

Mycelial Council Network: Temperature Dominates Routing in Multi-Agent LLM Code Synthesis — A Longitudinal Study with Category-Level Analysis

MCN Research Initiative

Anonymous Submission — February 2026

Abstract—We empirically study whether a learned router can improve upon the best single-agent baseline when a council of LLM agents differs only in system prompt or sampling temperature. Over ten experimental runs and three controlled ablations using Qwen2.5-Coder-7B-Instruct-AWQ on a 16-task Python benchmark (100 tasks per run), we find that temperature selection dominates routing strategy. A single agent at $T=0.3$ achieves 91%, while MCN-LinUCB (three homogeneous $T=0.3$ tribes) scores 86% (−5 pp) and MCN-GNN (heterogeneous $T=0.1/0.5/0.9$) scores 88% — identical to a $T=0.9$ single agent. A subsequent Phase 1B stratified evaluation scales to 400 live tasks across 8 algorithmic categories (50 per category), confirming the absence of routing specialisation (chi-squared $p=0.396$) while revealing stark category-level heterogeneity: string tasks achieve 100%, graph tasks 0% (a hard model-capability limit), and the LinUCB oracle gap is 12.8 pp — split equally between exploration cost, tie-breaking noise, and exploitation error. Bandit convergence was not reached at 400 tasks, requiring an estimated 2,000+ tasks for reliable per-category specialisation. We report a reproducible null-result baseline for multi-agent routing research with same-model tribes.

Index Terms—multi-agent systems, LLM code synthesis, contextual bandits, graph neural networks, temperature scaling, routing collapse, ablation study, category-level analysis, oracle gap decomposition

I. INTRODUCTION

Automated code synthesis with large language models (LLMs) has advanced rapidly, yet most deployments use a single model with fixed inference parameters. The Mycelial Council Network (MCN) hypothesis is that a council of agents with distinct styles, routed by a learned policy, can outperform any single agent — analogously to how software teams benefit from diverse specialists.

This paper reports ten experimental runs and three controlled single-agent ablations designed to test this hypothesis. Experiments proceed through three phases:

Phase 1 (Runs 1–6): Infrastructure validation and GNN vs. LinUCB comparison on a 12-task benchmark. After correcting three infrastructure defects, the GNN router reached 98% and LinUCB reached 97%, but LinUCB collapsed to a single tribe (100/0/0 routing).

Phase 2 (Runs 7–8): Task-set expansion to 16 types, revealing a 14 pp pass-rate drop from the 12-task ceiling and identifying three new reference-solution bugs.

Phase 3 (Runs 9–10 + ablations): Controlled five-condition comparison — three single-agent ablations ($T=0.1$, $T=0.3$, $T=0.9$) and two MCN variants (LinUCB homogeneous, GNN heterogeneous).

The overall conclusion is that temperature selection matters more than routing strategy under these conditions, and routing with same-model tribes imposes a net exploration cost with no compensating specialisation benefit.

We make the following claims, strictly grounded in observed results: (1) A single agent at $T=0.3$ achieves 91%,

outperforming all MCN variants. (2) MCN-LinUCB (homogeneous $T=0.3$) scores 86% — 5 pp below single-agent, with no statistically significant routing specialisation (chi-squared $p=0.74$). (3) MCN-GNN (heterogeneous $T=0.1/0.5/0.9$) scores 88% = $T=0.9$ single agent exactly. T_2 achieves 95% in isolation, but the router incurs a −7 pp cost relative to an oracle that always selects T_2 . (4) Test-infrastructure correctness dominated measured outcomes in Phase 1 more than routing strategy did.

II. SYSTEM DESCRIPTION

MCN comprises five components operating as Ray remote actors inside a Docker Compose stack: vLLM inference server, Redis state store, ChromaDB vector store, MLflow tracking, and an mcn-runner container.

A. Tribes

Three tribe actors (T_0 , T_1 , T_2) share the same base model (Qwen2.5-Coder-7B-Instruct-AWQ) but receive distinct system prompts: "Reliable Coder," "Fast Coder," and "Creative Coder." No weight updates are performed. Phase 3 introduces heterogeneous temperatures: $T_0=0.1$ (deterministic), $T_1=0.5$ (balanced), $T_2=0.9$ (high-variance). This creates genuine output diversity: T_0 generates near-identical code across attempts; T_2 explores a broader algorithmic space at the cost of higher failure variance.

