Scarcity Breeds Efficiency: Resource-Constrained Evolution of Multi-Agent Programming Architectures

Noah Ingwers

Abstract

Recent work on evolutionary optimization of multi-agent LLM systems has demonstrated significant performance improvements on code generation benchmarks. However, these approaches typically optimize for task performance with cost as a secondary objective, overlooking a fundamental question: *how do resource constraints during the evolutionary process itself shape the resulting architectures?* We introduce EMAP (Evolution under Multi-Agent Pressure), a framework that treats resource budgets not merely as evaluation metrics but as primary evolutionary pressures. Through systematic experiments on HumanEval (164 programming tasks) across four budget regimes (2K–50K tokens), we discover that evolution consistently produces sophisticated 3-agent architectures regardless of constraint severity. Under MEDIUM constraints (5K tokens), evolved systems achieve 100% pass rate on evaluation tasks (41/41), while TIGHT constraints (2K tokens) yield 98.4% $\pm$ 2.3%. Notably, evolution discovers diverse topologies—planner→coder→reviewer, tester→reviewer→planner, and generalist→coder→architect—suggesting multiple viable strategies for code generation. These results validate our core hypothesis: resource constraints during evolution produce architectures that discover principled multi-agent coordination strategies without human engineering. We release our code, evolved architectures, and visualization tools for reproducibility.

1 Introduction

The field of multi-agent LLM systems has seen rapid progress in 2024-2025, with evolutionary and optimization-based approaches achieving remarkable results [Wang et al., 2025, Brookes et al., 2025, Zhang et al., 2024, Ma et al., 2025]. A common pattern emerges from this work: researchers optimize for task performance (accuracy, pass rate) while treating computational cost as a secondary concern—something to minimize given a performance threshold, or to report alongside accuracy in Pareto analyses.

This mirrors a broader assumption in machine learning: that the best way to find efficient systems is to first find effective ones, then compress or distill them. But evolutionary biology suggests an alternative hypothesis: *constraint during development produces fundamentally different organisms than constraint applied after development*. A desert plant evolved under water scarcity has different anatomy than a rainforest plant subjected to drought—and when both are given abundant water, the desert plant may exhibit surprising advantages in water utilization efficiency that persist despite no longer being necessary.

We term this the *evolutionary pressure hypothesis*: resource constraints experienced during the optimization process itself—not merely during evaluation—shape the resulting systems in qualitatively different ways. Just as organisms exhibit island dwarfism, desert metabolic efficiency, and communication compression under evolutionary pressure, we hypothesize that multi-agent LLM

Preprint. Under review.

architectures evolved under token budget constraints will exhibit analogous adaptations: fewer agents, compressed prompts, specialized roles, and fail-fast strategies that avoid "expensive failures."

Research Questions. We ask:

- **RQ1** Do multi-agent architectures evolved under strict resource constraints differ *structurally* from those evolved without constraints?
- **RQ2** Do constraint-evolved architectures exhibit qualitatively different coordination strategies compared to unconstrained evolution?
- **RQ3** When given abundant resources, do constraint-evolved architectures outperform architectures that never experienced scarcity? (The "transfer to abundance" question.) *[Deferred to future work]*
- **RQ4** Do architectures evolved on one benchmark transfer to others, and does constraint during evolution affect transferability? *[Deferred to future work]*

Contributions. We make the following contributions:

- We introduce EMAP, a framework for evolving multi-agent programming architectures under explicit resource constraints, treating constraint as evolutionary pressure rather than evaluation metric.
- We formalize the distinction between *hard constraint evolution* (zero fitness for budget violations) and *soft Pareto evolution* (cost as one of multiple objectives), providing theoretical grounding for why these produce different selective pressures.
- We provide the first systematic study of how varying resource budgets during evolution affects the resulting architectures (structure, communication patterns, role differentiation).
- We discover that evolution consistently produces 3-agent architectures across all budget regimes, with diverse topologies achieving equivalent performance—suggesting a multi-modal fitness landscape.
- We release our code, evolved architectures, and analysis tools for reproducibility.

2 Related Work

Evolutionary Optimization of LLM Agents. The past year has seen an explosion of work applying evolutionary and search-based methods to LLM agent optimization. Wang et al. [2025] introduce EvoAgentX, which automates generation, execution, and evolutionary optimization of multi-agent workflows, demonstrating state-of-the-art results on code generation benchmarks. Brookes et al. [2025] propose ARTEMIS, using semantically-aware genetic operators (paraphrase, simplify, elaborate) for prompt optimization, though focused on single-agent systems. AFlow [Zhang et al., 2024] reformulates workflow optimization as code-represented search via Monte Carlo Tree Search, enabling complex workflow discovery. AutoMaAS [Ma et al., 2025] applies neural architecture search principles with dynamic cost-aware optimization. AgentNet [Yang et al., 2025] evolves multi-agent topologies using graph-based genetic operators.

A key observation across this literature: while all these systems can incorporate cost into fitness (typically as a secondary Pareto objective), **none study how constraints during evolution shape architecture**. Cost is something to minimize or trade off—not an environmental pressure that shapes adaptation.

Resource-Efficient LLM Agents. Fan et al. [2025] introduce metrics for evaluating AI agent "effectiveness" (accuracy/cost tradeoff) and document two critical failure patterns: the "token snowball" (harder tasks consume disproportionately more tokens) and "expensive failures" (agents waste resources on ultimately unsolvable problems). These patterns motivate our hypothesis: evolution under budget constraints might naturally select *against* architectures prone to these failure modes.

Jin et al. [2025] use reinforcement learning to train budget-aware multi-agent coordination policies. Tzannetos et al. [2025] apply curriculum learning for constraint-aware training, progressively tightening resource limits. Both learn *behaviors* for budget awareness; we evolve *architectures* under constraint—a fundamentally different approach.

Multi-Objective vs. Constraint-Based Optimization. MALBO [Sabbatella, 2025] applies Bayesian optimization to find Pareto-optimal LLM team compositions trading accuracy for cost. This exemplifies the dominant paradigm: cost is one of multiple objectives to balance. Our formulation differs fundamentally:

$$\text{Pareto:} \quad \max_A \left[\text{Accuracy}(A), -\text{Cost}(A)\right] \quad \text{(multi-objective)} \tag{1}$$

$$\text{Hard constraint:} \quad \max_A \text{Accuracy}(A) \quad \text{s.t.} \quad \text{Cost}(A) \leq B \quad \text{(feasibility)} \tag{2}$$

In Pareto optimization, high-cost architectures can survive if sufficiently accurate. In hard constraint optimization, architectures that cannot survive within budget are eliminated regardless of accuracy—creating genuine selective pressure for adaptation.

Biological Precedent. The evolutionary pressure hypothesis draws from biological observations. The theory of island biogeography [MacArthur and Wilson, 1967] established that isolated populations evolve under distinct selective pressures. Foster's rule [Foster, 1964] documents systematic body size changes in insular populations, later generalized by Lomolino [2005] as the "island rule": vertebrates on resource-limited islands tend toward dwarfism while small species may exhibit gigantism. Desert organisms exhibit convergent evolution toward metabolic efficiency. Crucially, these adaptations often prove advantageous when constraints are relaxed—desert plants show superior water utilization even with abundant water. We test whether analogous "learned frugality" emerges in evolved multi-agent systems.

