MAWGE: A Parametric and Synthetic Multi-Agentic Workflow Generator Library

Goshgar C. Ismayilov[1,*]

[1]*Independent Researcher, Istanbul, Turkey*

Abstract

Multi-agent systems powered by large-language models have been increasingly employed in complex reasoning tasks, requiring coordination of agents with diverse capabilities. Despite growing interest in multi-agent systems, neither large-scale publicly-available datasets for agentic workflows nor systematic multi-agentic workflow generator currently exists in the literature. Available workflow generators also fail to integrate the characteristics of multi-agent systems (e.g., reasoning patterns, inference overhead). To fill this literature gap, we introduce MAWGE (Multi-Agentic Workflow Generator Engine), a library for generating parametric and synthetic multi-agentic workflows with distinct topologies, compositions and execution characteristics. MAWGE implements a four-stage pipeline: (i) *generator* builds workflows by using six operator topologies (*single-agent*, *chain-of-thought*, *self-consistency*, *tree-of-thought*, *debate*, *self-refine*) and the first-order Markovian transition, (ii) *serializer* transforms potentially-cyclic workflows into acyclic workflows by instantiating agents from a collection of 75 LLM models with realistic throughput, latency and pricing data, (iii) *simulator* executes workflows through event-driven scheduling and (iv) *visualizer* renders workflows as hierarchical graphs. We believe that MAWGE provides a standardized toolkit for generating reproducible synthetic benchmarks on which researchers can evaluate their techniques.

Keywords

multi-agent systems, agentic workflows, large-language models, workflow generation, workflow simulation

1. Introduction

Large language models have advanced artificial intelligence to a great extent by enabling autonomous agents to perform complex tasks [2]. Recent advancements demonstrate that merging individual agents into agentic workflows can alleviate the cognitive and functional limitations of single agents [3]. Contemporary multi-agent systems implement diverse reasoning topologies: sequential decomposition (*chain-of-thought*) [4], parallel exploration with aggregation (*self-consistency*) [5], tree-structured search (*tree-of-thought*) [6], adversarial discussion (*debate*) [3] and iterative refinement loops (*self-refine*) [7]. Existing frameworks including AutoGen [8] and MetaGPT [9] enable multi-agent collaboration by conversation programming and role assignment. In addition, certain frameworks including AFlow [10] and EvoFlow [11] automate agentic workflow optimization.

Advancing artificial intelligence further requires a robust and reproducible experimental setup that currently remains underdeveloped. The community currently lacks large-scale datasets of heterogeneous and real-world agentic workflows with varying complexity and performance profiles. For instance, the work [12] clearly states the lack of publicly-available datasets for agentic workflows. Migration of existing scientific workflow generators to agentic workflows [13, 14, 15, 16] is limited since they lack the domain-specific characteristics of LLM-based agents such as inference latency profiles. In addition, integration of agentic workflow optimization tools [8, 9, 10, 11] does not resolve the issue since they do not provide parametric control. Therefore, we introduce MAWGE in this work to resolve the issue stated in [12] by proposing a dynamic solution where researchers can generate their own diverse and reproducible set of datasets.

NL4AI 2026: Ninth Workshop on Natural Language for Artificial Intelligence, October 6-9th, 2026, Perugia, Italy [1]

*Corresponding author.

goshgar.ismayilov@boun.edu.tr (G. C. Ismayilov)

0009-0007-9307-3637 (G. C. Ismayilov)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-05

CEUR-WS.org/Vol-4315/paper16.pdf

1. MAWGE **Library.** We introduce MAWGE (Multi-Agentic Workflow Generator Engine), a library for generating synthetic agentic workflows in parametric and reproducible setting. MAWGE supports workflow design with six fundamental agentic patterns, including *single-agent (SA)*, *chain-of-thought (CoT)*, *self-consistency (SC)*, *tree-of-thought (ToT)*, *debate (Db)* and *self-refine (SR)*, enabling the construction of heterogeneous and hybrid agentic workflows. Dynamic transitions between these patterns are modeled through the first-order Markovian transition matrix.

2. **Modular Architecture.** We design a modular pipeline consisting of four sequential modules: (i) *generator*: generates initial cyclic workflows with user specifications and parameters, (ii) *serializer*: serializes cyclic workflows into directed acyclic graphs by instantiating concrete agents from a database of 75 LLM models spanning 7 major vendors, (iii) *simulator*: executes workflows with event-driven simulation by preserving workflow dependencies, (iv) *visualizer*: visualizes workflows with collections of nodes and edges.

3. **Event-Driven Workflow Simulation.** We develop an event-driven realistic simulator that traces agentic workflow execution by tracking seven event types (for agent life-cycle and tool invocations) with queue scheduling. The simulator accounts for both computational costs (tokens per second, time to first token) and I/O costs (external tool latency, inference latency), enabling end-to-end simulation for complex multi-agentic workflows.

4. **Multidimensional Complexity Characterization.** We define a deterministic workflow complexity metric based on six structural measures (agent count, edge count, average degree, maximum depth, width, cycle count) and five functional measures (pattern diversity, total tool count, total latency, total inference time, total cost).

5. **Experimental Analysis.** We conduct systematic empirical evaluation in multiple dimensions: (i) computational efficiency analysis measuring workflow generation time costs across the generator, serializer, simulator and visualizer modules, (ii) sensitivity analysis for the probabilistic pattern transition matrix and (iii) illustrative multi-agentic case study.

2. Related Work

Multi-Agent Systems. In the literature, there exist several multi-agent systems where agents collaborate through workflows to solve complex reasoning tasks. For instance, AutoGen [8] provides conversation programming to streamline the development of LLM-based applications with two stages: (i) defining conversable agents with certain capabilities and (ii) programming the interactions between agents via conversation-centric computation. MetaGPT [9] encodes standardized operating procedures into agentic workflows, especially enabling robust software development by assigning diverse roles such as product manager, architect and engineer to the agents. AFlow [10] introduces a Monte Carlo Tree Search-based agentic workflow generation optimization by iteratively sampling, evaluating and refining candidate workflows. The objective is to maximize task performance while minimizing human intervention. For the identical problem, EvoFlow [11] employs evolutionary algorithms to evolve populations toward high-performing workflows with niching-based solution selection. While these frameworks advance the design and optimization of multi-agent systems, they are computationally complex and do not provide parametric workflow generation capabilities. MAWGE addresses this gap by enabling users to generate agentic workflows with diverse topology, composition and complexity.
Workflow Generation. The task scheduling and high-performance computing communities have developed synthetic workflow generators for benchmarking scheduling and resource allocation algorithms. For instance, Pegasus [13] provides a workflow management system to support scientific workflows including Montage for astronomy, and Epigenomics for bioinformatics. WorkflowSim [14] extends CloudSim to simulate scientific workflows, enabling evaluation of scheduling algorithms on cloud platforms. STG (The Standard Task Graph) [15] set offers manually curated DAG benchmarks representing parallel applications while TGFF (Task Graphs For Free) [16] generates random task

