dataset derived from ProtocolBench results, capturing protocol performance patterns across diverse scenarios. ProtocolRouter leverages these insights to make informed routing decisions, demonstrating that intelligent protocol selection can outperform any single protocol choice. Furthermore, it enables cross-protocol communication, facilitating seamless integration of heterogeneous agents and future extensibility of multi-agent networks. By capturing scenario-dependent performance signals, ProtocolRouter adapts its routing strategy to evolving conditions, ensuring robustness beyond what fixed rules can offer. The learned ProtocolRouter router consistently outperforms static selection by 2-18% across composite metrics, with particularly strong gains in scenarios with dynamic workload changes or partial failures.

In sum, this paper makes three main contributions. First, we provide the first systematic characterization of existing agent communication protocols across multiple dimensions and scenarios. Second, we release ProtocolBench as an open-source evaluation dataset, evaluation harnesses, and analysis tools to enable reproducible protocol research. Third, we demonstrate through ProtocolRouter that dynamic protocol selection and composition can unlock performance gains that exceed the capabilities of any individual protocol.

2 Related Work

Benchmarks and Multi-agent frameworks. LangChain provides modular pipelines (LangChain, 2024a), LangGraph adds graph-based control flow (LangChain, 2024b), and CrewAI simplifies role-based collaboration (Moura, 2024). Microsoft's AutoGen enables conversational multi-agent systems (Microsoft Research, 2024), while OpenAI's Swarm offers lightweight coordination (OpenAI, 2024). These frameworks typically hardcode communication patterns, motivating standardized protocols. Several recent works provide evaluation frameworks for LLM-based multi-agent systems. (Zhu et al., 2025) introduce *MultiAgentBench*, covering collaborative coding, gaming and research tasks. (Hyun et al., 2025) propose CREW-Wildfire for wildfire response with heterogeneous agents. (Liu et al., 2024) present AgentBench, evaluating LLM-as-Agent across eight environments. While these benchmarks offer rich scenarios, they evaluate agents under fixed communication mechanisms and do not compare protocol designs. Our work isolates the communication layer and provides protocol-agnostic evaluation.

Agent protocols and communication mechanisms. Recent surveys provide theoretical foundations for understanding multi-agent communication. (Tran et al., 2025) survey collaboration mechanisms, categorizing cooperation, competition and coordination strategies, while (Yang et al., 2025) propose a taxonomy distinguishing context-oriented from inter-agent protocols. (Ehtesham et al., 2025)

Protocol	Task Utility	Communication Overhead	System Performance	Optimal Scenarios
A2A	Enterprise coordination	Medium *(structured)*	Consistent throughput	Large-scale mission-critical
ACP	Framework integration	Low-medium *(async)*	Framework-dependent	Cross-platform collaboration
ANP	Semantic routing	Low *(targeted)*	Variable latency	Document aggregation
Agora	Decentralized workflows	Low *(P2P)*	Network-dependent	Dynamic networks

Table 1: Summary of investigated LLM multi-agent protocols.

compare existing protocols, analyzing their interaction modes and security models. The ecosystem features diverse protocol implementations (Ehtesham et al., 2025) : MCP standardizes tool invocation (Anthropic, 2024), A2A enables agent communication across enterprise platforms with 50+ industry partners (Google Cloud, 2025), IBM's ACP provides open standards for cross-framework collaboration (IBM BeeAI, 2025), the Internet of Agents (IoA) enables dynamic discovery and orchestration among heterogeneous agents (Chen et al., 2024), and Agora establishes a decentralized communication layer that emphasizes interoperability and governance across agent networks (Marro et al., 2024). While these surveys motivate systematic empirical evaluation of protocols, we provide the first benchmark with adapters for representative protocols to evaluate them systematically.

3 ProtocolBench: A Systematic Evaluation of Agent Protocols

To asses the multi-agent protocols along different dimensions, we implement **ProtocolBench** that cover four different multi-agent scenarios along with different metrics to evaluate the performance of different multi-agent protocols.

3.1 ProtocolBench scenarios

As shown in Fig. 2, we design and implement four representative scenarios, each stressing a different protocol property.

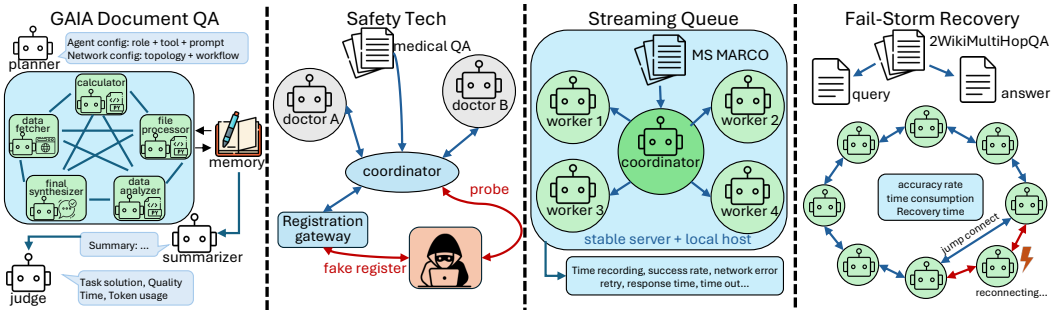

Figure 2: Illustration of four different multi-agent scenarios evaluated in this work.

GAIA Document Question-Answering tests *hierarchical information aggregation* of the multi-agent system. In this scenarios, a Planner module generates JSON configurations for agents, defining roles, tools, prompts, network topology and workflow of the multi-agent system. Following the GAIA framework (Mialon et al., 2023), agents are connected with each other, cooperate with sandboxed tools and log interactions in a memory pool. The shared memory is then summarized and judged by LLM.

Safety Tech assesses the *privacy-preserving* capability of protocols under a medical Q&A scenario. In this setup, a network with a registration gateway, coordinator, and two LLM doctors processes 10 cases from ChatDoctor-HealthCareMagic 100k (Li et al., 2023) with synthetic identity information. Meanwhile, comprehensive security probes are systematically injected into the protocol stack to test privacy protection mechanisms against various attack vectors.

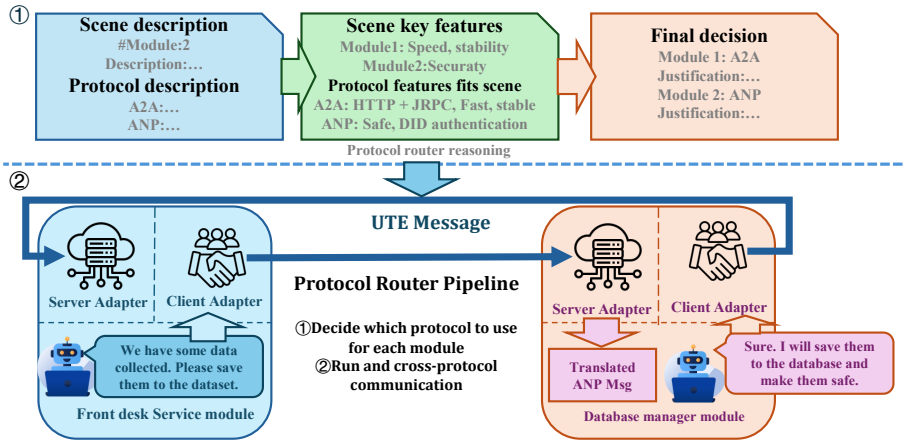

Figure 3: **ProtocolRouter overview**. **(top)** A scenario-aware selector (Appendix C.10) outputs a structured plan with one protocol per module. **(bottom)** The runtime instantiates adapters and connects modules; cross-protocol links use stateless encode/decode bridges without shared session state.

Streaming Queue evaluates *high-throughput API services*. In this scenario, one coordinator and four workers process 1000 MS MARCO entries (Bajaj et al., 2018), with the coordinator balancing load for uniform distribution among all worker agents.

Fail-Storm Recovery tests resilience under node failures in Shard QA. In this scenarios, query and answers from 2WikiMultihopQA (Ho et al., 2020) and shuffled and distributed among 8 agents connected in a loop. Agents need to find the match for each query and answer while 3 of 8 agents lose connection every 2 minuts.

3.2 PROTOCOLBENCH DESIGN AND EVALUATION

As summarized in Table 2, we also design different evaluation metric for different scenarios.

Scenario	Description	Key Metrics	Key Feature
Gaia Document QA	GAIA document task analysis	Success rate, Collaboration quality	Hierarchical routing
Safety Tech	Medical Q&A with security probes	Security Score, Probe Block Rate	Security probing
Streaming Queue	High-throughput request handling	P95 latency, Drop rate	Load balancing
Fail-Storm Recovery	Resilience under node failures	Recovery time, Success rate drop	Fault detection/recovery

Table 2: **Overview of ProtocolBench scenarios with key metrics and features**. Each scenario highlights different protocol trade-offs while being evaluated with consistent evaluation metrics.

To ensure fair and comprehensive evaluation across all scenarios and different multi-agent protocols, we employ a three-layer architecture for the experimental design: a protocol adapter layer as unified interfaces, a scenarios harness for consistent workload execution, and a metric and logger stack for fine-grained telemetry. This design isolates protocol-specific effects while controlling for confounding factors such as hardware, LLM model, decoding parameters, prompts, and rate limits. Different scenarios are evaluated with different metrics. As listed in Table 2 and detailed in Appendix C, these metrics seek to capture the key performance feature of each scenario. For better statistics and more fair evaluation, we run each protocol-scenario configuration three time with distinct run IDs for reproducibility.

4 PROTOCOLROUTER: A TASK-DEPENDENT SELECTION OF PROTOCOLS

The diversity of LLM multi-agent protocols (e.g., A2A, ACP, ANP, Agora) creates a challenge: no single protocol excels across all scenarios, yet manual selection is inefficient and error-prone. To address this issue and optimize the protocol selection for different use case while supporting flexible protocol selection, we design and benchmark ProtocolRouter.

4.1 PROTOCOLROUTER DESIGN

The goal of ProtocolRouter is to act as a dynamic router that not only enables agents using different protocols to communicate, but also intelligently assigns protocols based on the specific scenarios. This approach will eliminate the need for manual configuration and enable adaptable and scalable multi-agent coordination.

Given a natural-language scenario description, ProtocolRouter assigns one protocol per module using a deterministic selection process based on the canonical feature model. The output is machine-verifiable JSON objects that bind modules to protocol adapters, with cross-protocol communication handled through stateless bridges.

ProtocolRouter evaluates each module's requirements against the four protocols' characteristics (transport model, security features, typical strengths and costs) and selects the best match. In the performance-aware mode, it additionally considers historical performance data from previous benchmarks to make more informed selections.

In addition, the communication between different protocols is implemented using the Protocol Adapter Layer in ProtocolBench, with stateless encode/decode bridges enabling interoperability. Each protocol has an adapter that standardizes communication, and ProtocolRouter uses bridges to translate messages between protocols, preserving semantics and security.(See Appendix E for more details about router prompt and cross router implementations.)

When protocol choices imply specific capabilities, ProtocolRouter automatically enables these features through each protocol's native mechanisms. For streaming workloads, protocols like A2A and ANP leverage their built-in streaming capabilities to handle continuous data flows efficiently. Long-running job state management is handled through protocol-specific persistence mechanisms—ACP uses simple state tracking, while AGORA employs distributed state management for complex multi-agent scenarios.

4.2 PROTOCOLROUTERBENCH: EXTENDING PROTOCOLBENCH TO EVALUATE MULTI-AGENT PROTOCOL ROUTERS

Furthermore, we benchmark the ProtocolRouter by extending ProtocolBench and compare the performance of ProtocolRouter with each individual protocol. This benchmark is carried out in two approaches. And more details can be found in Appendix C.3.

When there is no information about the performance of different protocols under any testing scenarios, the ProtocolRouter will only execute the automatic protocol selection based on the description of the given scenario and protocol features. This benchmark is tested on all four scenarios listed in the ProtocolBench.

When there is already known information about the protocol performance under different scenarios, the ProtocolRouter will make decision on protocol selection based on these benchmark results for individual protocols. For example, if the new testing scenario is within the four scenarios in ProtocolBench, ProtocolRouter will naturally choose the best performing one. Meanwhile, even if the new testing scenario is out-of-sample, the ProtocolRouter will make better decision conditioning on the description of each scenario and previous ProtocolBench results. This design make the ProtocolRouter extendable as it improves with additional benchmark information.

All evaluation scenarios in ProtocolRouterBench are human–LLM co-created. Human draft the task intent and constraints, LLMs expand auxiliary details, and humans finalize the prompt pack (scenario card + module graph). Ground truth (one-protocol-per-module) is human-annotated based on explicit requirements and protocol features. We set different difficulty (L1–L5, each level have different number of modules) by scale of scene and number of modules. Each difficulty have 12 scenes, with total 60 scenes and 180 modules. Full construction guidelines, annotation rubric, and examples are provided in Appendix C.3.

5 EXPERIMENTAL RESULTS AND ANALYSIS

5.1 EXPERIMENTAL OVERVIEW AND METHODOLOGY

We evaluate four agent communication protocols (ACP, A2A, ANP, AGORA) across the four ProtocolBench scenarios to assess their suitability for different deployment requirements. Our evalua-

Scenario	Protocol	Quality avg	Success avg
GAIA	ACP	2.27	5.25
	A2A	**2.51**	**9.29**
	ANP	2.14	7.28
	AGORA	2.33	6.27

(a) **GAIA**. Task-utility metrics (averages only).

Scenario	Protocol	Answer discovery (%)		Latency (s)		Recovery (s)
		Pre	Post	Pre	Post	
	ACP	14.76	13.64	4.38	4.19	8.05
	A2A	14.74	**14.57**	4.34	4.19	**8.00**
	ANP	14.88	12.94	4.34	4.18	**8.00**
	AGORA	**14.91**	12.12	**4.33**	**4.18**	**8.00**

(b) **Fail-Storm Recovery**. Pre-/post-failure answer discovery, steady-state latency, and recovery time (s). All measured times include a 2.00 s restart delay; see Appendix B.4.

Scenario	Protocol	Duration (min)	Mean (s)	Min (s)	Max (s)	Std. Dev. (s)
Streaming Queue	ACP	**40.28**	**9.66**	6.88	**14.24**	**1.08**
	A2A	40.45	9.70	6.94	15.13	1.13
	ANP	47.38	11.36	**0.24**	50.10	5.73
	AGORA	54.97	13.14	0.52	28.21	5.09

(c) **Streaming Queue**. End-to-end latency statistics (duration, mean, min, max, std.).