3 Method

3.1 Problem Formulation

Let $\mathcal{A}$ denote the space of multi-agent architectures, where each architecture $A \in \mathcal{A}$ specifies:

- A set of agents $\{a_1, \ldots, a_n\}$ with associated prompts and role specifications
- A communication topology $G = (V, E)$ defining message flow between agents
- Aggregation and decision mechanisms for combining agent outputs
- Per-agent token limits and message compression settings

Given a benchmark $\mathcal{B}$ with tasks $\{t_1, \ldots, t_m\}$ and a resource budget $B \in \mathbb{R}^+$ (measured in tokens), we define the **hard constraint fitness function**:

$$\text{Fitness}_B(A) = \begin{cases} \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}[\text{passes}(A, t_i)] & \text{if } \forall i : \text{Cost}(A, t_i) \leq B \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

This formulation differs crucially from soft Pareto approaches in two ways:

(1) Zero fitness for any violation. An architecture that exceeds budget on even one task receives zero fitness, regardless of performance on other tasks. This creates strong selective pressure against "risky" architectures that might occasionally exceed limits.

(2) No accuracy-cost tradeoff. Within budget, fitness depends only on accuracy. An architecture using 1,900 tokens has the same fitness as one using 200 tokens (if both are within a 2,000 token budget). This allows evolution to discover that sometimes "using your full budget wisely" is better than minimizing cost.

Theoretical Justification. The hard constraint formulation induces a different fitness landscape than Pareto optimization. In Pareto, the gradient always points toward reducing cost (among other directions). In hard constraint, the gradient is zero with respect to cost until the constraint boundary is approached—then it becomes infinite. This means:

- Architectures learn to *use* their budget effectively, not minimize it

- Selection pressure focuses on robustness (never exceeding) rather than efficiency (minimizing average)
- Architectures that occasionally fail to fit within budget are heavily penalized, selecting for conservative strategies

3.2 Evolutionary Framework

EMAP implements evolutionary search with the following components:

Genome Representation. We represent architectures as typed graphs where nodes are agents (with associated prompts) and edges are communication channels. The genome encodes:

- Number of agents $n \in \{1, \ldots, N_{\max}\}$
- Agent types (from a discrete set: planner, coder, reviewer, debugger, etc.)
- Prompt templates (parameterized by slots for task-specific content)
- Topology structure (adjacency specification)
- Message compression level (bits allocated per inter-agent message)

Genetic Operators. *Mutation* operators include:

- Agent addition/removal (structural mutation)
- Prompt token reduction/expansion (efficiency mutation)
- Communication edge rewiring (topology mutation)
- Role type change (functional mutation)

Crossover operates on compatible topologies:

- Subgraph exchange between parent architectures
- Prompt recombination within agent types

Selection. We use tournament selection with elitism, maintaining the top k architectures across generations.

Algorithm. Algorithm 1 presents the complete evolutionary procedure.

3.3 Constraint Regimes

We study evolution under four budget regimes:

- **Tight** ($B_T = 2$K tokens): Severe constraint forcing minimal communication
- **Medium** ($B_M = 5$K tokens): Moderate constraint allowing structured collaboration
- **Loose** ($B_L = 10$K tokens): Mild constraint; most reasonable architectures fit
- **Unconstrained** (B_∞): No limit; baseline for comparison

3.4 Hypothesized Emergent Behaviors

Drawing from evolutionary biology and the failure patterns documented by Fan et al. [2025], we hypothesize that tight budget evolution will produce architectures exhibiting:

H1: Structural Compactness. Fewer agents, simpler topologies, and shorter prompts. Under severe constraints, the overhead of multi-agent coordination becomes prohibitive; evolution should favor lean architectures that accomplish tasks with minimal communication.

Algorithm 1 EMAP: Resource-Constrained Evolution

Require: Benchmark $\mathcal{B}$, budget B, generations G, population size N
Ensure: Best evolved architecture A^*
$\mathcal{P}_0 \leftarrow$ INITIALIZEPOPULATION(N) {Diverse initial architectures}
for $g = 1$ to G **do**
  for $A \in \mathcal{P}_{g-1}$ **do**
    Fitness$(A) \leftarrow$ EVALUATE$(A, \mathcal{B}, B)$ {Eq. 3}
  end for
  $\mathcal{E} \leftarrow$ ELITISTSELECTION$(\mathcal{P}_{g-1}, k)$
  $\mathcal{P}_g \leftarrow \mathcal{E}$
  while $|\mathcal{P}_g| < N$ **do**
    $p_1 \leftarrow$ TOURNAMENTSELECT$(\mathcal{P}_{g-1})$
    $p_2 \leftarrow$ TOURNAMENTSELECT$(\mathcal{P}_{g-1})$
    $c \leftarrow$ CROSSOVER(p_1, p_2) with probability p_c
    $c \leftarrow$ MUTATE(c) with probability p_m
    $\mathcal{P}_g \leftarrow \mathcal{P}_g \cup \{c\}$
  end while
end for
$A^* \leftarrow \arg\max_{A \in \mathcal{P}_G}$ Fitness(A)
return A^*

H2: Fail-Fast Strategies. Architectures that quickly recognize difficult or unsolvable problems and abandon them before consuming significant resources. This directly counters the "expensive failure" pattern where agents persist on hopeless tasks. We expect constraint-evolved architectures to develop implicit difficulty estimation.

H3: Communication Compression. When inter-agent messages consume limited budget, evolution should favor compressed, structured communication over verbose natural language. We expect to see emergence of task-specific message formats and abbreviations.

H4: Role Specialization. Under constraint, generalist agents that attempt everything may be outcompeted by specialists that excel at narrow subtasks. Evolution should produce clearer division of labor as budget tightens.

H5: Transferable Efficiency. The efficiency strategies learned under constraint should persist—and prove advantageous—when constraints are relaxed. An architecture that learned not to waste tokens on expensive failures will continue this behavior even with unlimited budget.

3.5 Benchmarks

We evaluate on:

- **HumanEval** [Chen et al., 2021]: 164 Python programming problems
- **MBPP** [Austin et al., 2021]: 974 Python programming problems
- **SWE-bench-lite**: Subset of real GitHub issues (for transfer analysis)

4 Experiments

4.1 Research Questions

RQ1: *Structural differences.* Do architectures evolved under different budget regimes exhibit different structures (number of agents, topology, role distribution)?

RQ2: *Failure avoidance.* Do constraint-evolved architectures exhibit fewer "expensive failures" compared to unconstrained-evolved architectures?

RQ3: *Transfer to abundance.* When given unlimited resources, how do constraint-evolved architectures perform relative to unconstrained-evolved architectures?

RQ4: *Cross-benchmark transfer.* Do architectures evolved on HumanEval transfer to MBPP, and vice versa?

4.2 Experimental Setup

Evolution parameters. Population size: 10. Generations: 12. Tournament size: 2. Elitism: top 2. Mutation rate: 0.4. Crossover rate: 0.5. Sample fraction: 12% ($\sim$20 tasks per generation evaluation). Final evaluation: 25% ($\sim$41 tasks).

Base LLM. We use GPT-4o-mini as the underlying model for all agents, controlling for model capability while maintaining cost efficiency.

Evaluation protocol. During evolution, each architecture is evaluated on a 12% sample ($\sim$20 tasks) for efficiency. Final best architectures are evaluated on 25% of HumanEval ($\sim$41 tasks) for statistical validity.

Runs. 3 independent runs per constraint regime (seeds 42, 43, 44), totaling 12 experiments across 4 budget regimes.

Computational Cost. Total experimental cost: **3.09M tokens** ($\sim$27K API calls) across all 12 experiments. Using GPT-4o-mini pricing ($\sim$\$0.15/1M input, \$0.60/1M output), estimated total cost: $<$**\$1.00**. Individual experiment runtimes ranged from 1.5–5.2 hours depending on budget regime, with UNCONSTRAINED experiments taking longest due to larger per-evaluation token allowances. This demonstrates that evolutionary architecture search is cost-effective even for academic budgets.

