TCP-MCP: LANDSCAPE-GUIDED CO-EVOLUTION OF PROMPTS AND COMMUNICATION TOPOLOGIES FOR MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Effective multi-agent systems cannot be designed by selecting prompts or communication graphs in isolation. Agent behavior depends on the information an agent receives, while the usefulness of a communication edge depends on how the receiving agent interprets and uses that information. We propose **TCP-MCP** (Topology-Coupled Prompting for Multi-Agent Collaborative Problem-Solving), a co-evolution framework that searches agent prompts and communication topologies as a unified genome. TCP-MCP couples this executable representation with a structure-aware evolutionary search procedure, using an initialization-time landscape probe to calibrate early search behavior and Pareto-front diagnostics to adapt exploration under three objectives: task performance, token cost, and structural complexity. Using the same DeepSeek-V3.2 backbone across all methods, TCP-MCP achieves 82.66%, 89.96%, 96.61%, and 77.31% accuracy on MMLU-Pro, MMLU, GSM8K, and WorkBench, respectively. Across the four benchmarks, it consistently outperforms automated graph-generation baselines and achieves competitive accuracy relative to debate-style systems, while using up to 5.71$\times$ fewer tokens than those systems at the reported operating points. These results show that jointly evolving prompts and communication structure provides a practical route to cost-aware and task-adaptive multi-agent system design in controlled evaluations.

1 INTRODUCTION

LLM-based multi-agent systems (MAS) support reasoning by assigning roles, exchanging intermediate reasoning, and aggregating outputs. As these systems become more capable, performance depends not only on each agent's prompt, but also on how agents are organized, what information passes between them, and which topology governs exchange. MAS construction is therefore a system-level design problem, rather than a direct extension of single-agent prompt engineering.

Many existing approaches, however, often fix the communication structure, optimize topology separately from prompts, or rely on reusable templates. This separation misses a central property of MAS design: prompt behavior and topology are coupled. A prompt that works under one communication pattern may fail under another, and an edge is useful only when the receiving agent is instructed to interpret and use the transmitted reasoning. This coupling raises two challenges: how to move beyond fixed architectural priors and optimize communication structures together with prompts, and how to search the high-dimensional prompt–topology space without converging too early.

We propose TCP-MCP, a landscape-guided co-evolution framework that treats prompts and communication topology as a coupled executable design object. The key methodological idea is to encode node-level agent behavior and inter-agent communication structure within the same genome, so that structural changes and prompt behavior can be evaluated and optimized jointly rather than in separate search stages. TCP-MCP searches this coupled space through structure-aware crossover and mutation, prompt inheritance and adaptation, and Pareto-guided exploration. It returns a Pareto set of reusable MAS designs over task performance, token cost, and structural complexity, from which a representative genome is selected and frozen for deployment across unseen queries.

Our contributions are as follows:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

- **Coupled Executable Genome.** We formulate multi-agent system design through an executable genome that couples node-level role–prompt configurations with communication topology.
- **Structure-Aware Co-Evolution.** We develop TCP-MCP to jointly search prompts and topology using structure-aware evolutionary operators and Pareto optimization.
- **Experimental Validation.** We evaluate TCP-MCP on MMLU, MMLU-Pro, GSM8K, and WorkBench, showing improvements over G-Designer and clear performance–cost trade-offs.

2 Related Work

General frameworks such as AutoGen Wu et al. (2024), ChatDev Qian et al. (2024), and MetaGPT Hong et al. (2024) simplify multi-agent workflow construction, while dynamic collaboration methods such as DyLAN Liu et al. (2024) adapt agent teams and communication structures to the task. Topology-oriented methods such as GPTSwarm Zhuge et al. (2024) and G-Designer Zhang et al. (2025) further automate communication-structure design, while EvoMAS (Hu et al., 2026) extends evolutionary adaptation to structured multi-agent configurations. These methods establish strong foundations for automated MAS construction, but differ in how agent behavior and communication structure are represented, optimized, and reused across queries.

TCP-MCP considers a design and deployment regime by jointly optimizing role–prompt configurations and communication topology as a coupled executable genome. This representation allows prompt behavior and information flow to be evaluated together under task performance, token cost, and structural complexity objectives. Search is performed on development data, after which the selected genome is frozen and reused across unseen queries without query-specific evolutionary search. This separates one-time system design from repeated deployment and enables the searched structure to be amortized across queries. A detailed discussion of single-agent reasoning, multi-agent collaboration, topology optimization, and evolutionary MAS design is provided in Appendix A.

3 Preliminaries

3.1 Notations and Problem Formulation

We model a multi-agent system (MAS) as a directed graph $G = (V, E)$, where each node is a role-specific agent and each edge specifies a communication link. Each node $v_i \in V$ is associated with a role–prompt pair $(role_i, p_i)$, and a candidate system is represented by a unified genome:

$$M = (G, P), \tag{1}$$

where P contains node-level role–prompt assignments. In TCP-MCP, prompt templates are stored independently of topology, but are instantiated at runtime with topology-dependent context:

$$\tilde{p}_i = \text{BuildPrompt}\left(p_i, N^-(v_i; G)\right), \tag{2}$$

where $N^-(v_i; G)$ is the incoming neighborhood of v_i. As a result, the behavior of a node can change with G even when its stored template p_i is inherited without modification. The final prediction is produced by a decision node v_{dec}, which aggregates the relevant outputs from sink nodes.

Prompt behavior depends on communication context, and the utility of an edge depends on the behavior induced at the receiving node. MAS design is therefore a coupled optimization problem over prompts and topology. Given an evaluation set D, we optimize task accuracy, token cost, and structural complexity. Let $A(M)$ denote task accuracy, $C(M)$ denote token cost, and $K(G) = |V| + |E|$ denote graph complexity. We define the multi-objective maximization problem as:

$$F(M) = (A(M), -C(M), -\ln(1 + K(G))). \tag{3}$$

The logarithmic transformation compresses large differences in structural complexity while preserving relative ordering, providing a more balanced numerical scale for downstream multi-objective diagnostics and diversity calculations.

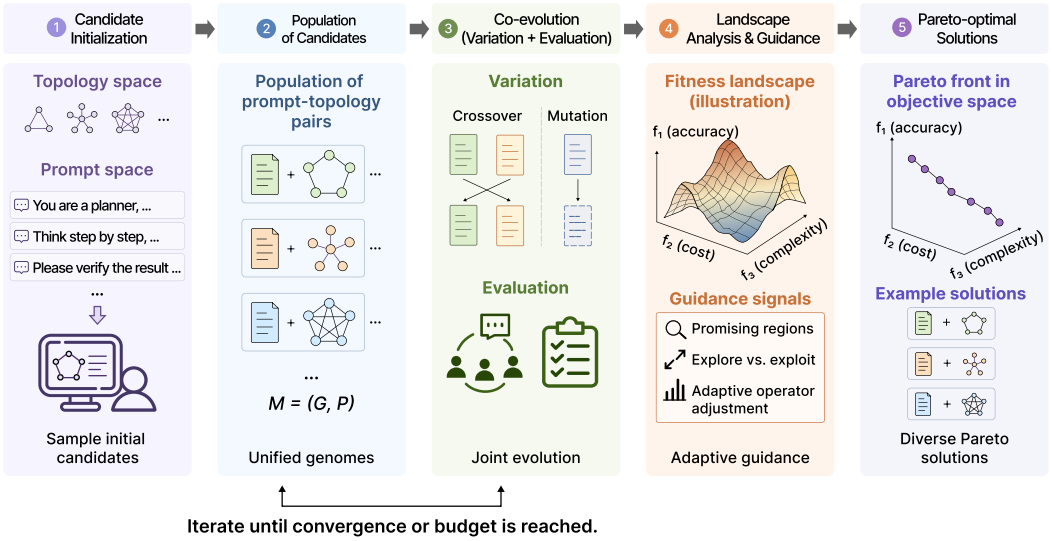

Figure 1: Overall framework of TCP-MCP for joint prompt–topology co-evolution. Each candidate is represented as a unified prompt–topology genome. The numbered blocks show the main functional modules; the text groups them into initialization, co-evolution, and final Pareto selection.

3.2 Design and Deployment Protocol.

TCP-MCP optimizes a Pareto set of genomes on a development set drawn from the target task distribution. A genome $M^\star = (G^\star, P^\star)$ is selected on the development set and frozen before held-out evaluation. The same genome is then reused across test queries without query-specific evolutionary search, topology construction, prompt optimization, or system selection. Consequently, the evolutionary search cost is incurred during system design and amortized across queries, while deployment requires execution of the selected system. Runtime prompt instantiation incorporates the current query and incoming messages but does not modify the searched genome.

3.3 Initialization-Time Landscape Probe

The unified genome defines a hybrid search space $\mathcal{S} = \mathcal{G} \times \Theta$ over communication topologies and role–prompt configurations. To obtain a coarse signal of the initial search landscape, TCP-MCP computes Fitness Distance Correlation (FDC) (Jones & Forrest, 1995) over the initialized population:

$$\mathrm{FDC} = \mathrm{Spearman}\left(\{S(M_j)\}_{j=1}^{n_0}, \{d(M_j, M_0^\star)\}_{j=1}^{n_0}\right), \tag{4}$$

where $S(M)$ is an auxiliary preference score combining task performance, token cost, and structural complexity, and $M_0^\star$ is the highest-scoring initial candidate. FDC is computed only once after initialization to calibrate the early search bias. The preference-score definition and implementation details are provided in Appendix B.

4 Methods

4.1 Overall Framework

As shown in Figure 1, TCP-MCP formulates MAS construction as a system-level design problem. Each candidate is encoded as a genome $M = (G, P)$, where G specifies the communication topology and P contains node-level role–prompt assignments. This representation allows prompts and communication structure to be optimized jointly within the same search process. The framework proceeds in three stages. First, TCP-MCP constructs a population using Latin Hypercube Sampling (LHS) McKay et al. (1979) and applies the initialization-time landscape probe described in Section 3.3. Details of topology and prompt initialization are provided in Appendix C.2. Second, TCP-MCP performs structure-aware co-evolution over the coupled genome. It uses NSGA-II Deb et al. (2002) as

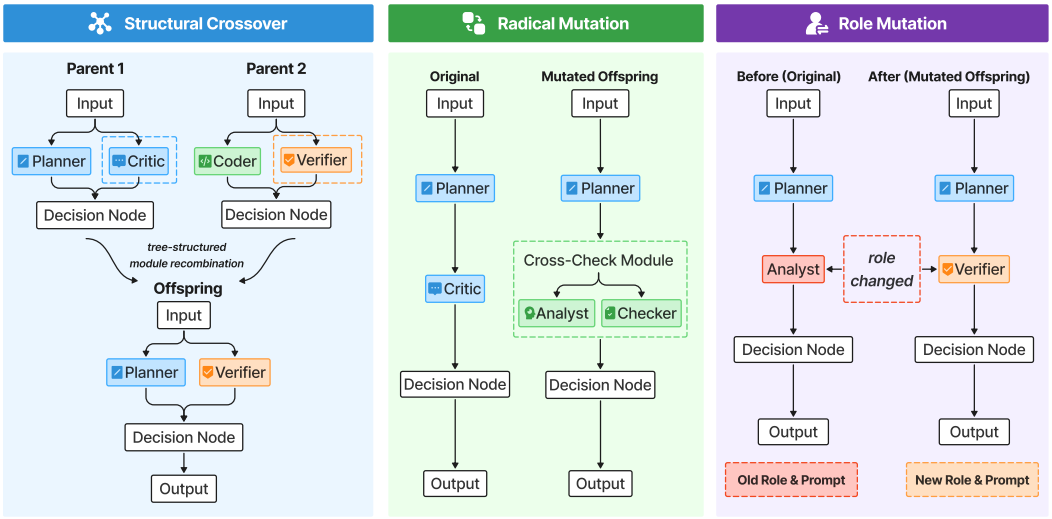

Figure 2: Illustration of structural crossover, radical mutation, and role mutation in TCP-MCP. The structural crossover panel uses PMI.

the multi-objective environmental-selection backbone, while structural crossover, topology and role mutation, prompt handling, and adaptive exploration are defined around the coupled prompt–topology representation. During evolution, Pareto-front diagnostics Zitzler et al. (2000); Deb (2001) guide adaptive exploration and help preserve coverage across trade-off regions. Finally, the search returns a Pareto elite set over task performance, token cost, and structural complexity, from which an operating point is selected on the development set and frozen for held-out evaluation.

