Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Study on Quantitative Modeling of Task Completion Probability in AI Agents Using DAG and State-Machine

Akira Otsuki

otsuki.akira@nihon-u.ac.jp

Nihon University College of Economics

Research Article

Keywords: AI agents, Task orchestration, Directed acyclic graph (DAG), State machine, Reliability engineering, Redundancy allocation, Monte Carlo simulation

Posted Date: November 19th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11565438/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Study on Quantitative Modeling of Task Completion Probability in AI Agents Using DAG and State-Machine

Akira Otsuki[1*]
[1]Nihon University College of Economics, Tokyo, Japan
*Corresponding author: Akira Otsuki, otsuki.akira@nihon-u.ac.jp

Abstract

Autonomous AI agents increasingly decompose user instructions into task lists, but widely used approaches (AutoGPT, BabyAGI, MetaGPT) generate simple linear, ID-ordered sequences that cannot express conditional branching or parallel execution. Newer orchestration frameworks such as LangGraph, the LLM Compiler, and the Claude Agent SDK add DAG (Directed Acyclic Graph) or state-machine execution control for non-linear workflows, yet none provides a quantitative, probabilistic estimate of how likely a plan is to succeed, or how much a redundant (alternative) path improves that likelihood. This study integrates DAG-based dependency modeling, prompt engineering for conditional branching, and a state-machine execution engine, and formulates the resulting task-completion probability P(Goal) with reliability engineering. The model is generalized to 1-out-of-n (parallel) path redundancy and to general DAG topologies with parallel branching and AND-joins, and a cost-constrained redundancy-allocation problem and a Beta-prior estimator of path success probabilities from execution logs are given. Assuming independent task outcomes, Monte Carlo simulations (10,000–100,000 trials) of a retrieval-and-summarization task (72.7% completion for the linear model, 84.4% with redundancy) agree with the closed-form predictions to within 0.18 percentage points across 24 parameter combinations and within 1 point under all conditions. Redundancy at the bottleneck stage yields about 3.9 times the gain at the most reliable stage, as the marginal-utility formula predicts. Alternative paths help most when failures are persistent; correlated failures reduce the gain. With a real LLM agent (100 runs), completion rates predicted from probabilities estimated on half the runs fell within the 95% intervals observed on the other half.

Keywords: AI agents; Task orchestration; Directed acyclic graph (DAG); State machine; Reliability engineering; Redundancy allocation; Monte Carlo simulation

1 Introduction

AI agents are already capable of efficiently substituting for a large portion of human tasks. For example, they can perform nearly any task, including text generation, conversation summarization, content classification, data extraction, answering questions, and proposing next steps [1]. Furthermore, research on generative AI that automates the entire

research process is also progressing, led by The AI Scientist [2,3], published by Sakana AI.

However, in conventional task-generation models represented by AutoGPT [4], BabyAGI [5], and MetaGPT [6], the LLM outputs task lists in a simple linear structure ordered by ID when decomposing instructions, and therefore cannot handle complex task dependencies. Specifically, task lists in real-world, complex projects require “conditional branching (if A succeeds, proceed to B; if it fails, proceed to C)” and “parallel execution,” which the current ID-order sorting cannot accommodate in non-linear workflows. This is where execution-control frameworks such as LangGraph [7], the LLM Compiler [8], and Claude Agent SDK [9] emerged, providing orchestration via DAG (Directed Acyclic Graph) [10,11] or State-Machine [12], enabling these frameworks to support non-linear workflows (graph networks) as well. However, they still lack mechanisms for prioritizing tasks based on each task’s success probability, or for quantitatively estimating the completion probability when a redundant path is added. Therefore, this study builds on the execution infrastructure of DAG and State-Machine and formulates completion probability based on the framework of reliability engineering (Section 3.5, Billinton and Allan [13]), validating its agreement with simulated values through Monte Carlo simulation and with the completion rates observed for a real LLM agent.

Through the evaluation experiments, the values predicted by the proposed model fell within the 95% confidence intervals of the values simulated by Monte Carlo simulation [14] in every configuration except one, which lay 0.03 points outside its interval (Section 4.7). This shows that the analytical model and the numerical simulation agree under the assumption that task outcomes are independent. How far this carries over to real agents is examined separately, through sensitivity analyses of retries and of correlated failures (Sections 4.7 and 4.8) and an experiment with a real LLM agent (Section 4.10), in which the completion rates predicted from probabilities estimated on half of 100 runs fell within the 95% confidence intervals of the rates observed in the other half. Consequently, as long as this assumption holds, without running a massive simulation every time, the overall task completion probability can now be calculated instantly simply by substituting the success probabilities of the original and alternative paths into the proposed model. Furthermore, using the proposed model, it is also possible to estimate how much the completion probability improves depending on where a redundant path is added, i.e., to quantify that improvement effect.

2 Related Work and Prior Cases

2.1 Prior Research and Cases on AI Agents

As of August 2026, an overview of the major AI agents published in open-source communities such as GitHub shows that AutoGPT [4] is a pioneering autonomous agent

that generates and executes its own tasks once given a goal, while BabyAGI [5] is an agent that combines a task-management system with an LLM to prioritize and execute tasks. MetaGPT [6], meanwhile, is a framework that distributes software development among agents (such as a programmer, a PM, and a reviewer) to complete it collaboratively. In 2026, agents such as Gemini Deep Research and Deep Research Max (built on Gemini 3.1 Pro and released in April 2026 [15]) that automate multi-stage search and summarization have also come into practical use. As for the elemental technologies underlying these AI agents, the foundational theories of "task decomposition" and "self-repair" that these agents draw on include the following: ReAct (Reasoning and Acting) [16] defines the basic LLM structure that combines an agent's reasoning and action; Self-Refine [17] describes the foundations of logic for revising one's own output based on generated feedback (self-repair); and Chain-of-Thought (CoT) [18] discusses the foundations of a thought process that decomposes a task step by step.

2.2 Orchestration Technologies Based on DAG and State-Machine

Furthermore, moving into 2026, orchestration technologies for production use based on DAG and State-machines are also being rapidly developed. LangGraph [7] is a framework that compiles an agent's processing graph into a state machine, enabling execution and resumption across steps as well as self-correction, while the LLM Compiler [8] provides a mechanism in which a Function Calling Planner generates the task dependencies in a DAG structure and a Task Fetching Unit executes tasks in parallel as soon as the dependencies are resolved. Claude Agent SDK [9], meanwhile, adopts an architecture in which sub-agents run in parallel within independent context windows and return only a summary to the orchestrator; according to Anthropic's research [19], a parallel multi-agent configuration using Claude Opus 4 as the lead agent and Claude Sonnet 4 as sub-agents outperformed a single-agent configuration by 90.2% in the company's internal research evaluation.