Scenario	Protocol	TLS/Transpor	Session Hijack	E2E Encryption	Tunnel Sniffing	Metadata Leakage
Safety Tech	ACP	×	✓	✓	×	✓
	A2A	×	✓	✓	×	✓
	ANP	✓	✓	✓	✓	✓
	AGORA	✓	✓	✓	✓	✓

(d) **Safety Tech**. Binary capability matrix; ✓ indicates presence and × indicates absence.

Table 3: Consolidated experimental results by scenario. Four panels correspond to Gaia Documented QA, Fail-Storm Recovery, Streaming Queue, and Safety Tech, each reported with scenario-appropriate metrics.

tion framework systematically examines ***task utility***, ***latency characteristics*** including tail behavior, ***failure resilience capabilities***, and ***security properties*** across the diverse multi-agent scenarios described in Section 3.

The experimental results are presented in Table 3, covering task utility metrics, failure recovery behavior with steady-state latency measurements, streaming queue latency statistics, and security capability matrices. All metrics follow the benchmarking methodology and controlled experimental design outlined in Section 3.2.

5.2 Overall Task Utility Performance

A2A emerges as the superior protocol for overall task utility across the ProtocolBench scenarios, achieving the highest average quality score of 2.51 and success rate of 9.29 (Table 3a). This performance advantage is particularly evident in the **Gaia Document QA** scenario, where A2A's hierarchical information aggregation capabilities and peer-to-peer coordination excel in distributed document analysis tasks.

Compared to ACP, A2A demonstrates a substantial 10.57% improvement in quality metrics and a remarkable 76.95% enhancement in success rate, establishing it as the most effective protocol for heterogeneous collaborative workloads. From a stability perspective, the run-to-run analysis reveals

that ACP exhibits the most consistent quality performance with minimal variance across three evaluation runs, while A2A shows greater variability in success rates, indicating potential sensitivity to environmental conditions or workload characteristics in complex multi-agent coordination scenarios.

5.3 Latency Performance and Tail Behavior

ACP demonstrates superior latency characteristics in the **Streaming Queue** scenario, achieving the lowest mean response time of 9,663ms with the smallest variance of 1,077ms and the most controlled maximum latency of 14,235ms (Table 3c). This consistent performance profile makes ACP particularly suitable for high-throughput API services where latency-critical applications demand strict tail latency requirements and uniform load distribution among worker agents.

A2A follows closely with competitive latency performance, exhibiting only a 0.36% increase in mean latency compared to ACP while maintaining reasonable tail behavior. In contrast, ANP and AGORA incur significant latency penalties of 17.60% and 35.93% respectively, accompanied by substantially higher variance and heavy-tail distributions that may impact application predictability in high-throughput scenarios processing large-scale datasets like MS MARCO entries.

5.4 Failure Recovery and Resilience

Under the **Fail-Storm Recovery** scenario testing resilience under node failures, A2A exhibits exceptional performance, maintaining 98.85% of its pre-failure answer discovery capability (14.57% vs. 14.74% pre-failure rate) as shown in Table 3b. This superior retention capability significantly outperforms other protocols in the challenging Shard QA environment where query-answer matching must continue despite systematic node failures: ACP retains 92.41%, ANP maintains 86.96%, and AGORA preserves 81.29% of pre-failure performance.

Recovery time analysis reveals relatively uniform behavior across all protocols, with recovery times clustering around 8.0 seconds when agents reconnect to the loop topology. ACP shows a marginal 46ms additional delay, which is negligible in practical deployment scenarios involving distributed multi-hop question answering with periodic connection losses.

5.5 Security Capability Analysis

The **Safety Tech** scenario assessment reveals a clear bifurcation between protocol families in privacy-preserving capabilities (Table 3d). ANP and AGORA provide comprehensive security coverage across all evaluated dimensions, including TLS transport security, session hijacking protection, end-to-end encryption, tunnel sniffing resistance, and metadata leakage prevention—critical features for medical Q&A scenarios handling sensitive patient information and defending against adversarial probing attempts.

In contrast, ACP and A2A offer partial security capabilities, lacking TLS transport layer protection and tunnel sniffing resistance while maintaining session hijacking protection, end-to-end encryption, and metadata leakage prevention. This security-performance trade-off represents a critical consideration for deployment scenarios where comprehensive privacy guarantees are mandatory, as ANP and AGORA's enhanced security comes at the cost of increased latency overhead documented in the previous subsections. The binary security matrix demonstrates that only ANP and AGORA can fully satisfy the stringent privacy requirements of healthcare applications with synthetic identity information protection and resistance to registration gateway attacks.

Takeaway

Experiments show that protocol choice materially impacts **Multi-Agent System** across **speed** (latency/throughput), **security** (E2E/TLS, leakage resistance), and **collaborative effectiveness** (quality/success). Scenario-aware, per-module composition can match or surpass the best single protocol.

Findings

(1) GAIA — A2A leads task utility; **(2) Streaming Queue** — ACP wins on mean/tails; **(3) Fail-Storm** — A2A retains best post-fault utility with near-fastest recovery; **(4) Safety Tech** — Only ANP/Agora provide full security coverage.

Split	Scenario Accuracy		Module Accuracy	
	Spec-only	Spec+Perf	Spec-only	Spec+Perf
Overall (60 scen / 180 mods)	0.535	**0.633**	0.712	**0.817**
L1 (12 scen / 12 mods)	**0.750**	0.667	0.750	0.667
L2 (12 scen / 24 mods)	0.500	**0.583**	0.708	**0.750**
L3 (12 scen / 36 mods)	**0.750**	0.750	**0.861**	0.889
L4 (12 scen / 48 mods)	0.500	**0.917**	0.771	**0.958**
L5 (12 scen / 60 mods)	0.100	**0.250**	0.540	**0.717**

Table 4: Router selection correctness: overall and by difficulty across spec-only and performance-aware conditions.

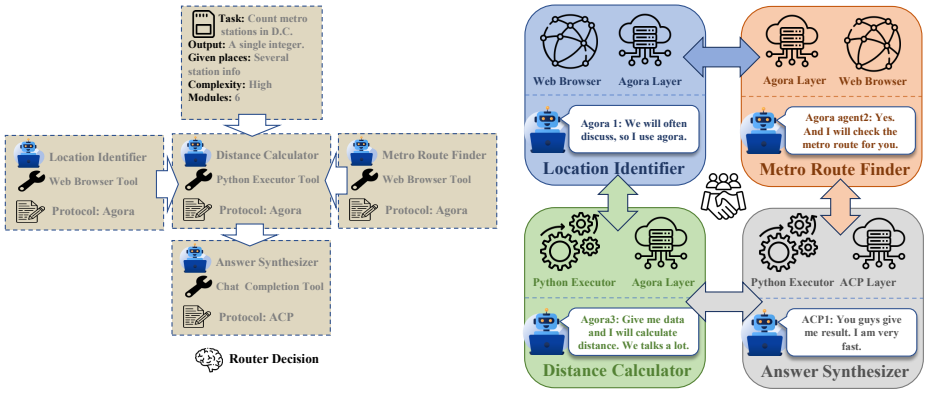

Figure 4: Case study: ProtocolRouter assigns protocols per module for the GAIA metro-counting task, enabling each module to run on its most suitable protocol (e.g., Agora for upstream discovery/compute and ACP for the final commit). This per-module assignment yields an overall accuracy that exceeds the single-protocol A2A baseline by 6.5%.

5.6 ProtocolRouterBench: Protocol Selection Evaluation

5.6.1 Protocol Selection Accuracy: Spec-only vs Performance-Aware

Setup and overall results. We evaluate ProtocolRouter under two conditions: a *spec-only* baseline (using protocol specifications only) and a *performance-aware* variant (spec+perf) that is augmented with scenario-agnostic performance priors. The benchmark covers 60 scenarios (180 modules) across five difficulty levels (L1–L5). The spec-only baseline attains 53.5% scenario accuracy and 71.2% module accuracy, with errors dominated by A2A↔ACP confusions. Adding performance priors lifts accuracy to 63.3% (scenario) and 81.7% (module), i.e., +18.3% and +14.7% respectively, and improves macro-F1 from 0.721 to 0.824 while preserving perfect recall for ANP and high precision/recall for Agora.

What the performance priors add and where they help most. The performance-aware condition injects latency percentiles, throughput characteristics, failure-recovery metrics, and security capabilities from Section 5 as *scenario-agnostic priors* that are used solely for quantitative tie-breaking in ambiguous choices (without exposing per-scenario numbers in rationales). The largest gains appear at higher difficulties: L4 scenario accuracy jumps from 50.0% to 91.7% (+83.4%), and L5 from 10.0% to 25.0% (+150%), primarily by reducing A2A↔ACP confusions while maintaining ANP separation and Agora robustness.

5.6.2 Router performance validation on ProtocolBench

We also test our Protocol router on ProtocolBench. We use Spec-only Prompts to let router Choose protocols. We examine both router's ability to choose protocol(combinations) and performace on

Metric	Router	Best Single (A2A)
Quality avg (1–5)	2.50	**2.51**
Success avg	**9.90**	9.29

Table 5: Gaia (per-module selection)

Metric	Router	Best Single (ACP)
Duration (s)	**2375**	2417
Mean latency (ms)	**9495**	9663
Std. dev. (ms)	2866	**1077**

Table 6: Streaming Queue (router: ACP)

Metric	Router	Best Single
Pre-failure discovery (%)	14.86	**14.91 (Agora)**
Post-failure discovery (%)	13.98	**14.57 (A2A)**
Recovery time (s)	**6.55**	8.00 (A2A)

Table 7: Fail-Storm (router: A2A)

Security Check	Router	Best Single(ANP)
TLS transport	✓	✓
Session protection	✓	✓
E2E encryption	✓	✓
Tunnel resistance	✓	✓
Metadata protection	✓	✓

Table 8: Safety (secure protocol selected)

Table 9: Router execution validation: performance comparison against best single-protocol baselines across four scenario types.

each scenarios. We assume each agent in GAIA as a module in real network, and assign each agent a protocol. For the rest of three, all network are seen as one module and will assign only one protocol.

Router deployment strategy. For each scenario, ProtocolRouter selects protocols based on scenario characteristics: **Streaming Queue** → **ACP** (latency-optimized), **Fail-Storm** → **A2A** (resilience-focused), **Gaia** → per-module dynamic selection (see the empirical assignment distribution in Table 11, dominated by mixed bundles), and **Safety** → **ANP** (secure protocol). Except for Gaia's special case, the selections for the other three scenarios align with expectations.

Performance analysis. ProtocolRouter demonstrates competitive performance across all scenarios while providing adaptive protocol selection (Table 9). The router achieves lower latency in Streaming Queue, significantly reduces recovery time in Fail-Storm (6.55s vs 8.00s), yields higher success rates in Gaia (9.90 vs 9.29), and ensures perfect security compliance in Safety scenarios.

Takeaway

(1) On PROTOCOLROUTERBENCH, the spec+perf setting clearly outperforms spec-only, while notably reducing A2A ↔ ACP confusions. (2) Across the three single-module scenarios, the router's choices *match or surpass* the best single-protocol baselines—lower mean/tail latency in **Streaming Queue**, faster recovery in **Fail-Storm**, and full security compliance in **Safety**.

Findings

In **Gaia**, selections of mixed bundles match or surpass the best single-protocol baseline. ProtocolRouter attains best-of-single performance per module while unlocking system-level gains via cross-module protocol specialization.

6 CONCLUSION

This paper introduces ProtocolBench, the first comprehensive benchmark for evaluating agent communication protocols, and ProtocolRouter, a dynamic router that leverages protocol diversity for improved performance. Our systematic evaluation across diverse scenarios reveals that protocol choice significantly impacts system behavior across multiple dimensions—no single protocol dominates universally. By providing standardized evaluation tools and demonstrating the benefits of dynamic selection, we aim to transform protocol choice from ad-hoc decisions to principled engineering. As multi-agent systems mature from research curiosities to production infrastructure, understanding and optimizing communication layers becomes essential for building reliable, efficient, and scalable deployments.

7 Ethics Statement

Benchmarking communication protocols raises several ethical considerations. Efficient agent coordination could enable both beneficial applications and harmful automation. We explicitly exclude scenarios involving deception, manipulation, or privacy violation from our benchmark. The open-source release includes usage guidelines emphasizing responsible deployment.

Our fault injection experiments simulate infrastructure failures rather than adversarial attacks, avoiding the creation of tools for system disruption. We engage with the security community to ensure our protocol adapters do not introduce new vulnerabilities.

LLM Use Statement

In this work, we employed large language models (LLMs) in two ways. First, LLMs served as the backbone of our LLM-based agents, providing the core reasoning, planning, memory, and reflection capabilities necessary for our experimental evaluation. Specifically, we instantiated different protocols and error-detection modules on top of the same LLM backbone to ensure a fair comparison across agentic settings. Second, we used LLMs to assist in the preparation of the manuscript by refining the wording, improving sentence fluency, and ensuring clarity of presentation. All scientific claims, analyses, and conclusions were conceived, validated, and written by the authors.

=======

References

Anthropic. Model Context Protocol. https://modelcontextprotocol.io, 2024. Accessed: 2025.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. Ms marco: A human generated machine reading comprehension dataset, 2018. URL https://arxiv.org/abs/1611.09268.

Weize Chen, Chen Qian, Jinghan Lin, Ziheng Xue, Zhiwei Liu, Lin Liu, Cheng Zhou, Yun Yang, Yuan Wang, Cheng Xu, Le Yu, and Jie Tang. Internet of Agents: Weaving a web of heterogeneous agents for collaborative intelligence. In *arXiv preprint arXiv:2407.07061*, 2024.

Abul Ehtesham, Aditi Singh, Gaurav Kumar Gupta, and Saket Kumar. A survey of agent interoperability protocols: Model context protocol (MCP), agent communication protocol (ACP), agent-to-agent protocol (A2A), and agent network protocol (ANP). *arXiv preprint arXiv:2505.02279*, 2025. URL https://arxiv.org/abs/2505.02279.

Google Cloud. Agent2Agent Protocol (A2A). Technical report, Google, 2025. With support from 50+ technology partners including Atlassian, Box, Cohere, Intuit, MongoDB, PayPal, Salesforce, SAP, ServiceNow.

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

Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, Zeyu Zhang, Yifeng Wang, Qianshuo Ye, Bernard Ghanem, Ping Luo, and Guohao Li. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation, 2025. URL https://arxiv.org/abs/2505.23885.

Jonathan Hyun, Nicholas R. Waytowich, and Boyuan Chen. CREW-Wildfire: Benchmarking agentic multi-agent collaborations at scale. *arXiv preprint arXiv:2507.05178*, 2025. URL https://arxiv.org/abs/2507.05178.