4.2 Structural Crossover and Prompt Handling Mechanism

TCP-MCP uses Prompt Minimal Inheritance (PMI): crossover transfers topological subgraphs rather than prompt fragments. Recipient-parent nodes keep their prompt templates, while transplanted nodes inherit donor templates when available and otherwise use lightweight defaults.

Given two parent genomes, $M_a = (G_a, P_a)$ and $M_b = (G_b, P_b)$, TCP-MCP selects anchor nodes to define crossover modules. It first searches for exact matches between canonical role names. If none exist, it uses the boundary-degree heuristic:

$$b(v) = \text{in_degree}(v) + \text{out_degree}(v), \tag{5}$$

and samples high-ranked boundary nodes as module interfaces. The selected subgraphs are then exchanged and stitched into the recipient topology.

Prompt construction is deferred to runtime. Although crossover changes the communication graph, stored prompt templates are not rewritten. Instead, topology-dependent context is added during execution through $\text{BuildPrompt}(\cdot)$, keeping templates stable while allowing node behavior to adapt to the current graph and its incoming messages after structural changes.

4.3 Mutation Operators

After structural crossover, TCP-MCP mutates the unified genome to maintain exploration beyond module recombination. Given a genome $M = (G, P)$, mutation produces an offspring:

$$M' = \mathcal{M}(M), \tag{6}$$

through a probabilistic scheme. Topology mutation and role mutation are sampled as stochastic operators, while radical mutation is activated with a probability determined by stagnation signals.

Topology mutation. Topology mutation applies local edits to the communication graph,

$$\mathcal{M}_{\text{topo}} \in \{\textsc{AddEdge}, \textsc{RemoveEdge}, \textsc{AddNode}, \textsc{RemoveNode}\}, \tag{7}$$

with sampling weights adjusted during search. Edits that create cycles are rejected and retried. If repeated attempts fail, the genome is left unchanged.

Role mutation. Role mutation changes identities using a three-tier role pool. When a role changes, its prompt is regenerated by `PromptMutator` so that the new role and prompt remain aligned.

Radical mutation. Radical mutation introduces larger structural changes when the search appears stagnant. Its activation is controlled by stagnation signals computed from Pareto-front diagnostics, rather than by a fixed mutation schedule. The corresponding activation probability is defined in Section 4.5, and further implementation details are given in Appendix C.4.

4.4 Environmental Selection and Diversity Preservation

At each generation, TCP-MCP applies elitist environmental selection after offspring generation. Selection follows NSGA-II with constraint dominance, together with additional mechanisms for preserving diversity across cost and structural regimes in the elite population. Let the fitness vector be defined as in Eq. 3. Feasibility is determined by a conservative dynamic accuracy floor:

$$\tau = \mathrm{best_acc}(t) - \delta, \tag{8}$$

where $\mathrm{best_acc}(t)$ is the best accuracy observed at generation t and δ is a fixed tolerance. This threshold reduces feasibility oscillation when the best individual changes. Feasible candidates are prioritized under constrained dominance, with tie-breaking details given in Appendix C.5.

After non-dominated sorting, TCP-MCP preserves diversity in the cost and structure space through bin-based niching. Candidates are assigned to quantile-based bins over token cost and structural complexity, and each bin has a capped occupancy to avoid crowding around similar trade-off points. Any remaining elite slots are filled according to the original NSGA-II ranking.

TCP-MCP also imposes a structural quota on the elite set to avoid collapse toward trivial low-cost solutions. The quota retains complex candidates if they satisfy the accuracy floor. Without this constraint, preliminary runs can collapse to 1–2 node topologies that exploit the cost objective and weaken the Pareto front. An external elite archive improves diversity and robustness by periodically reintroducing high-performing genomes. Detailed settings are provided in Appendix C.5.

4.5 Landscape Perception and Adaptive Control

After the initialization-time FDC calibration described in Section 3.3, TCP-MCP controls subsequent exploration using cross-generational Pareto diagnostics, including normalized HV, spacing, frontier gaps, and regional coverage. These diagnostics are combined into a response index that drives adaptive exploration and candidate injection:

$$s = w_{\mathrm{HV}} s_{\mathrm{HV}} + w_{\mathrm{acc}} s_{\mathrm{acc}} + w_{\mathrm{tok}} s_{\mathrm{tok}} + w_{\mathrm{div}} s_{\mathrm{div}}, \tag{9}$$

which summarizes hypervolume stagnation, accuracy stagnation, cost pressure, and diversity deficit. The response index is mapped to an exploration probability:

$$p_{\mathrm{stag}} = \sigma(\alpha s + \beta), \tag{10}$$

which increases mutation and injection pressure when the frontier stagnates or diversity declines. Detailed normalization and response rules are given in Appendix C.6.

5 Experiments

We evaluate TCP-MCP along four axes: Q1 (Superiority), Q2 (Resilience), Q3 (Effectiveness), and Q4 (Sensitivity). Q1–Q3 are reported in this section, and additional discussion of sensitivity and design choices, including implementation-constant and budget effects, is provided in the appendix.

5.1 Experimental Setup

Datasets and Evaluation. We evaluate TCP-MCP on MMLU (Hendrycks et al., 2021), MMLU-Pro (Wang et al., 2024), GSM8K (Cobbe et al., 2021), and WorkBench (Styles et al., 2024), covering

Table 1: Main results on MMLU, MMLU-Pro, GSM8K, and WorkBench. MMLU, MMLU-Pro, and GSM8K report accuracy (%), while WorkBench reports raw task success (%). Token usage is reported in millions. Small annotations show baseline-minus-TCP-MCP differences; upward and downward arrows indicate values above and below TCP-MCP, respectively.

Method	MMLU		MMLU-Pro		GSM8K		WorkBench	
	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Succ.	Tok.
Zero-shot	84.10 ↓5.86	3.42M ↓84.61	54.36 ↓28.30	2.79M ↓94.49	95.78 ↓0.83	0.21M ↓5.81	45.38 ↓31.93	1.01M ↓6.94
PHP	87.96 ↓2.00	9.13M ↓78.90	74.71 ↓7.95	14.46M ↓82.82	96.21 ↓0.40	0.71M ↓5.31	50.00 ↓27.31	4.33M ↓3.62
Complete Graph	85.92 ↓4.04	82.71M ↓5.32	73.95 ↓8.71	132.45M ↑35.17	92.11 ↓4.50	7.65M ↑1.63	68.85 ↓8.46	2.41M ↓5.54
GPTSwarm	81.70 ↓8.26	12.60M ↓75.43	74.23 ↓8.43	20.62M ↓76.66	77.71 ↓18.90	1.21M ↓4.81	52.12 ↓25.19	5.72M ↓2.23
LLM-Blender	87.23 ↓2.73	4.68M ↓83.35	63.55 ↓19.11	6.43M ↓90.85	92.87 ↓3.74	0.33M ↓5.69	47.69 ↓29.62	5.56M ↓2.39
G-Designer	87.90 ↓2.06	28.04M ↓59.99	79.32 ↓3.34	32.87M ↓64.41	95.81 ↓0.80	4.44M ↓1.58	64.81 ↓12.50	30.02M ↑22.07
DyLAN	90.27 ↑0.31	13.26M ↓74.77	82.36 ↓0.30	27.60M ↓69.68	95.78 ↓0.83	0.92M ↓5.10	63.27 ↓14.04	26.10M ↑18.15
Complex CoT	87.49 ↓2.47	4.87M ↓83.16	79.74 ↓2.92	11.04M ↓86.24	94.18 ↓2.43	0.45M ↓5.57	46.54 ↓30.77	1.03M ↓6.92
Self-Consistency	90.36 ↑0.40	12.47M ↓75.56	76.12 ↓6.54	45.69M ↓51.59	96.05 ↓0.56	1.96M ↓4.06	47.31 ↓30.00	5.45M ↓2.50
LLM-Debate	90.84 ↑0.88	404.09M ↑316.06	83.67 ↑1.01	553.41M ↑456.13	95.48 ↓1.13	24.61M ↑18.59	76.73 ↓0.58	45.38M ↑37.43
Random Graph	90.40 ↑0.44	255.06M ↑167.03	82.59 ↓0.07	331.48M ↑234.20	95.81 ↓0.80	14.78M ↑8.76	68.85 ↓8.46	2.39M ↓5.56
AutoGen	80.37 ↓9.59	2.27M ↓85.76	55.72 ↓26.94	3.18M ↓94.10	88.79 ↓7.82	0.39M ↓5.63	50.96 ↓26.35	5.39M ↓2.56
ChatDev	83.71 ↓6.25	4.14M ↓83.89	62.19 ↓20.47	5.47M ↓91.81	96.13 ↓0.48	0.44M ↓5.58	42.88 ↓34.43	9.83M ↑1.88
MetaGPT	86.20 ↓3.76	1.86M ↓86.17	59.03 ↓23.63	2.53M ↓94.75	93.06 ↓3.55	0.12M ↓5.90	47.50 ↓29.81	13.41M ↑5.46
TCP-MCP	**89.96**	**88.03M**	**82.66**	**97.28M**	**96.61**	**6.02M**	**77.31**	**7.95M**

broad-domain question answering, high-distraction option analysis, multi-step mathematical reasoning, and tool-oriented agent tasks. For search, we use fixed development subsets for all benchmarks and report final results on held-out evaluation data. We use accuracy for MMLU, MMLU-Pro, and GSM8K, raw success for WorkBench, and token cost as main metrics. We also track normalized Hypervolume (HV) and frontier trends to analyze search. Auxiliary scalar signals, such as the preference score, are used only for landscape probing.

Baselines and Fairness Control. We compare TCP-MCP against DeepSeek-V3.2 Zero-shot, Complex CoT Fu et al. (2023), Self-Consistency Wang et al. (2023), and PHP Zheng et al. (2024); fixed or sampled MAS topologies including Complete Graph and Random Graph; LLM-Debate Du et al. (2024) and LLM-Blender Jiang et al. (2023); general MAS frameworks including AutoGen Wu et al. (2024), ChatDev Qian et al. (2024), and MetaGPT Hong et al. (2024); dynamic collaboration with DyLAN Liu et al. (2024); and automated topology-design methods GPTSwarm Zhuge et al. (2024) and G-Designer Zhang et al. (2025). All main comparisons in Table 1 use the same DeepSeek-V3.2 backbone and inference configuration. Structural-search baselines are rerun under the same evaluation protocol and comparable budget constraints. EvoMAS Hu et al. (2026) is evaluated separately because it performs evolutionary system search during each query, placing its per-query inference cost on a substantially different scale from TCP-MCP's fixed-system deployment; in our matched runs, its token and model-call costs are roughly one to two orders of magnitude higher. We therefore use matched 500-query subsets of MMLU and MMLU-Pro, following the subset-based evaluation principle studied by tinyBenchmarks (Maia Polo et al., 2024). To make this comparison controlled rather than relying on subset size alone, we additionally report subset representativeness checks, paired statistical tests, and complete accuracy–cost results in Appendix N. We further report a full WorkBench comparison under the same backbone and evaluation protocol. Full inference settings are provided in the appendix.