2.3 Positioning of This Study

However, all of these frameworks merely provide "engineering for execution control" through DAGs or state machines, and none of them offers a quantitative, probabilistic model for estimating each task's success probability or how much the completion probability improves when a redundant path is added. Furthermore, Zhang et al. [20] have proposed EvoFSM, which allows the state-transition structure of a finite state machine (FSM) itself to self-evolve in a controllable manner, advancing research toward automating the design of the state machine itself. The novelty of this study lies in integrating, on top of execution control via DAG and State-Machine, a model of completion probability that draws on the framework of reliability engineering (Section 3.5, Billinton and Allan [13]) together with an evaluation design that validates its theoretical values through Monte Carlo simulation — a combination that differentiates this work by substantiating

“why, and how much, it is effective,” an aspect not addressed by existing frameworks, from a mathematical standpoint, with simulation-based verification and sensitivity analyses.

The author has a related manuscript under review at another journal (IEEE Access), which proposes SH-GD/SM-L (Schema-level Hypergraph with Learned Graded Dependency / State Machine). Both works start from the same problem setting, an AI agent that can switch to an alternative path when a task fails, and both use the standard parallel-redundancy formula of reliability engineering. Their objectives and contributions differ. SH-GD/SM-L attaches a confidence grade and a learned relation strength to every dependency edge, learns schema-pair substitutability from validation logs as Beta posteriors, and selects alternative tasks dynamically at run time from the whole task network; its evaluation concerns the calibration and the realized completion rate of such learned, dynamically re-planned policies. The present study, in contrast, takes the alternative paths of a plan as given and asks what completion probability a fixed DAG with redundancy achieves: it derives closed-form expressions for serial, 1-out-of-n and AND-join structures (Equations 6–11), formulates the cost-constrained redundancy-allocation problem (Equations 12 and 13), and examines these expressions against simulation, retries, correlated failures and exact optimization. It thus provides the static, analytical baseline against which dynamic methods such as SH-GD/SM-L can be compared, and it contains none of their learning or re-selection mechanisms. The evaluation scenario of Sections 4.1, 4.5 and 4.6 deliberately uses the same reference values as SH-GD/SM-L so that the two studies can be compared on a common setting. Because that manuscript has not yet been accepted, it is mentioned here rather than listed in the references; it will be cited once accepted.

3 Proposed Model

3.1 Overview of the Proposed Model

First, this study introduces three elements — (1) introduction of a DAG (Directed Acyclic Graph), (2) prompt engineering for conditional branching, and (3) transition to a State-Machine — to manage tasks not as a mere list but as a graph (DAG) with dependencies, and to implement logic that dynamically determines the next path to proceed along according to the execution result (Success/Failure).

Note that, hereafter, the three terms “task,” “step,” and “stage” are used as distinguished below. A **Task** is a “node” in the DAG, referring to the smallest unit of processing to be executed. Both an original task (path) and its alternative (backup) task are each treated as one independent task. A **Step** refers to the execution order (1st, 2nd, …) when multiple tasks are arranged in series, a concept that is meaningful only when the DAG is a simple single path with no branching. A **Stage** refers to a broader concept meaning “one

processing unit that must be passed through in order to achieve the goal," and one stage consists of one original task and its alternative tasks (zero or more). A stage can apply to either a serial or a parallel structure.

For example, consider a task A→B→C in which B has an alternative task B′. In this case there are four tasks (A, B, B′, C); the steps form the serial order A = 1st, B (or B′) = 2nd, C = 3rd; and there are three stages: Stage 1 = {A}, Stage 2 = {B, B′}, Stage 3 = {C}. On the other hand, when A, B, and C are combined in parallel via an AND-join, the tasks remain the same three (A, B, C) executed simultaneously, but since all three run at the same time, an order (step) such as "1st, 2nd, …" cannot be defined. Even so, stages in the sense of "processing units that must be passed through" still hold: Stage 1 = {A}, Stage 2 = {B}, Stage 3 = {C}. In what follows, the term "stage" is mainly used so as to accommodate both serial and parallel contexts.

3.2 Introduction of a DAG (Directed Acyclic Graph)

Tasks are defined as "nodes" and dependencies as "edges," and the LLM is made to output them in the form depends_on: [id_1, id_2, …, id_n], thereby controlling complex execution order. Specifically, if V denotes the set of tasks and E the dependencies between tasks, the agent's plan is represented as a directed graph G = (V, E). The condition under which task v_j becomes executable is that all of its predecessor tasks Pre(v_j) have completed in the Success state, as shown in Equation 1.

$$Ready(v_j) \Leftrightarrow \forall v_i \in Pre(v_j),\ Status(v_i) = Success \qquad (1)$$

Here,

- Pre(v_j): the set of tasks that must be completed before task v_j
- ∈: belongs to (is a member of) the following set
- v_i: an individual predecessor task
- ∀ (the universal quantifier): "for all"

Thus, $\forall v_i \in Pre(v_j)$ means, "for task v_j, across all of its predecessor tasks v_i," and that $Status(v_i)$ = Success means that its status is Success. In other words, this defines the condition that "all prerequisites (predecessor tasks) are in place." Note that, in our implementation example, gpt-oss-120b-Q8 [21] was used as the LLM for semantically parsing the instruction text and inferring dependency relationships. This LLM belongs to the implementation example only and was not used in the evaluations of Chapter 4 (the real-agent experiment of Section 4.10 uses the model described there).

3.3 Prompt Engineering for Conditional Branching

We have the LLM generate a routing task that sets the destination ID for branching according to a task's output result (Success/Failure). After task v_i completes, the next prioritized task v_{next} is determined by the conditional expression in Equation 2.

$$v_{next} = \begin{cases} on_{success}(v_i) & \text{if } R_i = 1 \\ on_{failure}(v_i) & \text{if } R_i = 0 \end{cases} \qquad (2)$$

Here, R_i denotes the score (return value) obtained as a result of executing task v_i. In other words, through this prompt engineering, the LLM pre-builds the mapping of on_success and on_failure in JSON format at planning time. This makes an "intelligent avoidance action" possible at execution time.