graphs with parametric degree distributions and communication costs. However, these generators lack domain-specific characteristics of LLM-based agents (e.g., not modeling inference latency, cost functions on per-token pricing). In addition, existing frameworks generate structural graphs without considering the semantic foundation of agentic reasoning patterns. MAWGE bridges this gap by integrating model performance profiles, reasoning topologies and Markovian pattern composition. Refer to Table 1 for the comparison of these systems with MAWGE.

Table 1
Comparison of existing workflow systems in the literature with the proposed MAWGE library.

Feature	**TGFF** [16]	**STG** [15]	**WorkflowSim** [14]	**AFlow** [10]	**EvoFlow** [11]	MAWGE
LLM-aware workflow generation	✗	✗	✗	✓	✓	✓
Parametric workflow control	✓	✗	✓	✗	✗	✓
Workflow optimization	✗	✗	✗	✓	✓	✗
Event-driven simulation	✗	✗	✓	✗	✗	✓
Inference latency modeling	✗	✗	✗	✗	✗	✓
Tool/API latency modeling	✗	✗	✗	✗	✗	✓
LLM pricing/cost modeling	✗	✗	✗	✗	✗	✓
Workflow visualization	✗	✗	✓	✗	✗	✓

3. Multi-Agent Systems as Directed Workflows

To enable systematic workflow generation and simulation, we formalize multi-agentic workflows as directed graphs and establish three core abstractions: the *agent model* (Section 3.1), the *tool model* (Section 3.2) and the *workflow model* (Section 3.3).

3.1. Agent Model

Let $\mathcal{A} = \{A_0, A_1, \ldots, A_{|\mathcal{A}|-1}\}$ denote a finite set of agents in a workflow. Each agent $A_i \in \mathcal{A}$ refers to a computational unit that encapsulates a large language model capable of reasoning, inference and interaction with external tools. Let $\mathcal{M} = \{M_0, M_1, \ldots, M_{|\mathcal{M}|-1}\}$ be a finite set of available LLM models where each agent $A_i \in \mathcal{A}$ is associated with a single model $M_j \in \mathcal{M}$. Each agent A_i is characterized by a tuple of attributes $A_i = (\text{id}_i, \theta_i, \rho_i, \gamma_i, \tau_i)$ where:

- $\text{id}_i \in \mathbb{N}$ is a unique identifier for agent A_i within the workflow.
- $\theta_i = (M_i, T_i)$ represents the *model configuration*, where $M_i \in \mathcal{M}$ is the assigned LLM model and $T_i \in \{0.0, 0.1, \ldots, 1.0\}$ is the sampling temperature controlling inference randomness.
- $\rho_i = (\lambda_i, t_i^{\text{ttft}}, t_i^{\text{inf}}, c_i)$ captures the *performance characteristics*, where $\lambda_i \in \mathbb{R}^+$ denotes token generation throughput (tokens/s), $t_i^{\text{ttft}} \in \mathbb{R}^+$ is the time-to-first-token latency (seconds), $t_i^{\text{inf}} \in \mathbb{R}^+$ is the total inference time (milliseconds) and $c_i \in \mathbb{R}^+$ is the inference cost (USD).
- $\gamma_i = (n_i^{\text{in}}, n_i^{\text{out}}, \mathcal{T}_i, \ell_i)$ defines the *data and tool specifications*, where $n_i^{\text{in}} \in \mathbb{N}$ and $n_i^{\text{out}} \in \mathbb{N}$ are the input and output token sizes, $\mathcal{T}_i \subseteq \mathcal{T}$ is the subset of external tools accessible to agent A_i from the global tool set $\mathcal{T}$ and $\ell_i \in \mathbb{R}^+$ is the total tool invocation latency (milliseconds).
- $\tau_i = (\delta_i^{\text{in}}, \delta_i^{\text{out}}, \mathcal{S}_i)$ is the *graph structure*, where $\delta_i^{\text{in}} \in \mathbb{N}$ and $\delta_i^{\text{out}} \in \mathbb{N}$ are the number of incoming and outgoing edges and $\mathcal{S}_i \subseteq \mathcal{A}$ is the set of successor agents that get output from agent A_i.

For each model $M_j \in \mathcal{M}$, we maintain vendor information $v_j \in \mathcal{V}$ (where $\mathcal{V}$ is the set of vendors), model-specific throughput $\bar{\lambda}_j$, base time-to-first-token $\bar{t}_j^{\text{ttft}}$ and output pricing p_j (USD per million tokens). Refer to Table 4 in Appendix B for definitions of complete agent attributes and Table 5 in Appendix C for individual model characteristics and pricing.

3.2. Tool Model

Let $\mathcal{T} = \{t_0, t_1, \ldots, t_{|\mathcal{T}|-1}\}$ denote a finite set of external tools available to agents in the workflow. Each tool $t_k \in \mathcal{T}$ represents an external computational resource or API that extends agent capabilities beyond pure language model inference such as with web search, code execution and database queries. Each tool t_k is characterized by a tuple $t_k = (\text{tid}_k, \ell_k, r_k)$ where:

- $\text{tid}_k \in \mathbb{N}$ is a unique identifier for tool t_k within the global tool set $\mathcal{T}$.
- $\ell_k \in \mathbb{R}^+$ represents the per-invocation latency (milliseconds) required to execute tool t_k, modeling network delays, API response times and computational overhead.
- $r_k \in \mathbb{N}^+$ denotes the number of invocations (requests) that an agent makes to tool t_k during its execution, reflecting the intensity of tool usage.

For an agent A_i with tool subset $\mathcal{T}_i \subseteq \mathcal{T}$, the total tool execution overhead is:

$$\ell_i = \sum_{t_k \in \mathcal{T}_i} \ell_k \cdot r_k \tag{1}$$

where ℓ_i aggregates the cumulative latency across all tool invocations made by agent A_i. Note that this formulation assumes sequential tool execution within an agent. The tool latencies ℓ_k intra-network and inter-network APIs and the number of requests r_k are sampled from a uniform distribution.