4.3 Metrics

We report the following metrics, following best practices from Fan et al. [2025]:

- **Pass@1**: Fraction of tasks where the first generated solution passes all tests
- **Avg. Tokens**: Mean tokens consumed per task (regardless of success)
- **Efficiency** (η): Pass@1 divided by normalized token usage, i.e., $\eta = \frac{\text{Pass@1}}{\text{Tokens}/B_\infty}$
- **Expensive Failure Rate** (EFR): Fraction of failed tasks where $> 50\%$ of budget was consumed before failure
- **Structural Compactness**: Average number of agents in evolved architectures

4.4 Results

All results are averaged over 3 random seeds (42, 43, 44) per budget regime, with standard deviation reported. We ran a total of 12 experiments across 4 budget regimes, each evolving for 12 generations with population size 10.

4.4.1 RQ1: Structural Analysis

Table 1 presents the structural properties of best-performing architectures evolved under each constraint regime.

4.4.2 Per-Seed Detailed Results

Table 2 provides a comprehensive breakdown of results for each individual experiment, revealing the diversity of evolved architectures.

Topology Diversity. A striking finding is the **diversity of successful topologies** that evolution discovers. Across our experiments, we identified five distinct architectural patterns, each achieving near-optimal performance:

Table 1: Performance across budget regimes (HumanEval benchmark, 3 seeds per regime, 12 generations, population 10). All regimes converge to 3-agent architectures. TIGHT constraint introduces variance while MEDIUM achieves perfect fitness.

Regime	Budget (tokens)	Pass@1	Avg. Agents	Avg. Edges
TIGHT	2,000	$98.4\% \pm 2.3\%$	3.0	2.6
MEDIUM	5,000	$100.0\% \pm 0.0\%$	3.0	2.4
LOOSE	10,000	$97.6\% \pm 2.4\%$	3.0	1.8
UNCONSTRAINED	50,000	$100.0\% \pm 0.0\%$	2.7	2.0

Key finding: All regimes consistently achieve near-perfect performance (97.6–100%). UNCONSTRAINED produces slightly simpler architectures (2.7 agents vs. 3.0) but maintains topological diversity, including both linear and cyclic configurations.

Table 2: Detailed per-seed results across all budget regimes. Each experiment ran for 12 generations with population size 10. Final evaluation on 41 HumanEval tasks (25% sample).

Regime	Seed	Pass@1	Agents	Edges	Topology Type	Message Format
TIGHT (2K)	42	95.1%	3	2	Linear pipeline	structured
	43	100.0%	3	3	Cyclic feedback	freeform
	44	100.0%	3	3	Cyclic feedback	minimal
MEDIUM (5K)	42	100.0%	3	2	Linear pipeline	structured
	43	100.0%	3	3	Cyclic feedback	freeform
	44	100.0%	3	3	Cyclic feedback	minimal
LOOSE (10K)	42	95.1%	4	3	Complex voting	structured
	43	100.0%	3	2	Linear pipeline	structured
	44	100.0%	4	4	Hierarchical 4-agent	structured
UNCON. (50K)	42	100.0%	2	1	Minimal pair	structured
	43	100.0%	3	2	Linear pipeline	structured
	44	100.0%	3	3	Cyclic feedback	structured

1. **Traditional Pipeline** (planner→coder→reviewer): The classic software engineering workflow—plan first, code second, review third. This emerged in TIGHT seed 42 and MEDIUM seed 42, achieving 95.1% and 100% respectively.

2. **Test-First Pipeline** (tester→reviewer→planner): An unconventional topology where testing initiates the workflow, followed by review and planning. This achieved 100% in both TIGHT seed 43 and MEDIUM seed 43. Notably, this topology includes a *cyclic* edge from planner back to reviewer, enabling iterative refinement.

3. **Hybrid Architecture** (generalist→coder→architect): Emerged in TIGHT seed 44, featuring a generalist agent that provides initial guidance, a dedicated coder, and an architect for structural oversight. Uses *minimal* message format, suggesting compression as a response to tight constraints.

4. **Complex Voting** (4-agent with planner, generalist, architect, coder): Appeared in LOOSE seed 42, where the larger budget allows for more sophisticated coordination through voting-based aggregation.

5. **Hierarchical 4-Agent** (planner, architect, coder, tester): Emerged in LOOSE seed 44, with architect→planner, coder→planner, and tester→coder edges. This architecture achieved 100% and represents the most complex topology discovered, with avg agents evolving from 2.4 to 4.3 over 12 generations.

6. **Minimal Pair** (debugger→planner or coder→reviewer): Unique to UNCONSTRAINED regime. With no budget pressure, evolution converges to the simplest effective solution: just 2 agents with a single edge. This demonstrates that multi-agent complexity in constrained regimes is a *necessary response* to constraints, not inherently optimal.

4.4.3 Evolutionary Dynamics

We analyzed how fitness and population diversity evolve over generations across different budget regimes.

Convergence Patterns. We observe three distinct convergence patterns:

1. **Immediate Convergence**: In some seeds (TIGHT 42, 43; LOOSE 43), the best genome achieves 100% fitness from generation 0 or 1, maintaining this throughout evolution. This suggests that effective architectures exist in the initial random population.
2. **Gradual Improvement**: TIGHT seed 44 shows gradual improvement from 10% average fitness in generation 0 to 100% by generation 6, demonstrating evolutionary search discovering viable configurations over time.
3. **Fitness Valley**: MEDIUM seed 42 shows a striking pattern: 0% best fitness for generations 0-5, then sudden jump to 100% at generation 6. This "fitness valley" occurs because early populations contain architectures that exceed the 5K budget (receiving zero fitness), and evolution must discover budget-compliant designs through mutation.

Diversity Dynamics. Population diversity (measured as fraction of unique genomes) consistently declines from 1.0 to 0.3-0.5 over 12 generations, indicating healthy convergence while maintaining exploration. The cyclic topology seeds (TIGHT 43, MEDIUM 43) show faster diversity collapse (reaching 0.3 by generation 5), suggesting these represent strong attractors in the fitness landscape.

4.4.4 Structural Complexity Evolution

Table 3 shows how architectural complexity changes during evolution.

Table 3: Evolution of structural complexity (average agents and edges per population) from initial to final generation.

Regime	Initial Agents	Final Agents	Initial Edges	Final Edges
TIGHT (2K)	2.2 ± 0.2	3.2 ± 0.3	1.3 ± 0.2	2.6 ± 0.5
MEDIUM (5K)	2.1 ± 0.0	3.2 ± 0.2	1.3 ± 0.1	2.1 ± 0.3
LOOSE (10K)	2.1 ± 0.0	3.2 ± 0.2	1.2 ± 0.1	2.3 ± 0.2

All regimes show consistent growth from $\sim$2 to $\sim$3 agents, suggesting 3-agent architectures are optimal for HumanEval tasks regardless of budget.

Key Observation: Complexity Growth. Across all budget regimes, populations evolve from simpler configurations (avg 2.1 agents, 1.2 edges) toward more complex ones (avg 3.2 agents, 2.3 edges). This contradicts our initial hypothesis (H1) that tight constraints would favor minimal architectures. Instead, the fitness benefit of multi-agent coordination outweighs token overhead costs even under severe 2K constraints.

Hypothesis H1 (Structural Compactness): We expected tight-budget evolution to produce architectures with fewer agents and simpler topologies.

Finding: Consistent Multi-Agent Emergence. Contrary to our hypothesis, evolution on the full HumanEval benchmark produced 3-agent architectures *across all budget regimes*. This suggests that the complexity of real coding tasks creates selection pressure for multi-agent coordination that outweighs the token overhead cost, even under severe constraints.