3.4 Transition to a State Machine

Rather than a fixed list, we adopt a state-machine-type architecture that dynamically generates and determines the next task according to a task's current state (State). The execution engine is defined as a state machine that monitors the "state" of the entire system and loops until convergence. Regarding the update of system state S, the system state S_t at time t is the set of the statuses of all tasks, and is expressed as shown in Equation 3.

$$S_t = \{Status(v) \mid v \in V\} \qquad (3)$$

By monitoring this overall system state of viability, S_t, the execution engine is built as a state machine that iterates until the system — the collection of "all tasks," "generated code," "memory (variables) in use during execution," "output files," and so on — reaches a steady state. The states a task can take are as follows:

- **Pending**: The initial state in which a task has been generated. A phase in which predecessor tasks are not yet complete and no execution resources have been allocated.
- **Running**: A state in which all dependencies have been resolved and the LLM is generating code or actively processing a subprocess.
- **Success**: A confirmed state in which the process's exit code is 0 and the integrity of the output data has been verified.
- **Failed**: A state that could not be resolved even after reaching the retry limit. This invokes the on_failure function, guiding the system to an alternative path.
- **Skipped**: A path rendered unnecessary by conditional branching, or a task that has become unreachable. Explicitly marked to prevent deadlocks and conserve computational resources.

Regarding the stopping (convergence) condition, the mission is judged to be “terminated” once either of the conditions in Equation 4 is met.

$$Terminated \Leftrightarrow (S_t = S_{t-1}) \vee (\forall v \in V,\ Status(v) \in \{Success, Failed, Skipped\}) \qquad (4)$$

This equation connects, via a logical OR (∨), two cases in which the agent judges that it has either “done everything it can” or “cannot proceed any further.” The first term, $S_t = S_{(t-1)}$, represents the situation in which “there is no change in the overall system compared to one step earlier, $S_{(t-1)}$, versus now, S_t” — this is set in place to prevent infinite loops. The second term, ∀v∈V, Status(v) ∈ {Success, Failed, Skipped}, means that “for every task v, its status is one of Success, Failed (non-retryable), or Skipped” — that is, it represents the situation in which every task on the graph has been classified into one of the statuses that signify termination.

3.5 Mathematical Advantage of the Proposed Model

We describe the mathematical advantage of the proposed model’s completion probability P(Goal) over conventional AI-agent task-generation models. Letting p denote the success probability of the original task and p′ the success probability of the alternative path, and examining the difference in goal-achievement probability, the conventional model, as shown in Equation 5, behaves like a series circuit in which a single failure anywhere causes the whole to collapse, so its success probability decays exponentially.

$$P(Goal)_{linear} = \prod_{i=1}^{n} p_i \qquad (5)$$

That is, in a linear process (Linear Model), achieving the overall goal requires that every step from Step 1 through Step n succeed. In other words, when each task is independent and each has a success probability of p, the overall success probability P is determined by the product of the probabilities of each step, as shown in Equation 6.

$$P(Goal)_{linear} = p_1 \times p_2 \times \cdots \times p_n = p^n \qquad (6)$$

This “multiplication (product)” structure means it is mathematically identical to the structure of a series circuit, in which “if even one link is broken (fails) anywhere, no current flows (overall failure).” For example, even if each task’s success probability is 90% (p = 0.9), as shown below, although the individual success probability is relatively high, the overall success probability decreases exponentially as the steps accumulate — a phenomenon known as “exponential decay.”

- 1 task: 0.9 (90%)
- 5 tasks: $0.9^5 \approx 0.59$ (about 59%)
- 10 tasks: $0.9^{10} \approx 0.35$ (about 35%)

In contrast, as shown in Equation 7, the proposed model probabilistically improves resilience against errors by incorporating “redundancy” at planning time.

$$P(Goal)_{DAG} = \prod_{i=1}^{n}(p_i + (1 - p_i) \cdot p_i') \qquad (7)$$

Here, $(p_i + (1 - p_i) \cdot p'_i)$ shows that simply adding one alternative path improves the overall success rate, making it possible to deliver goal completion more reliably even in a highly uncertain external environment. Specifically, p_i is the probability of success via the original means; 1 minus p_i is the probability that the original means fails, and p'_i is the probability that the alternative means (Plan B) succeeds given that failure. This sum is always greater than p_i alone, because a “rescue term” is added to the original p_i. For example, consider a project with a per-task success probability of p = 0.8 (80%) and 5 stages. In the conventional (linear) model, the success probability is reduced by a factor of 0.8 at every step, so that, as shown below, the overall success probability ultimately falls to 32.8%.

- At the end of Stage 1: $0.8^1 = 0.8$
- At the end of Stage 2: $0.8^2 = 0.64$
- (omitted)
- At the end of Stage 5: $0.8^5 = 0.32768$

In contrast, for the proposed model (with an alternative path), if we take the alternative path’s success probability to be $p' = 0.5$ (50%), then, as shown in Equation 8, simply preparing a “Plan B” with only 50% accuracy improves the task’s success probability from p = 0.8 to 0.9.

$$P(Goal) = 0.8 + (1 - 0.8) \times 0.5 = 0.9 \qquad (8)$$

That is, considering 5 stages in the same way as the conventional model, the values progress as follows:

- At the end of Stage 1: $0.9^1 = 0.9$
- At the end of Stage 2: $0.9^2 = 0.81$
- (omitted)
- At the end of Stage 5: $0.9^5 \approx 0.590$

This improvement is dramatic because the success-probability calculation is a “multiplication (power)”: in the conventional linear model, as the number of steps n increases, the small “uncertainty (1 − p)” snowballs and drives the overall success probability toward zero, whereas the proposed model (DAG-type) “filters the 20% failure risk at each stage through a further 50% filter (the alternative path’s success probability),

diluting it down to 10%," allowing the survival probability to avoid a sharp drop even as stages accumulate.

3.6 Generalization: Extension to a 1-out-of-n (Parallel) Redundancy Model and DAG Topology

The model shown in Equations 7 and 8, which adds "one alternative path," is in fact merely a special case ($n_i = 2$) of the general redundancy model shown below. Here we generalize to the case where n_i alternative strategies (Plan A, B, C, …) exist for a given stage i, each assumed to succeed or fail independently. Letting the success probability of each strategy be $p_i^{(1)}$, $p_i^{(2)}$, …, $p_i^{(ni)}$, the probability P_i that the stage succeeds via at least one strategy can be expressed, drawing on the formulation of parallel redundant systems in reliability engineering (Billinton and Allan [13]), as shown in Equation 9.

$$P_i = 1 - \prod_{k=1}^{n_i} \left(1 - p_i^{(k)}\right) \qquad (9)$$

In Equation 7, the expression $(p_i + (1 - p_i)\cdot p'_i)$ is mathematically identical to the special case of Equation 9 with $n_i = 2$ $(1 - (1 - p_i)(1 - p'_i) = p_i + p'_i - p_i \cdot p'_i = p_i + (1 - p_i)p'_i)$. Because a stage succeeds as soon as at least one of its ni strategies succeeds, Equation 9 describes a 1-out-of-n (parallel) redundant system, not a general k-out-of-n system, which would require at least k successes. In other words, the discussion in Section 3.5 corresponds to the minimal configuration (with only one alternative strategy) of the general formula derived in this section.