Search and Reporting Protocol. On each dataset, TCP-MCP starts from an initial population of 16 candidate systems and performs multi-objective search across generations. Environmental selection retains 8 elites per generation. Unless stated, all experiments use the same population and elite settings. We report the generation-21 Pareto operating point, where HV trends indicate a stabilized Pareto front. Since TCP-MCP returns a Pareto set rather than a single accuracy-only system, we select a representative operating point on the development set by considering task performance together with token cost and structural complexity before held-out evaluation. Details are provided in Appendix I. Table 1 reports held-out inference tokens; Appendix F reports held-out API-call counts, search-time tokens, search-time API calls, and wall-clock cost.

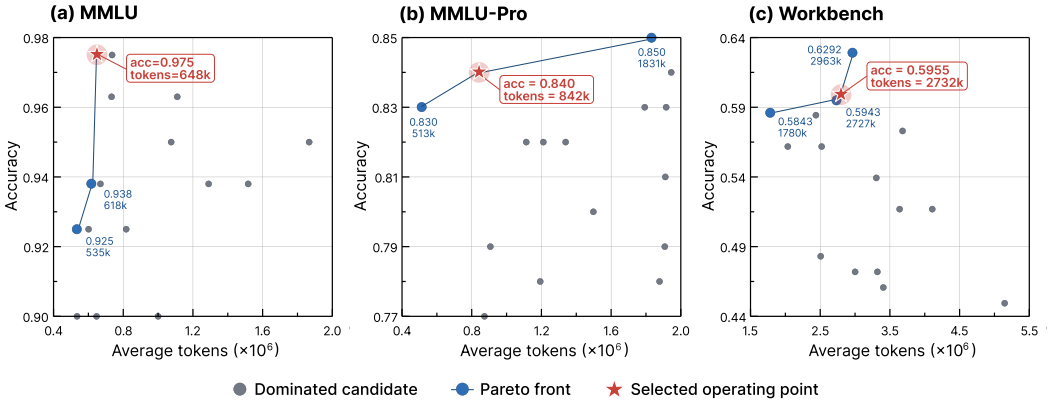

Figure 3: Development-set Pareto fronts. Stars mark the representative TCP-MCP points selected on the development-set Pareto front for held-out evaluation. The selected points are chosen from the development-set Pareto front by considering both accuracy and token cost. We show MMLU, MMLU-Pro, and WorkBench because they contain visible trade-off fronts; GSM8K is omitted because the selected operating region does not form a visually informative trade-off front.

5.2 Performance Comparison (Q1)

We evaluate selected Pareto solutions on held-out test sets against single-agent baselines, fixed-topology MAS variants, dynamic collaboration methods, and automated design baselines under the same backbone and evaluation protocol. Table 1 shows that TCP-MCP outperforms G-Designer on all four benchmarks, raising accuracy from 87.90 to 89.96 on MMLU, 79.32 to 82.66 on MMLU-Pro, and 95.81 to 96.61 on GSM8K, while improving raw success from 64.81 to 77.31 on WorkBench. These gains use more tokens than G-Designer on the three reasoning benchmarks, whereas TCP-MCP uses substantially fewer tokens on WorkBench, requiring 7.95M compared with 30.02M tokens.

WorkBench further extends the comparison from final-answer reasoning to tool-executing agent tasks. TCP-MCP improves raw success over DyLAN from 63.27 to 77.31 while reducing token usage from 26.10M to 7.95M. LLM-Debate reaches 76.73 raw success, only 0.58 percentage points below TCP-MCP, but requires 45.38M tokens compared with 7.95M for TCP-MCP. On MMLU-Pro, TCP-MCP also slightly exceeds Random Graph while using about 3.41$\times$ fewer tokens. Overall, TCP-MCP remains competitive across both reasoning and tool-use settings while exposing explicit trade-offs between task performance and inference cost. The corresponding performance and token-cost trade-offs are visualized in Appendix Figure 6.

Figure 3 shows the development-set Pareto fronts used to choose the TCP-MCP operating points. The selection is made before held-out evaluation using the same deterministic rule across benchmarks: among Pareto candidates within a fixed cost envelope, we select the candidate with the highest development-set task performance and freeze the selected genome thereafter. The formal selection rule and tie-breaking procedure are given in Appendix I.

5.3 Search Resilience (Q2)

We study TCP-MCP's resilience from three angles: Pareto-front stability, reproducibility across evolutionary runs, and reuse of structures beyond their original search setting.

Figure 4 plots token cost and structural complexity across generations. Appendix L reports normalized Hypervolume (HV) at generations 1, 11, and 21. These HV diagnostics show that the Pareto front improves mainly in early generations and stabilizes as evolution approaches generation 21. Average token usage generally decreases or stabilizes, while average structural complexity remains bounded rather than growing monotonically. Together, these diagnostics suggest that TCP-MCP does not improve the frontier simply by enlarging systems or spending more tokens.

We further examine whether the reported performance depends on a single favorable evolutionary trajectory. Across the original run and two additional complete runs under the same search and

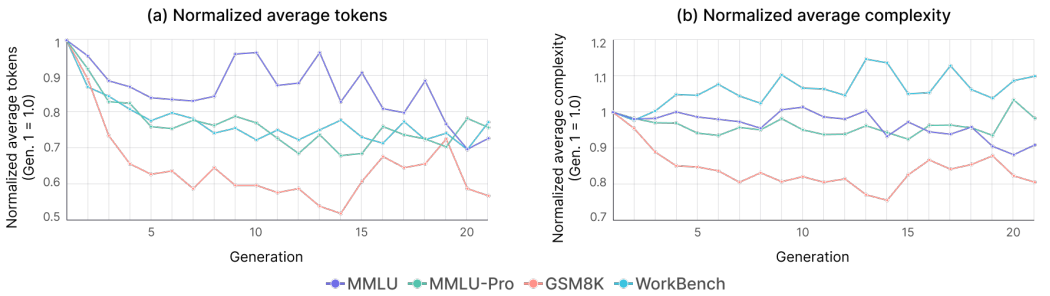

Figure 4: Evolutionary trends (Q2) showing token cost and structural complexity across generations.

Table 2: Cross-dataset transfer. Native rows report in-domain results; transfer rows reuse the searched genome on a different target dataset without re-running evolutionary search.

Source	Target	Acc.	Drop
MMLU	MMLU	89.96	–
MMLU-Pro	MMLU-Pro	82.66	–
MMLU	MMLU-Pro	81.12	1.54
MMLU-Pro	MMLU	88.76	1.20

Table 3: Cross-backbone transfer. Structures searched with DeepSeek-V3.2 are kept fixed and reused with GPT-4 on the same benchmarks, without re-running evolutionary search.

Method	MMLU	M-Pro	GSM8K	WB
Vanilla	82.14	55.35	85.40	54.81
Self-Cons.	83.65	69.88	86.14	63.45
G-Designer	84.50	74.79	86.14	49.04
TCP-MCP	**85.48**	**78.63**	**96.53**	**73.85**

evaluation protocol, TCP-MCP obtains mean accuracies of 89.55% on MMLU and 82.68% on MMLU-Pro, with sample standard deviations of 0.43 and 0.47 percentage points, respectively. Although the runs converge to different final structures, their full-benchmark accuracies remain close. This result indicates that TCP-MCP's performance is reproducible across stochastic search runs rather than being determined by one favorable trajectory. Full per-run results are reported in Appendix E.

We next examine whether the evolved structures remain useful beyond their original search setting. Table 2 and Table 3 cover two transfer settings. For cross-dataset transfer between the two multiple-choice benchmarks, we apply a searched prompt–topology genome to another dataset without re-running search or using the target development set. The transferred genomes retain strong performance, with accuracy drops of 1.54 points from MMLU to MMLU-Pro and 1.20 points from MMLU-Pro to MMLU. For cross-backbone transfer, we reuse structures searched with DeepSeek-V3.2 under GPT-4 without re-running evolutionary search. TCP-MCP achieves the highest performance among the compared methods on all four benchmarks. On WorkBench, the transferred TCP-MCP structure reaches 73.85% task success, compared with 63.45% for Self-Consistency, 54.81% for Vanilla, and 49.04% for G-Designer. Appendix G further reports a second held-out inference run, where the selected systems follow the same overall trend as the main results.

Together, these results support reproducible performance across evolutionary runs. The Pareto front improves while structural complexity remains bounded and average token costs decline or stabilize. Transfer experiments support partial reuse across related datasets and compatible backbones, although target-specific search may be needed when the deployment setting changes substantially.

5.4 Design Effectiveness (Q3)

We examine whether TCP-MCP discovers task-specific collaboration structures and how search components affect the final systems. We first analyze structures selected by the final Pareto elites and then study operator and coupling ablations. Operator ablations examine individual search components, while the coupling ablation compares joint prompt–topology optimization with staged optimization.

The final Pareto elites exhibit different structures across tasks. On GSM8K, the selected system is compact, with a mathematical analysis role interpreting the problem before passing it to a solver. On MMLU, the structure places emphasis on verification, combining knowledge checking with option-level validation before final solving. On MMLU-Pro, the system identifies definitions and decomposes

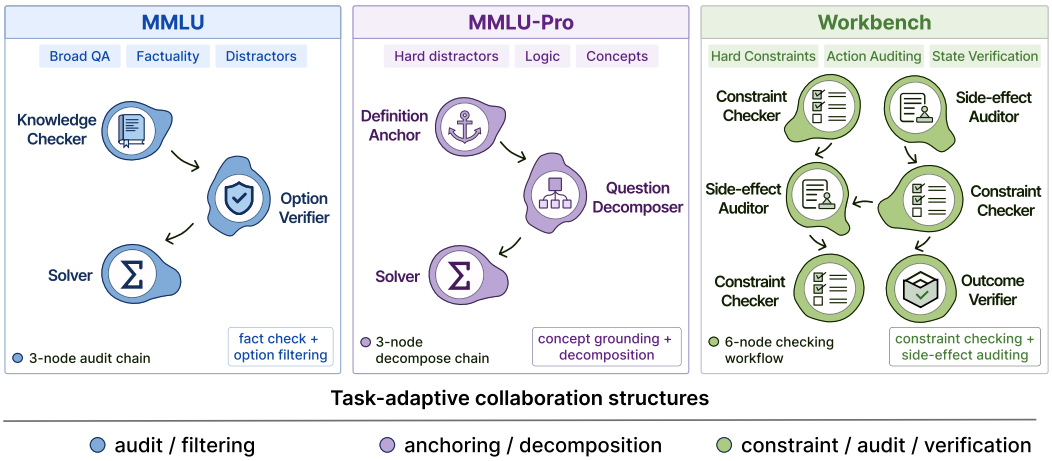

Figure 5: Representative structural prototypes evolved on MMLU, MMLU-Pro, and WorkBench. The final decision node is omitted for clarity.

Table 4: Ablation accuracy (%) on MMLU, MMLU-Pro, GSM8K, and WorkBench.