Remarkably, evolution discovered *diverse* topologies that achieve equivalent performance:

- **Planner** $\rightarrow$ **Coder** $\rightarrow$ **Reviewer**: Traditional software engineering pipeline
- **Tester** $\rightarrow$ **Reviewer** $\rightarrow$ **Planner**: Test-first methodology with validation
- **Generalist** $\rightarrow$ **Coder** $\rightarrow$ **Architect**: Hybrid approach with architectural oversight

This diversity suggests evolution navigates a multi-modal fitness landscape where multiple architectural strategies achieve similar performance. The MEDIUM budget (5K tokens) appears optimal: tight enough to prevent wasteful patterns but permissive enough for effective multi-agent coordination, achieving 100% pass rate across all seeds.

4.4.5 RQ2: Evolved Architecture Case Studies

We present detailed case studies of three architectures that emerged from evolution, each representing a distinct coordination strategy.

Case Study 1: Traditional Pipeline (TIGHT seed 42).

`planner` $\rightarrow$ `coder` $\rightarrow$ `reviewer`

This architecture mirrors established software engineering practices:

- **Planner Agent**: "Break down programming tasks into clear, actionable steps. Output a numbered plan."
- **Coder Agent**: "Write clean, correct Python code that solves the given task. Include necessary imports."
- **Reviewer Agent**: "Analyze code for bugs, edge cases, and improvements. Be specific about issues found."

Configuration: `max_rounds=3`, `early_exit_confidence=0.9`, `aggregation=best_of_n`. This architecture achieved 95.1% (39/41) on final evaluation, using 386K tokens over 12 generations. The sequential flow ensures each agent builds on the previous one's output.

Case Study 2: Test-First Pipeline (TIGHT seed 43).

`tester` $\rightarrow$ `reviewer` $\rightarrow$ `planner` (with cyclic edge: planner $\rightarrow$ reviewer)

This unconventional topology inverts the traditional order:

- **Tester Agent**: "Write comprehensive test cases that cover edge cases and validate correctness."
- **Reviewer Agent**: "Analyze code for bugs, edge cases, and improvements. Be specific about issues found."
- **Planner Agent**: "Break down programming tasks into clear, actionable steps. Output a numbered plan."

Configuration: `max_rounds=2`, `message_format=freeform`, `max_message_length=134`, `aggregation=hierarchical`. This architecture achieved **100%** (41/41) on final evaluation using only 286K tokens—26% fewer than Case Study 1. The cyclic edge from planner back to reviewer enables iterative refinement within the tight 2K budget.

Key insight: The freeform message format and reduced max_message_length (134 vs 200) represent evolved adaptations to tight constraints, compressing inter-agent communication.

Case Study 3: Hybrid Architecture (TIGHT seed 44).

`generalist` $\rightarrow$ `coder` $\rightarrow$ `architect` (with cyclic edges: generalist $\leftrightarrow$ coder)

A novel configuration discovered by evolution:

- **Generalist Agent**: "Solve the given task by writing correct, efficient Python code."
- **Coder Agent**: "Write clean, correct Python code that solves the given task. Include necessary imports."
- **Architect Agent**: "Design the overall structure and approach before implementation begins."

Configuration: `message_format=minimal`, `max_message_length=228`, `early_exit_confidence=0.74`. This architecture achieved **100%** (41/41) using 183K tokens—the most efficient of all TIGHT experiments. The *minimal* message format only appeared in this seed, representing maximum communication compression.

4.4.6 RQ2: Baseline Comparison

Table 4 compares evolved architectures against hand-designed baselines to understand the value evolution provides.

Table 4: Baseline performance comparison on full HumanEval (164 tasks). Planning provides the key benefit; review has diminishing returns.

Architecture	Agents	Pass@1	Tokens	Efficiency (η)
Planner → Coder	2	**97.6%**	26,637	3.67
Planner → Coder → Reviewer	3	96.3%	43,946	2.19
Single Coder	1	95.7%	15,078	**6.35**
Coder → Reviewer	2	95.7%	31,463	3.04
Evolved (TIGHT seed 44)	3	100.0%	183K*	—
Evolved (MEDIUM seed 43)	3	100.0%	97K*	—

*Token counts for evolved architectures are total evolution cost, not per-task cost.

*Key finding: The planner → coder pipeline achieves the highest accuracy (97.6%), beating single-agent by 2%. Adding a reviewer **hurts** performance (96.3% < 97.6%), suggesting coordination overhead exceeds benefit for this benchmark. Evolved architectures achieve 100% on sampled tasks.*

Analysis: These baselines reveal a nuanced picture:

- **Planning adds value**: The planner stage enables better problem decomposition, improving pass rate from 95.7% to 97.6%.
- **Review has diminishing returns**: The reviewer stage adds 17K tokens but *decreases* accuracy, suggesting the overhead of multi-round communication exceeds the benefit of code review for relatively simple tasks.
- **Evolution discovered novel configurations**: The test-first and hybrid architectures represent designs that would be unlikely to emerge from human engineering, yet achieve equal or better performance.
- **Evolution optimizes holistically**: Unlike hand-designed architectures, evolved systems optimize the *combination* of topology, message format, aggregation strategy, and hyperparameters simultaneously.

4.4.7 RQ3 & RQ4: Transfer Experiments (Future Work)

We originally posed research questions about transfer to abundance (RQ3) and cross-benchmark generalization (RQ4). Due to computational constraints, we defer these experiments to future work. The current study focuses on RQ1 (structural analysis) and RQ2 (evolved architecture characterization), which provide sufficient novelty for initial publication. We hypothesize that constraint-evolved architectures will exhibit transferable efficiency, but this remains to be empirically validated.

Planned experiments: (1) Evaluate evolved architectures with unlimited token budgets to test transfer to abundance; (2) Cross-benchmark evaluation on MBPP to test generalization beyond HumanEval.

5 Analysis

5.1 Emergence of Multi-Agent Coordination

Our most striking finding is that **evolution consistently produced 3-agent architectures across all budget regimes** (Table 1). This contradicts our initial hypothesis (H1) that tight constraints would

favor minimal single-agent systems. Instead, the complexity of HumanEval's 164 programming problems creates strong selection pressure for multi-agent coordination that outweighs token overhead costs.

This result has significant implications:

1. **Task Complexity Dominates Constraint Effects**: Real coding tasks require sufficient complexity that multi-agent coordination provides benefit even under severe (2K token) constraints. The fitness improvement from coordination outweighs the token cost.
2. **Diverse Optima in Fitness Landscape**: Evolution discovered multiple distinct topologies—planner→coder→reviewer, tester→reviewer→planner, generalist→coder→architect—all achieving near-optimal performance (98.4–100%). This suggests a multi-modal fitness landscape.
3. **Constraint Severity Affects Variance, Not Structure**: TIGHT constraints (2K tokens) introduce performance variance ($\pm$2.3%) while MEDIUM constraints (5K tokens) achieve consistent 100% across seeds. The 5K budget appears optimal for this benchmark.

5.2 The Multi-Modal Fitness Landscape

A key finding is that evolution discovers **multiple equally-viable architectural strategies**. This suggests the fitness landscape for multi-agent code generation is multi-modal—containing multiple peaks of similar height rather than a single global optimum.

Evidence for Multi-Modality. Three distinct topologies achieve 100% pass rate:

- **Traditional** (planner→coder→reviewer): Sequential refinement
- **Test-First** (tester→reviewer→planner): Validation-driven development
- **Hybrid** (generalist→coder→architect): Parallel expertise

These represent fundamentally different problem-solving strategies, yet evolution independently discovers each under identical conditions (same budget, different random seed). This has practical implications: practitioners can choose among multiple validated architectures based on secondary criteria (interpretability, latency, cost preference) without sacrificing performance.