3.3. Workflow Model

Let $\mathcal{W} = (\mathcal{A}, \mathcal{E})$ denote a multi-agentic workflow, where $\mathcal{A} = \{A_0, A_1, \ldots, A_{|\mathcal{A}|-1}\}$ is a finite set of agents as defined in Section 3.1 and $\mathcal{E} \subseteq \mathcal{A} \times \mathcal{A}$ is a set of directed edges representing execution dependencies between agents. Each edge $(A_i, A_j) \in \mathcal{E}$ indicates that agent A_j depends on the output of agent A_i where A_i must complete execution before A_j can begin. The workflow $\mathcal{W}$ forms a directed graph that encodes the coordination structure of the multi-agent system.

Execution Semantics. Workflow execution follows a data-driven model as:

1. *Initialization*: All entry agents in the workflow $A_i \in \mathcal{A}_{\text{entry}}$ are scheduled for execution at time $t = 0$.
2. *Dependency satisfaction*: An agent A_j becomes ready for execution when all of its predecessors $\{A_i \mid (A_i, A_j) \in \mathcal{E}\}$ have completed execution.
3. *Agent execution*: When agent A_i executes, it performs LLM inference consuming input of size n_i^{in} tokens and producing output of size n_i^{out} tokens, then invokes external tools $t_k \in \mathcal{T}_i$ as needed. The total execution time is $t_i^{\text{agent}} = t_i^{\text{inf}} + \ell_i$ where t_i^{inf} is inference time and ℓ_i is total tool latency.
4. *Data propagation*: Upon completion, agent A_i propagates its output to all successors $A_j \in \mathcal{S}_i$, satisfying their input dependencies.
5. *Termination*: Workflow execution completes when all exit nodes in the workflow $A_j \in \mathcal{A}_{\text{exit}}$ have finished execution.

The total workflow execution time t_{exec} is determined by the critical path: $t_{\text{exec}} = \max_{A_i \in \mathcal{A}_{\text{exit}}} T(A_i)$, where $T(A_i)$ is the completion time of agent A_i computed recursively as $T(A_i) = \max_{(A_j, A_i) \in \mathcal{E}} T(A_j) + t_i^{\text{agent}}$ with $T(A_i) = 0$ for entry nodes. This computation considers both sequential dependencies and parallel execution for heterogeneous multi-agentic workflows.

4. MAWGE: Multi-Agentic Workflow Generator Library

Based on the formal abstractions in the previous section, we now propose MAWGE by presenting the operator topologies that form the compositional vocabulary of MAWGE (Section 4.1), the Markovian operator transition for multi-pattern workflow generation (Section 4.2), the internal structure of the modules (Sections 4.3–4.6) and the multidimensional complexity computation (Section 4.7).

4.1. Operator Topologies

MAWGE supports six fundamental operator topologies that capture diverse reasoning in multi-agent systems. Each topology defines a structural pattern governing agent interactions, data flow and execution dependencies. These patterns can be composed hierarchically to construct complex heterogeneous workflows:

- *Single-Agent (SA).* pattern represents the simplest topology consisting of a single agent A_0 that processes the entire task independently. This pattern simply serves as the atomic building block for more complex compositions.
- *Chain-of-Thought (CoT).* pattern structures reasoning as a sequential pipeline of n agents $\{A_0, A_1, \ldots, A_{n-1}\}$ where $n \geq 2$, forming a linear dependency chain. Each agent A_i processes the output of A_{i-1}, enabling progressive problem decomposition and stepwise reasoning.
- *Tree-of-Thought (ToT).* pattern introduces parallel exploration through a root agent A_0 that branches into b child agents $\{A_1, A_2, \ldots, A_b\}$ where $b \geq 2$. The root agent A_0 generates an initial problem decomposition and each child agent A_j independently pursues a distinct solution strategy in parallel.
- *Self-Consistency (SC).* pattern generalizes tree-of-thought by adding an aggregation stage. The pattern consists of b parallel agents $\{A_0, A_1, \ldots, A_{b-1}\}$ where $b \geq 2$, followed by a single aggregator agent A_b that synthesizes outputs.
- *Debate (Db).* pattern models adversarial multi-turn reasoning between two agents A_0 and A_1 exchanging arguments over r rounds where $r \geq 2$. Each agent observes the prior response from its counterpart, enabling iterative refinement through critique, rebuttal and synthesis.
- *Self-Refine (SR).* pattern combines sequential reasoning with iterative refinement loops. The pattern consists of a chain of ℓ agents $\{A_0, A_1, \ldots, A_{\ell-1}\}$ where $\ell \geq 2$, with each agent executing r refinement rounds where $r \geq 2$. Each agent A_i refines its output r times with iterative self-critique.

4.2. Markovian Probabilistic Operator Transition

To enable the generation of heterogeneous multi-pattern workflows, MAWGE employs a Markovian probabilistic transition mechanism for composing operator topologies. Let $\mathcal{P} = \{SA, CoT, ToT, SC, Db, SR\}$ denote the set of six operator patterns defined in Section 4.1. Let $\mathbf{P} \in [0,1]^{6\times 6}$ be a matrix where entry P_{ij} is the conditional probability of transitioning from pattern i to pattern j:

$$P_{ij} = \Pr(\text{pattern}_{t+1} = j \mid \text{pattern}_t = i), \quad \forall i, j \in \{0, 1, \ldots, 5\} \tag{2}$$

with the constraint that each row sums to unity: $\sum_{j=0}^{5} P_{ij} = 1$ for all i. The Markovian property implies that pattern selection at step $t+1$ depends only on the current pattern π_t. A memory-based mechanism considering the full history of $\{\pi_0, \ldots, \pi_{t-1}\}$ might be designed as well. The resulting composite workflow forms a directed graph $\mathcal{W} = (\mathcal{A}, \mathcal{E})$ where $\mathcal{A} = \bigcup_{k=0}^{m-1} \mathcal{A}_{\pi_k}$ is the union of all pattern agents and $\mathcal{E} = \mathcal{E}_{\text{intra}} \cup \mathcal{E}_{\text{inter}}$ includes both intra-pattern edges (within each π_k) and inter-pattern edges (between successive patterns). Refer to Table 2 for the default transition probabilities.