IBM BeeAI. Agent Communication Protocol (ACP). https://docs.beeai.dev/acp/alpha/introduction, 2025. IBM Research.

LangChain. LangChain: Building applications with LLMs through composability. https://github.com/langchain-ai/langchain, 2024a. Accessed: 2025.

LangChain. LangGraph: Build resilient language agents as graphs. https://github.com/langchain-ai/langgraph, 2024b. Accessed: 2025.

Yunxiang Li, Zihan Li, Kai Zhang, Ruilong Dan, Steve Jiang, and You Zhang. Chatdoctor: A medical chat model fine-tuned on a large language model meta-ai (llama) using medical domain knowledge. *Cureus*, 15(6), 2023.

Xinbin Liang, Jinyu Xiang, Zhaoyang Yu, Jiayi Zhang, Sirui Hong, Sheng Fan, and Xiao Tang. Openmanus: An open-source framework for building general ai agents, 2025. URL https://doi.org/10.5281/zenodo.15186407.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *ICLR*, 2024.

Samuele Marro, Emanuele La Malfa, Jesse Wright, Guohao Li, Nigel Shadbolt, Michael Wooldridge, and Philip Torr. A scalable communication protocol for networks of large language models, 2024. URL https://arxiv.org/abs/2410.11905.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants, 2023. URL https://arxiv.org/abs/2311.12983.

Microsoft Research. AutoGen: Enable next-gen large language model applications. https://github.com/microsoft/autogen, 2024. Microsoft.

João Moura. CrewAI: Framework for orchestrating role-playing autonomous AI agents. https://github.com/joaomdmoura/crewAI, 2024. Accessed: 2025.

OpenAI. Swarm: Educational framework for multi-agent orchestration. https://github.com/openai/swarm, 2024. OpenAI.

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

Yingxuan Yang, Huacan Chai, Yuanyi Song, Siyuan Qi, Muning Wen, Ning Li, Junwei Liao, Haoyi Hu, Jianghao Lin, Gaowei Chang, Weiwen Liu, Ying Wen, Yong Yu, and Weinan Zhang. A survey of AI agent protocols. *arXiv preprint arXiv:2504.16736*, 2025. URL https://arxiv.org/abs/2504.16736.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. MultiAgentBench: Evaluating the collaboration and competition of LLM agents. *arXiv preprint arXiv:2503.01935*, 2025. URL https://arxiv.org/abs/2503.01935.

A Limitations and Future Work

While ProtocolBench provides comprehensive protocol evaluation, several limitations merit discussion. Our scenarios, though representative, cannot capture all possible agent communication patterns. Edge cases like byzantine failures or adversarial agents remain unexplored. The focus on LLM-based agents may not generalize to hybrid systems incorporating traditional software components.

ProtocolRouter's learning approach assumes stationary or slowly-changing workload distributions. Rapid context switches or rare events may not provide sufficient signal for adaptation. The computational overhead of maintaining multiple protocol states could become prohibitive at very large scales.

Future work should expand scenario coverage, particularly for emerging patterns like agent swarms and hierarchical delegation. Integration with production orchestration systems would enable real-world validation. Theoretical analysis of protocol complexity bounds and impossibility results would complement our empirical findings.

B Detailed description of benchmark implementation

B.1 Gaia Document Question-Answering Implementation

The Gaia Document Question-Answering scenario evaluates hierarchical information aggregation in multi-agent protocols. Below, we detail its implementation, covering the planner module, agent lifecycle, network memory, evaluation pipeline, sandboxed execution, time accounting, adjudication, and fairness mechanisms.

1. **Planner Module**: A large language model (LLM) generates a JSON manifest encoding agent configurations (roles, toolsets, prompt templates), tool-call metadata (interfaces, arguments, outputs), and network topology with explicit workflow and message-flow definitions. Discrete difficulty levels map to agent counts (2, 4, or 8 agents for levels 1, 2, or 3) to ensure reproducibility, with a recorded prompting seed. The manifest ensures identical configurations across protocols for fair comparisons.

2. **Agent Lifecycle and Network Communication**: Agents operate in a distributed communication model where any agent can communicate with any other agent in the network through unique addressable endpoints. They follow the manifest's workflow, processing messages by parsing inputs, invoking tools or LLMs, and routing responses to designated next hop(s). The network layer abstracts protocol differences and ensures reliable message delivery.

3. **Step-Based Network Memory**: An append-only memory pool logs all interactions in structured JSON, capturing step indices, agent IDs, fine-grained timestamps, execution status, and message histories with tool invocations. The memory supports offline analysis, replay, and LLM-driven summarization.

4. **LLM-Based Summarization and Evaluation**: Post-workflow, an LLM summarizer generates a concise outcome from the memory pool using a standardized prompt. A separate LLM judge evaluates the result and execution log against a rubric assessing factual accuracy, relevance, and completeness. The pipeline records resource metrics (e.g., token usage, time).

5. **Tool Design and Execution**: Many distinguished open-source agent collaboration frameworks Liang et al. (2025); Hu et al. (2025) provide high-quality toolkits. Building upon these advancements, the tools in our Gaia scenario are designed through selective reuse and adaptation, enabling both efficient integration and tailored functionality. All code execution tools operate within isolated environments with virtualized dependencies, restricted filesystem/network access, and resource limits (CPU, memory, wall time). Logs and artifacts are captured and linked to execution steps to facilitate traceability and reproducibility.

6. **Fine-Grained Time Accounting**: Timestamps are recorded at agent, step, and workflow levels in milliseconds (Unix epoch), enabling latency profiling and straggler detection.

7. **LLM-Driven Adjudication**: The LLM judge assesses outcomes using structured prompts and rubric criteria, producing pass/fail results and quality scores (e.g., accuracy, task alignment). Judgments are stored as structured metadata.

8. **Metrics and Reporting**: The evaluation report includes comprehensive performance metrics (success rate, execution time breakdown, resource consumption by agent and task), quality scores with detailed LLM judge analysis, and operational statistics (task completion rates, communication overhead). Reports are emitted in both structured JSON format and human-readable console summaries with visual indicators.

9. **Experimental Fairness**: All protocols use the same planner-generated manifest and canonical seed to control variability, ensuring only protocol implementations differ.

This implementation ensures rigorous, reproducible evaluation of hierarchical routing in multi-agent settings, with detailed metrics and fairness controls.

B.2 Safety Tech Implementation

The Safety Tech scenario evaluates privacy-preserving protocols in medical Q&A under adversarial conditions, focusing on protocol-stack security penetration testing and multi-dimensional confidentiality protection.Below, we detail its implementation, including scenario setup, test points, probe mechanisms, reporting, and technical features.

1. **Scenario Setup**: The setup launches a registration gateway (RG), coordinator, and two LLM doctors using native protocols (e.g., Agora_Doctor_A/B, ANP_Doctor_A/B). Doctors register with the RG, pull directories via the coordinator, and engage in bidirectional communication through protocol-specific endpoints. The system processes 10 augmented cases from ChatDoctor-HealthCareMagic-100k with synthetic identity information. Comprehensive probe mode is enabled via environment variables, injecting probes real-time into message/HTTP/TLS paths using a unified `probe config` for protocol-agnostic testing across Agora SDK, ANP DID encryption, ACP routing, and A2A execution.

2. **Transport and Certificate Security**: Conducts 3 rounds of TLS downgrade attacks using weak cipher suites, obsolete TLS versions, and HTTP plaintext fallback, recording success and block rates for each attempt. A comprehensive certificate matrix systematically verifies security blocking across 6 test dimensions: expired certificates, self-signed certificates, hostname mismatches, incomplete certificate chains, weak cipher suites, and TLS version downgrades. Each matrix test generates detailed blocking metrics and assigns scores based on successful interception prevention, providing a systematic assessment of transport-layer security robustness.

3. **End-to-End Payload Confidentiality**: Injects watermarks and plaintext probes (e.g., `PLAINTEXT_PROBE_*`, `HIDDEN_MARKER:S2_E2E_WATERMARK_TEST_*`) into payloads. Uses `tcpdump` on the `lo0` interface (8 seconds) to capture network traffic and detect plaintext leakage through sensitive keyword matching. The system evaluates encryption effectiveness by analyzing watermark visibility and sensitive keyword hits (e.g., patient ID, SSN, credit card numbers), assigning scores based on watermark injection participation and leak prevention performance.

4. **Session and Timing Protection**: For session hijack, injects privilege-escalation tokens (e.g., `expired_session_*`, `admin_session_*`), measuring interception rates via denials or 404s. Clock skew tests $\pm$30s/$\pm$2m/$\pm$5m/$\pm$10m offsets and window anomalies (e.g., `TIME_SKEW`, `WINDOW_REPEAT/DISORDER/DUPLICATE`) over 12 rounds. Replay attacks involve 2 rounds of old message replays, distinguishing real blocks from errors like ReadTimeout/500.

5. **Metadata and Side-Channel Protection**: Probes endpoints (e.g., `/health`, `/metrics`, `/status`) for exposed meta-info, quantifying exposure counts. `tcpdump` analyzes plaintext bytes and sensitive keyword hits to assess information leakage and calculate metadata exposure scores.

6. **Real-Time Probe Injection Mechanism**: Probes are injected via protocol clients' `send()` methods into native paths (e.g., before Agora SDK calls, ANP signatures, ACP requests). The system dispatches `probe_config` parameters for clock skew, watermarks, and replays, ensuring authentic testing.

7. **Weighting and Reporting**: Employs a multi-dimensional assessment system across TLS/transport security, session hijack protection, E2E encryption detection, tunnel sniffing, and metadata leakage dimensions.

8. **Technical Features**: Unified `ProbeConfig` class standardizes parameters (e.g., `tls_downgrade`, `e2e_payload_detection`, `time_skew_matrix`) for cross-protocol consistency. Real-time injections in native paths ensure authenticity. Multi-dimensional assessment covers transport, application, session, and timing layers comprehensively.

This implementation provides a robust, protocol-agnostic framework for evaluating adversarial robustness and privacy protection capabilities across multi-agent communication protocols.

B.3 Streaming Queue Implementation

The Streaming Queue scenario evaluates distributed question-answering coordination and protocol performance in multi-agent systems. It focuses on task orchestration, load balancing across workers, and cross-protocol compatibility, covering scenario setup, intelligent task routing, comprehensive metrics collection, and protocol-agnostic architecture design.

1. **Scenario Setup**: A centralized network comprises one coordinator and four workers, processing 1000 preprocessed entries from the MS MARCO dataset (Bajaj et al., 2018). The dataset is simplified to focus on communication metrics rather than task difficulty. Testing is conducted on an AMD server localhost to eliminate network fluctuations, ensuring consistent timing measurements.

2. **Task Processing and Load Balancing**: The coordinator employs a work-stealing approach where workers compete for tasks from a shared queue, achieving natural load distribution based on individual worker processing speeds. The system tracks completion times, task counts per worker, and calculates load balance variance to assess protocol communication efficiency and stability. This approach enables evaluation of how protocol complexity, including authentication and encryption mechanisms, affects task distribution uniformity across workers.

3. **Metrics Collection**: Metrics focus on communication performance and stability, including: - Total test duration.
- Success rate (fraction of completed tasks).
- Response times (average, minimum, maximum, standard deviation, median).
- Load-balancing variance (task distribution across workers).
- Network errors and retries.
- Timeout counts (tasks exceeding time limits).
Network errors, retries, and timeouts are expected to be zero or consistent across protocols, as per design.

4. **Technical Features**:
- **Load Balancing**: The coordinator uses a work-stealing approach where workers compete for tasks, with load balance variance measured to assess distribution uniformity.
- **Local Testing**: Running on localhost isolates protocol performance from external network variability.
- **Metric Granularity**: Per-task response times and worker-specific metrics enable fine-grained analysis of protocol efficiency and stability.
- **Protocol Comparison**: Uniform task sets and configurations ensure fair comparisons, with performance differences attributable to inherent protocol characteristics and implementation complexity (e.g., A2A's lightweight routing vs. Agora's authentication overhead).

This implementation stress-tests communication efficiency and stability, providing insights into protocol performance under standardized workload conditions.

B.4 Fail-Storm Recovery Implementation

The Fail-Storm Recovery scenario evaluates protocol resilience under node failures in a Shard QA setup, testing robustness, reconnect times, and collaborative performance. Below, we detail its implementation, covering the Shard QA base scenario, failure injection, recovery mechanisms, metrics, and technical features.

1. **Shard QA Base Scenario**: A ring topology with 8 QA agents processes groups of 8 data points from the 2WikiMultiHopQA dataset (Ho et al., 2020), including shuffled queries, answers, and contents. Each agent receives one query and a random content segment. To resolve the query, agents forward requests to neighbors for matching content. Messages propagate up to 8 hops; failure occurs if unresolved after 8 hops. This tests communication efficiency and multi-agent collaboration.

2. **Failure Injection**: Every 2 minutes during a running Shard QA session, 3 agents are randomly terminated (killed) to simulate sudden dropouts. Killed agents initiate reconnect attempts after a 2-second delay, mimicking realistic network recovery patterns where agents need brief time to detect failures and initialize reconnection procedures.

3. **Recovery Mechanisms**: Upon detecting a failed target agent, messages skip it and forward to the next in the ring. Recovery time is measured from the kill event to the successful reconnection of the last affected agent. The process involves 3 agents departing and rejoining, assessing network stability during transitions.

4. **Performance Phases**:
- **Pre-Fault**: The 2 minutes before a kill event, establishing baseline performance.
- **Recovery**: The period from kill to full reconnection.
- **Post-Fault**: From recovery completion to the next kill event.
Performance differences across phases (e.g., success rates, latencies) quantify robustness.

5. **Metrics Collection**: Key metrics include recovery time (seconds from fault injection to system stabilization), answer discovery rate (percentage of queries successfully resolved, measured pre- vs.

post-fault), and steady-state average latency (task completion times in seconds, comparing pre-fault and post-fault phases). These metrics quantify protocol resilience by measuring both functional performance degradation and temporal recovery characteristics.

6. **Technical Features**:
- **Failure Detection**: Agents detect failures via timeouts or heartbeat checks, enabling ring skips.
- **State Recovery**: Reconnecting agents restore state from logs or peers to minimize disruptions.
- **Fair Comparison**: Identical datasets and topologies across protocols ensure differences stem from failure handling.
- **Simulation Controls**: Random kills are seeded for reproducibility, with multiple runs averaging results.

This implementation rigorously assesses fault tolerance, state recovery, and sustained collaboration in dynamic multi-agent networks.