B. Router

Two routing implementations are compared.

LinUCB: Contextual bandit with 18-dim context. $\text{arm}^* = \text{argmax}_i [\mathbf{x}^T \theta_i + \alpha * \sqrt{\mathbf{x}^T \mathbf{A}_i^{-1} \mathbf{x}}]$. Epsilon-greedy warm-up (epsilon 0.5 \rightarrow 0.15, decay=0.995) prevents cold-start collapse. $\alpha=2.5$. An epsilon-restoration bug in early runs caused saved epsilon (near floor) to override the configured warm-up on restart; fixed in Run 9.

GNN Router: Lightweight 2-layer MLP approximating GraphSAGE on a bipartite task-tribe graph. Input: $[\text{task_context}(18), \text{tribe_embedding}(18)] = 36\text{-dim}$ concatenation. Three linear layers ($36 \rightarrow 32 \rightarrow 16 \rightarrow 1$) produce per-tribe scores. Online Adam optimisation ($\text{lr}=0.01$), 64-entry experience replay, mini-batches of 8. $\sim 2,000$ parameters. CPU-only inference.

C. Context Vector

Each task maps to an 18-dim vector: task-type one-hot (dims 0–2), exception one-hot (dims 3–12), and z-scored execution metrics — runtime, tests_passed/failed, test_count, failure_density (dims 13–17). Computed from the previous failure on the same task and used as the bandit context for the next routing decision.

D. Overseer and Test Generation

The OverseerActor generates three test classes per task without seeing generated code: (A) Hypothesis property-based fuzzing — integers bounded to $[-10k, 10k]$ (unbounded was a critical Phase 1 defect); (B) six adversarial static boundary tests including large inputs (10k elements) and large integers (10,000, not `sys.maxsize` — another Phase 1 defect); (C) mutation testing for test-suite sensitivity.

E. Patch Registry

Verified solutions are stored as context-vector embeddings. Top-K most similar prior solutions are retrieved and injected as few-shot hints into the selected tribe's prompt. When a reference solution is already present, the patch hint is suppressed to prevent conflicting code suggestions.

III. EXPERIMENTAL SETUP

Hardware: Single GPU workstation (Docker Compose). Model: Qwen/Qwen2.5-Coder-7B-Instruct-AWQ (4-bit AWQ). Inference: vLLM, temperature per condition, `max_tokens=2048`, `sandbox_timeout 10 s`.

Phase 1 task set (Runs 1–6): 12 Python functions — `sort_list`, `deduplicate`, `flatten`, `partition`, `reverse_string`, `is_palindrome`, `word_count`, `fibonacci`, `is_prime`, `gcd`, `invert_dict`, `running_sum`. $\sim 8\text{--}9$ attempts per type per 100-task run.

Phase 2–3 task set (Runs 7–10, ablations): Extended to 16 types by adding `has_cycle` (graph cycle via Kahn's topological sort), `permutations` (combinatorial enumeration), `search_insert` (binary search insertion position), and `unique_paths` (DP grid path counting). These harder tasks reduce the achievable ceiling and stress-test model capability

limits.

Five conditions in Phase 3: (A) $T=0.1$ ablation: Single tribe, `T_0` prompt, temperature=0.1. (B) $T=0.3$ ablation: Single tribe, `T_0` prompt, temperature=0.3. (C) $T=0.9$ ablation: Single tribe, `T_2` prompt, temperature=0.9. (D) MCN-LinUCB (Run 9): Three homogeneous $T=0.3$ tribes, LinUCB router. (E) MCN-GNN (Run 10): Three heterogeneous tribes ($T_0=0.1$, $T_1=0.5$, $T_2=0.9$), GNN router.

Phase 1B (stratified live evaluation): Following Phase 3, a separate 400-task experiment evaluates MCN-LinUCB (homogeneous $T=0.3$, $\alpha=2.5$) on a stratified sample of 8 algorithmic categories — string, data_structures, math, dynamic_programming, parsing, iterative, recursive, graph — with 50 tasks per category. All tasks executed via live vLLM inference (Qwen2.5-Coder-7B-Instruct-AWQ). Stratified sampling eliminates task-frequency confounds present in Phase 3's uniform task set, enabling direct category-level pass-rate comparison and oracle gap decomposition.