As a concrete example, suppose a given stage i has, in addition to the original path ($p_i^{(1)} = 0.8$), two alternative paths ($p_i^{(2)} = 0.5$, $p_i^{(3)} = 0.5$) prepared as well ($n_i = 3$). The success probability of that stage is then $P_i = 1 - (1 - 0.8)(1 - 0.5)(1 - 0.5) = 1 - 0.05 = 0.95$. Applying this to a DAG with 5 such stages arranged in series, $P(\text{Goal})_{DAG} = 0.95^5 \approx 0.774$ (about 77.4%), confirming that an even higher completion probability is obtained compared with the case of only one alternative path (Section 3.5, about 59.0%).

When the entire DAG consists of m sequential stages, and each stage i has its own redundancy configuration (n_i alternative strategies), the goal completion probability P(Goal) can be generalized, as shown in Equation 10, as the product of the redundant success probabilities of each stage.

$$P(Goal)_{DAG} = \prod_{i=1}^{m} \left[1 - \prod_{k=1}^{n_i} \left(1 - p_i^{(k)}\right)\right] \qquad (10)$$

Furthermore, the framework of Equation 10 extends directly to branching DAGs in which multiple predecessor tasks converge (via an AND-join) into a single successor task. When

B parallel branches converge into a single successor task, and P_j denotes the completion probability of branch j (computed via Equation 9), the executability probability of the join node is expressed as shown in Equation 11.

$$P_{join} = \prod_{j=1}^{B} P_j \qquad (11)$$

This translates into probabilistic terms the executability condition defined in Equation 1 — that "all predecessor tasks Pre(v) have succeeded" — and serves to integrate the structural properties of the DAG (Sections 3.2 and 3.3) with the redundancy model of Equation 9. In Section 4.5, we validate this general formula through simulation on an actual parallel-branching-plus-AND-join structure.

3.7 Redundancy Allocation Problem under Cost Constraints

Equations 9 and 10 show that the completion probability P(Goal) improves as more redundant paths are added; in practice, however, executing each alternative strategy incurs costs such as the number of API calls, token consumption, and execution time. The practical design question therefore reduces to a redundancy allocation problem: "in which stage, and how many redundant paths, should be placed?" Letting c_i denote the expected cost of adding one redundant path to stage i, and C the total available cost budget, this problem can be formulated as shown in Equation 12.

$$\text{maximize} \quad P(Goal)_{DAG} \quad \text{subject to} \quad \sum_{i=1}^{m} c_i \, (n_i - 1) \leq C \qquad (12)$$

This optimization problem is, in general, a combinatorial optimization problem (a variant of the 0-1 knapsack problem), and it has been shown that computing an exact solution to the redundancy allocation problem in a series system is not practically tractable in general [22] (for the small numbers of stages typical of agent plans, however, exhaustive enumeration is feasible; see Section 4.9). On the other hand, it is known that a greedy heuristic is effective for resource-allocation problems whose objective function exhibits diminishing marginal utility with respect to each element [23]; the guideline adopted in this study — "preferentially add redundancy to whichever stage currently has the lowest success probability (i.e., the bottleneck)" — is an approximate application of that heuristic. This is because, as shown in Equation 13, the marginal improvement that adding a redundant path to stage i contributes to the overall completion probability becomes larger the lower that stage's original success probability p_i is.

$$\frac{\Delta P(Goal)}{P(Goal)} = \frac{(1 - p_i) \cdot p_i{}'}{p_i} \qquad (13)$$

Equation 13 holds with equality, not as an approximation: replacing the factor pi of stage i by pi + (1 − pi)p′i multiplies P(Goal) by exactly 1 + (1 − pi)p′i/pi in every structure in which the stage enters P(Goal) as one factor, i.e., in all structures covered by Equations 10 and 11. Note also that the lowest pi coincides with the largest gain of Equation 13 only when the p′i of the candidate stages are equal; Section 4.9 compares the two rules. In Section 4.6, we confirm via simulation that this greedy allocation strategy achieves a higher completion probability than allocating to a stage other than the bottleneck, under the same cost budget (only one additional redundant path allowed). As an intuitive example, consider a DAG consisting of three stages with different success probabilities (p_A = 0.50, p_B = 0.75, p_C = 0.90), where the redundant-path budget is limited to just one (the success probability of the added alternative path is taken as p′ = 0.5 in every case). Allocating it to the weakest stage, A (the bottleneck), raises Stage A's success probability from 0.50 to 0.75 (a 1.5× increase), which directly translates into a 1.5× improvement in the DAG's overall completion probability. By contrast, allocating the same alternative path to the highest-success stage, C, only raises Stage C's success probability from 0.90 to 0.95 (about a 1.06× increase), yielding only a negligible improvement overall. This difference stems precisely from the term (1 − p_i) in Equation 13, meaning that the lower a stage's p, the greater the "room for rescue."

3.8 Estimation Method for p and p'

The p_i and p'_i described above can, in actual operation, be estimated from the frequency of execution logs. For a given stage i, let N_i be the number of times the original path was executed and S_i the number of those that succeeded; then the success probability of the original path, p_i, is given by Equation 14.

$$p_i = \frac{S_i}{N_i} \qquad (14)$$

On the other hand, since the alternative path is executed only when the original path fails, the denominator of its success probability p'_i is not the "total number of times the original path was executed, N_i," but rather the "number of times the original path failed, F_i (= N_i − S_i)." Out of F_i such times, letting S'_i be the number of times the alternative path succeeded, p'_i is expressed as shown in Equation 15.

$$p_i' = \frac{S_i'}{F_i} \qquad (15)$$

That is, p'_i is a conditional probability, conditioned on the event that "the original path failed," and it is precisely this conditioning that underlies the additive form P(Goal) = p_i + (1 − p_i)·p'_i in Equation 7. If the denominator of p'_i were instead taken to be the total number of executions, N_i, then the failure event would be double-counted in the product with (1 − p_i), causing the completion probability to be underestimated.

As a concrete example, suppose the original path was executed 100 times and succeeded 80 times (p_i = 80/100 = 0.80), and for the 20 failures, the alternative path was executed and succeeded 12 times (p'_i = 12/20 = 0.60); then, by Equation 7, that stage's success probability is P = 0.80 + (1 − 0.80) × 0.60 = 0.92. However, when the number of trials is small, a simple frequency ratio can easily swing to an estimate of 0 or 1. The evaluation experiments described later therefore use smoothing via a Beta prior, estimating as shown in Equation 16.

$$p_i = \frac{S_i + \alpha}{N_i + \alpha + \beta} \qquad (16)$$

Here, α is the hypothetical number of times deemed to have "succeeded" before observation begins, and β is likewise the hypothetical number of times deemed to have "failed"; α/(α+β) represents the success probability assumed a priori, and α+β represents the strength of that prior information (equivalent to how many observations it corresponds to). For example, if the alternative path is executed twice and succeeds both times, a simple frequency ratio would flatly conclude p'_i = 1.00, but setting α = β = 1 gives p'_i = 3/4 = 0.75, avoiding overly confident conclusions from a small number of observations. In the early stage of actual operation, before logs have accumulated, one could conceivably give initial values of α and β based on a tool's specifications or an LLM's stated estimate drawn from past performance, and then overwrite them with observations as execution proceeds. In the simulation experiments of Sections 4.1–4.9, this estimation is not performed; instead, the assumed values shown in Table 1 of Section 4.1 (80% / 70% / 90%) are given as fixed settings. The estimation is applied to real execution logs in Section 4.10.