Setting	MMLU	MMLU-Pro	GSM8K	WorkBench
Full	89.96	82.66	96.61	77.31
AB1	88.23	82.48	95.97	73.08
AB2	87.70	80.15	95.65	72.88
AB3	87.37	81.46	96.29	69.04
AB4	87.48	82.30	96.27	71.35

the question before solving it. On WorkBench, the evolved structure focuses on constraint checking, side-effect auditing, and outcome verification. These differences suggest that TCP-MCP adjusts both collaboration topology and role specialization to the requirements of each task.

We evaluate four variants of genome search over $M = (G, P)$. AB1 removes adaptive control, AB2 removes crossover, and AB3 removes mutation-related exploration. AB4 uses staged optimization, where prompts are optimized under a fixed topology and fixed while the topology is searched. Full TCP-MCP instead evolves prompts and topology jointly throughout the search.

As shown in Table 4, full TCP-MCP achieves the highest accuracy on all four benchmarks. Removing crossover causes a drop on MMLU-Pro, while removing mutation-related exploration causes the largest losses on MMLU and WorkBench. AB4 also performs worse than full TCP-MCP on all four benchmarks, with larger gaps on MMLU and WorkBench. This result supports joint optimization of prompts and topology rather than separating them into sequential stages.

6 Conclusion, Limitations, and Future Work

Conclusion. We presented TCP-MCP, a framework that jointly evolves prompts and communication topologies under accuracy, cost, and complexity objectives. Across MMLU, MMLU-Pro, GSM8K, and WorkBench, TCP-MCP learns task-specific collaboration structures and reveals clear Pareto trade-offs among performance, cost, and structural complexity. Ablations further show the benefit of jointly optimizing prompts and topology rather than optimizing them sequentially.

Limitations. TCP-MCP searches within a predefined genome and operator space, so its performance is bounded by the expressiveness of this search space.

Future Work. Future work can explore richer search spaces and more efficient evolutionary strategies, while studying how searched collaboration structures generalize across broader tasks and models.

AI USE STATEMENT

We used generative AI tools to assist with writing and language polishing, literature retrieval and discovery, and research ideation or execution. All AI-assisted materials were reviewed by the authors, who verified the cited sources, experimental results, and scientific claims. The authors take full responsibility for the final content of the paper.

REFERENCES

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Chichester, UK, 2001. ISBN 9780471873396.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi: 10.1109/4235.996017.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=yf1icZHC-l9.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Yuntong Hu, Yuting Zhang, Matthew Trager, Yi Zhang, Shuo Yang, Wei Xia, and Stefano Soatto. EvoMAS: Evolutionary generation of multi-agent systems. In *Forty-third International Conference on Machine Learning*, 2026. URL https://www.amazon.science/publications/evomas-evolutionary-generation-of-multi-agent-systems.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14165–14178, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.792. URL https://aclanthology.org/2023.acl-long.792/.

Terry Jones and Stephanie Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 184–192. Morgan Kaufmann, 1995.

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

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinybenchmarks: Evaluating llms with fewer examples. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 34303–34326. PMLR, 2024.

Michael D. McKay, Richard J. Beckman, and William J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979. doi: 10.1080/00401706.1979.10489755.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. WorkBench: a benchmark dataset for agents in a realistic workplace setting. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=4HNAwZFDcH.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=y10DM6R2r3.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=LpE54NUnmO.

Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. Progressive-hint prompting improves reasoning in large language models. In *AI for Math Workshop at ICML 2024*, 2024. URL https://openreview.net/forum?id=UkFEs3ciz8.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=uTC9AFXIhg.

Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000. doi: 10.1162/106365600568202.

A Related Work

A.1 Single-Agent Reasoning

Single-agent reasoning methods improve the inference process of a single LLM without explicit communication between agents. Zero-shot prompting provides the simplest baseline, in which

an LLM answers a question directly without decomposition or collaboration. Complexity-based prompting encourages more detailed intermediate reasoning for multi-step problems (Fu et al., 2023), while Self-Consistency samples multiple reasoning paths and selects the most consistent answer (Wang et al., 2023). Progressive-Hint Prompting improves reasoning within a single-model workflow by providing iterative hints (Zheng et al., 2024). These methods can achieve strong performance at relatively low cost, but they do not consider role specialization, communication between agents, or topology-dependent information flow.

A.2 Multi-Agent Collaboration and Aggregation

Multi-agent methods introduce multiple agents, intermediate communication, or output aggregation to improve reasoning. Fixed-topology baselines such as Complete Graph and Random Graph, following the setting of G-Designer (Zhang et al., 2025), examine whether communication among multiple agents alone is sufficient to improve performance. Complete Graph enables dense communication among agents, whereas Random Graph uses sampled communication structures as a topology baseline. Because these structures are not adapted jointly with node-level prompts, they may introduce unnecessary communication or fail to direct useful information to the appropriate agents. Multiagent Debate improves reasoning through iterative exchanges among multiple model instances (Du et al., 2024), while LLM-Blender combines outputs from multiple LLMs through pairwise ranking and generative fusion (Jiang et al., 2023). General multi-agent frameworks such as AutoGen, ChatDev, and MetaGPT provide abstractions for roles, message passing, and collaborative execution (Wu et al., 2024; Qian et al., 2024; Hong et al., 2024). These frameworks simplify the construction of multi-agent workflows, but do not primarily target joint optimization of communication topology and node-level prompts. GPTSwarm is more closely related, as it optimizes both node-level prompts and graph connectivity through node and edge optimizers (Zhuge et al., 2024). TCP-MCP instead represents role–prompt assignments and communication topology within a single executable genome, so that both are varied, evaluated, and selected jointly throughout the same evolutionary process.

A.3 Automated Topology and Dynamic Collaboration

Automated topology-design methods are closely related to TCP-MCP because they adapt the organization of multi-agent communication rather than relying on manually specified interaction patterns. DyLAN dynamically selects agents and constructs task-dependent communication structures through a two-stage procedure for team optimization and task solving (Liu et al., 2024). GPTSwarm formulates language-agent systems as optimizable graphs (Zhuge et al., 2024), while G-Designer learns multi-agent communication topologies with graph neural networks (Zhang et al., 2025). Together, these methods highlight the importance of agent selection and communication structure for reasoning performance.

TCP-MCP builds on this direction through structure-aware evolution of executable prompt–topology genomes. Structural crossover transfers subgraphs together with their associated roles and prompt templates, while mutation modifies communication patterns and role-conditioned behavior. Candidate systems are evaluated under accuracy, token cost, and structural complexity objectives, allowing useful combinations of agent behavior and information flow to be retained across generations. The selected genome is then frozen after development-set optimization and reused across held-out queries.

A.4 Evolutionary Multi-Agent System Design

Recent work extends automated MAS design beyond topology optimization to evolutionary search over broader system configurations. EvoMAS (Hu et al., 2026) formulates MAS generation as structured configuration search and evolves candidate systems directly in configuration space. It maintains a population of configurations and refines them through execution-trace-guided mutation and crossover, using feedback and accumulated experience during the search. This approach shows that evolutionary methods can search over executable multi-agent systems beyond fixed communication graphs.

TCP-MCP follows the same broad direction of evolutionary MAS design, but differs in both the search representation and the deployment setting. It jointly evolves node-level role–prompt behavior and communication topology within a single executable genome. Structural crossover, role mutation,

prompt regeneration and inheritance, and topology-conditioned runtime prompt construction all operate on this shared representation. TCP-MCP performs this search on development data to obtain a reusable system, freezes the selected genome before held-out evaluation, and applies it unchanged to unseen queries without query-specific evolutionary search. EvoMAS instead constructs and evolves systems at the query level. The two methods therefore study different settings of automated MAS design: query-level adaptive construction and dataset-level optimization of a reusable system.

B Formalization Notes

The notation in this paper defines the search object, objectives, and adaptive-control signals used by TCP-MCP. These equations are meant to specify the algorithm, not to provide theoretical guarantees. We do not claim convergence, approximation bounds, or a guarantee that the evolutionary process will find a globally Pareto-optimal solution. The unified genome $M = (G, P)$ defines the joint search space over communication topology and node-level prompting. The multi-objective fitness vector specifies how accuracy, token cost, and structural complexity are compared during selection. The scalar preference score is used only as an auxiliary signal for initialization-time landscape probing. It does not replace Pareto-based environmental selection. Fitness Distance Correlation (FDC) is also computed only after initialization to calibrate the early search bias. During evolution, TCP-MCP does not re-estimate FDC. It instead uses cross-generational Pareto-front diagnostics, including Hypervolume, spacing, frontier gaps, and regional coverage, to adjust exploration.

B.1 Initialization Preference Score

For the initialization-time landscape probe, we use the auxiliary preference score

$$S(M) = \frac{A(M)^k}{(C(M)/T_0)^\gamma \cdot \bar{K}(G)^\beta}, \tag{11}$$

where $\bar{K}(G) = 1 + K(G)/K_0$. Here, T_0 and K_0 normalize token cost and graph complexity, respectively, while $k, \gamma, \beta > 0$ control their relative contribution to the scalar probe. The score is used only to compute the initialization-time FDC and does not replace Pareto-based environmental selection.

Let

$$M_0^\star = \arg\max_{M_j \in \mathcal{P}_0} S(M_j). \tag{12}$$

FDC measures the Spearman correlation between the preference scores of the initial candidates and their genome distances to $M_0^\star$.

C Implementation Details

This section gives the implementation details of TCP-MCP. We describe how structural crossover, mutation, environmental selection, and adaptive control are implemented in the evolutionary search. The discussion complements the main method section and specifies the search procedure at a level that supports reproduction, without turning the appendix into code-level pseudocode.

C.1 Inference and Baseline Configuration

For the main comparisons, all methods use the same DeepSeek-V3.2 backbone, API endpoint, decoding configuration, and context budget. Unless otherwise specified, we use the official DeepSeek API defaults without manually overriding decoding hyperparameters: temperature $= 1.0$, top-p $= 1.0$, frequency penalty $= 0$, and presence penalty $= 0$.

Baselines involving structural search are rerun under the same evaluation protocol and comparable budget constraints. Token usage reported in the main comparison corresponds to held-out inference cost. Search-time token usage, API-call counts, and wall-clock cost for TCP-MCP are reported separately in Appendix F.

C.2 Initial Population Construction

TCP-MCP initializes each run with a diverse population of prompt–topology genomes. We use Latin Hypercube Sampling (LHS) to cover the main design dimensions of the initial search space, including graph size, edge density, topology pattern, role composition, and prompt style. Each LHS sample is then mapped to an executable genome $M = (G, P)$.

For topology initialization, TCP-MCP samples the number of non-input agents from a bounded range and instantiates a directed acyclic communication graph. The initial graph is drawn from a mixture of structural templates, including chain, tree, star, layered, and sparse random DAG patterns. The sampled LHS coordinate controls edge density. After graph construction, TCP-MCP checks reachability from the input node to the final decision node and rejects edges that violate the acyclic execution order. If a sampled graph is invalid, the implementation either resamples it or repairs it by adding the minimum links needed to make the candidate executable.

For role initialization, each non-input node receives a role from a task-conditioned role pool. The pool contains task-specific expert roles, domain-heuristic roles, and general reasoning roles. For MMLU and MMLU-Pro, the task-specific pool includes Question Decomposer, Option Verifier, Knowledge Checker, Definition Anchor, Trap Detector, Counterfactual Reasoner, and Consistency Checker. For GSM8K, it includes mathematical roles such as Math Solver, Mathematical Analyst, Programming Expert, Inspector, Formula Auditor, and Unit Checker. General roles, including Planner, Critic, Verifier, Summarizer, and Aggregator, are shared across datasets. The final decision or aggregation role is fixed when needed, so that all candidates expose the same external prediction interface.