4.3. Generator Module

The generator module (see Figure 1) constructs workflows by composing operator topologies in Section 4.1. Given a target workflow size N and pattern-specific parameters such as chain length n for *CoT*, the generator produces an ordered sequence of pattern objects $\Pi = \{\pi_0, \pi_1, \ldots, \pi_{|\Pi|-1}\}$ where each $\pi_i \in \mathcal{P} = \{SA, CoT, SC, ToT, Db, SR\}$ is a pattern instance.

Pattern Instantiation. The generator accepts explicit parameters for each topology: `-cot-length` n creates a *CoT* pattern with n sequential agents ($n \geq 2$); `-sc-num-parallel` b creates an *SC* pattern

Table 2
Pattern transition probabilities of the first-order Markovian chain to order patterns within a workflow.

From / To	SA	CoT	SC	ToT	Db	SR
SA	0.35	0.20	0.15	0.15	0.10	0.05
CoT	0.30	0.10	0.25	0.20	0.10	0.05
SC	0.35	0.25	0.10	0.15	0.10	0.05
ToT	0.30	0.25	0.20	0.10	0.10	0.05
Db	0.40	0.20	0.15	0.15	0.05	0.05
SR	0.40	0.20	0.15	0.15	0.08	0.02

with b parallel agents plus one aggregator ($b+1$ total); `-tot-num-branches` b creates a *ToT* pattern with one splitter plus b branches ($b+1$ total); `-debate-rounds` r creates a *Db* pattern with 2 unique agents reused across r rounds; `-refine-chain-length` ℓ and `-refine-rounds` r create a *SR* pattern with ℓ agents each executing r refinement iterations. After allocating agents to requested patterns, the generator fills the remaining budget $N - \sum_i |\mathcal{A}_{\pi_i}|$ with simply *SA* patterns. Table 3 in Appendix A summarizes the complete set of command-line arguments exposed by MAWGE.

Markovian Pattern Ordering. To produce heterogeneous workflows, the generator orders patterns using the transition matrix in Section 4.2. Let $\Pi_{\text{unordered}}$ denote the set of pattern instances to be ordered. The algorithm proceeds as follows: (i) initialize $\Pi_{\text{ordered}} \leftarrow \emptyset$ and select a random initial pattern $\pi_0 \sim \mathcal{U}(\Pi_{\text{unordered}})$; (ii) for each subsequent position, compute transition probabilities $\mathbf{w}$ for all remaining patterns $\pi_j \in \Pi_{\text{unordered}} \setminus \Pi_{\text{ordered}}$; (iii) normalize weights $\mathbf{w}' = \mathbf{w}/\|\mathbf{w}\|_1$ and sample the next pattern π_{t+1}; (iv) append π_{t+1} to Π_{ordered} and repeat until all patterns are ordered. This Markovian ordering produces workflows where pattern transitions reflect empirically-plausible reasoning compositions (e.g., *CoT* followed by *SC* for ensemble reasoning).

4.4. Serializer Module

The serializer module instantiates pattern specifications from the generator into concrete workflows by assigning LLM models, performance characteristics and tools to each agent. Given a pattern sequence $\Pi = \{\pi_0, \pi_1, \ldots, \pi_{|\Pi|-1}\}$, the serializer produces the agent instances $\mathcal{A} = [A_0, A_1, \ldots, A_{|\mathcal{A}|-1}]$ where each agent is fully characterized by the tuple $(id_i, \theta_i, \rho_i, \gamma_i, \tau_i)$, as defined in Section 3.1.

Agent Instantiation. For each pattern $\pi_k \in \Pi$, the serializer generates $|\mathcal{A}_{\pi_k}|$ agent instances with randomly sampled attributes: (i) LLM model $M_i \sim \mathcal{U}(\mathcal{M})$ where $|\mathcal{M}| = 75$ models; (ii) temperature $T_i = U(0,1)$ on a discrete 0.1 scale; (iii) agent latency $\ell_i \sim \text{LogNormal}(\mu = 5.12, \sigma = 0.58)$ milliseconds; (iv) output tokens $n_i^{\text{out}} \sim \mathcal{U}(50, 500)$; (v) inference time $t_i^{\text{inf}} = (t_i^{\text{ttft}} + n_i^{\text{out}}/\lambda_i)$ milliseconds where t_i^{ttft} and λ_i are model-specific throughput values from Table 5; (vi) cost $c_i = (n_i^{\text{out}}/10^6) \times p_i$ where p_i is the per-million-token pricing; (vii) tools $\mathcal{T}_i$ with each tool having latency $\ell_k \sim \mathcal{U}(5, 100)$ ms (intra-network) or $\mathcal{U}(100, 1000)$ ms (inter-network) and request count $r_k \sim \mathcal{U}(1, 5)$. Note that latencies and inference times are characterized on real-world data in the works [17] and [18] respectively.

Inter-Pattern Connections. After instantiating all patterns, the serializer adds inter-pattern edges $\mathcal{E}_{\text{inter}}$ to create heterogeneous workflows. For each agent A_i at the boundary of a pattern (e.g., last agent in *CoT*), the serializer randomly samples $k \sim \mathcal{U}(\delta_{\min}^{\text{out}}, \delta_{\max}^{\text{out}})$ successors from agents A_j with indices $j > i$ (ensuring the acyclic property) and connects the separate patterns to each other. Refer to Section 5.3 for a sample workflow output.

4.5. Simulator Module

The simulator module executes workflows with event-driven simulation that produces temporally-ordered execution traces:

Event Types. The simulator tracks seven event types: AGENT_START marks agent A_i initiation when its predecessors have completed; AGENT_TOOLS_START signals the beginning of tool execu-

tion, TOOL_EXECUTE traces each tool invocation, AGENT_TOOLS_END marks completion of tools, AGENT_INFERENCE_START and AGENT_INFERENCE_END show start and completion of LLM inference and AGENT_END signals agent completion and triggers successor activation.