IV. INFRASTRUCTURE DEFECTS

Six defects across the three phases are documented here because they illustrate a systematic challenge in automated LLM evaluation: test infrastructure errors produce misleading metrics that corrupt router training signals. The pass-rate improvement from 83% (Run 1) to 98% (Run 4) was driven primarily by fixing these defects, not by router learning.

Defect 1 — `sys.maxsize` in adversarial test B6 (Run 1): `fibonacci(sys.maxsize)` hangs indefinitely. Corrected to `fibonacci(10,000)` with a 5-second limit.

Defect 2 — Unbounded `st.integers()` in Hypothesis (Runs 1–3): The strategy generator produced `st.integers()` (unbounded) for integer parameters. Hypothesis probes `sys.maxsize`; even a correct fibonacci loops forever, producing `tests_passed=0`, `tests_failed=0` false negatives and corrupting router rewards for three runs. Corrected to `st.integers(min_value=-10000, max_value=10000)`.

Defect 3 — `PATCH_MIN_ATTEMPTS=2` (Run 1): Patch registration required two attempts, leaving the few-shot hint mechanism entirely inactive (0 patches stored). Corrected to `PATCH_MIN_ATTEMPTS=1`.

Defect 4 — `permutations` reference returns tuples (Phase 2): `itertools.permutations` yields tuples; tests expect `list[list[int]]`. Corrected: `[list(p) for p in sorted(set(itertools.permutations(xs)))]`.

Defect 5 — `unique_paths` crash on edge cases (Phase 2): `math.comb(m+n-2, m-1)` raises `ValueError` when $m=0$ or $n=0$. Corrected with guard: if $m \leq 0$ or $n \leq 0$: return 0.

Defect 6 — Epsilon restoration overwrites warm-up (Phase 2): `LinUCB.from_state_dict()` restored saved epsilon (~ 0.05 , floor), silently overriding the current `.env` warm-up schedule. On restart, the router skipped its warm-up phase,

causing 97/1/2 degenerate routing. Corrected by overriding epsilon, epsilon_min, and epsilon_decay from MCNConfig after from_state_dict().

V. RESULTS

Table I presents the five-condition Phase 3 comparison. Table II gives the complete ten-run history. The primary finding is that the single-agent T=0.3 ablation outperforms all MCN variants.

TABLE I FIVE-CONDITION PHASE 3 COMPARISON (100 tasks, 16-task benchmark)

Condition	T	Pass	Fib	Search Ins.	Routing
(A) T=0.1 solo	0.1	86%	40%	25%	—
(B) T=0.3 solo	0.3	91%	80%	100%	—
(C) T=0.9 solo	0.9	88%	40%	n/m	—
(D) MCN-LinUCB	0.3×3	86%	40%	n/m	71/13/16
(E) MCN-GNN	0.1/.5/.9	88%	60%	n/m	66/13/21

n/m = not measured for that condition. T=0.9 unique_paths=100%. Best result (B) highlighted.

TABLE II COMPLETE RUN HISTORY (RUNS 1–10)

Run	Router	Tasks	Pass %	Routing	Notes
1	GNN	12	83%	10/69/21	Infra bugs 1–3
2	GNN	12	90%	33/48/19	Partial fix
3	GNN	12	91%	33/48/19	Partial fix
4	GNN	12	98%	39/13/48	All bugs fixed
5	GNN*	12	98%	10/8/82	Inherited state
6	LinUCB	12	97%	100/0/0	Cold-start collapse
7-8	LinUCB	16	84.5%	mixed	Exploratory; ϵ -bug
9	LinUCB	16	86%	71/13/16	Post-fix; -5pp vs solo
10	GNN	16	88%	66/13/21	Hetero T=0.1/.5/.9

* Run 5 inherits GNN state from Run 4 (no --fresh). Runs 7–8 combined exploratory dataset.

A. Temperature Ablation — The Dominant Variable

Among the three single-agent ablations, T=0.3 achieves the highest pass rate (91%), outperforming both T=0.9 (88%) and T=0.1 (86%). This 5 pp spread across temperatures exceeds the improvement attributable to any routing strategy tested, establishing temperature as the dominant performance variable for Qwen2.5-Coder-7B on this task set.

T=0.1 failure mode: search_insert collapses to 25%. At very low temperature, the model locks into an incorrect algorithm (linear scan instead of binary search) and fails to self-correct across attempts.