Prompt initialization follows the assigned role and dataset domain. For each role, TCP-MCP either uses a role-conditioned seed template from the prompt registry or generates an initial instruction with the same PromptMutator used in later role mutation. The generated prompt specifies the role definition, expected reasoning behavior, input usage, and output format. Node-specific topology information is not stored in the prompt template. Instead, it is injected at runtime through BuildPrompt according to the node's incoming neighbors. This keeps the initial prompt representation consistent with the rest of the evolutionary process.

The LHS population is therefore diverse along both structural and semantic dimensions. Candidates differ in graph size, communication pattern, edge density, role assignment, and prompt behavior. After initialization, all candidates are evaluated once on the development subset. Their fitness values are used to compute the initialization-time preference score and Fitness Distance Correlation (FDC), which calibrates the early crossover bias before the main evolutionary loop begins.

C.3 Structural Crossover and Prompt Minimal Inheritance

TCP-MCP performs structural crossover by recombining topology-level modules, rather than by mixing prompt strings directly. Given two parent genomes, the operator treats one parent as the recipient and the other as the donor. It selects a subgraph module from the donor and transplants it into the topology of the recipient. The recipient keeps the rest of its original graph, while the donor provides the selected structural module. The resulting topology is checked before acceptance, and any crossover output that violates the directed acyclic communication constraint is rejected. The module-matching procedure first tries to align nodes by exact role match. When exact role matching is not available, the operator uses a structural anchor based on boundary degree, defined as the sum of the in-degree and out-degree of a node. This fallback favors high-interface nodes, which are more likely to preserve useful information flow after module transplantation. This allows crossover to remain applicable even when the two parents do not have identical role assignments.

Prompt inheritance follows Prompt Minimal Inheritance (PMI). Nodes retained from the recipient parent keep their original roles and prompt templates unchanged. Nodes introduced from the donor module inherit the donor-side role and prompt template when a valid donor prompt assignment is available. If a transplanted donor node does not have a valid prompt–role entry, the implementation uses a lightweight role-conditioned default prompt instead of invoking full prompt regeneration. In this way, crossover can make substantial changes to the structural context while changing prompt text only when necessary. This design reflects how prompts are represented in TCP-MCP: prompt templates are stored independently of the concrete topology, and topology-dependent context is added only at runtime through the prompt-construction function. The same inherited prompt template

can therefore lead to different behavior after crossover, because the node may receive different upstream messages. PMI preserves useful local prompt behavior while still allowing the surrounding communication structure to change.

C.4 Mutation Operators

TCP-MCP uses three classes of mutation: local topology mutation, role mutation with prompt regeneration, and radical structural mutation. Local topology mutation edits the communication graph through elementary operations, including edge addition, edge removal, node addition, and node removal. Each proposed structural edit is checked before acceptance. Edits that introduce cycles are rejected, so the candidate system remains executable under the feed-forward communication protocol. Role mutation changes the functional identity of a node. The candidate role pool is domain-aware and is assembled from dataset-specific expert roles, domain-heuristic roles, and general-purpose reasoning roles. For MMLU and MMLU-Pro, the expert pool includes roles such as Question Decomposer, Option Verifier, Knowledge Checker, Trap Detector, Definition Anchor, Counterfactual Reasoner, and Consistency Checker. For GSM8K, the pool includes mathematical roles such as Math Solver, Mathematical Analyst, Programming Expert, and Inspector. Domain-heuristic and general-purpose roles include Calculator, Mathematician, Statistician, Quantitative Analyst, Debugger, Tester, FactChecker, StepPlanner, FormulaAuditor, UnitChecker, ConflictDetector, ConstraintExplicator, SemanticDifferentiator, and GroundingEnforcer. Output-critical roles, such as final decision or output aggregation roles, are excluded from mutation to avoid changing the external behavior of a candidate system.

When a role changes, TCP-MCP regenerates the corresponding prompt body instead of only replacing the role name. PromptMutator produces a new instruction body conditioned on the new role and the dataset domain. The generated prompt must start with the new role definition, include explicit execution actions, and satisfy role-type-specific requirements: checker roles must include verification behavior, solver roles must include concrete solution steps, and constraint-oriented roles must explicitly enforce task constraints. Node identity is not included in the generated prompt text, because node-specific topology information is added at runtime. Prompt regeneration uses a bounded retry mechanism. If all retries fail, the system first uses a role-specific registry template when one exists, and otherwise uses a minimal role-conditioned fallback prompt. Radical mutation is used when ordinary local edits are not sufficient. It can introduce larger structural changes, such as chain, tree, star, or layered topologies, and its probability is controlled by the adaptive control index described in Section C.6. In practice, radical mutation is rare when the search is progressing well and becomes more likely when the Pareto frontier stagnates or the population loses diversity.

C.5 Environmental Selection and Diversity Preservation

TCP-MCP uses constrained NSGA-II for environmental selection. Each candidate genome is evaluated with a multi-objective fitness vector over task accuracy, token cost, and structural complexity. Accuracy is maximized, whereas token cost and complexity are minimized. Since low-cost candidates are not useful when their accuracy is too low, TCP-MCP applies a dynamic feasibility constraint before standard Pareto ranking. The feasibility threshold is defined relative to the current best accuracy. A candidate is feasible only if its accuracy lies within a fixed margin of the current best candidate. Feasible candidates dominate infeasible candidates regardless of token cost or complexity, preventing the search from selecting very cheap but ineffective systems only because they occupy favorable cost regions. After feasibility filtering, candidates are ranked by standard non-dominated sorting. Within each Pareto front, crowding distance favors candidates in less crowded regions of the objective space. The tie-breaking order is feasibility, Pareto rank, crowding distance, bin capacity, and deterministic iteration order.

TCP-MCP further preserves diversity through bin-based niching, a structural quota, and an external elite archive. Bin-based niching partitions the cost–complexity space using quantile-based boundaries, and each bin can contribute only a limited number of elites. This prevents a dense region of the Pareto frontier from dominating the selected population. The bin cap depends on the diversity deficit: it remains stricter when diversity is low and gradually relaxes when the population is sufficiently diverse. The structural quota prevents premature collapse toward trivial low-node topologies by retaining structurally richer candidates when they satisfy the dynamic accuracy floor and remain

competitive under Pareto ranking. The external elite archive can reintroduce validated elites from earlier generations with duplicate checks. If archive injection cannot fill the reserved slots, the implementation falls back to cloning selected elites with duplicate filtering. These mechanisms allow the search to recover useful genomes that may have disappeared from the current generation.

C.6 Adaptive Control

TCP-MCP uses adaptive control to regulate exploration during evolution. The controller does not replace Pareto-based selection. It adjusts mutation pressure and targeted injection behavior using cross-generational diagnostics of the Pareto frontier and the selected population. The main adaptive signal is a scalar control index $s \in [0, 1]$, which combines hypervolume progress, elite-best accuracy progress, token-cost pressure, and diversity deficit. Each component is clipped before aggregation, and the final control index is also clipped to $[0, 1]$. The weighting scheme changes across generations: early generations give more weight to diversity preservation, while later generations shift toward hypervolume and accuracy. This schedule gives a simple transition from exploration to exploitation. The control index is mapped to the radical-mutation probability through the sigmoid function in Eq. 10, where the implementation uses a steep sigmoid. Radical mutation is therefore unlikely under mild pressure, but its probability rises quickly when the control index becomes large.

The controller also adjusts topology and role mutation rates from their base values using the control index and runtime landscape features, including diversity, neutrality, multimodality, and ruggedness proxies computed from recent population history. These runtime features are distinct from the initialization-time Fitness Distance Correlation (FDC). FDC is computed once after initialization and is used only to calibrate early crossover bias; it is not repeatedly estimated during evolution. At the population level, TCP-MCP can perform targeted injection. The targeted injector monitors Pareto-front diagnostics, including hypervolume growth, spacing, maximum frontier gap, and regional coverage. When it detects a recognizable failure mode, it generates repair-oriented genomes: regression recovery introduces conservative candidates based on recently strong regions, multi-island split introduces bridging structures, semantic deficiency targets missing Pareto regions, diversity loss introduces structurally varied candidates, and radical innovation introduces more exploratory reasoning structures. These injected candidates are conditioned on the diagnosed failure mode and the underrepresented Pareto region, but they must still compete under constrained NSGA-II and are retained only if they satisfy the accuracy and Pareto-selection criteria.

C.7 Prompt Templates and Mutation Examples

TCP-MCP stores each node prompt as a role-conditioned template. Topology-dependent context is added only at runtime by BuildPrompt. This section gives representative examples to make the prompt representation and mutation process explicit.

For multiple-choice benchmarks such as MMLU and MMLU-Pro, decomposition and verification roles use short task-specific instructions. A decomposition role classifies the question type, extracts key entities, and states the goal in a fixed JSON format. A verification role independently checks whether each candidate answer is factually correct and relevant to the question, and defers calculation-heavy cases to the math solver. A representative verification template is shown below.

```
## IDENTITY
You are the Option Fact Checker.

## TASK
1. Fact Check: Check whether each option is
   factually true.
2. Relevance: Check whether each option answers
   the question.

## CRITICAL RULE
- If this is a math question, output:
  "Defer to MathSolver."
- Do not copy other agents' analyses.
  Check the facts independently.
```

```
## OUTPUT
- Option [X]: [Correct/Incorrect]
  - [Brief Reason]
```

For GSM8K, solver prompts are more calculation-oriented. The math solver receives hints from upstream agents, solves the problem step by step, re-reads the final sentence before solving, and must end with a fixed integer-answer format, `The answer is <integer>`. Few-shot examples, when used, are attached to this template in the released configuration files.

For WorkBench, prompts focus on tool use, hard task constraints, side effects, and final-state verification.

A representative Constraint Checker template is shown below.

```
Translate every must/only/before/after condition
into explicit checks.
Reject actions that violate hard task constraints.
The benchmark grades final sandbox state
against ground truth.
Use tools for actions and do not invent
tool results.
```

A representative Side-effect Auditor template is shown below.

```
Classify proposed calls as read-only or
state-changing.
Block unrelated sends/deletes/updates/creates.
Inspect for unintended changes.
The benchmark grades final sandbox state
against ground truth.
Use tools for actions and do not invent
tool results.
```

A representative Outcome Verifier template is shown below.

```
Make the final evidence-backed completion
judgment.
Check sandbox state and forbidden side effects.
Never claim success from prose alone.
The benchmark grades final sandbox state
against ground truth.
Use tools for actions and do not invent
tool results.
```

These examples show that the same prompt representation applies to both reasoning and tool-use tasks. The stored template specifies the local behavior of a role, while topology-dependent context is supplied at runtime.

Role mutation regenerates the prompt body instead of only replacing the role name. For example, when a node mutates from a general reasoning role to a trap-detection role, PROMPTMUTATOR generates an instruction that asks the node to check for unit errors, scope errors, and option-level inconsistencies. This keeps the mutated role aligned with its prompt behavior. For tool-use tasks, the same mechanism can generate behaviors such as constraint checking, side-effect auditing, and outcome verification.