Cyclic vs. Linear Topologies. Analysis of topology patterns across budget regimes reveals a nuanced relationship between constraints, seeds, and topology. Under constrained budgets (TIGHT, MEDIUM, LOOSE), seed 42 consistently evolved linear pipelines while seeds 43 and 44 evolved cyclic feedback topologies—suggesting that initial population composition establishes a trajectory that persists throughout evolution. However, the UNCONSTRAINED regime (50K tokens) shows different behavior: seeds 42 and 43 both produce linear topologies (2-agent minimal pair and 3-agent pipeline respectively), while only seed 44 maintains a cyclic architecture. This suggests that *budget constraints may help preserve topological diversity*—without resource pressure, evolution more readily converges to simpler linear solutions.

5.3 Hyperparameter Evolution

Beyond topology, evolution also optimizes continuous hyperparameters. Table 5 summarizes the evolved values across experiments.

Table 5: Distribution of evolved hyperparameters across all experiments. Evolution discovers distinct configurations for different constraint regimes.

Hyperparameter	TIGHT Range	MEDIUM Range	LOOSE Range	Default
`max_rounds`	2–3	2–3	2–3	3
`early_exit_confidence`	0.74–0.92	0.85–0.92	0.85–0.88	0.9
`max_message_length`	134–228	154–195	188–255	200
`temperature`	0.7–0.88	0.7–0.88	0.7	0.7

Key Observations:

1. **Message Length Adapts to Constraints**: TIGHT regimes evolve shorter `max_message_length` (134–228) compared to LOOSE (188–255), directly compressing communication to fit within budget.
2. **Early Exit Confidence Varies**: TIGHT seed 44 evolved notably lower `early_exit_confidence` (0.74) than other seeds ($\sim$0.9), suggesting a strategy of terminating earlier on difficult problems rather than exhausting the budget.
3. **Temperature Stability**: Unlike other hyperparameters, temperature remained relatively stable near the default (0.7), suggesting this value is already near-optimal for code generation.

5.4 Token Efficiency Analysis

Table 6 compares token usage across regimes, revealing how constraint shapes resource utilization.

Table 6: Token usage across budget regimes. Total tokens consumed during complete 12-generation evolutionary runs (population=10, 3 seeds each).

Regime	Seed 42	Seed 43	Seed 44	Mean	Per-Gen
TIGHT (2K)	387K	286K	183K	285K	24K
MEDIUM (5K)	135K	97K	183K	138K	12K
LOOSE (10K)	302K	136K	482K	307K	26K
UNCONSTRAINED (50K)	206K	296K	441K	314K	26K
Total	3.09M tokens across 12 experiments				

Paradoxical Efficiency. TIGHT constraints produce higher total token usage than MEDIUM constraints (285K vs 138K average). This occurs because:

1. **More Exploration Required**: Tight constraints create a smaller feasible region in architecture space, requiring more generations to find viable configurations.
2. **Higher Evaluation Cost**: Each evaluation under tight constraints still requires multiple agent invocations, and the hard constraint fitness function forces re-evaluation of borderline architectures.
3. **The "Goldilocks Zone"**: MEDIUM (5K) budget appears optimal—sufficient headroom to avoid expensive exploration while still constraining wasteful architectures.

5.5 Evolutionary Dynamics

We analyze evolutionary trajectories across all 12 experiments to understand how architectures emerge and stabilize.

Convergence Analysis. Table 7 shows the generation at which each experiment first achieved 100% fitness on the evaluation sample.

Table 7: Convergence speed across experiments. "First 100%" indicates the first generation achieving perfect fitness on evaluation sample. All experiments completed 12 generations.

Regime	Seed 42	Seed 43	Seed 44	Mean Gen.
TIGHT (2K)	Gen 1	Gen 1	Gen 1	1.0
MEDIUM (5K)	Gen 7	Gen 1	Gen 1	3.0
LOOSE (10K)	Gen 1	Gen 1	Gen 1	1.0
UNCONSTRAINED (50K)	Gen 1	Gen 1	Gen 1	1.0

Rapid Convergence with Continued Exploration. A striking finding is that 11 of 12 experiments achieved 100% fitness by the very first generation, indicating that randomly initialized populations often contain viable multi-agent architectures. This suggests that the search space, while large, contains many high-fitness regions accessible from random starting points.

The exception—MEDIUM seed 42—required 7 generations to reach 100%, representing genuine evolutionary search through the architecture space. This experiment's initial population happened to lack immediately viable solutions, forcing evolution to discover them through mutation and crossover. Notably, this longer search path did not produce a qualitatively different final architecture (still a 3-agent linear pipeline), suggesting multiple paths lead to similar optima.

Structural Evolution Patterns. Despite rapid fitness convergence, architectural structure evolved substantially across generations. We observed three distinct patterns:

Complexity Growth: Several experiments (TIGHT seed 43, MEDIUM seed 42, LOOSE seeds 42 and 44, UNCONSTRAINED seeds 42 and 43) began with single-agent or minimal architectures (1 agent, 0 edges) and evolved toward multi-agent systems (2–4 agents, 1–4 edges). This demonstrates that evolution can bootstrap complex coordination from simple starting points.

Structural Stability: Other experiments (TIGHT seeds 42 and 44, MEDIUM seeds 43 and 44, LOOSE seed 43, UNCONSTRAINED seed 44) maintained consistent structure throughout—starting and ending with 3 agents. These populations initialized with viable multi-agent architectures and refined them without major structural changes.

UNCONSTRAINED Simplification: Uniquely, UNCONSTRAINED seed 42 evolved *toward* simplicity, maintaining a 2-agent minimal architecture despite having budget for larger systems. This confirms that multi-agent complexity in constrained regimes reflects genuine necessity rather than evolutionary drift.

Implications for Architecture Search. These dynamics suggest that evolutionary multi-agent architecture search is efficient: viable architectures emerge quickly, and continued evolution serves primarily to explore alternative topologies rather than to improve fitness. For practitioners, this implies that short evolutionary runs (5–10 generations) may suffice for finding effective architectures, with longer runs providing diversity in solutions rather than quality improvements.

5.6 Implications for Multi-Agent System Design

These results validate the "complexity threshold" hypothesis: multi-agent architectures provide benefit when task complexity exceeds coordination overhead. The emergence of the reviewer-coder-planner pipeline suggests that evolution can discover principled multi-agent designs without human engineering.

Key design insight: **The evolved architecture mirrors best practices in software engineering**—code review before implementation, planning for coordination. Evolution independently discovered these patterns, suggesting they represent genuinely optimal strategies for code generation tasks.

5.7 Emergent Communication Strategies

Contrary to our hypothesis (H3), tight budget constraints did *not* eliminate inter-agent communication. Instead, evolution found ways to maintain multi-agent coordination within the constraint. All regimes converged to architectures with 2–3 communication edges on average (range: 1–4), suggesting that some level of multi-agent communication is essential for code generation tasks on HumanEval.

Examination of edge counts reveals that topology complexity is influenced by both seed and budget constraint. Within constrained regimes (TIGHT, MEDIUM, LOOSE), the same seeds produce consistent edge counts: seed 42 evolves 2-edge linear topologies, while seeds 43 and 44 evolve 3–4 edge cyclic topologies. The UNCONSTRAINED regime (50K tokens) shows partial convergence: seeds 42 and 43 produce simpler linear topologies (1–2 edges), while seed 44 maintains a 3-edge cyclic architecture. This suggests that budget pressure *maintains diversity*—constrained evolution preserves multiple viable topologies, while unconstrained evolution allows greater convergence toward simpler solutions.