T=0.9 failure mode: fibonacci and permutations show higher failure rates than T=0.3, despite T=0.9 succeeding more on unique_paths. High variance occasionally finds solutions for tricky DP problems but increases failures on tasks with a uniquely correct iterative pattern.

B. MCN-LinUCB vs. Single Agent T=0.3 (Run 9)

MCN-LinUCB with three homogeneous T=0.3 tribes scored 86% — 5 pp below the single-agent T=0.3 ablation (91%). Routing distribution: 71/13/16 (Gini=0.387). Chi-squared test for routing independence from task type: p=0.737. No statistically significant specialisation was detected.

The 5 pp loss arises from exploration overhead. Epsilon-greedy routing sends approximately 29% of tasks to non-dominant tribes during warm-up. Because all tribes use the same model at the same temperature, these explorations produce no useful diversity signal: the model generates statistically identical code regardless of which tribe receives the task. Variance introduced by routing is a pure cost with no compensating specialisation benefit.

Task-level losses (MCN vs. single-agent): fibonacci -40 pp, search_insert -25 pp, unique_paths -50 pp. These represent cases where routing variance accumulated failures during unlucky epsilon windows that a single-agent run would have resolved deterministically.

C. MCN-GNN (Heterogeneous) vs. Single Agents (Run 10)

MCN-GNN with heterogeneous tribes (T₀=0.1, T₁=0.5, T₂=0.9) scored 88% — exactly matching the T=0.9 single-agent ablation and 3 pp below T=0.3. Routing: 66/13/21 (Gini=0.353). Chi-squared p=0.762. No significant specialisation detected.

Critically, T₂ (T=0.9) achieved 95% pass rate in isolation within Run 10 — the highest single-tribe result in the entire study. Yet the full MCN system scored only 88%, quantifying a -7 pp routing cost relative to an oracle that always selects T₂. The GNN's inability to reliably route all tasks to T₂ stems from two failure modes: (i) search_insert is handled worse by T₂ (0%) than by T₀, so the oracle would need to route that task type away from T₂; (ii) unique_paths is handled poorly by T₁ (0%). The GNN must implicitly learn a per-task-type routing policy from ~6 attempts per type — insufficient signal.

Concretely, on fibonacci: T₀ (T=0.1) fails repeatedly with deterministic naive recursion; T₂ (T=0.9) succeeds through high-variance exploration of the search space. MCN-GNN fibonacci improved from 40% (Run 9, homogeneous) to 60% (Run 10), showing that T₂'s variance does help — but the GNN must also avoid routing fibonacci to T₀, which it only partially achieves.

TABLE III MCN VARIANTS vs. BEST SINGLE AGENT

Metric	T=0.3 Solo	MCN-LinU CB	MCN-GNN
Pass rate	91%	86% (−5pp)	88% (−3pp)
fibonacci	80%	40% (−40pp)	60% (−20pp)
Routing dist.	—	71/13/16	66/13/21
Chi-sq. p	—	0.737	0.762
Specialisation?	—	No	No
T ₂ isolation	88% (T=0.9)	n/a	95% (within-run)
Oracle gap	—	n/a	−7pp vs T ₂ solo
Verdict	BEST	−5pp vs best	=T=0.9 solo

T₂ isolation = T₂ tribe pass rate computed within Run 10, not a standalone ablation. Oracle gap = 95%−88% = 7 pp routing cost.

D. Model Capability Limits

Two of the four new tasks reveal hard limits of the 7B model, independent of routing.

has_cycle achieves 0% across all conditions. A Kahn's topological sort reference solution was provided but the model consistently generates syntactically valid yet algorithmically incorrect implementations (missing in-degree initialisation, incorrect BFS termination). The overseer's adversarial tests — directed cycles, self-loops, disconnected components — catch all errors. This is a model capability limit, not a routing failure.

permutations achieves 0% before the type-annotation fix (Defect 4). The critical insight: itertools.permutations returns tuples, but tests expect list[list[int]]. With the corrected reference solution, small models reliably copy the wrapping pattern. This is a reference-solution quality issue, not a routing issue.