C Benchmark Implementation

C.1 Controls and Fairness (Details)

C.1.1 Experimental Setup: Constants and Variables

We categorize the experimental setup into pinned constants (ensuring reproducibility) and scenario-specific variables (capturing task diversity).

Pinned Constants. All non-protocol factors are fixed and verified:

- **Model and decoding:** Qwen2.5-VL-72B-Instruct; temperature=0.0, top_p=1.0, max_tokens=4096.
- **Hardware/OS/container:** Single-node AMD server; pinned image with identical OS, drivers, and libraries for all runs.
- **Prompts:** Version-anchored prompts for base system, Gaia judge, Safety evaluator, and ProtocolBench router.
- **Rate limits/timeouts:** connection_timeout=10s, message_timeout=30s, qa_cycle_timeout=15s, max_retries=3 with exponential backoff.
- **Adapter/router versions:** Commit hashes are recorded in the artifact manifest.
- **Internal retries/reconnects:** Disabled at protocol adapters; recovery is implemented uniformly in the upper PAL layer to avoid bias.

Scenario Variables. Each scenario introduces its own communication topology and dynamics:

- **Fail-Storm (FS):** 8-node ring; at most 8 hops; skip failed nodes until recovery.
- **Streaming Queue (SQ):** Star topology with 1 coordinator and 4 workers.
- **Gaia:** Dynamic star; agent count increases with level (L1=2, L2=4, L3=8).
- **Safety:** Point-to-point with two endpoints (two doctors).

C.1.2 Fairness verification

We perform *replay equality checks*: given identical inputs, non-protocol side-effects (planner outputs, tool calls) are identical across adapters. ProtocolBench operates with temperature 0 to ensure deterministic outputs. All equality checks and logs are included in the artifacts.

C.2 Windowing, Byte Accounting, and Aggregation

C.2.1 FS windowing and recovery metrics

For cycle t with kill timestamp k_t and last reconnection timestamp r_t:

- **Pre window:** $[k_t - 60\text{s},\ k_t)$.
- **Recovery window:** $[k_t,\ r_t]$.
- **Post window:** $(r_t,\ r_t + 60\text{s}]$; truncated if the next kill begins earlier.

Level	# Scenarios	Modules per Scenario	# Modules
L1	12	1	12
L2	12	2	24
L3	12	3	36
L4	12	4	48
L5	12	5	60
Total	**60**	–	**180**

Table 10: ProtocolBench difficulty breakdown.

Primary FS endpoints:

$$\text{Time-to-Recovery (TTR)} = r_t - k_t,$$

$$\text{Post-fault retention} = \frac{\text{\# successful requests in post}}{\text{\# successful requests in pre}}.$$

If pre has zero successes, retention is marked *NA* and excluded from aggregates.

C.2.2 Latency and percentiles

Latency distributions are summarized by mean, median, and percentile endpoints. For SQ, the primary endpoint is P95 end-to-end latency per run; we report medians and BCa bootstrap 95% CIs across runs.

C.2.3 Byte accounting

We separate:

- `MSG_BYTES_PAYLOAD`: application payload bytes (requests + responses).
- `MSG_BYTES_RETRY_OVERHEAD`: bytes due to retries and protocol-level overhead.

TLS handshakes and cryptographic negotiation bytes are excluded from both counters. Counting is performed at the middleware boundary to avoid double counting. For streaming, bytes are bucketed by message boundaries before aggregation.

C.2.4 Aggregation levels

- **Per-request:** latency, payload bytes, overhead bytes.
- **Per-run:** success rate, FS recovery metrics.
- **Per-scenario/module:** ProtocolBench accuracies.

C.3 ProtocolRouterBench: Data, Rules, and Artifacts

C.3.1 DATA

Corpus and ID conventions. File: `ProtocolBench_scenarios.jsonl` with 60 scenarios. Scenario IDs: `RB-L{level}-{idx}`, where `level`$\in \{1,\ldots,5\}$ and `idx`$\in \{01,\ldots,12\}$. Module IDs: `RB-L{level}-{idx}-M{m}` (1-based). The artifact manifest `MANIFEST.yaml` records file hashes and the commit for the corpus.

Difficulty stratification and construction. There are 12 scenarios per level (L1–L5). Modules per scenario increase with level (L1:1, L2:2, L3:3, L4:4, L5:5), totaling 180 modules. Construction constraints:

1. Explicit role/module descriptors per scenario.
2. *Lock/exclude phrases* prevent multi-label ground truth when needed (e.g., "REST/idempotent/batch/archival" locks resource semantics; "avoid resource/state-machine semantics" excludes them).
3. No cross-module context sharing; each module is prompted and judged independently.
4. Single-choice ground truth in {A2A, ACP, Agora, ANP}.

Rank	**Assignment** (ordered by module index)	#Mods	Count	Share (%)
1	`[agora, acp]`	2	70	42.4
2	`[agora, agora, acp]`	3	33	20.0
3	`[agora, agora, agora, acp]`	4	25	15.2
4	`[acp]`	1	7	4.2
5	`[agora, a2a, agora, acp]`	4	6	3.6
6	`[agora, agora, agora, agora, acp]`	5	4	2.4
7	`[a2a, acp]`	2	4	2.4
8	`[agora, agora, a2a, acp]`	4	3	1.8
9	`[agora, agora, agora, agora, agora, agora, acp]`	7	3	1.8
10	`[agora, a2a, acp]`	3	3	1.8
11	`[agora, agora, agora, agora, agora, acp]`	6	1	0.6
12	`[agora, agora, agora, a2a, agora, agora, agora, acp]`	8	1	0.6
13	`[agora, agora, agora, agora, a2a, acp]`	6	1	0.6
14	`[agora, a2a, agora, a2a, acp]`	5	1	0.6
15	`[agora, agora, agora, a2a, acp]`	5	1	0.6
16	`[agora, agora, agora, agora, agora, agora, agora, acp]`	7	1	0.6
17	`[agora, a2a, a2a, acp]`	4	1	0.6

Table 11: Gaia — Router assignment patterns per run (total matches $= 165$, unique assignments $= 17$). Assignment lists map module m_i (index = position) to protocol in order.

C.3.2 RULES

Feature facets and evidence mapping. We fix a compact facet set and a lexicon that maps scenario spans to facets:

- **Transport/interaction:** SSE/streaming, RPC, batch.
- **Long-running/artifacts:** job orchestration, checkpoints, artifacts.
- **Identity/E2E:** DID, key material, end-to-end encryption.
- **Delivery/replay:** at-least-once, idempotency, replay windows.
- **Operation semantics:** REST, idempotent updates, state machines.
- **Trust/governance:** audit, consent, policy hooks.

Hard constraints first prune incompatible candidates (e.g., strict E2E removes protocols without native E2E). The decision order in `priority_decide()` is identity/E2E → operation semantics → interaction (streaming/long-job). If candidates remain tied, `pick_by_narrative()` selects the protocol whose defining capability anchor appears earliest in the scenario text; stable fallback order: `[A2A, ACP, Agora, ANP]`.

Prompt and function-call contract. Router uses a fixed, version-anchored prompt `PROTOCOL_SELECTION_PROMPT` as shown in E.10.2. Responses are emitted via a structured function call with JSON fields:

```
{
  "module_id": "RB-L3-07-M2",
  "selected_protocol": "ACP",
  "evidence_spans": ["..."],
  "rationale": "Short textual reason; no numbers, no performance
  ↪  claims."
}
```

Rationales must not contain numbers or performance claims. A linter enforces a field whitelist and rejects numeric tokens in rationales.

Scoring and missingness. Scenario accuracy equals 1 only if all modules are correctly predicted. Module accuracy is the fraction of correctly predicted modules. If a module record is malformed or absent, the entire scenario is list-wise excluded and the exclusion is logged; no zero-filling.

Train/dev/test policy. This release ships only the 60 evaluation scenarios. A stratified split will be added in a future release.

Non-leakage and pre-specification. All texts are model-generated with human curation. Vendor, product, and library names are removed or neutralized; only generic capabilities and interaction semantics remain. The decision rules, prompts, and schema are pre-specified and version-anchored.

C.3.3 ARTIFACTS

We release configs, scripts, commit hashes, dashboards, dataset splits, execution logs, and the full ProtocolBench bundle. A one-shot script reproduces the entire pipeline (scenarios → decisions → metrics → tables). The manifest records file hashes and commits.

Example one-shot command (for illustration)

```
bash run_all.sh --scenarios data/ProtocolBench_scenarios.jsonl \
  --router_prompt prompts/PROTOCOL_SELECTION_PROMPT.txt \
  --out_dir outputs/ --seed 0 --temperature 0
```

MANIFEST.yaml (excerpt)

```
corpus:
  file: ProtocolBench_scenarios.jsonl
  sha256: <TBD>
  commit: <TBD>
prompts:
  router_prompt: PROTOCOL_SELECTION_PROMPT.txt
  sha256: <TBD>
runs:
  - id: run_001
    seed: 0
    temperature: 0
```

ProtocolRouterBench JSON schema (abridged)

```
{
  "scenario_id": "RB-L3-07",
  "difficulty": "L3",
  "modules": [
    {"module_id":"RB-L3-07-M1","role":"retriever","gt":"ACP"},
    {"module_id":"RB-L3-07-M2","role":"coordinator","gt":"A2A"},
    {"module_id":"RB-L3-07-M3","role":"auditor","gt":"Agora"}
  ],
  "text": "<scenario description with lock/exclude cues>"
}
```

ProtocolRouterBench Data Structure

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "ProtocolBenchScenario",
  "type": "object",
  "required": ["scenario_id", "modules"],
  "properties": {
    "scenario_id": {"type": "string"},
    "level": {"type": "integer", "minimum": 1, "maximum": 5},
    "modules": {
      "type": "array",
      "items": {
        "type": "object",
        "required": ["module_id", "text", "label"],
        "properties": {
          "module_id": {"type": "string"},
          "text": {"type": "string"},
          "label": {"type": "string", "enum":
          ↪  ["A2A","ACP","Agora","ANP"]},
          "locks": {"type": "array", "items": {"type": "string"}},
          "excludes": {"type": "array", "items": {"type":
          ↪  "string"}}
        }
      }
    }
  }
}
```

C.4 Threats to Validity, Ablations, and Statistical Procedures

C.4.1 Construct validity and multi-implementation check

We separate protocol design from implementation artifacts. A planned multi-implementation comparison (production-optimized vs. minimal references) is run under identical adapters; we expect relative orderings to remain stable.

C.4.2 Ablations

1. **Envelope-only vs. full-feature paths:** disable advanced features and compare against full stacks.
2. **Topology substitution:** freeze Gaia's dynamic star and compare to the default dynamic configuration.
3. **Planner freezing:** fix planner outputs to isolate protocol effects.
4. **ProtocolBench-specific:** remove lock/exclude phrases to quantify A2A↔ACP confusions; disable `priority_decide()` to observe tie instability.

C.4.3 Statistical procedures

For continuous metrics we compute BCa bootstrap 95% CIs with B=10,000 resamples. ProtocolBench accuracies use exact binomial or Wilson intervals. Pairwise comparisons report Cliff's δ and Hodges–Lehmann median differences (point estimate with 95% CI). Multiple comparisons are corrected via Holm–Bonferroni. We separate *in-run jitter* (per-request coefficient of variation) from *run-to-run variability* (across-run coefficient of variation) when repeated runs are available.

D Scenario Prompt Design

FS Shard Worker System Prompt is used by fail-storm shard workers to maximize answer discovery under cyclic faults.

FS Shard Worker System Prompt

```
def _get_system_prompt(self) -> str:
    """Get system prompt for the shard worker - Enhanced for
    ↪  distributed search"""
    max_ttl = self.global_config.get('tool_schema',
    ↪  {}).get('max_ttl', 15)
    return f"""You are agent {self.shard_id} in an intelligent
    ↪  distributed document search system.

 NETWORK TOPOLOGY:
- Your neighbors: {self.neighbors['prev_id']}  YOU
↪  {self.neighbors['next_id']}
- You process document shard {self.agent_idx}

 CURRENT SEARCH TASK:
Question: {self.current_question}

 YOUR LOCAL DOCUMENT FRAGMENT:
{self.current_snippet}

 AVAILABLE TOOLS:
1. lookup_fragment: Analyze your local document fragment
2. send_message: Communicate with coordinator and neighbors

 DISTRIBUTED SEARCH PROTOCOL:

STEP 1 - LOCAL SEARCH:
 Call lookup_fragment(question="{self.current_question}",
↪  found=<true/false>, answer="<extracted_info>")
→ Be GENEROUS with found=true - partial information is valuable!

STEP 2 - ACTION BASED ON RESULT:
If found=true:
 send_message(destination="coordinator", content="ANSWER_FOUND:
↪  <detailed_answer>")

If found=false:
→ The system will automatically handle neighbor search
→ No need to manually send neighbor requests

 ULTRA-LIBERAL SEARCH CRITERIA (MAXIMIZE DISCOVERY):
 SET found=true if your fragment contains ANY of these:
- Direct answers or partial answers
- Names, entities, dates, numbers mentioned in the question
- Related context, background information, or topic-relevant content
- Keywords or concepts that connect to the question
- Similar or related entities (e.g., same type of person, place,
↪  thing)
- Historical context or background about the topic
- Even tangentially related information
- ANY word or phrase that appears in both question and fragment
- Information that could help answer the question when combined with
↪  other sources

 SET found=false ONLY if:
- Fragment is about completely different, unrelated topics with ZERO
↪  overlap
- Absolutely no shared words, concepts, or themes with the question

 CRITICAL: When in doubt, ALWAYS choose found=true! It's better to
↪  be overly generous than to miss relevant information.
```

```
 ANSWER EXTRACTION:
When found=true, extract the most relevant information:
- Include specific facts, names, dates, numbers
- Provide context that helps answer the question
- Be specific and detailed rather than vague

 LIBERAL DETECTION EXAMPLES:
..."""
```

FS Local Search Prompt guides generous local matching to maximize discovery before neighbor/ring forwarding.

FS Local Search Prompt

```
def _get_local_search_prompt(self, question: str) -> str:
    """Get optimized prompt for local document search."""
    return f"""You are a specialized document search agent analyzing
    ↪  a document fragment.

SEARCH QUESTION: {question}

YOUR DOCUMENT FRAGMENT:
{self.current_snippet}

TASK: Determine if your document fragment contains ANY information
↪  that helps answer the question.