Execution Semantics. The event queue $\mathcal{Q}$ is initialized with AGENT_START events for all entry nodes $\mathcal{A}_{\text{entry}} = \{A_i \mid \delta_i^{\text{in}} = 0\}$ at $t = 0$. For each event e_k dequeued from $\mathcal{Q}$: (i) AGENT_START schedules AGENT_TOOLS_START at $t_k + \ell_i$ after initial agent latency; (ii) AGENT_TOOLS_START schedules sequential TOOL_EXECUTE events, dispatching each tool $t_j \in \mathcal{T}_i$ only after the preceding invocation completes, and schedules AGENT_TOOLS_END at $t_k + \sum_{t_j \in \mathcal{T}_i}(\ell_j \times r_j)$; (iii) AGENT_TOOLS_END schedules AGENT_INFERENCE_START immediately; (iv) AGENT_INFERENCE_START schedules AGENT_INFERENCE_END at $t_k + t_i^{\text{inf}}$; (v) AGENT_END decrements predecessor counters for successors $A_j \in \mathcal{S}_i$ and schedules AGENT_START for A_j when its predecessors complete. The simulation terminates when the queue is empty.

4.6. Visualizer Module

The visualizer module transforms the serialized workflows into human-readable graphical layouts, relying heavily on the Graphviz library [19]. The output files are generated in the PNG format with left-to-right convention. This module primarily serves researchers by enabling visual debugging of workflow topology and facilitating the presentation of workflows in their own works. Refer to Section 5.3 for example visualizations.

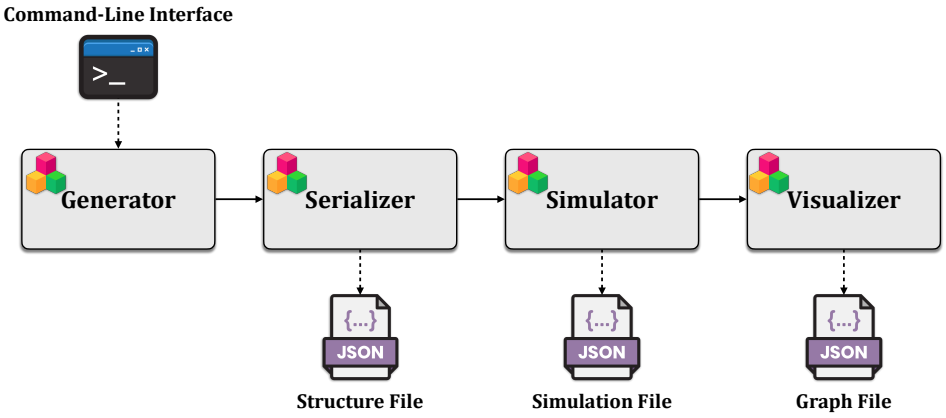


Figure 1: The flow of MAWGE library with Generator, Serializer, Simulator and Visualizer modules including their output files.

4.7. Workflow Complexity

MAWGE computes a complexity score for each workflow $\mathbf{c}(\mathcal{W})$. The complexity calculation includes six structural properties: the number of agents $|\mathcal{A}|$, the number of edges $|\mathcal{E}|$, the average connectivity degree $\frac{1}{N}\sum_{A_i \in \mathcal{A}}(\delta_i^{\text{in}} + \delta_i^{\text{out}})$, the width W and the cycle count C; and five functional properties: the distinct pattern count $|\mathcal{P}|$, the total tool count $\sum_{A_i \in \mathcal{A}} |\mathcal{T}_i|$, the total tool latency $\sum_{A_i \in \mathcal{A}} \ell_i$, the total inference time $\sum_{A_i \in \mathcal{A}} t_i^{\text{inf}}$ and the total cost $\sum_{A_i \in \mathcal{A}} c_i$. To produce a single scalar complexity indicator, MAWGE computes a normalized weighted score:

$$\mathbf{c}(\mathcal{W}) = \left(\sum_{k=1}^{11} w_k \cdot |c_k| \right) \tag{3}$$

where c_k is the k-th complexity metric and $|c_k|$ undergoes min-max normalization. The current implementation simply uses uniform weighting ($w_k = 1/11$).

5. Experimental Study

In this section, we evaluate MAWGE through systematic empirical analysis across three dimensions: (i) computational efficiency to measure workflow generation time costs across modules, (ii) workflow complexity analysis by the effects of configuration parameters and (iii) an illustrative case study. All experiments are performed on MacBook Air (Apple M2 chip, 8 GB memory and 8 CPU cores).

5.1. On Workflow Generation Time Cost

We evaluate the computational efficiency of MAWGE by measuring the execution time of each module across workflow sizes ranging from 10 to 1000 agents. Figure 2 presents both absolute execution time and relative percentage for module share from the total time. The generator exhibits negligible overhead, scaling from 0.01ms to 0.19ms, validating the efficiency of our Markovian pattern transition mechanism. The serializer demonstrates exponential growth from 0.19ms to 132.06ms (700× increase). The simulator also grows from 0.23ms to 230.16ms (1000× increase). The visualizer accounts for the majority of computational cost, ranging from 71.26% to 99.38% of total execution time, with absolute times scaling from 68.78ms to 1071.38ms. To gain more efficiency, users can turn off the simulator and visualizer modules. Aggregating all four modules, end-to-end workflow generation scales from 69.21ms for 10 agents to 1433.80ms for 1000 agents. Our analysis identifies the visualizer as the primary bottleneck, consuming 71–99% of total execution time. This bottleneck is due to Graphviz [19], which our library heavily relies on for graph rendering. Nonetheless, current performance remains within the practical limits for the majority of research scenarios.