PMI follows the same template-level representation. During crossover, nodes retained from the recipient keep their prompt templates, while transplanted donor nodes inherit the donor-side role and prompt template when available. The stored template contains no edge information or upstream-node identifiers; those are injected by BUILDPROMPT under the current topology. Generated prompts are subject to hard constraints, including a 3000-character limit, removal of greetings and generic assistant phrases, and the requirement that each prompt specify at least one explicit reasoning behavior, such as elimination, step-by-step solving, consistency checking, constraint checking, or state verification.

D Implementation Sensitivity and Design Choices

We discuss the main implementation choices that affect TCP-MCP's search behavior, focusing on the adaptive-control schedule, population and elite sizes, generation budget, and the complexity objective. Full sweeps over evolutionary hyperparameters are costly, so we use the development subsets to examine robustness in search behavior rather than to rank hyperparameter settings. We focus on the main factors that affect exploration: the adaptive-control schedule, population and elite sizes, generation budget, and the complexity objective.

The adaptive controller determines when the search moves from local refinement to stronger exploration. The control index s is mapped to the probability of radical mutation through a sigmoid function. If this mapping is too flat, radical mutation reacts weakly to frontier stagnation; if it is too sharp, small changes near the transition region can make mutation behavior unstable. The setting used in the main experiments keeps radical mutation rare during normal progress and increases it only under persistent stagnation or diversity loss. The weighting schedule follows the same logic. Early generations emphasize diversity to keep multiple topology and cost regimes active, while later generations emphasize hypervolume and accuracy to refine the Pareto frontier.

Population size, elite size, and generation budget control the cost–coverage trade-off of evolutionary search. Smaller populations and elite sets reduce evaluation cost but can discard useful structures before they are recombined or refined. Larger settings improve coverage but increase token use directly and leave less room for efficient search under a fixed budget. In the main configuration, bin-based niching and the structural quota reduce this sensitivity by preventing one cost–complexity region from dominating the elite set and by avoiding early collapse to trivial low-node graphs. Across the monitored runs, most hypervolume gains occur in the early and middle generations, after which the frontier tends to stabilize. This pattern suggests that the chosen generation budget is sufficient to reach a stable trade-off region under the available evaluation budget.

The complexity objective controls pressure against unnecessarily large communication graphs. If this pressure is too weak, the search may retain systems whose additional nodes and edges bring little accuracy gain. If it is too strong, the population may collapse to small topologies before useful collaborative structures are explored. We therefore treat structural complexity as a Pareto objective rather than a hard graph-size constraint. The structural quota only guards against premature collapse: structurally richer candidates must still satisfy the dynamic accuracy floor and remain competitive under Pareto ranking. Final operating points are selected from the development Pareto frontier to reflect the accuracy–cost–complexity trade-off, not to favor the largest or most expensive candidate found during search.

E Robustness Across Evolutionary Runs

Evolutionary search involves stochastic initialization, crossover, and mutation. We therefore conduct two additional complete TCP-MCP runs on MMLU and MMLU-Pro to examine whether the reported performance depends on a particular search trajectory. All runs use the same DeepSeek-V3.2 backbone, population size of 16, elite size of 8, generation-21 reporting point, development-evaluation budget, and final evaluation protocol. The original main run is retained without selecting the best result among the three runs.

Table 5: Full-benchmark accuracy across three complete evolutionary runs. The final column reports the mean and sample standard deviation across runs.

Dataset	Main	Seed 1	Seed 2	Mean $\pm$ s.d.
MMLU	89.96	89.11	89.59	$\mathbf{89.55 \pm 0.43}$
MMLU-Pro	82.66	82.21	83.15	$\mathbf{82.68 \pm 0.47}$

On MMLU, the three complete runs achieve accuracies between 89.11% and 89.96%, with a mean of 89.55% and a sample standard deviation of 0.43 percentage points. On MMLU-Pro, accuracy ranges from 82.21% to 83.15%, with a mean of 82.68% and a sample standard deviation of 0.47 percentage points. In both cases, the variation across complete searches remains below one percentage point.

The originally reported result is also close to the three-run mean, particularly on MMLU-Pro, where it differs from the mean by only 0.02 percentage points.

The runs reach similar full-benchmark accuracy despite stochastic search and non-identical final genomes. This suggests that TCP-MCP can discover multiple competitive prompt–topology designs rather than relying on one isolated solution. These results complement the within-run Hypervolume analysis in Appendix L: the Hypervolume results show that the Pareto front stabilizes during evolution, while the repeated complete runs show that the resulting performance is also stable across independent search trajectories. Together, they provide evidence that TCP-MCP's reported gains are reproducible and are not determined by a single favorable run.

F Compute Resources

TCP-MCP does not require local model training or GPU-based fine-tuning. All experiments can be launched from a local machine or a standard CPU worker, with the main computational cost coming from external LLM API inference. We therefore report token usage, API-call counts, and wall-clock time as the primary compute-cost measures. The token counts reported in the main result table correspond to the final held-out evaluation of the selected Pareto operating point, not to the full evolutionary search cost. The evolutionary search itself is performed on the fixed development subsets and incurs additional API inference cost when candidate genomes are evaluated.

Table 6: Compute summary for the selected TCP-MCP systems on held-out evaluation. Token usage follows Table 1. Node counts include all LLM-executed nodes, including the final decision node. API calls are estimated by multiplying the number of held-out examples by the number of executed nodes.

Dataset	Held-out examples	Selected nodes	Tokens	API calls
MMLU	13,961	4	88.03M	55,844
MMLU-Pro	12,032	4	97.28M	48,128
GSM8K	1,240	3	6.02M	3,720
WorkBench	520	7	7.95M	3,640

The API-call counts in Table 6 are for final held-out evaluation only. The selected systems contain three agent nodes and one final decision node on MMLU and MMLU-Pro, and two agent nodes and one final decision node on GSM8K. We count one LLM call for each executed node on each held-out test example, including the final decision node. These counts exclude failed requests, retries, preliminary experiments, baseline reruns, and all API calls used during evolutionary search on the development subsets. Search-time cost depends on the number of evaluated genomes, the active nodes in each candidate topology, the prompt length of each node, and the generation budget. In our implementation, search-time token usage and API calls are recorded in the checkpoint logs and are treated separately from the held-out evaluation cost used for the main accuracy–token comparisons.

To make the search-time inference cost explicit, Table 7 reports token usage and API calls recorded in the main checkpoint files for each benchmark. These logged counts measure LLM inference used for candidate evaluation on the development subset during evolutionary search.

Table 7: Search-time LLM inference usage recorded from the main checkpoint files. Counts correspond to development-set candidate evaluation during evolution and exclude validation and elite folders, preliminary experiments, baseline reruns, and final held-out evaluation.

Benchmark	Logged search-time tokens	Logged search-time API calls
MMLU	351.05M	135,880
MMLU-Pro	486.05M	174,500
GSM8K	128.02M	66,120
WorkBench	998.19M	172,543
Total	**1,963.31M**	**549,043**

Wall-clock time is reported as an operational measure because it depends on API latency, rate limits, retry behavior, and endpoint availability. Table 8 reports the recorded wall-clock time for evolutionary search on each benchmark under our API setting.

Table 8: Recorded search-time wall-clock cost under our API setting. Times correspond to evolutionary search on the fixed development subsets and exclude held-out evaluation, preliminary experiments, and baseline reruns.

Benchmark	Recorded wall-clock time
MMLU	20.0h
MMLU-Pro	25.7h
GSM8K	9.7h
WorkBench	20.0h
Total	**75.4h**

The supplemental code package includes the scripts, configuration files, and evaluation commands used to reproduce the API-based experiments.

Scope and cost. TCP-MCP is evaluated under a fixed backbone and fixed development-subset protocol, with final results reported on held-out test sets. This design avoids test-set leakage and keeps comparisons controlled, but broader validation across additional backbones and search budgets remains future work. Since TCP-MCP is a Pareto-based optimizer, the selected high-performance operating points may require more tokens than lower-cost baselines; however, searched prompt–topology patterns can be reused as fixed templates or transferred to related settings, which can amortize part of the initial search cost.

Additional Cost Visualization Figure 6 visualizes the accuracy and token cost reported in Table 1. Across MMLU, MMLU-Pro, GSM8K, and WorkBench, TCP-MCP is not always the lowest-token method, but it remains in a high-performance region while avoiding the much larger token costs of high-interaction baselines such as LLM-Debate and Random Graph. This figure complements the table by showing that the selected TCP-MCP operating points emphasize task performance while keeping token usage below the most expensive multi-agent protocols.

G Second Evaluation on Main Datasets

We conduct a second evaluation on the main datasets as an additional stability check. This second evaluation repeats inference for the selected systems under the same held-out evaluation protocol, but does not rerun evolutionary search. As shown in Table 9, the second-run results follow the same overall trend as the main results, with TCP-MCP remaining in the high-performance region across all four datasets. We therefore use this check as additional robustness evidence rather than as a multi-seed evolutionary result.

Table 9: Second held-out evaluation of the selected TCP-MCP systems. The second run repeats inference under the same held-out protocol without rerunning evolutionary search.

Dataset	Main result	Second run	Difference
MMLU	89.96	89.10	-0.86
MMLU-Pro	82.66	82.33	-0.33
GSM8K	96.61	96.40	-0.21
WorkBench	77.31	77.88	+0.57

H Statistical Considerations

Reported performance may vary with evolutionary initialization, LLM stochasticity, and development-set selection. Ablation and transfer results are interpreted within the evaluated configurations and experimental settings.