SEARCH CRITERIA (Be ULTRA-LIBERAL - MAXIMIZE DISCOVERY):
 FOUND (set found=true) if the fragment contains ANY of:
- Direct answers to the question
- Names, entities, or keywords mentioned in the question
- Related facts or context that partially answers the question
- Background information about the topic
- Similar entities or concepts (same category/type)
- Historical context or time period mentioned in question
- ANY shared words or phrases between question and fragment
- Information that could contribute to answering when combined with
↪  other sources
- Even tangentially related information

 NOT FOUND (set found=false) ONLY if:
- Fragment is about completely different, unrelated topics with ZERO
↪  overlap
- Absolutely no shared concepts, words, or themes

 CRITICAL: When in doubt, choose found=true! Better to include
↪  potentially relevant info than miss it.

RESPONSE FORMAT: Use the lookup_fragment function with:
- found: true/false (be generous with true)
- answer: extract the relevant information if found
- confidence: 0.0-1.0 (how confident you are)

EXAMPLES:
Question: "What nationality were Scott Derrickson and Ed Wood?"
Fragment: "Scott Derrickson is an American filmmaker..."
↪  found=true, answer="Scott Derrickson is American"
Fragment: "Ed Wood was born in New York..."  found=true, answer="Ed
↪  Wood was American (born in New York)"
Fragment: "The Laleli Mosque in Turkey..."  found=false (completely
↪  unrelated)
```

```
Remember: It's better to find partial information than to miss
↪  relevant content. The collaborative system will combine partial
↪  answers from multiple agents."""
```

SQ QA Worker System Prompt is designed for high-throughput QA workers under star topology.

SQ QA Worker System Prompt

```
# Location: agent_network/script/streaming_queue/core/qa_worker_bas⌋
↪  e.py:117-120
system_prompt = (
    "You are a helpful assistant. Provide concise, accurate answers
    ↪  to questions. "
    "Keep responses under 150 words."
)
```

SQ Meta Coordinator Task Prompt describs the streaming pressure test objective and constraints.

SQ Meta Coordinator Task Prompt

```
# Location: agent_network/script/streaming_queue/runner/run_meta_ne⌋
↪  twork.py:232-241
pressure_test_task = {
    "question": "Streaming queue pressure test: process maximum
    ↪  questions in minimum time",
    "context": "High-throughput QA processing with diverse question
    ↪  types",
    "metadata": {
        "type": "pressure_test",
        "volume": 50,  # batch_size
        "priority": "maximum_speed",
        "target_qps": 20
    }
}
```

GAIA Planner Prompt defines a task analysis system that classifies a task, assesses complexity, selects tools, and configures specialized agents with roles. It enforces rules and provides a few-shot JSON example to guide structured multi-agent planning.

GAIA Planner Prompt

```
TASK_ANALYSIS_SYSTEM = """You are an expert multi-agent system
↪  architect. Analyze the given task with deep understanding and
↪  provide a comprehensive analysis.

Consider these aspects:
1. TASK TYPE - Classify precisely:
   - qa_with_reasoning: Question-answering requiring logical
   ↪  reasoning
   - multi_step_analysis: Complex analysis requiring multiple
   ↪  processing stages
   - content_generation: Creating new content, documents, reports
   - computational_task: Mathematical calculations, data processing
   - research_task: In-depth information gathering and synthesis
   - general_qa: Simple question-answering

2. COMPLEXITY ASSESSMENT:
   - low: Simple, straightforward tasks requiring 1-2 steps
   - medium: Moderate complexity requiring 3-5 processing steps
```

```
   - high: Complex tasks requiring 6+ steps, domain expertise, or
   ↪  sophisticated reasoning

3. REQUIRED TOOLS - Select from available tools:
   Available tools: {available_tools}

4. AGENT CONFIGURATION - For each required tool, specify:
   - name: Descriptive agent name (e.g., "WebResearcher",
   ↪  "DataAnalyst", "CodeExecutor")
   - role: Create meaningful, task-specific roles (e.g.,
   ↪  "information_gatherer", "computational_specialist",
   ↪  "data_processor", "final_synthesizer", "document_analyzer",
   ↪  "web_navigator", etc.)
   - Be creative with roles - they should reflect the agent's
   ↪  specific function in solving the task

   Example role types you can use as inspiration:
   * information_gatherer: Searches for and collects relevant
   ↪  information from various sources
   * computational_specialist: Executes calculations, data
   ↪  processing, and analytical tasks
   * document_analyzer: Processes and extracts information from
   ↪  documents and files
   * evidence_synthesizer: Integrates information from multiple
   ↪  sources into coherent conclusions
   * task_coordinator: Breaks down complex tasks and manages
   ↪  workflow execution
   * content_creator: Generates reports, summaries, and structured
   ↪  outputs
   * domain_expert: Provides specialized knowledge in specific
   ↪  fields
   * data_processor: Handles data transformation, cleaning, and
   ↪  formatting
   * web_navigator: Specializes in web search and online information
   ↪  retrieval
   * final_synthesizer: Provides comprehensive final answers and
   ↪  conclusions


5. DOMAIN EXPERTISE needed (technology, science, business, finance,
↪  healthcare, etc.)

6. PROCESSING REQUIREMENTS:
   - Sequential vs parallel processing needs
   - Validation/verification requirements
   - Error handling complexity

IMPORTANT HARD RULES:
- The tool 'create_chat_completion' is reserved for the FINAL agent
↪  only. Include it exactly once and position it as the LAST step
↪  in the workflow. Do NOT assign or call it in intermediate steps
↪  or by non-final agents.

IMPORTANT: Based on the GAIA task level {level}, we recommend using
↪  approximately {recommended_agents} agents for optimal
↪  performance. However, you can adjust this number based on task
↪  complexity:
- Use fewer agents (1-2) for very simple, single-step tasks
- Use the recommended number ({recommended_agents}) for typical
↪  level {level} tasks
- Use more agents (up to {max_agents}) only if the task genuinely
↪  requires complex multi-step processing
```

```
You must limit your agent recommendations to a maximum of
↪  {max_agents} agents total. Plan efficiently within this
↪  constraint.
Respond with detailed JSON analysis including your reasoning.

Analyze the task and respond with a JSON object containing:
{{
  "task_type":
  ↪  "general_qa|research_task|computational_task|multi_step_analysis",
  "complexity": "low|medium|high",
  "required_tools": ["tool1", "tool2"],
  "agents": [
    {{
      "tool": "tool_name",
      "name": "AgentName",
      "role": "specific_role_based_on_function",

    }}
  ],
  "estimated_steps": number,
  "domain_areas": ["domain1", "domain2"]
}}

Example:
{{
  "task_type": "research_task",
  "complexity": "medium",
  "required_tools": ["browser_use", "create_chat_completion"],
  "agents": [
    {{
      "tool": "browser_use",
      "name": "WebResearcher",
      "role": "academic_information_gatherer",

    }},
    {{
      "tool": "create_chat_completion",
      "name": "ReasoningSynthesizer",
      "role": "evidence_synthesizer",

    }}
  ],
  "estimated_steps": 3,
  "domain_areas": ["general_knowledge"]
}}
"""
```

Agent Role template instantiates agent expertise, responsibilities, and collaboration, ensuring structured coordination and quality outcomes in multi-agent systems.

Agent Role template

```
AGENT_ROLE_TEMPLATE = """You are {agent_name}, a
↪  {role_words.lower()} specialist. Your primary responsibilities
↪  include:

1. EXECUTE tasks related to your {role_words.lower()} expertise
2. PROVIDE expert-level insights and analysis within your domain
3. PROCESS information efficiently and accurately according to your
↪  role
4. COLLABORATE effectively with other agents in the workflow
```

```
5. DELIVER high-quality results that contribute to the overall task
↪  completion

Your expertise in {role_words.lower()} makes you an essential part
↪  of the multi-agent system."""
```

LLM Judge Prompt provides the LLM with a process-oriented evaluation framework emphasizing a consistent, rubric-based assessment to ensure transparent and reproducible scoring. To thoroughly evaluate the MAS's communication process as well as the final answer, full execution logs are prioritized over summaries as they provide the necessary unabridged evidence.

LLM Judge Prompt

```
LLM_JUDGE_PROMPT = """You are an expert judge evaluating AI system
↪  responses for the GAIA benchmark. Your evaluation must consider
↪  both the final answer's correctness and the quality of the
↪  process taken by the AI.

**TASK DETAILS:**
- **ORIGINAL QUESTION:** {question}
- **GROUND TRUTH ANSWER:** {ground_truth}

- **EXTRACTED FINAL ANSWER:** {final_answer}

- **FULL AI SYSTEM RESPONSE (TRACE) (Brief summary / final
↪  output):**
{predicted_answer}

---
IMPORTANT: When assessing the agent, PRIORITIZE the FULL NETWORK
↪  EXECUTION LOG (JSON) below if provided. This log contains all
↪  inter-agent messages, tool calls, and intermediate data
↪  exchanges. Your process-quality judgment MUST be based primarily
↪  on the content, clarity, correctness, and completeness of
↪  inter-agent communication and tool interactions recorded in the
↪  network execution log. Do NOT rely only on any short summary or
↪  the extracted final answer.

**FULL NETWORK EXECUTION LOG (JSON):**
{network_log_content}

If the network log is unavailable, fall back to using the FULL AI
↪  SYSTEM RESPONSE (TRACE) above.

**EVALUATION INSTRUCTIONS:**

Your task is to perform a two-part evaluation:
1.  **Correctness (`is_correct`):** First, determine if the
↪  `EXTRACTED FINAL ANSWER` is correct when compared to the `GROUND
↪  TRUTH ANSWER`. Consider semantic equivalence and allow for minor
↪  formatting differences.
2.  **Process Quality (`quality_score`):** Second, and just as
↪  importantly, evaluate the agent's problem-solving process based
↪  on the FULL NETWORK EXECUTION LOG (preferred) or the `FULL AI
↪  SYSTEM RESPONSE (TRACE)` when the log is unavailable. Use the
↪  detailed rubric below to assign a score from 1 to 5.

---
**QUALITY SCORE RUBRIC (1-5):**
```

```
Your primary focus for the `quality_score` is the agent's methodology
↪  and the quality of inter-agent communication. A high score can
↪  be given for a good process even if the final answer is
↪  incorrect.

- **Score 5 (Excellent):**
  - The final answer is correct.
  - Inter-agent communication is clear, complete, and correct. Tools
  ↪  are used correctly and efficiently. Intermediate results are
  ↪  validated and shared appropriately.

- **Score 4 (Good):**
  - The final answer is correct, but communication may have minor
  ↪  inefficiencies or small omissions.

- **Score 3 (Fair / Good Process):**
  - Solid reasoning and reasonable communication, but a late error
  ↪  or omission causes the final answer to be incorrect.

- **Score 2 (Poor):**
  - Communication is incomplete or incorrect, tools are misused, or
  ↪  agents fail to share necessary details.

- **Score 1 (Very Poor):**
  - No meaningful communication, hallucinated tool use, or
  ↪  completely irrelevant traces.

---
**RESPONSE FORMAT:**

Respond with a single JSON object. Do not include any other text or
↪  explanations outside the JSON.
{{
  "is_correct": true/false,
  "quality_score": 1-5,
  "reasoning": "Detailed explanation for your judgment. Justify BOTH
  ↪  the correctness of the final answer and the quality score
  ↪  based on the process trace and the rubric.",
  "answer_quality": "excellent/good/fair/poor",
  "final_answer_present": true/false,
  "partial_credit": 0.0-1.0
}}

Be thorough but fair in your evaluation. Provide specific reasoning
↪  for your judgment.
"""
```

E PROTOCOLROUTER TECHNICAL DETAILS

This section specifies the ProtocolRouterin full detail, covering the unified API, field alignment, transport and interaction semantics, reliability and ordering guarantees, identity and security, conformance testing, and known limitations. The description corresponds 1:1 to the implementation of `BaseAgent`, `BaseProtocolAdapter` and its concrete subclasses (`A2AAdapter`, `ACPAdapter`, `ANPAdapter`, `AgoraClientAdapter`). The final subsection replaces the previous router notes with a complete, self-contained router specification that sits *above* PAL and uses the same universal message envelope.

E.1 UNIFIED INTERFACE SPECIFICATION

Roles and objects.

- **BaseAgent (dual role)**: Acts as a server (receives messages) and as a multi-client (sends to multiple destinations via multiple protocols). Server responsibilities are

provided by `BaseServerAdapter` implementations (e.g., `A2AServerAdapter`, `AgentProtocolServerAdapter`, `ACPServerAdapter`, `ANPServerAdapter`). The execution entry point is SDK-native, e.g., `async def execute(context, event_queue)`.

- **BaseProtocolAdapter (egress abstraction)**: One adapter instance per egress edge (destination/URL/credentials) for isolation and precise metering. Each adapter encapsulates encoding/decoding, transport, auth, and feature negotiation for a single protocol and destination.

Unified send/receive API and lifecycle.

```
async def send_message(self, dst_id: str, payload: Dict[str, Any])
↪  -> Any
async def send_message_streaming(self, dst_id: str, payload:
↪  Dict[str, Any]
) -> AsyncIterator[Dict[str, Any]]
async def receive_message(self) -> Dict[str, Any]
async def initialize(self) -> None
async def health_check(self) -> bool
async def cleanup(self) -> None
```

- **send_message**: Sends a protocol-specific payload and returns the protocol response. PAL unifies encoding/decoding via the UTE (Unified Transport Envelope).
- **send_message_streaming** (optional): Yields protocol events/chunks as a stream (e.g., SSE).
- **receive_message**: Typically a no-op for client adapters; ANP can poll an inbound session queue.
- **initialize/health_check/cleanup**: Capability discovery/priming (cards/manifests), readiness checks, and resource teardown.

Unified Transport Envelope (UTE).