4 Evaluation Experiments

4.1 Outline of the Evaluation Experiment

The instruction (prompt) used for evaluation was "Retrieve and summarize an economic news article," a composite task including error recovery: "If useful information cannot be obtained from the main news site A, retrieve it from alternative news site B instead. Finally, summarize the content and report it." To keep the sampling error of the estimated rates small (the standard error of a rate near 80% is about 0.4 points at 10,000 trials and about 0.13 points at 100,000 trials), a Monte Carlo simulation [14] using the probability distributions of the uncertainties illustrated in Table 1 was run 10,000 times (100,000 times in Sections 4.3–4.9). Each measured rate is reported with its 95% confidence interval (normal approximation, ±1.96 standard errors); no hypothesis test of the difference between the models is implied by these intervals.

Table 1 Definition of uncertainties used in the Monte Carlo method

Variable	Definition
Success rate of the main site (e.g., Yahoo)	80%
Success rate of the alternative site (e.g., Google)	70%
Success rate of the summarization task (LLM stability)	90%

The definitions in Table 1 are assumed values set for this study based on real-world web-scraping and LLM behavior, and are not measured values estimated from actual operational logs; they are the same reference values as in the author's related work SH-GD/SM-L (Section 2.3) (the estimation method is described in Section 3.8, and the limitations in Section 4.11). Several other uncertainties besides these three are also conceivable; however, for the instruction used here ("retrieve and summarize an economic news article"), these three were selected as the main factors considered most directly suited to evaluating the advantage of the new model — dynamically switching routes based on execution results (DAG + state machine).

Achievement was defined as the state in which the agent successfully completes the final task (summarization/report generation) and outputs the final deliverable the user requested. Achievement consists of two categories: Success, the number of successes via the main route, and Saved by Alt-Path, the number of cases where the main route failed but the alternative path succeeded.

4.2 Results of the Evaluation Experiment

The results of the evaluation experiment are shown in Fig. 1. Compared with the conventional linear model's achievement rate of 72.7% (95% CI ±0.9 points), the proposed model achieved 84.4% (±0.7 points) when Success and Saved by Alt-Path are summed. A complex human instruction has no single correct route; it requires dynamic judgments during execution, such as "if A then B, if C then D." However, the conventional linear model cannot interpret such dynamic "if…then" branching at the code level, so, for example, the moment site A has no information, the data passed to the subsequent summarization task becomes empty and the mission is deemed a failure. The proposed model (DAG + state machine), in contrast, autonomously transitions to "retrieving from site B" the instant it detects "the failure of site A," making it possible to avoid the error and reach the final goal. This is the "1,277 cases succeeded via the alternative path (Saved by Alt-Path)" and suggests that the model can respond to the dynamic judgments described above more flexibly than the conventional model.

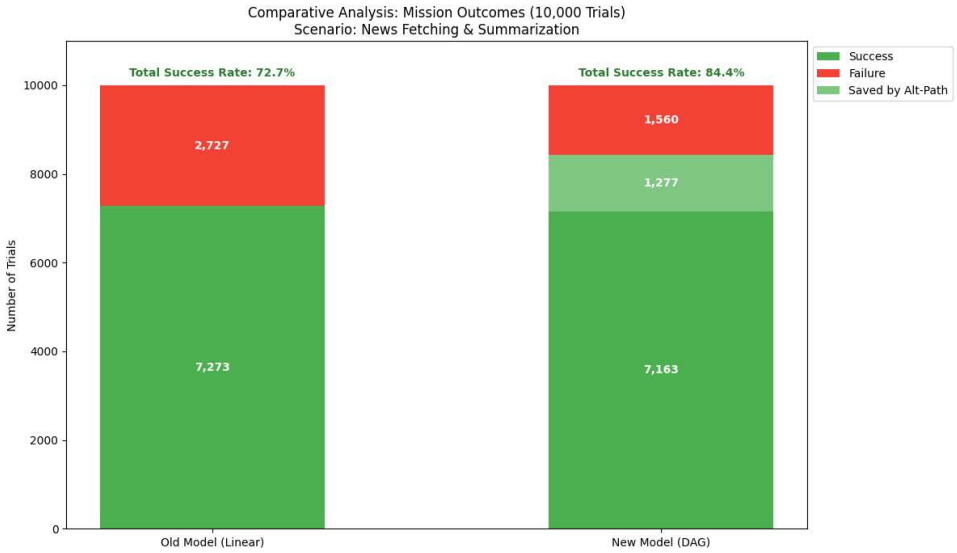


Fig. 1 Results of the evaluation experiment. Comparative outcome counts (10,000 trials) for the conventional linear model (Old Model) and the proposed DAG + state-machine model (New Model), broken down into Success, Saved by Alt-Path, and Failure

Note that, in this simulation, the proposed model results in Failure only when both retrieval paths fail (the rescue term is exhausted) or when the summarization task fails; failures of planning itself are not part of the simulation. In real operation, Failure can also arise when planning (DAG construction) fails, that is, when the user's instruction is so ambiguous that the LLM cannot extract tasks, or when the LLM generates a graph containing a circular reference (A depends on B, and B depends on A); in such cases the process either errors out before execution begins or terminates without finding a single executable task. Feeding such cases back to a human (or using them for re-planning) is considered to be the next step toward greater autonomy.

The Success count of the New Model (DAG) (7,163) is lower than that of the Old Model (Linear) (7,273). This difference is not caused by planning failures, which are not modeled (Table 1 contains no planning-failure probability). In both models the primary route succeeds with the same probability $p \cdot q = 0.80 \times 0.90 = 0.72$, so both counts have expectation 7,200 and standard deviation $\sqrt{(10{,}000 \times 0.72 \times 0.28)} \approx 44.9$. The difference of 110 cases is about 1.7 standard deviations of the difference between two independent counts (≈ 63.5; two-sided $p = 0.08$ in a two-proportion z test) and is therefore explained by random sampling variation; in a re-run with the author's code and a fixed seed, the corresponding counts were 7,207 and 7,184. The gain of the proposed model comes entirely from the Saved by Alt-Path cases.