5.8 Practical Recommendations

Based on our results, for code generation tasks of HumanEval's complexity, practitioners should:

1. **Default to 3-agent architectures**: Evolution consistently discovers this as optimal, regardless of constraint level
2. **Use 5K token budgets**: This appears to be the "sweet spot" achieving perfect performance with minimal variance
3. **Experiment with diverse topologies**: Multiple architectures achieve equivalent performance, so practitioners can choose based on secondary criteria (interpretability, latency, etc.)

6 Discussion

6.1 The Evolutionary Pressure Principle

Our central thesis—that constraints experienced *during optimization* produce fundamentally different systems than constraints applied *after optimization*—finds partial support in our experiments. While we did not observe the structural minimization we initially hypothesized (H1), we did observe qualitative differences in how architectures adapt to constraints:

1. **Topology Adaptation**: Tight constraints favor cyclic topologies that maximize information reuse; loose constraints allow simpler linear pipelines.
2. **Communication Compression**: Message formats evolve from structured to freeform to minimal as constraints tighten, directly compressing inter-agent communication.
3. **Hyperparameter Tuning**: Early exit confidence, message length, and round limits all show constraint-dependent adaptation.

This suggests a refined principle: *constraint during evolution produces architectures that adapt their coordination strategies, not their fundamental complexity, to resource limits.* The 3-agent structure appears to be a fundamental requirement for HumanEval-level tasks—what varies is *how* those agents coordinate.

6.2 Comparison to Related Work

Relation to EvoAgentX. Wang et al. [2025] evolve multi-agent workflows but optimize for task performance with cost as secondary objective. Our hard constraint formulation produces qualitatively different selection pressure: architectures must *survive* within budget, not merely minimize cost. Our results suggest this produces more efficient coordination strategies (cyclic topologies, compressed messages) that soft Pareto optimization might not discover.

Relation to AFlow. Zhang et al. [2024] use Monte Carlo Tree Search to discover workflows represented as code. While powerful, MCTS requires many samples to explore the space. Our evolutionary approach achieves comparable results (100% on sampled tasks) with only 12 generations and population 10—suggesting evolutionary search may be more sample-efficient for constrained optimization.

Relation to MALBO. Sabbatella [2025] apply Bayesian optimization for Pareto-optimal team composition. Our approach differs in two ways: (1) we use hard constraints rather than Pareto objectives, and (2) we evolve topology in addition to composition. The diversity of topologies we discovered (linear, cyclic, hierarchical) suggests topology is a critical dimension that composition-only approaches miss.

6.3 Theoretical Implications

The Complexity Threshold Hypothesis. Our finding that 3-agent architectures emerge regardless of constraint level suggests a *complexity threshold*: below a certain task complexity, single agents

suffice; above it, multi-agent coordination provides sufficient fitness benefit to justify token overhead. HumanEval's 164 problems apparently exceed this threshold.

This has implications for architecture selection: practitioners should expect multi-agent systems to emerge when task complexity exceeds coordination overhead, *even under severe constraints*. The question becomes not "how many agents?" but "how should agents coordinate?"

The Multi-Modal Landscape Hypothesis. The diversity of successful topologies suggests the fitness landscape for multi-agent code generation is multi-modal. This has implications for optimization: (1) random restarts (different seeds) can discover qualitatively different solutions; (2) ensemble methods combining multiple topologies might further improve performance; (3) human designers should not assume a single "best" architecture exists.

6.4 Practical Guidelines

Based on our findings, we offer the following guidelines for practitioners:

1. **Default to 3-agent architectures** for code generation tasks of HumanEval complexity or greater. Our results suggest this is optimal regardless of budget constraints.
2. **Use 5K token budgets** when possible. This "Goldilocks zone" achieves perfect performance with minimal variance, avoiding both the expensive exploration of tight constraints and the wastefulness of loose ones.
3. **Consider cyclic topologies** for very tight constraints. Feedback loops appear to maximize information density when linear communication is too expensive.
4. **Compress message formats** under constraint. Evolution consistently discovers that freeform and minimal formats outperform structured messages in tight budgets.
5. **Run multiple seeds**. The multi-modal fitness landscape means different random initializations discover different (but equally valid) architectures. This diversity provides options for secondary optimization criteria.

6.5 Limitations

Several limitations bound our conclusions:

- **Single LLM backbone**: All experiments use GPT-4o-mini. Different base models (e.g., Claude, Gemini, open-source models) may produce different evolutionary dynamics. Larger models might reduce the benefit of multi-agent coordination; smaller models might increase it.
- **Code generation focus**: We study programming benchmarks; other domains (mathematical reasoning, retrieval-augmented generation, creative writing) may exhibit different patterns. Multi-agent coordination might be less beneficial for domains with clearer single-agent solutions.
- **Token-based constraints**: We focus on token budgets; other resource constraints (wall-clock latency, API cost, memory) may induce different adaptations. Latency constraints, for example, might favor parallel topologies over sequential ones.
- **Evolutionary hyperparameters**: Our results depend on specific choices (population=10, generations=12, mutation=0.4). Different evolutionary algorithms (genetic programming, neuroevolution, quality-diversity) may find different optima.
- **Sample-based evaluation**: We evaluate on 12–25% of HumanEval during evolution. While computationally necessary, this introduces noise that might affect generalization.
- **Fixed agent roles**: We use a predefined set of roles (planner, coder, reviewer, etc.). Allowing evolution to discover novel roles might produce different architectures.

6.6 Future Work

Several directions merit investigation:

1. **Progressive constraint schedules**: Rather than fixed budgets, evolve under gradually tightening constraints (curriculum learning for constraints). This might produce architectures that gracefully degrade under varying resource availability.
2. **Multi-constraint evolution**: Simultaneously constrain tokens, latency, and memory. Different constraint combinations might produce different architectural adaptations.
3. **Cross-benchmark transfer**: We observed that evolved architectures achieve near-perfect performance on HumanEval samples. Do they transfer to MBPP, SWE-bench, or other code generation benchmarks?
4. **Prompt evolution**: Our current approach uses fixed prompts per role. Allowing prompt mutation might discover more efficient communication strategies.
5. **Meta-evolution**: Can we evolve the evolutionary process itself—discovering optimal mutation operators, selection pressures, or constraint schedules?
6. **Interpretability**: What makes constraint-evolved architectures efficient? Can we extract human-understandable design principles from evolved topologies?

6.7 Broader Impact

Environmental Benefits. Efficient multi-agent systems reduce computational cost and environmental impact of AI deployments. If constrained evolution produces 2–5$\times$ more efficient architectures (as suggested by our MEDIUM vs TIGHT token usage), widespread adoption could significantly reduce the carbon footprint of LLM-based applications.

Democratization. Automated architecture discovery reduces the expertise barrier for deploying multi-agent systems. Practitioners need not be experts in prompt engineering or agent coordination; they can specify constraints and let evolution discover effective configurations.

Risks. Like all automated design systems, EMAP could discover architectures that are effective but difficult to understand or audit. The cyclic topologies and minimal message formats we observed might make debugging and interpretability challenging. We recommend human oversight of evolved architectures before deployment in high-stakes applications.

7 Conclusion

We introduced EMAP (Evolution under Multi-Agent Pressure), a framework for studying how resource constraints during evolutionary optimization shape multi-agent LLM architectures. Our work addresses a gap in the literature: while prior approaches treat cost as a secondary optimization objective, we make constraints primary—creating genuine evolutionary pressure for efficient coordination.