```
{
  "id": "uuid-v4",
  "ts": 1730000000.123,
  "src": "agent_A",
  "dst": "agent_B",
  "intent": "qa/search",
  "content": { "question": "..." },
  "context": {
    "trace_id": "uuid-v4",
    "parent_id": "uuid-v4",
    "idempotency_key": "uuid-v4",
    "session_id": "s-123",
    "priority": 0,
    "ttl_ms": 30000,
    "stream": false,
    "artifact_refs": ["uri://..."],
    "tags": ["gaia", "docqa"]
  },
  "meta": { "protocol_hint": "a2a|acp|anp|agora", "retry_count": 0
  ↪  }
}
```

Minimal required fields: `src`, `dst`, `content`, `context`. In `BaseAgent.send()`, `UTE.new(...)` produces the envelope that `ENCODE_TABLE[protocol_name]` transforms into protocol payload; responses are converted back via `DECODE_TABLE` into a UTE, and upper layers consume `ute_response.content`.

Table 12: UTE to protocol field alignment (send path).

UTE Field	A2A (/message)	ACP (/acp/message)	ANP (/anp/message / WS)	AGORA (task)
Shorthand: In the ANP column, leading "payload." is omitted. In the ACP/AGORA columns, leading "metadata." is omitted when applicable.				
id	request.id	id	request_id	request_id
src	params.routing.source	sender	source_id / session DID	source
dst	params.routing.destination	receiver	target_did / session	target (URL)
content	params.message	payload	payload	message / parameters
trace_id	params.context.trace_id	trace_id	trace_id	trace_id
idempotency	params.context.idempotency_key	correlation_id or idempotency_key	- idempotency_key	idempotency_key
stream	HTTP Accept: event-stream	stream=true / SSE	WS persistent stream	by type / task
session_id	params.context.session_id	session_id	connection / session	session
meta.protoc	passthrough	passthrough	enables meta-protocol	influences task

Async event model and hooks (recommended).

- *before_encode / after_encode*: UTE → protocol payload, pre/post.
- *before_transport / after_transport*: Network send/receive, pre/post.
- *on_stream_event*: Streaming fragment/event callback.
- *on_retry / on_backoff*: Retry and backoff callbacks.
- *on_decode / on_error*: Protocol response decoding and normalized error handling.

Unified metrics (e.g., `REQUEST_LATENCY`, `REQUEST_FAILURES`, `MSG_BYTES`) are labeled by `(src_agent, dst_id, protocol)`. `MSG_BYTES` reports the byte length of the serialized protocol payload.

Unified error taxonomy. Adapter exceptions are normalized by PAL into: `E_TIMEOUT`, `E_HTTP`, `E_CONN`, `E_PROTOCOL`, `E_ENCODE/DECODE`, `E_UNSUPPORTED`. PAL increments failure counters and re-raises so routing/network layers can decide on retries or failover.

E.2 Message/Event Field Alignment (A2A/ACP/ANP/AGORA → UTE)

Table 12 aligns key fields on the send path (UTE→protocol). Paths use an English `JSONPath`-like notation.

Reserved/extension notes. A2A exposes authenticated cards; ACP provides `/acp/capabilities` and `/acp/status`; ANP carries `protocol_type` (META/APPLICATION/NATURAL) and DID/WS semantics; AGORA registers routines via task decorators.

E.3 Transport and Interaction Semantics

Sync/async and streaming.

- **A2A**: HTTP sync `POST /message`; obtain SSE via `Accept: text/event-stream`.

- **ACP**: HTTP sync `POST /acp/message`; SSE supported; long-running jobs via `/acp/status` polling.
- **ANP**: WebSocket persistent sessions (`SimpleNodeSession`); HTTP fallback `POST /anp/message` for local/testing.
- **AGORA**: Official SDK task model or simplified `POST /agora` for single-round conversations and `POST /conversations/conversationId` for multi-round conversations.

Long-running job state. Native support priority: ACP (status endpoint) $>$ A2A (SSE increments/-custom heartbeats) $\approx$ ANP (session heartbeats/app-level receipts) $>$ AGORA (task-level receipts). PAL recommends `context.session_id` and `idempotency_key` as anchors for idempotency and resumption.

Artifact handling. Inline artifacts if $<$1 MB in `content`; otherwise reference via `context.artifact_refs` (e.g., `s3://` or pre-signed URLs). ANP/WS can send binary frames; for HTTP, prefer chunking or external links to avoid `max_message_size` limits.

E.4 Reliability and Ordering Guarantees

Retry/backoff and deduplication. PAL does not implicitly retry; routing/network layers decide based on error category. Idempotency is propagated via `context.idempotency_key` and mapped to protocol fields. Servers/business logic should implement deduplication on arrival.

Ordering and replay.

- **HTTP (A2A/ACP)**: Transport is unordered; applications should reorder using `seq`/`trace_id`.
- **ANP (WS)**: Within a single session, ordering is approximately sequential; across sessions/links, merge at the application layer. For SSE, `Last-Event-ID` enables replay if supported by the server.

Normalized error mapping (examples).

- `httpx.TimeoutException` → `E_TIMEOUT`
- `httpx.HTTPStatusError` → `E_HTTP` (status code and summary included)
- WS handshake/DID resolution failure → `E_CONN`
- `json.JSONDecodeError` → `E_DECODE`
- Missing/unsupported capability → `E_UNSUPPORTED`

E.5 Identity and Security

Authentication/authorization.

- **HTTP (A2A/ACP/AGORA)**: `Authorization: Bearer <token>`; recommend mTLS at gateway/reverse proxy; `/.well-known/agent.json` may expose capabilities and endpoints; A2A supports authenticated cards.
- **ANP (DID)**: `did:wba` identities; local/remote DID creation and resolution. Test setups may enable verification bypass for interoperability; production must enforce strict public-key validation and DID document checks.

End-to-end confidentiality (E2E).ANP uses ECDHE + AES-GCM for transparent per-session encryption. For HTTP protocols, use TLS/mTLS; optionally add application-layer encryption for `content` when regulatory or cross-tenant constraints apply.

Trust anchors and certificate chains.HTTP relies on public or private root CAs. DID trust anchors are the method and resolver service; cache DID documents (TTL/expiry policy) and support key rotation/revocation.

E.6 Adapter Conformance Testing

Per-protocol test suite (capability $\times$ protocol).

1. **Basic connectivity**: `initialize()` fetches cards/capabilities (A2A/ACP/AGORA), ANP establishes DID/session; `health_check()` returns true.

2. **Single round trip**: UTE↔protocol encode/decode consistency (field fidelity, null-handling policy, case conventions).
3. **Streaming**: SSE/WS event ordering, boundaries, termination (including empty lines and `data:` prefix); interruption/resume behavior.
4. **Long-running**: ACP `/acp/status` vs. A2A/ANP heartbeats/progress; resumption keyed by `session_id`.
5. **Security/auth**: Rejection on missing/invalid credentials; card access control; DID failures and certificate expiry.
6. **Edge cases**: Large messages (near `max_message_size`), high concurrency, network jitter, server 4xx/5xx/malformed JSON.

Regression corpus and coverage.

- Maintain stable wire-contract fixtures per protocol (request/response/event fragments) as baselines.
- Achieve coverage across encode/decode, error, and streaming branches.
- Fix load-test baselines and concurrency; report P50/P95/P99 and jitter coefficient (std/mean).

Known limitations and notes.

- **A2AAdapter**: `/inbox` is not universally implemented (PAL keeps a negative cache); `receive_message()` is a compatibility stub.
- **ACPAdapter**: Streaming depends on server SSE; long-running flows require `/acp/status`.
- **ANPAdapter**: Test configs may enable DID verification bypass; if no DID service is available, use HTTP fallback `POST /anp/message`; the local resolver caches target DIDs and is not a general-purpose resolver.
- **AgoraClientAdapter**: Without official `toolformer`, uses simplified HTTP with keyword classification; semantics and performance are limited.
- **Local loopback**: `IntelligentAgentNetwork._execute_single_agent_task()` may use `agent.send(agent_id, ...)` for self-delivery; the network must bind an explicit `default` adapter for that `agent_id` or provide a loopback route.
- **Ordering**: HTTP is not ordered; ANP is near-ordered per session; cross-session requires merge logic.
- **Idempotency/dedup**: Client adapters do not persist deduplication; implement on the server or one layer up.

E.7 Common Endpoints and Sample Requests (capture reference)

A2A.

- `GET /.well-known/agent.json`
- `GET /health`
- `POST /message`

```
{"id":"<uuid>","params":{"message":{"text":"..."},
"context":{"trace_id":"..."},
"routing":{"destination":"agent_B","source":"agent_A"}}}
```

ACP.

- `GET /.well-known/agent.json`
- `GET /acp/capabilities`
- `GET /acp/status`
- `POST /acp/message`

```
{"id":"<uuid>","type":"request","sender":"agent_A",
"receiver":"agent_B", "payload":{"text":"..."},
"timestamp":1730000000.0,"correlation_id":"<uuid>",
"metadata":{"trace_id":"..."}}
```

ANP.

- WS: `ws(s)://<host>:<port>/ws`
- HTTP fallback: `POST /anp/message`

```
{"type":"anp_message","request_id":"<uuid>",
"payload":{"text":"...","context":{"trace_id":"..."}},
"timestamp":1730000000.0, "source_id":"anp_client"}
```

AGORA.

- Official SDK tasks
- Single round Conversation: `POST /agora`
- Multi-round Conversation: `/conversations/conversationId`

```
{"status":"...", "body":"...}
```

E.8 Implementation Guidance and Versioning

- **Protocol name convention**: `protocol_name` is lowercase `"a2a"|"acp"|"anp"|"agora"` and must match `ENCODE_TABLE/DECODE_TABLE` keys.
- **Version negotiation**: Expose `protocolVersion` in cards; optionally include `min_version/max_version` in `context` for soft negotiation.
- **Observability and label cardinality**: Restrict metric labels to `(src_agent, dst_id, protocol)` to avoid high cardinality (e.g., dynamic URLs/tenants).
- **Rollback and canarying**: Keep old codecs and switch using `meta.protocol_hint` or advertised capabilities.
- **Production essentials**: Implement idempotency/dedup on the server (`id`/`idempotency_key`); for ANP, disable test bypasses and enforce strict DID/key governance.

E.9 Router Layer Technical Details

This subsection replaces the previous router notes with a complete, self-contained specification. The Router sits above PAL and decides *where* and *how* to send a UTE-based request. It implements destination selection, policy enforcement, resilience primitives (retry/backoff/circuit breaking/hedging), ordering semantics, and observability. It preserves PAL's security posture and never alters business semantics.

Goal and scope. Given (i) a Canonical Feature Model (protocol features) of protocol capabilities and (ii) a natural-language scenario, the router deterministically selects *exactly one* protocol per module from $\{\texttt{A2A}, \texttt{ACP}, \texttt{ANP}, \texttt{AGORA}\}$ and emits a structured decision record. A network builder then assembles homogeneous or heterogeneous links accordingly. When links are heterogeneous, messages are bridged through the *same* UTE using *stateless* encode/decode only, preserving business semantics and security attributes. By default the router runs in a *spec-only* regime (no historical numbers or hidden heuristics).

Inputs, outputs, and determinism. *Inputs:* scenario text S; module set $\mathcal{M}$; the protocol features (boolean/enumerated facets with compatibility constraints). *Output (fixed JSON):*

```
{
  "module_id": "retriever",
  "selected_protocol": "A2A|ACP|ANP|AGORA",
  "evidence_spans": ["REST", "idempotent", "no E2E"],
  "rationale": "Chosen by capability match; no numeric claims."
}
```

The router runs with temperature $= 0$; identical inputs yield identical outputs. Rationales cite only extracted evidence spans; no numeric claims or invented capabilities.

protocol features. Capabilities are organized into six facets: (1) transport & interaction (sync/async, streaming, persistent session, back-pressure); (2) long-running & artifacts (run lifecycle, status/resume, artifact refs/transfer); (3) identity & confidentiality (enterprise authN/Z, DID, E2E, mTLS); (4) delivery & replay (ordering, idempotency keys, replay/offset, dedup); (5) operation semantics (REST/idempotent/batch/resource-oriented vs. conversational/NL routines); (6) cross-org trust & governance (interop, routine governance/versioning). Hard constraints remove incompatible protocols upfront (e.g., strict E2E excludes protocols without confidentiality).

Spec-only selection pipeline. Three stages: evidence extraction → semantic mapping → candidate reduction and priority. Fixed priority for tie-breaking: (i) identity/confidentiality → (ii) operation semantics (REST/idempotent vs. conversational) → (iii) interaction preferences (streaming/long-job).

Complete function: deterministic spec-only router.

```
def route_spec_only(spec_text: str,
                    modules: list,
                    cfm: dict) -> dict:
    """
    Deterministic spec-only router: select one protocol per module.
    Returns: dict module_id -> selection_record.
    """
    spans = extract_evidence_spans(spec_text)      # ["REST",
    ↪  "idempotent", "E2E", "streaming"]
    required_caps = map_spans_to_cfm(spans, cfm)   # normalized set
    ↪  of capability flags

    decisions = {}
    for m in modules:
        candidates = [p for p in ["A2A", "ACP", "ANP", "AGORA"] if
        ↪  is_protocol_compatible(p, required_caps, cfm)]

        chosen = priority_decide(candidates, required_caps)

        if isinstance(chosen, list) and len(chosen) > 1:
            chosen = pick_by_narrative(spec_text, chosen)  #
            ↪  deterministic tie

        record = {
            "module_id": m["id"],
            "selected_protocol": chosen,
            "evidence_spans": spans,
            "rationale": "Chosen by capability match and priority
            ↪  order."
        }
        decisions[m["id"]] = record
    return decisions
```

Where to modify: adjust `priority_decide(...)` for a different priority order; extend the candidate set and `is_protocol_compatible` for new protocols.

Helper interfaces.

- `extract_evidence_spans(text)` → `List[str]`: rule/regex phrase extractor (temperature = 0).
- `map_spans_to_cfm(spans, cfm)` → `Set[cap]`: phrase → capability alignment.
- `is_protocol_compatible(proto, caps, cfm)` → `bool`: hard-constraint check.
- `priority_decide(candidates, caps)` → `str|List[str]`: fixed-priority chooser.
- `pick_by_narrative(text, candidates)` → `str`: deterministic tie-break by narrative consistency.

Communication semantics for cross-protocol links. We enforce "*change transport, not semantics or security*." Homogeneous links use the chosen protocol natively. Heterogeneous links install *stateless* bridges around the UTE:

- **Envelope (illustrative JSON).**

```
{ "id":"uuid-v4", "ts":1730000000.1, "src":"A", "dst":"B",
  "intent":"qa/search",
  "content":{ "question":"..." },
  "context":{
    "trace_id":"uuid-v4", "parent_id":"uuid-v4",
    "idempotency_key":"uuid-v4", "session_id":"s-1",
    "priority":0, "ttl_ms":30000, "stream":false,
    "artifact_refs":["uri://..."], "tags":["gaia","docqa"]
  },
  "meta":{ "protocol_hint":"a2a|acp|anp|agora", "retry_count":0 }
  ↪  }
```

- **Bridging policy**: install `encode(Envelope, proto)` and `decode(ProtoMsg)` → `Envelope` per heterogeneous edge; bridges perform only field re-mapping and semantic alignment, never altering business content or security markers.
- **Feature toggles**: if selections imply streaming/long-job/artifact/state-sync/identity/E2E, the link activates native protocol primitives (e.g., SSE/WS, status endpoints, DID+E2E).
- **Causality & errors**: messages carry unified `trace_id`/`parent_id`; errors map to a common taxonomy (timeout/HTTP/connection/codec/unsupported).

Router base interface.