4.3 Comparison between Theoretical Values and Simulated Measurements

To verify how well the theoretical formulas of Section 3.5 predict measured values, we again ran a Monte Carlo simulation of 100,000 trials under the same settings as Section

4.2. Table 2 compares those measured values, with their 95% confidence intervals, with the theoretical values from Equations 6 and 7.

Table 2 Comparison between theoretical values and Monte Carlo measured values

Model	Theoretical P(Goal)	Measured, 100,000 trials (95% CI)	Error
Conventional model (linear)	72.00%	71.95% (±0.28)	0.05 pt
Proposed model (DAG + redundancy)	84.60%	84.70% (±0.22)	0.10 pt

The errors are 0.10 points or less and lie inside the 95% confidence intervals (±0.28 and ±0.22 points), so the remaining differences are within sampling error. Under the assumption that task outcomes are independent, Equations 6 and 7 therefore reproduce the simulated completion rates. This agreement is expected by construction, because the simulator draws each outcome from the same independent probabilities that the formulas multiply; it verifies the derivation and its implementation, not the behavior of real agents, which is examined in Sections 4.7, 4.8 and 4.10.

4.4 Systematic Validation via Parameter Sweep

To verify whether the general formula of Equation 9 holds not only under a single condition but across a wide parameter range, we ran a Monte Carlo simulation of 100,000 trials each for all 24 combinations of the primary strategy's success probability $p \in \{0.5, 0.6, 0.7, 0.8, 0.9, 0.95\}$ and the alternative strategy's success probability $p' \in \{0.3, 0.5, 0.7, 0.9\}$ (Table 3).

Table 3 Error statistics between theoretical and measured values across 24 parameter-sweep combinations

Statistic	Value
Maximum error	0.18 pt
Mean error	0.07 pt
Standard deviation of error	0.05 pt
Mean |error| expected from sampling alone	0.07 pt
Range of 95% CI half-widths of the 24 measured rates	±0.04 to ±0.30 pt

The mean error (0.07 points) equals the mean absolute error expected from binomial sampling alone at 100,000 trials (0.07 points), so no systematic deviation remains. Furthermore, Table 4 shows the progression of theoretical and measured values for

representative values of n, with the number of serial stages n varied from 1 to 10 (p = 0.8, p′ = 0.6 fixed).

Table 4 Theoretical and measured values with a varying number of serial stages n

n	Conventional: theoretical	Conventional: measured (95% CI)	Proposed: theoretical	Proposed: measured (95% CI)
1	80.00%	80.06% (±0.25)	92.00%	92.12% (±0.17)
2	64.00%	64.28% (±0.30)	84.64%	84.80% (±0.22)
3	51.20%	51.21% (±0.31)	77.87%	77.75% (±0.26)
4	40.96%	41.03% (±0.30)	71.64%	71.50% (±0.28)
5	32.77%	32.53% (±0.29)	65.91%	65.89% (±0.29)
7	20.97%	20.92% (±0.25)	55.78%	55.66% (±0.31)
10	10.74%	10.65% (±0.19)	43.44%	43.47% (±0.31)

As n increases, the conventional model's success probability converges exponentially toward 0, whereas the proposed model shows only gradual decay; for every value of n, the error between theoretical and measured values was at most 0.28 points (conventional model, n = 2), below 1 point and inside the 95% confidence interval in every case. Under the independence assumption, Equation 10 therefore reproduces the simulation for every number of stages tested (n = 1–10), not just for a single case.

4.5 Validation on a General DAG Structure (Parallel Branching + AND-Join)

To verify whether Equations 10 and 11 apply not only to a simple serial structure but also to a general DAG structure that includes parallel branching and AND-joins, we ran a simulation on a structure in which three parallel branches (representing data-collection tasks, with primary/secondary success probabilities of 0.8/0.6, 0.7/0.5, and 0.9/0.4, respectively) converge into a single composite task (success probability q = 0.85) (Table 5). The main-route probabilities 0.80/0.70/0.90 and q = 0.85 are the same reference scenario as in SH-GD/SM-L (Section 2.3).

Table 5 Theoretical and measured values for a 3-parallel-branch + AND-join DAG

Configuration	Theoretical value	Measured, 100,000 trials (95% CI)	Error
No redundancy (main route of each branch only)	42.84%	42.85% (±0.31)	0.01 pt

Configuration	Theoretical value	Measured, 100,000 trials (95% CI)	Error
Proposed model (each branch made redundant via Equation 9)	62.48%	62.59% (±0.30)	0.11 pt

Through redundancy, the completion probability improved from 42.8% to 62.5%, an improvement of about 20 points (45.8% in relative terms), and in every configuration the error between theoretical and measured values stayed within 0.11 points, inside the 95% confidence intervals. This result shows that, under the independence assumption, Equations 10 and 11 reproduce the simulation also for a general DAG structure that includes parallel branching and AND-joins, not only for the simple serial model of Section 3.5.

4.6 Redundancy Allocation Experiment under Cost Constraints

To verify the theory derived in Section 3.7 — that "preferentially allocating to the bottleneck is most effective" — we compared the overall success probability for different allocation targets under a condition in which only one additional redundant path was allowed, using a 3-branch configuration with the same main-route success probabilities as Section 4.5 (stock-price data retrieval A: p = 0.80; economic-indicator data retrieval B: p = 0.70; related news article retrieval C: p = 0.90; alternative-strategy success probability uniformly $p' = 0.6$; composite task q = 0.85) (Table 6). Note that, unlike the intuitive example of Section 3.7 ($p_A = 0.50$, $p_B = 0.75$, $p_C = 0.90$), this section uses the same success probabilities as the experiment in Section 4.5 (the reference scenario shared with SH-GD/SM-L). Each configuration was also simulated with 100,000 trials.

Table 6 Overall success probability for different redundant-path placements (budget = 1 path)

Placement	Overall success probability (theory)	Improvement	Measured, 100,000 trials (95% CI)
No redundant path	42.84%	—	43.11% (±0.31)
Placed on stock-price data retrieval A (p = 0.80)	49.27%	+6.43 pt	49.40% (±0.31)
Placed on economic-indicator data retrieval B (p = 0.70, bottleneck)	53.86%	+11.02 pt	53.70% (±0.31)
Placed on related news article retrieval C (p = 0.90)	45.70%	+2.86 pt	45.62% (±0.31)

The improvement from placing the redundant path on the bottleneck stage with the lowest success probability (economic-indicator data retrieval B, p = 0.70) (+11.02 pt) reached about 3.9 times that from placing it on the stage with the highest success probability (related news article retrieval C, p = 0.90) (+2.86 pt). The simulated values agree with the theoretical values within their 95% confidence intervals. The ratio is consistent with Equation 13 (relative gains of 25.7%, 15.0% and 6.7% for B, A and C), which supports the design guideline of “preferentially reinforcing the weakest stage” under a limited budget when the alternatives are of equal quality and cost; Section 4.9 examines the case in which they are not.