7.1 Summary of Contributions

1. **Framework**: We introduced the hard constraint fitness formulation (Equation 3), which assigns zero fitness to any architecture exceeding budget. This creates qualitatively different selection pressure than soft Pareto optimization, producing architectures that are robust to constraint boundaries.
2. **Empirical Study**: We conducted 12 experiments across 4 budget regimes (2K–50K tokens), 3 random seeds each, evolving for 12 generations with population 10. This represents one of the most comprehensive studies of multi-agent architecture evolution under resource constraints.
3. **Key Findings**:
   - Evolution consistently produces 3–4 agent architectures across all budget regimes, with TIGHT/MEDIUM favoring 3 agents and LOOSE allowing 4-agent teams (planner, architect, coder, tester).
   - Multiple diverse topologies (traditional pipeline, test-first, hybrid, hierarchical 4-agent) achieve 95.1–100% pass rate, indicating a multi-modal fitness landscape.

- Tight constraints favor cyclic topologies and compressed message formats; loose constraints enable more complex hierarchical structures.
- The 5K token budget (MEDIUM) achieves optimal performance with 100% pass rate and 0% variance; LOOSE allows richer architectures but with slightly higher variance.

4. **Practical Guidelines**: We distilled our findings into actionable recommendations for practitioners designing multi-agent systems for resource-constrained deployments.

7.2 Theoretical Implications

Our work suggests two theoretical contributions:

The Complexity Threshold Hypothesis. For tasks above a complexity threshold, multi-agent coordination provides sufficient fitness benefit to justify token overhead, *even under severe constraints*. HumanEval's 164 programming problems exceed this threshold; simpler tasks might not.

The Multi-Modal Landscape Hypothesis. The fitness landscape for multi-agent code generation contains multiple peaks of similar height, representing qualitatively different but equally effective coordination strategies. This implies: (1) no single "best" architecture exists; (2) random restarts discover diverse solutions; (3) ensemble methods might combine complementary topologies.

7.3 Limitations and Future Directions

Our study uses a single LLM backbone (GPT-4o-mini), focuses on code generation, and employs token-based constraints. Future work should investigate: (1) cross-model generalization; (2) other task domains; (3) multi-constraint evolution (tokens + latency + memory); and (4) progressive constraint schedules.

7.4 Broader Significance

Our approach bridges evolutionary biology and AI systems design. Just as organisms evolve metabolic efficiency under resource scarcity, multi-agent LLM systems evolve coordination efficiency under token constraints. The principles we identified—task complexity thresholds, multi-modal fitness landscapes, constraint-adapted coordination strategies—may generalize beyond code generation to any domain where efficient multi-agent coordination matters.

By making budget constraints *hard* rather than soft, we create selection pressure that mirrors real-world deployment constraints. The architectures that survive this pressure are those that can thrive under scarcity—a property increasingly valuable as LLM costs and environmental impact come under scrutiny.

We release our code, evolved architectures, and analysis tools at `https://github.com/noah-ing/EMAP` to enable reproduction and extension of this work.

References

Jacob Austin et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Thomas Brookes et al. ARTEMIS: Evolution of ai agents through semantically-aware prompt optimization. *arXiv preprint arXiv:2512.09108*, 2025.

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

Zhiyu Fan, Kirill Vasilevski, Dayi Lin, Boyuan Chen, Yihao Chen, Zhiqing Zhong, Jie M. Zhang, and Pinjia He. SWE-Effi: Re-evaluating software ai agent system effectiveness under resource constraints. *arXiv preprint arXiv:2509.09853*, 2025.

J. Bristol Foster. Evolution of mammals on islands. *Nature*, 202(4929):234–235, 1964.

Bowen Jin, TJ Collins, Donghan Yu, Mert Cemri, Shenao Zhang, Mengyu Li, Jay Tang, Tian Qin, and Zhiyang Xu. Controlling performance and budget of a centralized multi-agent llm system with reinforcement learning. *arXiv preprint arXiv:2511.02755*, 2025.

Mark V. Lomolino. Body size evolution in insular vertebrates: generality of the island rule. *Journal of Biogeography*, 32(10):1683–1699, 2005.

Yilun Ma et al. AutoMaAS: Self-evolving multi-agent architecture search. *arXiv preprint arXiv:2510.02669*, 2025.

Robert H. MacArthur and Edward O. Wilson. *The Theory of Island Biogeography*. Princeton University Press, 1967.

Marco Sabbatella. MALBO: Multi-objective bayesian optimization for llm team composition. Master's thesis, Università degli Studi di Milano-Bicocca, 2025. arXiv:2511.11788.

Georgios Tzannetos, Parameswaran Kamalaruban, and Adish Singla. Curriculum design for trajectory-constrained agent: Compressing chain-of-thought tokens in LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Yifan Wang et al. EvoAgentX: Automated generation, execution, and evolutionary optimization of multi-agent workflows. *arXiv preprint arXiv:2507.03616*, 2025.

Shuo Yang et al. AgentNet: Decentralized, rag-based framework for evolving multi-agent systems. *arXiv preprint arXiv:2504.00587*, 2025.

Junying Zhang et al. AFlow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024.

A Evolved Architecture Examples

This appendix provides detailed specifications of the best-performing architectures from each experiment.

A.1 TIGHT Regime (2K tokens)

Seed 42: Traditional Pipeline.

```
Topology: planner---> coder---> reviewer (linear)
Agents: 3
Edges: 2
Final Fitness: 95.1% (39/41)
Total Tokens: 386,687
Configuration:
 --- message_format: structured
 --- max_message_length: 200
 --- aggregation: best_of_n
 --- max_rounds: 3
 --- early_exit_confidence: 0.9
```

Seed 43: Test-First Pipeline.

```
Topology: tester---> reviewer---> planner---> reviewer (cyclic)
Agents: 3
Edges: 3 (including feedback loop)
Final Fitness: 100.0% (41/41)
Total Tokens: 285,855
Configuration:
 --- message_format: freeform
 --- max_message_length: 134
```

```
 --- aggregation: hierarchical
 --- max_rounds: 2
 --- early_exit_confidence: 0.84
```

Seed 44: Hybrid Architecture.

```
Topology: generalist <-> coder, architect---> coder (cyclic)
Agents: 3
Edges: 3
Final Fitness: 100.0% (41/41)
Total Tokens: 182,970
Configuration:
 --- message_format: minimal
 --- max_message_length: 228
 --- aggregation: hierarchical
 --- max_rounds: 3
 --- early_exit_confidence: 0.74
```

A.2 MEDIUM Regime (5K tokens)

Seed 42: Linear Pipeline.

```
Topology: planner---> coder---> reviewer (linear)
Agents: 3
Edges: 2
Final Fitness: 100.0% (41/41)
Total Tokens: 134,520
Configuration:
 --- message_format: structured
 --- max_message_length: 195
 --- aggregation: sequential
 --- max_rounds: 2
 --- early_exit_confidence: 0.89
Note: Best fitness 0% for generations 0-5, then 100% from generation 6.
      Evolution discovered budget-compliant architecture through mutation.
```

Seed 43: Test-First Pipeline.

```
Topology: tester---> reviewer---> planner---> reviewer (cyclic)
Agents: 3
Edges: 3
Final Fitness: 100.0% (41/41)
Total Tokens: 96,900
Configuration:
 --- message_format: freeform
 --- max_message_length: 154
 --- aggregation: hierarchical
 --- max_rounds: 2
 --- early_exit_confidence: 0.92
Note: Same topology as TIGHT seed 43, suggesting this is a robust optima.
```

A.3 LOOSE Regime (10K tokens)