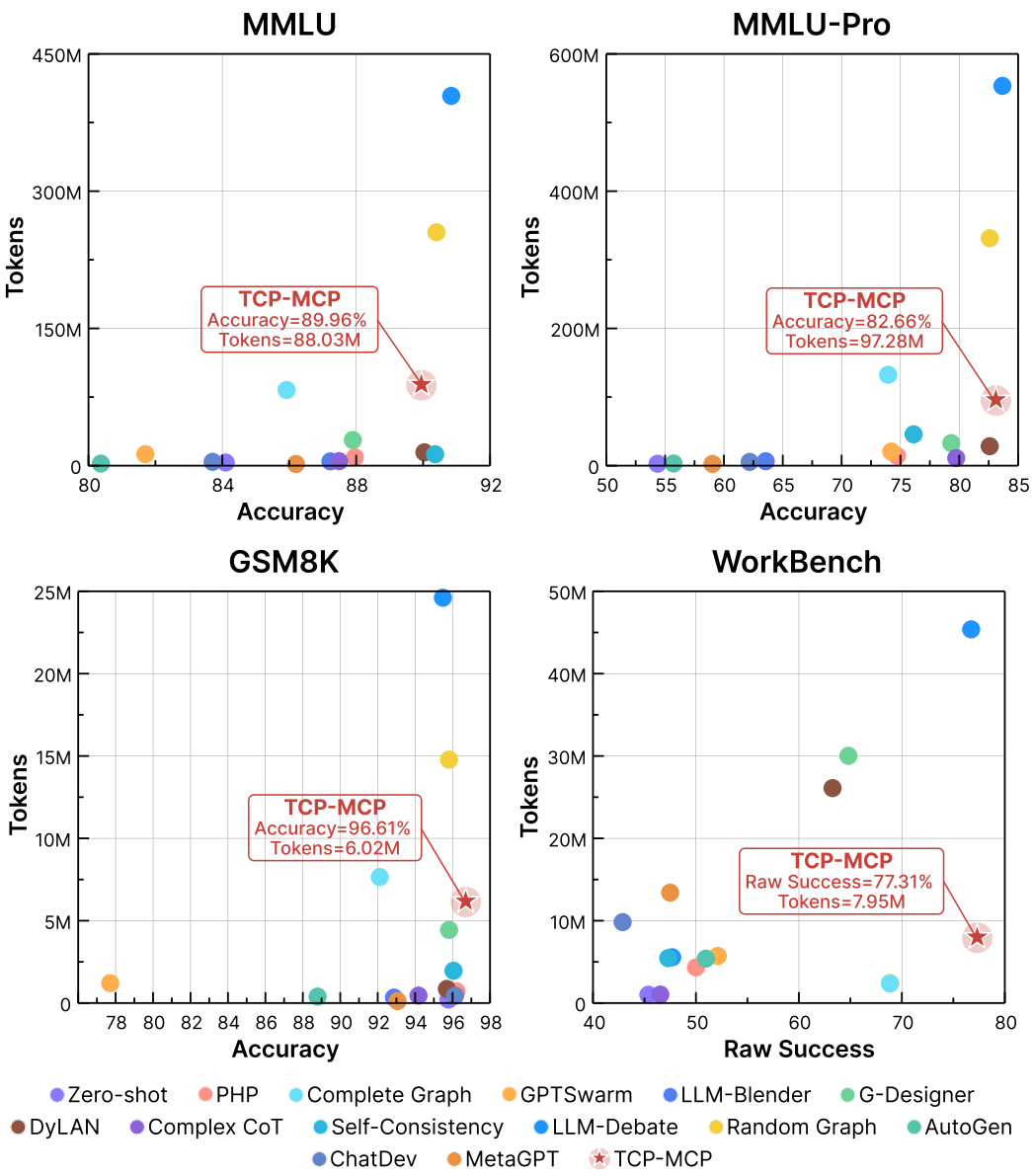

Figure 6: Supplementary visualization of accuracy and token cost for the main results in Table 1. Each point corresponds to one method on the held-out evaluation set. The horizontal axis shows accuracy, and the vertical axis shows total held-out inference tokens. TCP-MCP is highlighted with a dashed circle.

I Operating Point Selection

TCP-MCP returns a Pareto set rather than a single system. We therefore use a deterministic rule to select one operating point from the development-set Pareto front before held-out evaluation.

Let $\mathcal{P}$ denote the final development-set Pareto set and let $\widetilde{C}$ denote the median token cost among its candidates. We define a cost envelope

$$B = 1.05\,\widetilde{C}. \tag{13}$$

We retain Pareto candidates whose development-set token cost satisfies $C(M) \le B$ and select the candidate with the highest development-set task performance:

$$M^\star = \arg \max_{M=(G,P)\in\mathcal{P},\, C(M)\le B} \big(A(M), -K(G), -C(M)\big). \tag{14}$$

where the tuple is compared lexicographically. Thus, task performance is the primary criterion, while structural complexity and token cost are used only to break ties. The median-cost reference makes the rule insensitive to high-cost outliers on the Pareto front, while the fixed 5% slack avoids changing the selected system because of small differences in token usage.

The same rule is applied across benchmarks. The selected genome is fixed before held-out evaluation and is subsequently used unchanged on the evaluation set. Held-out performance is not used for genome selection or operating-point adjustment.

I.1 WorkBench Execution and Evaluation

WorkBench evaluates tool-use tasks by comparing the resulting environment state with a reference state. TCP-MCP executes the selected prompt–topology genome, with node-level prompts guiding constraint checking, action auditing, and outcome verification. These roles contribute to task execution, but their textual judgments do not determine benchmark success. State-changing actions are replayed in the benchmark environment, and the resulting state is used for evaluation. Action replay is therefore part of the execution and evaluation procedure, rather than an additional stage of prompt or topology optimization. A task is counted as successful according to the benchmark's state-based success criterion; a natural-language claim of completion alone is insufficient. We report raw success over the complete held-out set. The selected TCP-MCP genome remains fixed throughout held-out evaluation, without query-specific evolutionary search.

J Initialization Landscape-Probing Ablation

TCP-MCP uses Fitness Distance Correlation (FDC) only at initialization to obtain a coarse signal about the search landscape and calibrate the initial search behavior. To isolate the contribution of this choice, we compare the default FDC-guided initialization with a neutral initialization that removes only the FDC guidance. Both settings use the same population, search space, evolutionary operators, development examples, and evaluation budget.

Table 10 reports normalized Pareto hypervolume (HV) during the first three generations. The two settings start from the same HV at generation 1. By generation 3, FDC-guided initialization reaches an HV of 0.903400, compared with 0.883945 under neutral initialization, an absolute difference of 0.019455 (approximately 2.20% relative to the neutral value). The mean HV over the three generations is 0.889241 for FDC-guided initialization and 0.883351 for neutral initialization; the corresponding trapezoidal HV–AUC values are 1.774943 and 1.766999.

Table 10: Controlled initialization analysis. Normalized Pareto hypervolume (HV) over the first three generations. FDC guides initialization only; subsequent search components remain unchanged.

Initialization	Gen. 1	Gen. 2	Gen. 3
FDC-guided	0.882162	0.882162	**0.903400**
Neutral	0.882162	0.883945	0.883945

This lightweight, short-horizon analysis supports the intended role of FDC: it can guide the search toward a stronger early Pareto region. It does not establish that FDC alone determines final performance, nor does it replace a full multi-seed analysis of final evolutionary outcomes. Later-stage search is governed by the Pareto diagnostics, adaptive operators, and environmental selection described in the main paper.

K Ablation Study Details

We ablate the main search components of TCP-MCP to examine how they contribute to finding effective prompt–topology systems. All variants use the same backbone, development subset, held-out evaluation protocol, population size, elite size, and development-set operating-point selection rule as the full method.

AB1 removes adaptive rate control by fixing the crossover, topology mutation, and role mutation rates throughout evolution. The initialization-time FDC probe is retained, but subsequent rate adjustments based on stagnation and diversity signals are disabled. AB2 removes structural crossover, so offspring are generated only through mutation. This prevents the exchange of subgraphs together with their associated roles and prompt templates. AB3 removes mutation-related operations, leaving crossover as the variation mechanism without additional topology edits, role changes, or prompt regeneration.

AB4 replaces simultaneous co-evolution with sequential prompt-first topology search. It retains the same population and elite sizes and allocates the same total of 21 generations across two stages. Prompts are first optimized for 11 generations under a fixed topology. The selected development-set solution then supplies the fixed prompt configuration for 10 generations of topology optimization. The final system is selected using the same development-set rule and evaluated on the held-out set. This matches the generation budget, although realized token costs can differ across search procedures.

The distinction tested by AB4 is how the two components are optimized, rather than whether both are included in the search space. In full TCP-MCP, structural crossover transfers subgraphs with their role–prompt assignments, and role mutation regenerates the corresponding prompts. Each resulting combination is evaluated as an executable system, allowing subsequent search to retain or revise both its behavior and communication structure. AB4 also optimizes both components, but commits to a prompt configuration before searching the topology in its second stage.

Full TCP-MCP achieves the highest accuracy across all four benchmarks. Removing crossover produces the largest drop on MMLU-Pro, reducing accuracy by 2.51 percentage points. Removing mutation produces the largest drops on MMLU and WorkBench, by 2.59 and 8.27 points, respectively. These results support complementary roles for the two operators: crossover reuses existing role–prompt subgraphs in new structural contexts, while mutation introduces topology edits and role-conditioned prompt changes beyond inherited modules.

Compared with AB4, full TCP-MCP improves accuracy by 2.48 points on MMLU, 0.36 on MMLU-Pro, 0.34 on GSM8K, and 5.96 on WorkBench. The larger gains on MMLU and WorkBench support the benefit of allowing prompts and topology to adapt throughout the same search. In particular, the results suggest that selecting prompts under one fixed communication structure can constrain their usefulness when that structure subsequently changes. This comparison supports co-evolution over the tested prompt-first schedule under the same generation budget.

The ablations evaluate the complete search procedure through the systems it ultimately selects. Changes in accuracy can therefore accompany changes in deployment token cost and structural complexity. A lower-cost ablated system is not necessarily a worse trade-off, just as higher accuracy alone does not establish greater efficiency. Taken together, the results show that structural recombination, mutation, and joint updates help TCP-MCP find higher-performing systems under the shared evaluation and selection protocol.

L HYPERVOLUME DIAGNOSTICS

Table 11 reports normalized Hypervolume (HV) at generations 1, 11, and 21. These checkpoints summarize the Pareto-front trend after the first generation, in the middle of evolution, and at the final reported generation.

Table 11: Normalized Hypervolume (HV) diagnostics across generations.

Dataset	**Gen. 1**	**Gen. 11**	**Gen. 21**
MMLU	0.5905	0.6945	0.6963
MMLU-Pro	0.3698	0.5036	0.6098
GSM8K	0.7939	0.8399	0.8431

M IMPLEMENTATION SETTINGS.

The initialization probe uses the dissimilarity $d = (d_{\text{topo}} + 3d_{\text{role}})/4$, combining topology-set Jaccard distance with directional nearest-role semantic dissimilarity and penalties for role-count

differences. The reference genome maximizes the initialization preference score. Its Spearman FDC ρ sets the crossover probability to $0.4 - 0.1\sigma(-2\rho)$, while topology and role mutation start at 0.30, subject to low-diversity adjustment. Environmental selection uses a fixed accuracy tolerance $\delta = 0.08$, giving the dynamic feasibility threshold $\max(0, a_{\text{best},t} - 0.08)$; infeasible candidates receive lower priority rather than being automatically discarded. In the implementation, controller weights for HV stagnation, accuracy stagnation, token growth, and diversity deficit interpolate from $(0.30, 0.10, 0.10, 0.50)$ at $g \leq 2$ to $(0.50, 0.30, 0.05, 0.15)$ at $g = 12$. Radical mutation is attempted after ordinary topology mutation with conditional probability $\sigma(5s_t - 3.5)$. Default HV computation clips objectives to fixed bounds, uses reference point $(0, -2500000, -\log 1001)$, and normalizes by $2500000 \log 1001$. Selection uses 3×3 cost–complexity quantile bins with an initial cap of two elites per bin, relaxed when needed, and seeks to retain two high-node candidates satisfying the accuracy threshold.

N Cost-Controlled Comparison with EvoMAS

Motivation. EvoMAS (Hu et al., 2026) and TCP-MCP differ in when evolutionary optimization takes place. TCP-MCP searches for a prompt–topology genome on development data and deploys the selected system directly at test time. EvoMAS instead continues to evolve candidate multi-agent configurations while processing each query. TCP-MCP therefore pays the search cost once and reuses the selected genome across queries, whereas EvoMAS repeatedly incurs evolutionary cost during inference.

This difference makes a full EvoMAS evaluation substantially more expensive. In our 500-query evaluation, EvoMAS uses an average of 132,353 tokens per query on MMLU and 734,531 on MMLU-Pro, compared with 6,274 and 8,210 tokens for TCP-MCP. Extrapolating these measured averages to the complete held-out sets gives approximately 1.85 billion tokens for MMLU and 8.84 billion for MMLU-Pro, together with roughly 0.64 million and 1.71 million model calls. A complete EvoMAS evaluation on both benchmarks would thus require more than 10 billion tokens and 2 million model calls.

We therefore evaluate EvoMAS on fixed subsets of MMLU and MMLU-Pro. This choice follows the motivation of tinyBenchmarks (Maia Polo et al., 2024), which studies lower-cost evaluation when exhaustive testing of large language models is prohibitively expensive. WorkBench is much smaller, so we evaluate EvoMAS on the complete held-out set.