4.7 Comparison with Retries

A simpler way to make a linear agent more reliable is to retry a failed task. To separate the contribution of the DAG alternative path from that of retrying, we compared, for the task of Section 4.1, a linear agent with k = 0, 1 or 2 retries of every task, the DAG with an alternative path and no retries, and the DAG with an alternative path and one retry of every task. Retries of the same task are not necessarily independent: when the main site does not carry the article, retrying the same site fails again. The success probability of repeated attempts of the same task was therefore drawn from a Beta distribution with mean p and intra-class correlation ρr (a beta-binomial model), so that ρr = 0 corresponds to independent retries and ρr → 1 to persistent failures; the probability that a task with k retries succeeds, 1 − E[(1 − θ)^(k+1)], then has a closed form. Table 7 reports the theoretical value, the Monte Carlo measurement (100,000 trials) with its 95% confidence interval, and the mean number of task executions per run as a cost measure.

Table 7 Completion probability with retries and with an alternative path (theory / measured (95% CI), %)

Configuration	ρr = 0	ρr = 0.5	ρr = 0.9	Mean executions (ρr = 0 / 0.9)
Linear, no retry	72.00 / 72.07 (±0.28)	72.00 / 72.00 (±0.28)	72.00 / 71.92 (±0.28)	1.80 / 1.80
Linear + 1 retry	95.04 / 95.08 (±0.13)	83.16 / 83.29 (±0.23)	74.17 / 74.15 (±0.27)	2.26 / 2.10
Linear + 2 retries	99.10 / 99.11 (±0.06)	87.69 / 87.67 (±0.20)	75.23 / 74.99 (±0.27)	2.34 / 2.37

Configuration	ρr = 0	ρr = 0.5	ρr = 0.9	Mean executions (ρr = 0 / 0.9)
DAG alternative path, no retry	84.60 / 84.85 (±0.22)	84.60 / 84.64 (±0.22)	84.60 / 84.56 (±0.22)	2.14 / 2.14
DAG alternative path + 1 retry	98.64 / 98.62 (±0.07)	92.29 / 92.38 (±0.16)	86.23 / 86.36 (±0.21)	2.35 / 2.48

When retries are independent (ρr = 0), one retry of every task (95.0%) outperforms the alternative path alone (84.6%) at a similar cost (2.26 vs. 2.14 executions per run); the alternative path is therefore not superior per se. When failures are persistent (ρr = 0.9), retries add little (74.2% with one retry, 75.2% with two), whereas the alternative path, whose failures are independent of those of the main site in this setting, keeps 84.6%. Combining both gives the highest rate in every case. The contribution of the alternative path is thus largest exactly when retrying the same route cannot help. In terms of Equation 9, a retry is simply a further strategy that is correlated with the first attempt, so both mechanisms are covered by the same framework once this correlation is modeled. In one of the 15 configurations (DAG, ρr = 0) the error (0.25 points) slightly exceeded the 95% confidence interval (±0.22), as is expected for about one in twenty intervals.

4.8 Sensitivity to Correlated Failures

Section 4.11 notes that the main and alternative sites may fail together, for example when both depend on the same CDN or API provider. To quantify this, we kept the marginal success probabilities p and p′ fixed, as they would be if each source were benchmarked on its own, and introduced a Pearson correlation ρ between the failure indicators of the primary and the alternative path. The stage success probability then becomes $P_i = p_i + (1 - p_i)p'_i - \rho\sqrt{(p_i(1 - p_i)p'_i(1 - p'_i))}$, which reduces to Equation 7 at ρ = 0; ρ is bounded by the marginals (ρ ≤ 0.764 for p = 0.8 and p′ = 0.7). Table 8 shows the completion probability of the task of Section 4.1 (q = 0.9) as ρ increases.

Table 8 Completion probability under correlated primary/alternative failures (task of Section 4.1)

ρ	Theory	Measured, 100,000 trials (95% CI)	Overestimate by Eq. 7 (pt)	Conditional p′ (Eq. 15)
0.0	84.60%	84.43% (±0.22)	0.00	0.700
0.1	82.95%	82.97% (±0.23)	1.65	0.608
0.2	81.30%	81.30% (±0.24)	3.30	0.517

ρ	Theory	Measured, 100,000 trials (95% CI)	Overestimate by Eq. 7 (pt)	Conditional p′ (Eq. 15)
0.3	79.65%	79.75% (±0.25)	4.95	0.425
0.4	78.00%	78.00% (±0.26)	6.60	0.333
0.5	76.35%	76.43% (±0.26)	8.25	0.242
0.6	74.70%	74.55% (±0.27)	9.90	0.150
0.7	73.05%	73.05% (±0.27)	11.55	0.058

Each increase of ρ by 0.1 removes 1.65 points; at ρ = 0.7 the alternative path adds only 1.05 points to the linear model (72.00%), and Equation 7 with the unconditional p′ overestimates the completion probability by 11.6 points. For the AND-join DAG of Section 4.5 (where ρ ≤ 0.27 because of branch C), the completion probability falls from 62.48% at ρ = 0 to 58.57%, 54.83% and 52.30% at ρ = 0.1, 0.2 and 0.27 (measured 62.64%, 58.54%, 54.79% and 52.53%), still above the 42.84% without redundancy, while the independence model overestimates by up to 10.2 points. If, however, p′ is estimated as in Section 3.8, that is, conditionally on the failure of the primary path (Equation 15), the correlation is already contained in the estimate (last column of Table 8), and Equation 7 remains exact. The overestimation arises only when p′ is taken from an unconditional measurement of the alternative path alone, which is a further reason to estimate p′ with Equation 15. Correlation among three or more strategies, and between stages, still requires a common-cause model such as the beta-factor method.

4.9 Bottleneck-First Greedy Allocation versus the Exact Optimum

Section 3.7 adopted a bottleneck-first greedy rule for the allocation problem of Equation 12. To measure its distance from the optimum, we generated random serial DAGs with m = 3, 4 and 5 stages (pi ~ U(0.5, 0.95); every alternative path added to stage i has success probability p′i ~ U(0.3, 0.8); at most three alternatives per stage) and computed the exact optimum by exhaustive enumeration (at most 4^5 = 1,024 allocations). Two cost settings were used: uniform costs (ci = 1, budget C ∈ {2, 3}) and non-uniform costs (ci ∈ {1, …, 5}, C ∈ {3, …, 10}), with 1,000 random instances per setting (instances in which no path is affordable were excluded). We compared (a) the bottleneck-first rule, which adds a path to the affordable stage whose current success probability is lowest, and (b) a greedy rule that adds the path with the largest gain of Equation 13 per unit cost, ln(1 + ΔP/P)/ci. Table 9 shows the ratio of the completion probability reached to the exact optimum.