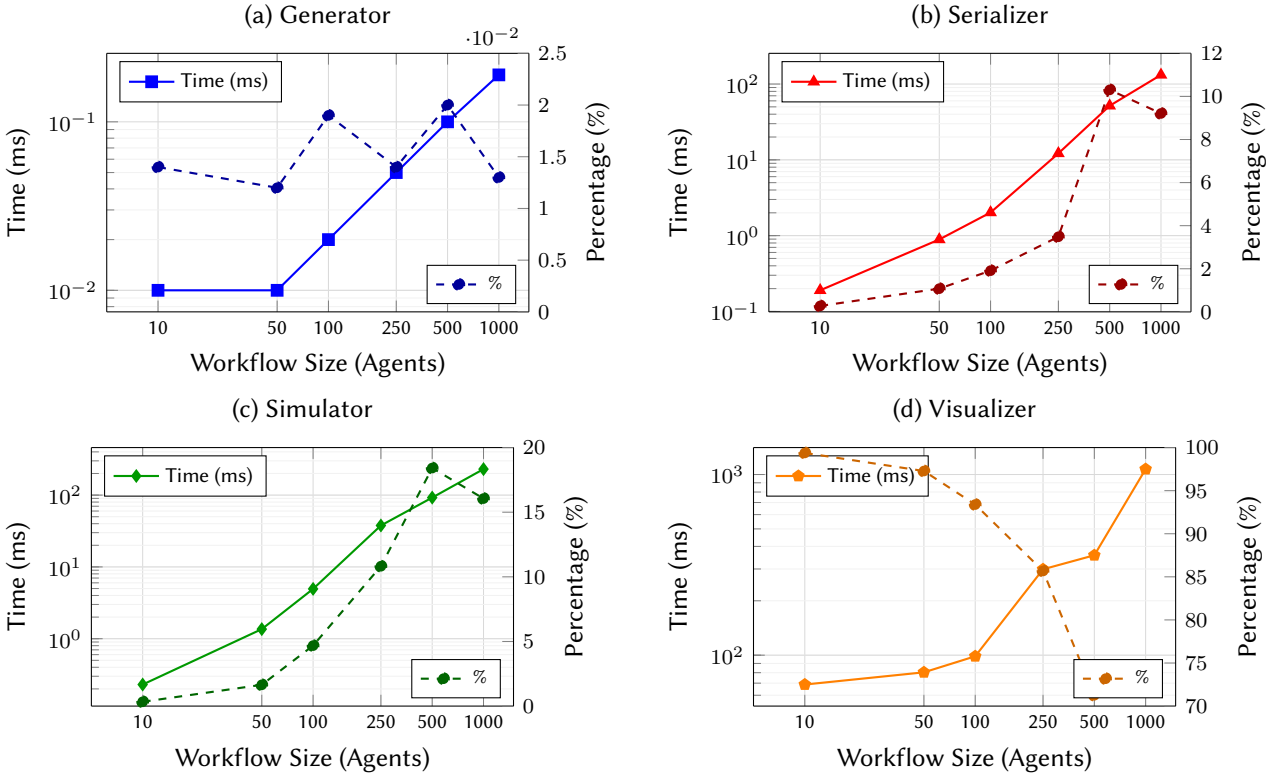
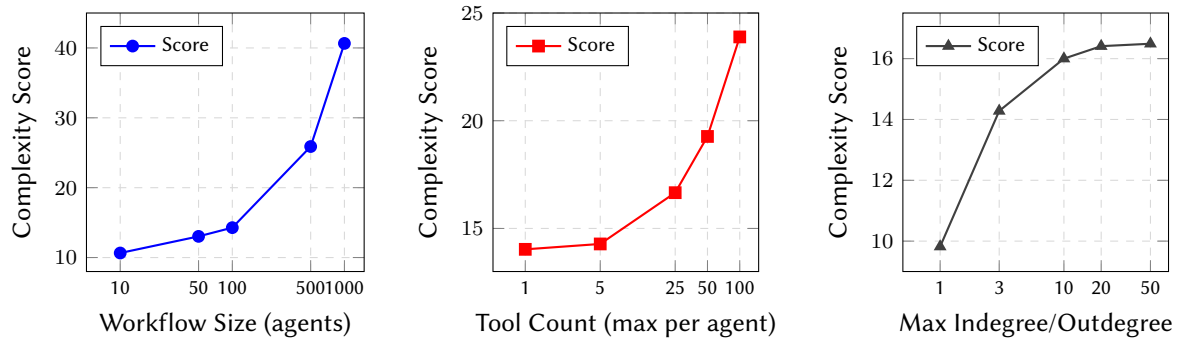


Figure 2: Log-scaled module execution time analysis with absolute time (solid lines) and relative percentage (dashed lines) for workflow sizes 10–1000 agents. **(a)** Generator. **(b)** Serializer. **(c)** Simulator. **(d)** Visualizer.

5.2. On Workflow Complexity Score

We evaluate the sensitivity of the proposed workflow complexity metric on three configuration parameters: workflow size, tool count and degree constraints. Figure 3 presents the empirical complexity scores at the scale of 0 and 100. Figure 3(a) demonstrates growing behavior with workflow size: the increase from 10 to 1000 agents (100× growth) raises the score from 10.65 to 40.65 (3.8× increase). Figure 3(b) also shows positive correlation between tool count and complexity. With workflow size fixed at 100 agents, the increase in tool count from 1 to 100 raises the score from 14.03 to 23.89 (1.7× increase). In this respect, workflow size has more dominance on complexity score than tool count. Figure 3(c) shows a different behavior (i.e., saturation). The initial increase from maximum degree 1 to 3 produces a 45% growth (9.82 to 14.28), but later rapidly saturates (degree 3 to 50 adds only 2.21 points or +15%), showing maximum individual contributions of parameters on the scores.



(a) For workflow size (degree=3, tool count=5).

(b) For tool count (workflow size=100, degree=3).

(c) For degree (workflow size=100, tool count=5).

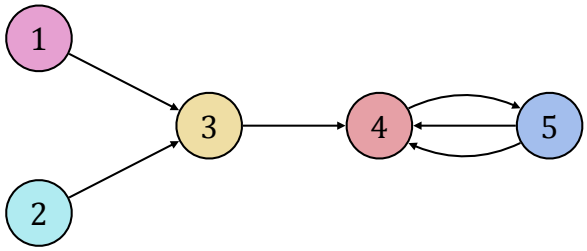
Figure 3: Sensitivity analysis of workflow complexity score under varying configuration parameters. (a) against workflow size (b) against total tool count (c) against maximum in-degree/out-degree.

5.3. Case Study

As a case study, we generate a simple agentic workflow with the total of 5 agents. Figure 4a visualizes the resulting directed graph: two parallel agents ([1]: Claude-4.6-Opus and [2]: GPT-5.4-Mini) initiate independent reasoning paths that converge at a merger agent ([3]: Grok-3-Mini), as the self-consistency pattern. The workflow then transitions to a debate pattern where two agents ([4]: Mimo-V2-Flash and [5]: Ministral-3-14B) engage in iterative refinement through a bidirectional cycle. The corresponding JSON output (see Figure 4b) captures the complete workflow specification including agent models temperature settings and tool assignments. It also shows the overall complexity metrics (e.g., complexity score of 9.23). This case study illustrates MAWGE's ability to synthesize heterogeneous multi-agentic workflows and provides both visual and text outputs for benchmarking and analysis.