Evaluation protocol. For MMLU and MMLU-Pro, we sample 500 queries without replacement from the held-out evaluation sets using random seed 42. The subsets are fixed before evaluation, and TCP-MCP and EvoMAS are tested on exactly the same question IDs. TCP-MCP is not searched again on these subsets; it directly reuses the genome selected on development data. EvoMAS follows its query-time evolutionary procedure, with cross-query memory and the configuration pool enabled throughout evaluation.

For WorkBench, both methods are evaluated on the complete held-out set without subset sampling. TCP-MCP reuses the main held-out result reported in Table 1. EvoMAS is independently evaluated three times over all 520 tasks, and we report its mean performance across the three complete runs. The MMLU and MMLU-Pro results therefore use matched 500-query subsets, while the WorkBench comparison covers the complete held-out set.

As a check on the subset evaluation, Table 12 compares TCP-MCP accuracy on the 500-query subsets with its corresponding complete held-out results.

Table 12: TCP-MCP accuracy on the fixed 500-query subsets and complete held-out sets.

Dataset	Size	Subset Acc.	Full Acc.	Difference
MMLU	500	90.20	89.96	+0.24
MMLU-Pro	500	84.80	82.66	+2.14

Implementation. We use the released EvoMAS implementation with the same DeepSeek-V3.2 backbone as in our main experiments. We retain the original evolutionary procedure and modify only

the components needed for API compatibility, benchmark support, and token and model-call accounting. The evaluated configuration uses $k = 2$ parent configurations and $T = 2$ evolutionary steps per query. Cross-query memory and the configuration pool remain enabled throughout evaluation.

For WorkBench, we keep the same EvoMAS evolutionary procedure and adapt the released implementation to the benchmark environment. We restore the required benchmark dependencies and tool interfaces following the WorkBench implementation, isolate per-process file locks, use atomic updates for shared experience memory, and keep the inference provider fixed throughout evaluation. The same setup is used for all three complete EvoMAS WorkBench runs.

Accuracy and inference cost. Table 13 compares TCP-MCP and EvoMAS across the three benchmarks. MMLU and MMLU-Pro are evaluated on the matched 500-query subsets. On WorkBench, both methods are evaluated on the complete held-out task set, with EvoMAS evaluated over three complete runs.

On MMLU, TCP-MCP reaches 90.20% accuracy with 6,274 tokens per query, compared with 84.60% and 132,353 tokens for EvoMAS. EvoMAS therefore uses 21.1× as many tokens per query.

On MMLU-Pro, TCP-MCP reaches 84.80% accuracy with 8,210 tokens per query, while EvoMAS reaches 76.80% with 734,531 tokens. The resulting token-cost ratio is 89.5×.

On the complete WorkBench held-out set, TCP-MCP achieves 77.31% raw success with 15,289 tokens per task. EvoMAS reaches a mean raw success rate of 57.24% across three complete runs, with an average cost of 257,462 tokens per task. TCP-MCP therefore improves raw success by 20.06 percentage points while EvoMAS uses approximately 16.8× as many deployment tokens per task.

Table 13: Comparison between TCP-MCP and EvoMAS. MMLU and MMLU-Pro use matched 500-query subsets, while WorkBench uses the complete held-out task set. Performance is accuracy (%) on MMLU and MMLU-Pro and raw success (%) on WorkBench. EvoMAS WorkBench performance is averaged over three complete runs. Token counts are mean deployment tokens per query or task, and ratios are relative to TCP-MCP.

Dataset	Method	Performance	Tokens	Ratio
MMLU	TCP-MCP	90.20	6,274	1.0×
MMLU	EvoMAS	84.60	132,353	21.1×
MMLU-Pro	TCP-MCP	84.80	8,210	1.0×
MMLU-Pro	EvoMAS	76.80	734,531	89.5×
WorkBench	TCP-MCP	77.31	15,289	1.0×
WorkBench	EvoMAS	57.24	257,462	16.8×

Paired accuracy analysis. Because TCP-MCP and EvoMAS are evaluated on identical question IDs for MMLU and MMLU-Pro, we also apply exact McNemar tests to their paired predictions.

On MMLU, TCP-MCP is correct and EvoMAS incorrect on 45 queries, while the reverse occurs on 17 queries. This corresponds to a 5.60 percentage-point accuracy difference and an exact McNemar p-value of 4.97×10^{-4}.

On MMLU-Pro, the corresponding discordant counts are 57 and 17, giving an 8.00 percentage-point accuracy difference and an exact McNemar p-value of 3.40×10^{-6}.

Table 14: Paired accuracy analysis on the matched 500-query subsets.

Dataset	TCP-MCP only	EvoMAS only	Difference	McNemar p
MMLU	45	17	+5.60	4.97×10^{-4}
MMLU-Pro	57	17	+8.00	3.40×10^{-6}

Inference-cost distribution. The deployment-cost gap follows directly from the two inference procedures. Once TCP-MCP has selected a genome, each held-out query executes the fixed multi-agent system once. EvoMAS instead evaluates multiple candidate configurations while processing

each query, which introduces repeated agent executions as well as additional meta-model and evaluation calls.

Table 15 reports the per-query token distributions for MMLU and MMLU-Pro. On MMLU, EvoMAS has a median cost of 103,288 tokens and reaches 1,276,094 tokens on the most expensive query, compared with 6,132 and 10,169 tokens for TCP-MCP. On MMLU-Pro, the corresponding EvoMAS costs are 355,126 and 14,610,070 tokens, compared with 7,506 and 26,501 for TCP-MCP.

Table 15: Distribution of per-query token usage on the matched 500-query subsets.

Dataset	Method	Median	Mean	P99	Maximum
MMLU	TCP-MCP	6,132	6,274	9,146	10,169
MMLU	EvoMAS	103,288	132,353	599,021	1,276,094
MMLU-Pro	TCP-MCP	7,506	8,210	19,204	26,501
MMLU-Pro	EvoMAS	355,126	734,531	7,578,167	14,610,070

Execution outcomes. For the MMLU and MMLU-Pro subset evaluations, we also record EvoMAS execution outcomes separately from ordinary incorrect predictions. Table 16 reports invalid generated configurations, executions that produce no valid answer, timeouts, and infrastructure failures.

Table 16: Execution outcomes for EvoMAS on the matched 500-query subsets.

Dataset	Invalid Config.	No Answer	Timeout	Infrastructure
MMLU	35	3	0	0
MMLU-Pro	28	18	2	1

WorkBench evaluation. WorkBench differs from MMLU and MMLU-Pro in both scale and evaluation protocol, so we use the complete held-out set rather than a sampled subset. TCP-MCP reuses the main held-out evaluation reported in Table 1, while EvoMAS is evaluated over three complete runs of all 520 tasks.

WorkBench evaluates tool-use behavior from the resulting environment state. State-changing actions generated by the system are replayed in the benchmark environment, and success is determined by comparing the resulting state with the reference state. Across three complete runs, EvoMAS succeeds on 293, 302, and 298 of the 520 tasks, corresponding to raw success rates of 56.35%, 58.08%, and 57.31%, respectively, with a mean of 57.24%. Its average deployment cost is 257,462 tokens per task. TCP-MCP succeeds on 402 of the 520 tasks, reaching 77.31% raw success with 15,289 tokens per task.

The complete WorkBench evaluation therefore preserves a substantial performance–cost gap: TCP-MCP improves raw success by 20.06 percentage points, while EvoMAS requires approximately 16.8× as many deployment tokens per task. The success-rate difference is computed from the unrounded task counts. This result complements the cost-controlled subset comparisons on MMLU and MMLU-Pro by evaluating tool-use performance over the full held-out workload.

Search-inclusive cost analysis. The deployment ratios above do not include TCP-MCP's one-time evolutionary search cost. We therefore also account for this cost on MMLU and MMLU-Pro. Let S_{TCP} denote the one-time TCP-MCP search cost and c_{TCP} its per-query deployment cost. For a workload of N queries,

$$C_{\mathrm{TCP}}(N) = S_{\mathrm{TCP}} + Nc_{\mathrm{TCP}}. \tag{15}$$

EvoMAS performs evolutionary adaptation during query processing. Using its measured average per-query cost c_{EvoMAS},

$$C_{\mathrm{EvoMAS}}(N) \approx Nc_{\mathrm{EvoMAS}}. \tag{16}$$

The break-even workload is therefore

$$N^\star = \frac{S_{\mathrm{TCP}}}{c_{\mathrm{EvoMAS}} - c_{\mathrm{TCP}}}. \tag{17}$$

Using the search-time token counts recorded from the main checkpoint files, TCP-MCP uses 351.05M search tokens on MMLU and 486.05M on MMLU-Pro. Including the complete held-out deployment costs reported in the main evaluation brings the corresponding combined search-evaluation and deployment costs to 439.08M and 583.33M tokens. Extrapolating the measured EvoMAS per-query costs to the complete held-out workloads gives 1,847.8M tokens on MMLU and 8,837.9M on MMLU-Pro. The resulting break-even workloads are approximately 2,784 queries for MMLU and 669 for MMLU-Pro.

Table 17: Search-inclusive cost comparison with projected complete-set EvoMAS cost. TCP Search reports the one-time evolutionary search cost, while TCP All-in includes held-out deployment cost.

Dataset	TCP Search	TCP All-in	EvoMAS Proj.	Break-even	Ratio
MMLU	351.05M	439.08M	1,847.8M	2,784	4.21×
MMLU-Pro	486.05M	583.33M	8,837.9M	669	15.15×

O Controlled LLM-Debate Configuration Sweep

LLM-Debate is one of the strongest baselines in our full held-out evaluation. As reported in Table 1, it exceeds TCP-MCP on MMLU and MMLU-Pro by 0.88 and 1.01 percentage points, respectively, but requires substantially more inference cost. We therefore examine whether this accuracy–cost pattern is specific to the reported debate configuration or persists across different interaction budgets.

We run a controlled configuration sweep on a matched MMLU-Pro evaluation sub-batch. The evaluation examples, DeepSeek-V3.2 backbone, prompts, decoding settings, answer extraction, scoring procedure, and aggregation rule are held fixed, while only the number of debate agents and rounds is varied. The purpose of this sweep is to characterize the accuracy–cost curve of LLM-Debate rather than replace the full held-out comparison in Table 1.

Table 18: Controlled LLM-Debate sweep on the matched MMLU-Pro evaluation sub-batch. All settings use the same backbone and evaluation protocol; token usage and model calls are reported per query.

Method	Agents	Rounds	Acc. (%)	Tokens/Query	Calls/Query
LLM-Debate	2	1	82.0	1,794.88	2
LLM-Debate	3	1	83.0	2,685.39	3
TCP-MCP	–	–	85.0	8,237.24	4
LLM-Debate	6	3	**88.0**	46,380.32	18

The sweep shows a clear accuracy–cost trade-off. With two or three agents and a single round, LLM-Debate costs less than TCP-MCP but reaches only 82.0% and 83.0% accuracy. Increasing the budget to six agents and three rounds raises accuracy to 88.0%, above TCP-MCP on the matched sub-batch, but requires 46,380.32 tokens and 18 model calls per query. TCP-MCP uses 8,237.24 tokens and 4 calls per query.

These results place the full held-out comparison in context. LLM-Debate can achieve higher accuracy with a sufficiently large interaction budget, consistent with its results in Table 1. When its budget is reduced enough to lower inference cost substantially, however, its accuracy also decreases. TCP-MCP therefore operates at a different point on the accuracy–cost curve: it does not outperform every debate configuration, but provides a reusable system with a competitive balance between accuracy and inference cost.