E. Phase 1B: Category-Level Analysis (400 Tasks)

Phase 1B evaluates MCN-LinUCB on 400 stratified live tasks. Overall pass rate: 245/400 = 61.2%, significantly below Phase 3's 86% on the 16-task benchmark. This drop reflects the harder category mix: graph tasks (has_cycle, topological_sort, count_components, num_islands, is_bipartite) achieve 0%, and recursive/iterative categories average 42–43%.

Category heterogeneity is extreme (Table IV). String tasks — palindromes, anagrams, compression, case conversion — achieve 100%, consistent with their dominance in pre-training corpora. Graph algorithms achieve 0% in all conditions, confirming the 7B model capability limit established in Phase 3. The 100 pp spread across categories dwarfs the 5 pp MCN-vs.-single-agent gap from Phase 3, establishing category as a far stronger predictor of outcome than routing strategy.

Routing specialisation: chi-squared test for routing independence from task category: $p=0.396$. No statistically significant specialisation was detected. Routing distribution shows dramatic T0→T1 drift across the 400 tasks, consistent with a bandit that has not converged. Analysis of UCB upper confidence bounds shows that the bandit requires approximately 2,000+ tasks for reliable per-category specialisation at this task diversity.

Oracle gap decomposition (Table V): The total oracle gap of 12.8 pp — MCN 61.2% vs. oracle 74.0% — decomposes into three roughly equal components: exploration cost 4.1 pp (ϵ -greedy routing to non-optimal arms during warm-up), tie-breaking noise 4.7 pp (near-equal UCB scores trigger random arm selection), and exploitation error 4.0 pp (insufficient per-category signal causes wrong arm selection even during exploitation). No single failure mode dominates, indicating the bandit is operating in a data-limited regime throughout.

TABLE IV PHASE 1B CATEGORY-LEVEL RESULTS (400 tasks, 8×50 stratified)

Category	Pass%	vs. Overall	Capability
string	100%	+38.8 pp	Full
data_structures	84%	+22.8 pp	High
math	86%	+24.8 pp	High
dynamic_prog.	76%	+14.8 pp	Moderate
parsing	66%	+4.8 pp	Moderate
iterative	43%	−18.2 pp	Low
recursive	42%	−19.2 pp	Low
graph	0%	−61.2 pp	None
Overall	61.2%	—	—

vs. Overall = category pass rate minus overall mean (61.2%). Capability = qualitative model-performance tier for Qwen2.5-Coder-7B. Graph 0% is a hard capability limit, not a routing failure.

TABLE V ORACLE GAP DECOMPOSITION (Phase 1B, 400 tasks)

Component	Gap (pp)	Mechanism
Exploration cost (Q1)	4.1	ϵ -greedy routes ~15% of tasks to non-optimal arms
Tie-breaking noise (Q2+Q3)	4.7	Near-equal UCB scores → random arm selection
Exploitation error (Q4)	4.0	Insufficient per-category signal → wrong arm
Total oracle gap	12.8	MCN 61.2% vs. oracle 74.0%

Oracle = per-task best-tribe selection with perfect knowledge. Q1–Q4 computed via counterfactual routing simulation on recorded decisions. All three components are roughly equal; no single source dominates.

VI. DISCUSSION

The central question — can a learned router improve upon the best single agent when tribes share the same base model? — receives a consistent negative answer across all three routing experiments and the Phase 1B stratified evaluation.

A. Why Routing Fails With Same-Model Tribes

The multi-armed bandit framing assumes that different arms have genuinely different expected rewards for different contexts. When tribes are functionally similar (same model, same temperature), all arms converge to the same expected reward, and exploration cost is never recovered.

Heterogeneous temperatures (0.1/0.5/0.9) create genuine output diversity but do not solve the routing problem, because no single temperature is uniformly best across all task types. $T=0.3$ beats $T=0.9$ on fibonacci and search_insert; $T=0.9$ beats $T=0.3$ on unique_paths. An oracle router would outperform any single temperature — but the GNN achieves $p=0.76$ routing independence from task type at 100 tasks, far from oracle behaviour. The -7 pp gap between T_2 -in-isolation (95%) and MCN-GNN (88%) directly measures the cost of imperfect routing even when a clearly superior tribe exists.

B. What Would It Take for Routing to Add Value?

For routing to add aggregate value, the inter-tribe performance variance on each task type must exceed the exploration cost. Formally, the expected routing gain equals $E[\max_i R_i(\text{task})] - E[R_{\text{routing}}(\text{task})]$, where R_{routing} is the reward received under the routing policy. When $\sigma^2(\text{performance} | \text{task})$ is small — as here, where tribes are functionally similar — this gain is negligible and the exploration cost dominates.