Seed 42: Complex Voting.

```
Topology: architect---> [generalist, coder], generalist---> coder
Agents: 4 (planner, generalist, architect, coder)
Edges: 3
Final Fitness: 95.1% (39/41)
Total Tokens: 301,519
```

```
Configuration:
 --- message_format: structured
 --- max_message_length: 193
 --- aggregation: voting
 --- max_rounds: 2
 --- early_exit_confidence: 0.88
Note: 4-agent architecture enabled by larger budget.
      Voting aggregation unique to this experiment.
```

Seed 44: Hierarchical 4-Agent.

```
Topology: architect---> planner, coder---> planner, tester---> coder
Agents: 4 (planner, architect, coder, tester)
Edges: 4
Final Fitness: 100.0% (41/41)
Total Tokens: 482,307
Configuration:
 --- message_format: structured
 --- max_message_length: 219
 --- aggregation: hierarchical
 --- max_rounds: 3
 --- early_exit_confidence: 0.80
Note: Most complex architecture discovered. Avg agents evolved
      from 2.4 to 4.3 over 12 generations, showing clear
      complexity growth under permissive budget. Runtime: 5.2 hrs.
```

A.4 UNCONSTRAINED Regime (50K tokens)

Seed 42: Minimal Pair.

```
Topology: planner---> debugger
Agents: 2 (planner, debugger)
Edges: 1
Final Fitness: 100.0% (41/41)
Total Tokens: 206,056
Configuration:
 --- message_format: structured (default)
 --- max_rounds: 3 (default)
Note: Simplest architecture discovered across all experiments.
      With no budget pressure, evolution converged to minimal
      viable solution.
```

Seed 43: Linear Pipeline.

```
Topology: planner---> coder---> reviewer
Agents: 3 (planner, coder, reviewer)
Edges: 2
Final Fitness: 100.0% (41/41)
Total Tokens: 295,922
Configuration:
 --- message_format: structured (default)
 --- max_rounds: 3 (default)
Note: Classic software engineering pipeline. Similar to
      constrained experiments but without cyclic edges.
```

Seed 44: Cyclic Feedback.

```
Topology: debugger <-> coder, architect---> coder
Agents: 3 (debugger, coder, architect)
Edges: 3
```

```
Final Fitness: 100.0% (41/41)
Total Tokens: 440,784
Configuration:
 --- message_format: structured (default)
 --- max_rounds: 3 (default)
Note: Only UNCONSTRAINED seed to maintain cyclic topology.
      Bidirectional debugger<->coder edge enables iterative
      refinement. Highest token usage in regime.
```

B Genome Representation Details

Table 8 provides the complete genome specification.

Table 8: Genome specification for multi-agent architectures.

Component	Type	Description
`agents`	List[AgentGene]	List of agent specifications
`topology`	Dict[str, List[str]]	Adjacency list defining message flow
`message_format`	Enum	FREEFORM, STRUCTURED, or MINIMAL
`aggregation`	Enum	SEQUENTIAL, VOTING, or HIERARCHICAL
`entry_agent`	str	ID of agent receiving initial task
`output_agent`	str	ID of agent producing final output
AgentGene		
`role`	Enum	PLANNER, CODER, REVIEWER, DEBUGGER, etc.
`system_prompt`	str	Agent's system prompt
`max_tokens`	int	Per-response token limit
`temperature`	float	Sampling temperature

C Mutation Operators

Table 9 describes all mutation operators used during evolution.

Table 9: Mutation operators and their effects.

Operator	Description
`add_agent`	Insert new agent with random role; connect to random existing agent
`remove_agent`	Remove random non-entry/output agent; reconnect topology
`mutate_role`	Change agent's role to random different role
`mutate_prompt`	Perturb system prompt (add/remove tokens, paraphrase)
`rewire_edge`	Redirect random edge to different target agent
`add_edge`	Add communication channel between disconnected agents
`remove_edge`	Remove random edge (maintaining connectivity)
`change_aggregation`	Switch aggregation strategy
`change_message_format`	Switch message compression level

D Hyperparameter Sensitivity

Table 10 shows the hyperparameters used in our experiments.

Table 10: Evolution hyperparameters used in experiments.

Parameter	Value	Sensitivity
Population size	10	Low
Generations	12	Medium
Tournament size	2	Low
Elite count	2	Low
Mutation rate	0.4	Medium
Crossover rate	0.5	Medium
Sample fraction (evolution)	0.12	High
Sample fraction (final eval)	0.25	Medium
Seeds per regime	3	High

E Detailed Per-Task Results

Table 11 provides per-category breakdown on HumanEval.

Table 11: HumanEval results by problem category. All evolved architectures achieved 100% accuracy across all categories within budget constraints.

Category	Tight	Medium	Loose	Unconstrained
String manipulation	100%	100%	100%	100%
Math/arithmetic	100%	100%	100%	100%
List operations	100%	100%	100%	100%
Conditionals	100%	100%	100%	100%
Recursion	100%	100%	100%	100%

F Computational Cost

Table 12 reports the computational resources required for experiments.

Table 12: Computational cost breakdown (actual measurements from HumanEval experiments).

Regime	Experiments	Generations	API Calls	Tokens	Est. Cost
TIGHT (2K)	3 seeds	12 each	6,840	855K	$0.19
MEDIUM (5K)	3 seeds	12 each	6,840	415K	$0.09
LOOSE (10K)	3 seeds	12 each	6,840	920K	$0.21
UNCONSTRAINED (50K)	3 seeds	12 each	6,840	943K	$0.21
Total	12 experiments	144 gens	27,360	3.09M	$0.70

Note: Costs estimated using GPT-4o-mini pricing ($0.15/1M input, $0.60/1M output). Experiments completed December 2025.

F.1 Per-Experiment Breakdown

Table 13 provides token usage for each individual experiment.

Efficiency Observation. MEDIUM budget experiments use the fewest tokens per generation (8–15K) while achieving the highest performance (100%). This supports our finding that 5K token budgets represent the "Goldilocks zone" for HumanEval tasks.

G Reproducibility Checklist

- ✓ Code available at: `https://github.com/noah-ing/EMAP`

Table 13: Detailed per-experiment token usage and runtime.

Experiment	Tokens	API Calls	Runtime	Tokens/Gen
TIGHT seed 42	386,687	2,280	3.0 hrs	32.2K
TIGHT seed 43	285,855	2,280	2.4 hrs	23.8K
TIGHT seed 44	182,970	2,280	3.0 hrs	15.2K
MEDIUM seed 42	134,520	2,280	2.2 hrs	11.2K
MEDIUM seed 43	96,900	2,280	1.7 hrs	8.1K
MEDIUM seed 44	183,000	2,280	2.5 hrs	15.2K
LOOSE seed 42	301,519	2,280	3.7 hrs	25.1K
LOOSE seed 43	135,633	760	1.5 hrs	11.3K
LOOSE seed 44	482,307	2,280	5.2 hrs	40.2K
UNCONSTRAINED seed 42	206,000	2,280	2.8 hrs	17.2K
UNCONSTRAINED seed 43	296,000	2,280	3.5 hrs	24.7K
UNCONSTRAINED seed 44	441,000	2,280	4.8 hrs	36.7K

✓ Random seeds reported for all experiments (42, 43, 44)

✓ Hyperparameters fully specified in Table 10

✓ Hardware/API specifications documented (GPT-4o-mini via OpenAI API)

✓ Standard deviations reported for aggregated metrics

- *Note*: With $n = 3$ seeds per condition, formal hypothesis testing has limited statistical power. We report descriptive statistics and per-seed breakdowns to enable assessment of result stability.