6. Conclusion

This work introduces MAWGE, a Python library for generating parametric and synthetic multi-agentic workflows with the construction of six reasoning patterns and Markovian probabilistic transitions between these patterns. By integrating a comprehensive database of 75 production LLM models spanning 7 major vendors, MAWGE bridges the gap between synthetic evaluation and real-world deployment constraints. MAWGE consists of four internal modules: *generator*, *serializer*, *simulator* and *visualizer*. Specifically, the event-driven simulator traces agentic workflow execution by considering computational and communicational costs, enabling an end-to-end and realistic simulation. Our experimental evaluation clearly shows the practicality and scalability of the library under the effects of distinct configuration parameters. A systematic comparison of our synthetically-generated workflows



(a) Workflow graph visualization.


```
{
  "workflow_size": 5,
  "indegree_range": {"min": 0, "max": 5},
  "outdegree_range": {"min": 1, "max": 5},
  "tool_max": 5, "seed": 12,
  "sc_num_parallel": 2, "debate_rounds": 2,

  "agents": [
    {
      "id": "1", "model": "claude-4.6-opus",
      "temperature": 0.0, "latency": 220.2,
      "inference": 10155.0, "ttft": 1.78,
      "tokens_per_sec": 40, "cost": 0.008375,
      "output_size": 335, "tool_size": 3,
      "tools": [
        {"is_intra": true, "latency": 55.3, "number_of_requests": 3},
        {"is_intra": true, "latency": 6.0, "number_of_requests": 3},
        {"is_intra": true, "latency": 66.1, "number_of_requests": 4}
      ],
      "indegree_size": 0, "outdegree_size": 1,
      "successors": ["2"]
    },
    ...
  ],

  "complexity": {
    "num_agents": 5, "num_edges": 6, "average_degree": 1.71,
    "max_depth": 3, "width": 2, "pattern_diversity": 3,
    "total_tool_count": 20, "total_latency": 846.9,
    "total_inference": 24322.3, "execution_time": 39923.5,
    "cycle_count": 2, "complexity_score": 9.23,
    "total_cost": 0.009044
  }
}
```


(b) Workflow structure JSON output.

Figure 4: Case study workflow, `./mawge-cli generate -workflow 5 -debate-rounds 2 -sc-num-parallel 2 -simulate -visualize -seed 12`. (a) Graph visualization. (b) Structure JSON File.

with realistic workflow traces remains an open direction where we plan to pursue this validation as future work. In addition, we are currently working on scheduling of multi-agent workflows on distributed systems such as cloud computing. Overall, we still envision that MAWGE will serve as a benchmarking tool to generate a large collection of distinct agentic workflows over which researchers can carry out their novel techniques such as scheduling algorithms.

Declaration on Generative AI

During the preparation of this work, the author(s) used Claude in order to: Paraphrase and reword, Improve writing style, Grammar and spelling check. After using these tool(s)/service(s), the author(s)

reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] A. Bondielli, G. Bonetta, E. Leonardelli, I. Siragusa, Preface to the Ninth Workshop on Natural Language for Artificial Intelligence (NL4AI), in: Proceedings of the Ninth Workshop on Natural Language for Artificial Intelligence (NL4AI 2026) co-located with 24th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2026), 2026.

[2] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al., Training language models to follow instructions with human feedback, Advances in neural information processing systems 35 (2022) 27730–27744.

[3] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, I. Mordatch, Improving factuality and reasoning in language models through multiagent debate, arXiv preprint arXiv:2305.14325 (2023).

[4] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al., Chain-of-thought prompting elicits reasoning in large language models, Advances in neural information processing systems 35 (2022) 24824–24837.

[5] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, D. Zhou, Self-consistency improves chain of thought reasoning in language models, arXiv preprint arXiv:2203.11171 (2022).

[6] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, K. Narasimhan, Tree of thoughts: Deliberate problem solving with large language models, Advances in neural information processing systems 36 (2023) 11809–11822.

[7] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, et al., Self-refine: Iterative refinement with self-feedback, Advances in neural information processing systems 36 (2023) 46534–46594.

[8] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, et al., Autogen: Enabling next-gen llm applications via multi-agent conversation, arXiv preprint arXiv:2308.08155 (2023).

[9] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, S. Yau, Z. Lin, L. Zhou, et al., Metagpt: Meta programming for a multi-agent collaborative framework, in: International Conference on Learning Representations, volume 2024, 2024, pp. 23247–23275.

[10] J. Zhang, J. Xiang, Z. Yu, F. Teng, X. Chen, J. Chen, M. Zhuge, X. Cheng, S. Hong, J. Wang, et al., Aflow: Automating agentic workflow generation, in: International Conference on Learning Representations, volume 2025, 2025, pp. 34040–34077.

[11] G. Zhang, K. Chen, G. Wan, H. Chang, H. Cheng, K. Wang, S. Hu, L. Bai, Evoflow: Evolving diverse agentic workflows on the fly, arXiv preprint arXiv:2502.07373 (2025).

[12] P. Yu, B. Liu, S. Tang, D. Li, W. Lin, Awto: A latency-optimized task offloading scheme for llm-driven agentic workflows on heterogeneous edge, Future Generation Computer Systems (2026) 108415.

[13] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. Da Silva, M. Livny, et al., Pegasus, a workflow management system for science automation, Future Generation Computer Systems 46 (2015) 17–35.

[14] W. Chen, E. Deelman, Workflowsim: A toolkit for simulating scientific workflows in distributed environments, in: 2012 IEEE 8th international conference on E-science, IEEE, 2012, pp. 1–8.

[15] T. Tobita, H. Kasahara, A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, Journal of Scheduling 5 (2002) 379–394.

[16] R. P. Dick, D. L. Rhodes, W. Wolf, Tgff: task graphs for free, in: Proceedings of the Sixth International Workshop on Hardware/Software Codesign.(CODES/CASHE'98), IEEE, 1998, pp. 97–101.

[17] F. Palumbo, G. Aceto, A. Botta, D. Ciuonzo, V. Persico, A. Pescapé, Characterization and analysis of cloud-to-user latency: The case of azure and aws, Computer Networks 184 (2021) 107693.

[18] BenchLM.AI, Llm speed benchmark, 2026. URL: https://benchlm.ai/llm-speed, accessed: 2026-07-23.
[19] E. R. Gansner, S. C. North, An open graph visualization system and its applications to software engineering, Software: practice and experience 30 (2000) 1203–1233.