The boundary conditions for positive routing value are: (1) True model diversity: Routing between qualitatively different models (e.g., 7B code-specialist vs. 13B reasoning-specialist) would create large inter-tribe performance variance on specific task classes. (2) Sufficient data: At 6–8 attempts per task type, the GNN lacks the signal to learn per-type preferences reliably. Hundreds of attempts per type would provide the learning signal needed for genuine specialisation. (3) Adaptive temperature: Treating temperature as a continuous router output rather than a fixed per-tribe hyperparameter could achieve better task-type coverage than discrete temperature choices.

The negative result here is therefore a boundary condition, not a universal truth: routing is not beneficial with same-model, same-architecture tribes on a 100-task dataset, but may be beneficial with genuinely diverse tribes at larger scale.

C. Category-Level Heterogeneity and Bandit Convergence

Phase 1B establishes that algorithmic category is the dominant predictor of task outcome, with a 100 pp spread from string (100%) to graph (0%). This heterogeneity creates an apparent opportunity for category-adaptive routing: a converged bandit that identifies graph tasks and halts exploration on those categories would reduce the 4.1 pp exploration cost.

However, the critical limiting factor is tribe homogeneity. At $T=0.3$, all three tribes generate statistically identical code for any given task. The bandit cannot learn a meaningful per-category preference because the arms do not differ. The 12.8 pp oracle gap is therefore irreducible under the current tribe configuration regardless of how many tasks are collected: the oracle for homogeneous tribes is the same as any single tribe, making the gap an artefact of the exploration protocol rather than a recoverable routing signal.

The 2,000-task convergence estimate applies to the case where tribes are genuinely diverse. With qualitatively different tribes (different base models, architectures, or fine-tuning), the per-category reward signal would differ across arms, and the bandit could learn a meaningful routing policy. Under those conditions, the category-level heterogeneity documented in Phase 1B — specifically the mid-tier cluster (parsing 66%, iterative 43%, recursive 42%, dynamic_programming 76%) — represents the most actionable routing opportunity: categories where model capability is partial and tribe-level variance could in principle be exploited.

VII. CONCLUSIONS

We have conducted ten experimental runs, three controlled ablations, and one Phase 1B stratified live evaluation (400 tasks, 8 categories) evaluating the Mycelial Council Network on Python code synthesis benchmarks. Primary conclusions:

- (1) Temperature dominates routing. $T=0.3$ achieves 91% (single agent); all MCN variants score 86–88%. The optimal multi-agent configuration does not exceed the optimal single-agent temperature.
- (2) Same-model routing imposes a net cost. MCN-LinUCB (homogeneous $T=0.3$) scores -5 pp vs. single-agent, with $p=0.74$ routing independence.
- (3) Heterogeneous temperatures do not enable specialisation. MCN-GNN matches $T=0.9$ single agent (both 88%). The -7 pp gap between T_2 -in-isolation (95%) and MCN-GNN (88%) quantifies the routing cost when a clearly superior tribe exists but cannot be selected reliably.
- (4) Infrastructure correctness dominated early measured outcomes. Six defects across the three phases caused systematic performance distortions unrelated to routing quality.
- (5) Model capability limits, not routing, explain the hardest failures. has_cycle achieves 0% in all conditions; this is a 7B model limit.
- (6) Category-level heterogeneity dwarfs routing effects. Phase 1B reveals a 100 pp spread from string (100%) to graph (0%), with a 12.8 pp oracle gap split equally among exploration cost (4.1 pp), tie-breaking noise (4.7 pp), and exploitation error (4.0 pp). No component dominates; the bandit is data-limited throughout. Reliable per-category specialisation requires approximately 2,000+ tasks — an order of magnitude more than tested.

Future directions: (A) model diversity — route between qualitatively different architectures to create genuine inter-tribe variance; (B) scale — $n=2,000+$ tasks for bandit convergence with diverse tribes; (C) adaptive temperature as a continuous router output rather than a fixed per-tribe hyperparameter.

Reproducibility: all code, Docker configuration, and MLflow artifacts available in the MCN repository. Raw data: `mcn-results:/results/runs.jsonl`.