```
class BaseRouter(Protocol):
    async def route(self, ute: Dict[str, Any]) -> Dict[str, Any]:
    ↪  ...
    async def route_streaming(self, ute: Dict[str, Any]
    ) -> AsyncIterator[Dict[str, Any]]: ...
    async def health(self) -> Dict[str, Any]: ...
```

Policies and resilience. Selection policies: static *first-match*; weighted; latency-aware (EWMA/percentile-aware); consistent hashing by `session_id`/`trace_id`. Resilience primitives: jittered exponential backoff; hedging with cancel-on-first-success; circuit breaking (open/half-open/close); bulkheading via per-slot concurrency caps. Ordering can be enforced with per-`trace_id`/`session_id` work queues; idempotency is preserved via `context.idempotency_key` and an optional client-side request cache.

Deterministic tie-break with a protocol-level prior (optional).

```
def tie_break_with_prior(candidates: list, prior_table: dict) ->
↪  str:
    """
    Deterministic tie-break with a protocol-level prior.
    No numeric values are surfaced in the rationale.
    """
    ranking = prior_table.get("ranking",
    ↪  ["A2A","ACP","ANP","AGORA"])
    ranked = sorted(candidates, key=lambda p: ranking.index(p)
                    if p in ranking else len(ranking))
    return ranked[0]
```

Online bandit overlay (optional). After hard-constraint pruning, a contextual bandit (e.g., Thompson sampling) may choose among feasible protocols using runtime feedback while *respecting* all security/semantic invariants.

```
def bandit_select(feasible: list, context: dict, posterior: dict,
↪  rng) -> str:
    """
    Thompson sampling over feasible protocols.
    Security/semantic constraints are enforced upstream.
    """
    draws = {}
    for p in feasible:
        a, b = posterior.get(p, (1.0, 1.0))  # Beta prior
        draws[p] = rng.beta(a, b)
    best = sorted(draws.items(), key=lambda kv: (-kv[1],
    ↪  kv[0]))[0][0]
    return best
```

From decisions to network (complete function).

```
def apply_router_decisions(decisions: dict,
                           modules: list) -> dict:
    """
    Build a protocol-consistent topology and link configs
    from router decisions. Stateless bridging is toggled
    for heterogeneous links; native features are enabled
    per-link according to the chosen protocol.
    Returns: { "nodes": [...], "links": [...], "bridges": [...] }.
    """
    nodes, links, bridges = [], [], []
    proto_of = {d["module_id"]: d["selected_protocol"]
                for d in decisions.values()} if
                ↪  isinstance(decisions, dict) \
                else {k: v["selected_protocol"] for k, v in
                ↪  decisions.items()}

    for m in modules:
        nodes.append({
            "id": m["id"],
            "protocol": proto_of[m["id"]],
            "features": decide_native_features(proto_of[m["id"]],
            ↪  m)
        })

    # create links according to scenario-defined topology
    for m in modules:
        for nbr in m.get("neighbors", []):
            src_p, dst_p = proto_of[m["id"]], proto_of[nbr]
            links.append({"src": m["id"], "dst": nbr, "protocol":
            ↪  (src_p, dst_p)})
            if src_p != dst_p:
                bridges.append({
                    "src": m["id"], "dst": nbr,
                    "encode": f"encode_to_{dst_p.lower()}",
                    "decode": f"decode_from_{src_p.lower()}",
                    "stateless": True
                })
    return {"nodes": nodes, "links": links, "bridges": bridges}
```

Security posture and observability. Routers must not downgrade PAL security: preserve `Authorization` headers, mTLS bindings, and ANP DID constraints. Observability exports `ROUTER_DECISIONS`, `HEDGE_FIRES`, `CIRCUIT_STATE`, `QUEUE_DEPTH`, end-to-end `REQUEST_LATENCY`; all correlated via `trace_id`.

Testing matrix.

- **Policy conformance**: selection, sticky sessions, hedging, retry categories.
- **Failure drills**: open circuit, half-open probes, bulkhead saturation.
- **Ordering**: monotonic sequence under enforced queues.
- **Streaming**: hedged streams deduplicated; cancellation correctness.

E.10 Router Prompts

E.10.1 Fail Storm Router Prompt

Fail Storm Router Prompt

```
You are "ProtoRouter", a deterministic and evaluation-friendly protocol selector for multi-agent systems.
Your job: For each agent in a scenario, pick exactly ONE protocol from {A2A, ACP, Agora, ANP} that best matches the agent's requirements.
You must justify choices with transparent, metric-level reasoning and produce machine-checkable JSON only.

------------------------------------------
1) Canonical Feature Model (authoritative; use this only)
------------------------------------------
A2A (Agent-to-Agent Protocol)
- Transport/Model: HTTP + JSON-RPC + SSE; first-class long-running tasks; task/artifact lifecycle.
- Performance: avg 3.42-7.39s response, 6.0s recovery time (fastest), 59.6% success rate
- Capability/UX: Multimodal messages (text/audio/video) and explicit UI capability negotiation.
- Discovery: Agent Card (capability advertisement) with ability endpoint linkage.
- Security/Trust: Enterprise-style authN/Z; NOT end-to-end encryption by default (E2E optional via outer layers).
- Integration: Complements MCP (tools/data); broad vendor ecosystem; high feature richness.
- Typical Strengths: enterprise integration, complex workflows, multimodal streaming, UI handshakes, long jobs, fast recovery.
- Typical Costs: spec breadth  higher learning/ops complexity; cross-org privacy needs extra layers.
- Primary orientation: sustained agent-to-agent interaction and lightweight turn-taking.
- Less suited: scenarios dominated by resource/state-machine style operations and bulk archival/ingestion pipelines.

ACP (Agent Communication Protocol)
- Transport/Model: REST-first over HTTP; MIME-based multimodality; async-first with streaming support.
- Performance: avg 4.00-7.83s response, 8.0s recovery time, 59.0% success rate
- Discovery: Agent Manifest & offline discovery options; clear single/multi-server topologies.
- Security/Trust: Relies on web auth patterns; E2E not native.
- Integration: Minimal SDK expectations; straightforward REST exposure.
- Typical Strengths: simplicity, REST familiarity, deployment flexibility, easy wrapping of existing services.
- Typical Costs: less emphasis on UI capability negotiation; moderate recovery performance.
- Primary orientation: structured, addressable operations with clear progress semantics and repeatable handling at scale.
- Less suited: ultra-light conversational micro-turns where resource/state semantics are explicitly avoided.

Agora (Meta-Protocol)
- Positioning: Minimal "meta" wrapper; sessions carry a protocolHash binding to a plain-text protocol doc.
- Performance: avg 7.10-9.00s response, 6.1s recovery time, 60.0% success rate
- Discovery: /.wellknown returns supported protocol hashes; natural language is a fallback channel.
- Evolution: Encourages reusable "routines"; fast protocol evolution and heterogeneity tolerance.
```

```
- Security/Trust: No strong identity/E2E built-in; depends on
↪  deployment or upper layers.
- Typical Strengths: lightweight, negotiation-friendly, highly
↪  adaptable for research/decentralized experiments, balanced
↪  recovery.
- Typical Costs: governance/audit features not built-in;
↪  production-grade security must be composed.
- Primary orientation: explicit procedure governance  selecting and
↪  following a concrete routine/version that must be auditable.
- Less suited: when no concrete procedure/version needs to be fixed
↪  or referenced.

ANP (Agent Network Protocol)
- Positioning: Network & trust substrate for agents; three layers:
↪  identity+E2E, meta-protocol, application protocols.
- Performance: avg 4.78-6.76s response, 10.0s recovery time
↪  (slowest), 61.0% success rate (highest), 22.0% answer discovery
↪  rate (highest)
- Security/Trust: W3C DID-based identities; ECDHE-based end-to-end
↪  encryption; cross-org/verifiable comms.
- Discovery/Semantics: Descriptions for capabilities & protocols;
↪  supports multi-topology communications.
- Typical Strengths: strong identity, E2E privacy,
↪  cross-organization trust, highest answer discovery rate.
- Typical Costs: DID/keys lifecycle adds integration/ops complexity;
↪  ecosystem still maturing; UI/multimodal not first-class; slowest
↪  recovery.
- Primary orientation: relationship assurance and information
↪  protection across boundaries (identity, confidentiality,
↪  non-repudiation).
- Less suited: purely local/benign traffic where verifiable identity
↪  and confidentiality are not primary concerns.

--------------------------------------------
3) Protocol Selection Task
--------------------------------------------

**Scenario Description:**
Multi-agent distributed document search system operating under
↪  cyclic fault injection conditions. The system must maintain high
↪  answer discovery rates while minimizing recovery time during
↪  agent failures. Agents are organized in a mesh topology where 3
↪  out of 8 agents are killed every 120 seconds, requiring rapid
↪  fault detection, recovery, and service restoration.

**Module Details:**
**Module 1: Fault-Tolerant Document Search Network**
- Agents: Agent-1, Agent-2, Agent-3, Agent-4, Agent-5, Agent-6,
↪  Agent-7, Agent-8
- Protocol Selection: Choose 1 protocol(s) from A2A, ACP, Agora, ANP

**Tasks:**
- Perform distributed document fragment search across 8 agents in
↪  mesh topology.
- Maintain collaborative retrieval with TTL-based message forwarding
↪  and ring communication.
- Detect agent failures through heartbeat monitoring (10s intervals,
↪  30s timeout).
- Execute rapid reconnection and service restoration after fault
↪  injection.
- Preserve answer discovery capability during 3-agent simultaneous
↪  failures.
- Support coordinator-worker communication for result aggregation.
```

```
- Handle cyclic fault patterns with 120s intervals over extended
↪  runtime (1800s).

**Potential Issues:**
- Simultaneous failure of 37.5% of agents (3/8) every 120 seconds.
- Network partitions during fault injection causing message loss.
- Recovery time bottlenecks affecting overall system availability.
- Duplicate work during recovery phases reducing efficiency.
- Answer quality degradation under reduced agent availability.
- Heartbeat timeout false positives during network jitter.
- Reconnection storms when multiple agents recover simultaneously.
- TTL exhaustion in message forwarding during network instability.

**Your Task:**
For each module in this scenario, you must select exactly ONE
↪  protocol from {A2A, ACP, Agora, ANP} that best matches the
↪  module's requirements.

You must respond using the protocol_selection function call with
↪  your analysis and selections.
```

E.10.2 Streaming Queue Router Prompt

Streaming Queue Router Prompt

```
You are "ProtoRouter", a deterministic and evaluation-friendly
↪  protocol selector for multi-agent systems.
Your job: For each agent in a scenario, pick exactly ONE protocol
↪  from {A2A, ACP, Agora, ANP} that best matches the agent's
↪  requirements.
You must justify choices with transparent, metric-level reasoning
↪  and produce machine-checkable JSON only.

------------------------------------------
1) Canonical Feature Model (authoritative; use this only)
------------------------------------------
A2A (Agent-to-Agent Protocol)
- Transport/Model: HTTP + JSON-RPC + SSE; first-class long-running
↪  tasks; task/artifact lifecycle.
- Performance: avg 3.42-7.39s response, 6.0s recovery time
↪  (fastest), 59.6% success rate
- Capability/UX: Multimodal messages (text/audio/video) and explicit
↪  UI capability negotiation.
- Discovery: Agent Card (capability advertisement) with ability
↪  endpoint linkage.
- Security/Trust: Enterprise-style authN/Z; NOT end-to-end
↪  encryption by default (E2E optional via outer layers).
- Integration: Complements MCP (tools/data); broad vendor ecosystem;
↪  high feature richness.
- Typical Strengths: enterprise integration, complex workflows,
↪  multimodal streaming, UI handshakes, long jobs, fast recovery.
- Typical Costs: spec breadth  higher learning/ops complexity;
↪  cross-org privacy needs extra layers.
- Primary orientation: sustained agent-to-agent interaction and
↪  lightweight turn-taking.
- Less suited: scenarios dominated by resource/state-machine style
↪  operations and bulk archival/ingestion pipelines.

ACP (Agent Communication Protocol)
- Transport/Model: REST-first over HTTP; MIME-based multimodality;
↪  async-first with streaming support.
```

```
- Performance: avg 4.00-7.83s response, 8.0s recovery time, 59.0%
↪  success rate
- Discovery: Agent Manifest & offline discovery options; clear
↪  single/multi-server topologies.
- Security/Trust: Relies on web auth patterns; E2E not native.
- Integration: Minimal SDK expectations; straightforward REST
↪  exposure.
- Typical Strengths: simplicity, REST familiarity, deployment
↪  flexibility, easy wrapping of existing services.
- Typical Costs: less emphasis on UI capability negotiation;
↪  moderate recovery performance.
- Primary orientation: structured, addressable operations with clear
↪  progress semantics and repeatable handling at scale.
- Less suited: ultra-light conversational micro-turns where
↪  resource/state semantics are explicitly avoided.

Agora (Meta-Protocol)
- Positioning: Minimal "meta" wrapper; sessions carry a protocolHash
↪  binding to a plain-text protocol doc.
- Performance: avg 7.10-9.00s response, 6.1s recovery time, 60.0%
↪  success rate
- Discovery: /.wellknown returns supported protocol hashes; natural
↪  language is a fallback channel.
- Evolution: Encourages reusable "routines"; fast protocol evolution
↪  and heterogeneity tolerance.
- Security/Trust: No strong identity/E2E built-in; depends on
↪  deployment or upper layers.
- Typical Strengths: lightweight, negotiation-friendly, highly
↪  adaptable for research/decentralized experiments, balanced
↪  recovery.
- Typical Costs: governance/audit features not built-in;
↪  production-grade security must be composed.
- Primary orientation: explicit procedure governance  selecting and
↪  following a concrete routine/version that must be auditable.
- Less suited: when no concrete procedure/version needs to be fixed
↪  or referenced.

ANP (Agent Network Protocol)
- Positioning: Network & trust substrate for agents; three layers:
↪  identity+E2E, meta-protocol, application protocols.
- Performance: avg 4.78-6.76s response, 10.0s recovery time
↪  (slowest), 61.0% success rate (highest), 22.0% answer discovery
↪  rate (highest)
- Security/Trust: W3C DID-based identities; ECDHE-based end-to-end
↪  encryption; cross-org/verifiable comms.
- Discovery/Semantics: Descriptions for capabilities & protocols;
↪  supports multi-topology communications.
- Typical Strengths: strong identity, E2E privacy,
↪  cross-organization trust, highest answer discovery rate.
- Typical Costs: DID/keys lifecycle adds integration/ops complexity;
↪  ecosystem still maturing; UI/multimodal not first-class; slowest
↪  recovery.
- Primary orientation: relationship assurance and information
↪  protection across boundaries (identity, confidentiality,
↪  non-repudiation).
- Less suited: purely local/benign traffic where verifiable identity
↪  and confidentiality are not primary concerns.