A. Command-Line Arguments

Table 3 lists the complete set of command-line arguments exposed by the MAWGE generator. The arguments are grouped into core parameters, agentic reasoning patterns and execution/output options.

Table 3
Command-line arguments for the MAWGE workflow generator.

Argument	Description
Core Parameters	
`-w, –workflow`	number of unique agents in the workflow
`–indegree-max`	maximum in-degree (incoming connections per agent)
`–outdegree-max`	maximum out-degree (outgoing connections per agent)
`–seed`	random seed for reproducible generation
Agentic Reasoning Patterns	
`–cot-length`	*chain-of-thought*: sequential reasoning chain length
`–sc-num-parallel`	*self-consistency*: number of parallel reasoning paths
`–tot-num-branches`	*tree-of-thought*: number of exploration branches
`–debate-rounds`	*debate*: number of adversarial reasoning rounds
`–refine-chain-length`	*self-refine*: iterative refinement chain length
`–refine-rounds`	*self-refine*: number of refinement iterations
Execution & Output	
`–simulate`	enable event-driven execution simulation
`-o, –output`	output file prefix for generated artifacts

B. Agent Attributes

Table 4 presents the individual attributes to compose an agent instance, with their descriptions, units/ranges and sources to generate.

Table 4
Agent component specification in the MAWGE library.

Attribute	Description	Unit/Range	Source
`id`	Unique agent identifier	–	Sequential
`model`	LLM model name	–	Random
`temperature`	Sampling temperature for inference	[0.0, 1.0]	Random
`latency`	External tool call latency	milliseconds	Lognormal
`inference`	Total LLM inference time	milliseconds	Calculated
`ttft`	Time to first token	seconds	Model-specific
`tokens_per_sec`	Token generation throughput	tokens/s	Model-specific
`input_size`	Input prompt size	tokens	Random
`output_size`	Generated output size	tokens	Random
`tool_size`	Total tool call overhead	milliseconds	Calculated
`cost`	Inference cost	USD	Calculated
`indegree_size`	Number of incoming edges	–	Pattern-dependent
`outdegree_size`	Number of outgoing edges	–	Pattern-dependent
`successors`	List of successor agent IDs	–	Pattern-dependent
`tools`	List of external tool identifiers	–	Random

C. Model Specifications

Table 5 reports the throughput, time-to-first-token and per-million-token pricing for each of the 75 LLM models considered in MAWGE.

Table 5
LLM model performance characteristics and pricing in MAWGE [18].

Model	Tput (t/s)	TTFT (s)	Price ($ per 1M)	Model	Tput (t/s)	TTFT (s)	Price ($ per 1M)
OpenAI Models				OpenAI Models			
gpt-5.4-pro	74	151.79	180.00	o1	98	32.29	60.00
gpt-5.4	74	151.79	15.00	o3	118	5.38	8.00
gpt-5.4-mini	201	3.85	4.50	o3-mini	160	7.12	4.40
gpt-5.4-nano	191	3.64	1.25	o3-pro	27	84.93	80.00
gpt-5.3-codex	79	88.26	14.00	o4-mini	161	21.94	0.00
gpt-5.2-codex	123	87.34	14.00	Anthropic Claude			
gpt-5.2	73	130.34	14.00	claude-4.1-opus	29	1.66	75.00
gpt-5.1	111	57.47	10.00	claude-4.1-opus-thinking	29	15.00	0.00
gpt-5	83	36.28	10.00	claude-4-sonnet	40	1.33	15.00
gpt-5-nano	137	83.30	0.40	claude-4.6-sonnet	44	1.48	15.00
gpt-5-mini	86	65.32	2.00	claude-4.5-opus	46	1.01	25.00
gpt-4.1	108	1.02	8.00	claude-4.6-opus	40	1.78	25.00
gpt-4.1-nano	181	0.63	0.40	claude-3-haiku	138	1.16	1.25
gpt-4.1-mini	80	0.76	1.60	Google Gemini			
gpt-4o	131	0.81	10.00	gemini-3.1-pro	109	29.71	12.00
gpt-4o-mini	33	3.16	0.60	gemini-3.1-flash-lite	205	7.50	1.50
gpt-4-turbo	30	2.84	30.00	gemini-3-pro	109	32.65	12.00
gpt-oss-120b	262	0.79	0.00	gemini-3-flash	159	1.19	3.00
gpt-oss-20b	313	0.65	0.00				
Google Gemini				Mistral Models			
gemini-2.5-pro	117	21.19	10.00	mistral-small-4	175	0.64	0.60
gemini-2.5-flash	221	0.50	2.50	mistral-medium-3	57	1.20	2.00
Meta Llama				mistral-large-3	48	1.04	1.50
llama-4-scout	128	0.70	0.00	mistral-large-2	38	1.45	0.00
llama-4-maverick	121	0.95	0.00	xAI Grok			
llama-3.1-405b	29	2.19	0.00	grok-4.20	233	10.33	6.00
Mistral Models				grok-4.3	209	12.36	2.50
ministral-3-3b	274	0.42	0.10	grok-4.1-fast	138	0.54	0.50
ministral-3-8b	182	0.52	0.15	grok-4	54	15.60	0.00
ministral-3-14b	110	0.60	0.20	grok-3-mini	190	0.54	0.50
xAI Grok				Other Models			
grok-code-fast-1	172	2.81	1.50	glm-5	74	1.64	3.20
Other Models				glm-4.7	82	1.10	0.00
nemotron-3-super-100b	367	0.71	0.00	glm-4.7-flash	95	0.91	0.00
nemotron-3-nano-30b	152	1.90	0.00	glm-4.5	51	1.45	2.20
deepseek-v3.2	35	3.75	0.42	glm-4.5-air	106	1.18	1.10
deepseek-r1-distill-qwen-32b	60	0.84	0.00	lfm2-24b-a2b	92	0.42	0.00
mercury-2-inception	789	3.88	0.75	step-3.5-flash	87	3.03	0.30
ling-2.6-flash	210	1.07	0.00	minimax-m2.7	45	2.53	1.20
nova-pro	141	0.81	0.00	minimax-m2.5	46	2.12	1.20
mimo-v2-flash	129	2.14	0.00	kimi-k2.5	45	2.38	3.00
qwen-3.5-397b	96	2.44	3.60	kimi-k2	43	1.51	2.50
				phi-4	35	2.02	0.00
				gemma-3-27b	31	2.04	0.00