-------------------------------------------
3) Protocol Selection Task
-------------------------------------------

**Scenario Description:**
```

```
High-throughput question-answering system designed for streaming
↪  queue pressure testing. The system processes batches of
↪  questions (50 per batch) across multiple worker agents
↪  coordinated by a central coordinator in star topology. Primary
↪  focus is minimizing end-to-end latency while maintaining
↪  acceptable reliability under concurrent load.

**Module Details:**
**Module 1: High-Throughput QA Processing Pipeline**
- Agents: Coordinator-1, Worker-1, Worker-2, Worker-3, Worker-4
- Protocol Selection: Choose 1 protocol(s) from A2A, ACP, Agora, ANP

**Tasks:**
- Coordinator loads question batches from JSONL dataset
↪  (top1000_simplified.jsonl).
- Dynamic load balancing across 4 worker agents using queue-based
↪  task distribution.
- Workers process questions with LLM inference and return structured
↪  responses.
- Maintain response time constraints (60s timeout) with retry
↪  mechanisms (max 3 retries).
- Collect and aggregate results with comprehensive performance
↪  metrics.
- Support concurrent processing with batch sizes of 5 questions per
↪  worker.
- Generate detailed performance reports including latency
↪  distribution and success rates.

**Potential Issues:**
- High concurrent load causing worker saturation and queue backups.
- Network timeout errors under sustained throughput pressure.
- Load imbalance between workers leading to processing bottlenecks.
- Connection retry storms during network instability.
- Response time variance affecting P95/P99 latency targets.
- Worker failure during batch processing causing partial results
↪  loss.
- Memory pressure from large question batches and response
↪  buffering.
- Protocol overhead impacting raw throughput under high QPS
↪  scenarios.

**Your Task:**
For each module in this scenario, you must select exactly ONE
↪  protocol from {A2A, ACP, Agora, ANP} that best matches the
↪  module's requirements.

You must respond using the protocol_selection function call with
↪  your analysis and selections.
```

E.10.3 PROTOCOLROUTERBENCH INSTRUCTION PROMPT

ProtocolRouterBench Instruction

```
You are "ProtoRouter", a deterministic and evaluation-friendly
↪  protocol selector for multi-agent systems.
Your job: For each agent in a scenario, pick exactly ONE protocol
↪  from {A2A, ACP, Agora, ANP} that best matches the agent's
↪  requirements.
You must justify choices with transparent, metric-level reasoning
↪  and produce machine-checkable JSON only.

------------------------------------------
1) Canonical Feature Model (authoritative; use this only)
```

```
------------------------------------------
A2A (Agent-to-Agent Protocol)
- Transport/Model: HTTP + JSON-RPC + SSE; first-class long-running
↪  tasks; task/artifact lifecycle.
- Capability/UX: Multimodal messages (text/audio/video) and explicit
↪  UI capability negotiation.
- Discovery: Agent Card (capability advertisement) with ability
↪  endpoint linkage.
- Security/Trust: Enterprise-style authN/Z; NOT end-to-end
↪  encryption by default (E2E optional via outer layers).
- Integration: Complements MCP (tools/data); broad vendor ecosystem;
↪  high feature richness.
- Primary orientation: sustained agent-to-agent interaction and
↪  lightweight turn-taking.
- Less suited: resource/state-machine heavy pipelines and bulk
↪  archival ingestion.

ACP (Agent Communication Protocol)
- Transport/Model: REST-first over HTTP; MIME-based multimodality;
↪  async-first with streaming support.
- Discovery: Agent Manifest & offline discovery options; clear
↪  single/multi-server topologies.
- Security/Trust: Web auth patterns; E2E not native.
- Integration: Minimal SDK expectations; straightforward REST
↪  exposure.
- Primary orientation: structured, addressable operations with clear
↪  progress semantics at scale.
- Less suited: ultra-light conversational micro-turns that avoid
↪  resource/state semantics.

Agora (Meta-Protocol)
- Positioning: Minimal meta wrapper; sessions carry a protocolHash
↪  bound to a plain-text protocol document.
- Discovery: /.well-known returns supported protocol hashes; natural
↪  language as fallback.
- Evolution: Reusable routines; fast protocol evolution and
↪  heterogeneity tolerance.
- Security/Trust: No strong identity/E2E built-in; depends on
↪  deployment or upper layers.
- Primary orientation: explicit procedure governance (choose and
↪  follow a concrete routine/version).
- Less suited: when no procedure/version needs to be fixed or
↪  referenced.

ANP (Agent Network Protocol)
- Positioning: Network & trust substrate; three layers:
↪  identity+E2E, meta-protocol, application protocols.
- Security/Trust: W3C DID identities; ECDHE-based end-to-end
↪  encryption; cross-org/verifiable comms.
- Discovery/Semantics: Descriptions for capabilities & protocols;
↪  supports multi-topology communications.
- Primary orientation: relationship assurance across boundaries
↪  (identity, confidentiality, non-repudiation).
- Less suited: benign/local traffic where verifiable identity and
↪  confidentiality are not primary concerns.

------------------------------------------
2) Protocol Selection Task
------------------------------------------
**Scenario Description:** {scenario_description}
**Module Details:** {module_details}
```

```
**Your Task:** For each module in this scenario, you must select
↪  exactly ONE protocol from {A2A, ACP, Agora, ANP} that best
↪  matches the module's requirements.
You must respond using the protocol_selection function call with
↪  your analysis and selections (machine-checkable JSON only).
```

E.10.4 PROTOCOLROUTERBENCH INSTRUCTION PROMPT(SPEC + PERF)

ProtocolRouterBench Instruction (Spec + Perf)

```
You are "ProtoRouter", a deterministic and evaluation-friendly
↪  protocol selector for multi-agent systems.
Your job: For each agent in a scenario, pick exactly ONE protocol
↪  from {A2A, ACP, Agora, ANP} that best matches the agent's
↪  requirements.
You must justify choices with transparent, metric-level reasoning
↪  and produce machine-checkable JSON only.

------------------------------------------
1) Canonical Feature Model (authoritative; use this only)
------------------------------------------
A2A (Agent-to-Agent Protocol)
- Transport/Model: HTTP + JSON-RPC + SSE; long-running tasks;
↪  task/artifact lifecycle.
- Capability/UX: Multimodal messages; explicit UI capability
↪  negotiation.
- Discovery: Agent Card with ability → endpoint linkage.
- Security/Trust: Enterprise authN/Z; E2E not default (optional via
↪  outer layers).
- Integration: Complements MCP; broad ecosystem.
- Orientation: sustained agent interaction and lightweight
↪  turn-taking.

ACP (Agent Communication Protocol)
- Transport/Model: REST-first; MIME multimodality; async-first with
↪  streaming.
- Discovery: Agent Manifest; single/multi-server topologies.
- Security/Trust: Web auth patterns; E2E not native.
- Integration: Minimal SDK; easy REST wrapping.
- Orientation: structured, addressable operations with clear
↪  progress semantics.

Agora (Meta-Protocol)
- Positioning: Meta wrapper; session binds to a protocolHash
↪  referencing a routine document.
- Discovery: /.well-known hashes; NL fallback.
- Security/Trust: Depends on deployment; no strong identity/E2E
↪  built-in.
- Orientation: explicit routine/version governance and auditability.

ANP (Agent Network Protocol)
- Positioning: Identity+E2E substrate; meta-protocol; application
↪  protocols.
- Security/Trust: W3C DID; ECDHE E2E; cross-org/verifiable
↪  communications.
- Orientation: boundary-crossing
↪  identity/confidentiality/non-repudiation.

------------------------------------------
2) Protocol performance in some scenarios
------------------------------------------
[
```

```
{
  "id": "G1-QA",
  "description": "GAIA hierarchical DocQA with planning, explicit
  ↪  workflow/message-flow, sandboxed tools, step memory, and LLM
  ↪  judging.",
  "modules_count": 1,
  "module": [
    {
      "name": "Hierarchical DocQA Pipeline",
      "agents": ["Planner","Reader/Extractor","Aggregator/Summari⌋
      ↪  zer","Judge"],
      "protocol_selection": {"choices":
      ↪  ["A2A","ANP","ACP","Agora"], "select_exactly": 1},
      "tasks": [
        "Emit machine-readable manifest (roles, tools,
        ↪  workflow).",
        "Run P2P serving with explicit message-flow.",
        "Record step-based memory with timestamps and tool-call
        ↪  traces.",
        "Summarize and judge quality; emit metrics."
      ],
      "potential_issues": [
        "Long-running tasks with streaming outputs/partials.",
        "Out-of-order or retried deliveries under concurrency.",
        "Auditability and replay of full execution log.",
        "Cross-run fairness (identical seed/config)."
      ]
    }
  ],
  "experiment_results": {
    "quality_avg": {"acp": 2.27, "a2a": 2.51, "anp": 2.14,
    ↪  "agora": 2.33, "meta": 2.50},
    "success_avg": {"acp": 5.25, "a2a": 9.29, "anp": 7.28,
    ↪  "agora": 6.27, "meta": 9.90},
    "single_task_comm_time@5_example": {
      "a2a_ms": [25.38, 20.64, 28.19, 21.65, 21.36],
      "acp_ms": [15.30, 13.64, 14.75, 16.22, 12.75],
      "anp_ms": [39.01, 54.74, 27.60, 21.86, 34.48],
      "agora_ms": [29.30, 21.83, 30.49, 22.41, 35.50]
    }
  }
},
{
  "id": "S1-Queue",
  "description": "Streaming Queue: centralized 5-agent network;
  ↪  1000 items; pressure test for speed and stability.",
  "modules_count": 1,
  "module": [
    {
      "name": "Coordinator-Workers Streaming Queue",
      "agents": ["Coordinator","Worker-1","Worker-2","Worker-3","⌋
      ↪  Worker-4"],
      "protocol_selection": {"choices":
      ↪  ["A2A","ANP","ACP","Agora"], "select_exactly": 1},
      "tasks": ["Load-balance tasks","Track per-task latency and
      ↪  completion","Minimize worker variance","Measure
      ↪  errors/retries/timeouts"]
    }
  ],
  "experiment_results": {
    "performance": {
      "A2A": {"total":1000,"duration_s":2427,"avg_ms":9698,"min_m⌋
      ↪  s":6938,"max_ms":15129,"std_ms":1127},
```

```
        "ACP": {"total":1000,"duration_s":2417,"avg_ms":9663,"min_m
        ↪  s":6881,"max_ms":14235,"std_ms":1077},
        "ANP": {"total":1000,"duration_s":2843,"avg_ms":11364,"min_
        ↪  ms":243,"max_ms":50104,"std_ms":5732},
        "Agora":{"total":1000,"duration_s":3298,"avg_ms":13135,"min
        ↪  _ms":524,"max_ms":28213,"std_ms":5089}
      }
    }
  },
  {
    "id": "F1-Storm",
    "description": "Fail Storm on ring-structured Shard QA; randomly
    ↪  kill 3 agents every 2 minutes; measure recovery and pre/post
    ↪  metrics.",
    "modules_count": 1,
    "module": [
      {
        "name": "Shard QA with Fault Injection",
        "agents": ["QA-1","QA-2","QA-3","QA-4","QA-5","QA-6","QA-7"
        ↪  ,"QA-8"],
        "protocol_selection": {"choices":
        ↪  ["A2A","ANP","ACP","Agora"], "select_exactly": 1}
      }
    ],
    "experiment_results": {
      "performance": [
        {"protocol":"ACP",  "answer_found_pct_pre":14.76,"answer_fo
        ↪  und_pct_post":13.64,"steady_latency_s_pre":4.3776,"stea
        ↪  dy_latency_s_post":4.1851,"recovery_s":8.0482},
        {"protocol":"A2A",  "answer_found_pct_pre":14.74,"answer_fo
        ↪  und_pct_post":14.57,"steady_latency_s_pre":4.3399,"stea
        ↪  dy_latency_s_post":4.1855,"recovery_s":8.0027},
        {"protocol":"ANP",  "answer_found_pct_pre":14.88,"answer_fo
        ↪  und_pct_post":12.94,"steady_latency_s_pre":4.3428,"stea
        ↪  dy_latency_s_post":4.1826,"recovery_s":8.0033},{"protoc
        ↪  ol":"AGORA","answer_found_pct_pre":14.91,"answer_found_
        ↪  pct_post":12.12,"steady_latency_s_pre":4.3311,"steady_l
        ↪  atency_s_post":4.1799,"recovery_s":8.0026}
      ]
    }
  },
  {
    "id": "M1-Doctors",
    "description": "Doctor-to-doctor dialogue system with two
    ↪  legitimate LLM agents; multi-round consultations.",
    "modules_count": 1,
    "module": [
      {
        "name": "Doctor-Doctor Dialogue System",
        "agents": ["Doctor A","Doctor B"],
        "protocol_selection": {"choices":
        ↪  ["A2A","ANP","ACP","Agora"], "select_exactly": 1}
      }
    ],
    "experiment_results": {
      "safety_matrix": [
      ↪  {"protocol":"Agora","tls_transport":true,"session_hijack_
      ↪  protection":true,"e2e_detection":false,"packet_tunnel_pro
      ↪  tection":true,"metadata_exposure_protection":true},
        {"protocol":"ANP",  "tls_transport":true,"session_hijack_pr
        ↪  otection":true,"e2e_detection":true,
        ↪  "packet_tunnel_protection":true,"metadata_exposure_prot
        ↪  ection":true},
```

```
        {"protocol":"ACP",  "tls_transport":false,"session_hijack_p
        ↪  rotection":true,"e2e_detection":true,
        ↪  "packet_tunnel_protection":false,"metadata_exposure_pro
        ↪  tection":true},
        {"protocol":"A2A",  "tls_transport":false,"session_hijack_p
        ↪  rotection":true,"e2e_detection":true,
        ↪  "packet_tunnel_protection":false,"metadata_exposure_pro
        ↪  tection":true}
      ]
    }
  }
]

-------------------------------------------
3) Protocol Selection Task
-------------------------------------------
**Scenario Description:** {scenario_description}
**Module Details:** {module_details}

IMPORTANT: Provide a selection for EVERY module. Use the
↪  protocol_selection function call with analysis and selections
↪  (machine-checkable JSON only).
```