Table 9 Greedy allocation versus the exact optimum (ratio to the optimum: mean / worst case / share of instances solved optimally)

Costs	m	Instances	(a) Bottleneck-first	(b) Eq. 13 gain per cost
Uniform	3	1,000	99.40% / 83.98% / 75.3%	100.00% / 100.00% / 100.0%
Uniform	4	1,000	98.88% / 82.75% / 67.4%	100.00% / 100.00% / 100.0%
Uniform	5	1,000	98.21% / 78.63% / 63.6%	100.00% / 100.00% / 100.0%
Non-uniform	3	992	98.84% / 75.25% / 77.3%	99.29% / 75.70% / 80.8%
Non-uniform	4	997	98.06% / 70.76% / 69.2%	99.10% / 71.68% / 78.6%
Non-uniform	5	997	96.79% / 50.91% / 61.8%	99.15% / 74.21% / 80.2%

With uniform costs, the greedy rule based on Equation 13 reached the exact optimum in all 3,000 instances, consistent with the diminishing marginal returns of each stage [23]. The simpler bottleneck-first rule reached 98.2–99.4% of the optimum on average but solved only 63.6–75.3% of the instances exactly, because the stage with the lowest pi is not necessarily the stage with the largest (1 − pi)p′i/pi when the p′i differ. With non-uniform costs, neither rule is always optimal: the cost-weighted rule reached 99.1–99.3% of the optimum on average (worst case 71.7%) and the bottleneck-first rule 96.8–98.8% (worst case 50.9%), the gap widening with m. The bottleneck-first guideline is therefore a good rule of thumb when the alternative paths are of similar quality and cost; otherwise Equation 13 divided by the cost should be used, and for the small DAGs typical of agent plans ($m \leq 5$) the exact optimum can simply be computed by enumeration.

4.10 Validation with a Real LLM Agent

Sections 4.2–4.9 share a circularity: the simulator draws each outcome from the same probabilities that the model multiplies. To break it, the task of Section 4.1 was executed by a real LLM agent, and p, p′ and q were estimated from its execution log with the method of Section 3.8. Task A selects an article on a given economic topic from a main news feed (the NHK economy RSS feed) and judges its relevance; if A fails, task B does the same with a Google News search feed for the topic (the on_failure edge); task C summarizes the selected article as JSON (50–200 characters) and is accepted only if the JSON is valid, the length constraint is met and a separate LLM call judges the summary faithful to the article. All success criteria are machine-checked, and the State-Machine engine of Section 3.4 executes the graph. The agent was run 100 times, each run on a topic drawn at random (with replacement, fixed seed) from 20 Japanese economic topics (2–11 runs

per topic). p, p′ and q were estimated from the first 50 runs with Equations 14–16 (α = β = 1), and the completion rates of the linear agent (A → C) and of the DAG agent predicted by Equations 6 and 7 were compared with the rates observed in the remaining 50 runs (hold-out). The linear baseline is read off the same runs: it succeeds exactly when A succeeds and C succeeds on A's article. The LLM was NVIDIA Nemotron-Nano-9B-v2-Japanese (8-bit GGUF quantization, Q8_0), served locally by llama.cpp on a CPU with its reasoning mode disabled, temperature 0.7 and at most 512 output tokens; the same model performed the faithfulness check. The 100 runs were made on 29 October 2026 and took about 8.5 hours. The LLM is the object of this evaluation, not a writing aid.

Table 10 Real LLM agent: estimated probabilities, and predicted versus observed completion rates (observed rates with Wilson 95% confidence intervals)

Quantity	Runs 1–50 (estimation)	Runs 51–100 (hold-out)	All 100 runs	Remark
p: task A, main feed (Eq. 14)	0.135 (6/50)	0.269 (13/50)	0.196 (19/100)	Posterior mean, Beta(1, 1)
p′: task B, alternative feed (Eq. 15)	0.935 (42/44)	0.923 (35/37)	0.940 (77/81)	Conditional on failure of A
q: task C, summarization (Eq. 16)	0.720 (35/48)	0.640 (31/48)	0.684 (66/96)	Runs that reached C
Linear agent: predicted / observed	–	9.7% / 16.0% [8.3, 28.5]	13.4% / 11.0% [6.3, 18.6]	Prediction from runs 1–50 (hold-out) or all runs (in-sample)
DAG agent: predicted / observed	–	67.9% / 62.0% [48.2, 74.1]	65.1% / 66.0% [56.3, 74.5]	Prediction from runs 1–50 (hold-out) or all runs (in-sample)

The probabilities of the real agent differ markedly from the reference scenario of Section 4.1. The main feed was the bottleneck: task A succeeded in only 19 of 100 runs, because the economy feed carries a limited number of recent articles, most of which do not concern the given topic (in 47 runs the model found no relevant article, and in 34 runs the article it selected did not mention the topic). The topic-specific alternative feed succeeded in 77 of the 81 runs in which it was invoked. Summarization succeeded in 66 of 96 runs; the 30 failures were 19 outputs that were not valid JSON (in 16 of them only the closing brace was missing) and 11 summaries shorter than 50 characters, while no summary was

judged unfaithful. The success criteria were fixed before the runs and were not relaxed afterwards.

On the hold-out half, the predicted completion rates of both agents lay within the 95% confidence intervals of the observed rates: 67.9% predicted versus 62.0% observed (48.2–74.1%) for the DAG agent, and 9.7% versus 16.0% (8.3–28.5%) for the linear agent. Most of the prediction error comes from the main feed itself, whose success rate rose from 0.135 in the first half to 0.269 in the second as the feed contents changed during the experiment, so p was not stationary over the 8.5 hours of the experiment. Over all 100 runs, Equations 6 and 7 with the estimated probabilities give 65.1% and 13.4% against observed 66.0% and 11.0%. In this regime of a weak primary path and a strong alternative, the alternative path raised the observed completion rate from 11.0% to 66.0% (a factor of 6.0), compared with a factor of 4.86 predicted by Equation 13 with the estimated probabilities (relative gain $(1 - p)p'/p = 3.86$); the difference reflects the lower summarization success on main-feed articles (57.9% vs. 71.4%, not significant; see below). As a check of the independence assumed between stages, the success rate of summarization did not differ significantly between articles from the main feed (11/19 = 57.9%, 95% CI 36.3–76.9%) and from the alternative feed (55/77 = 71.4%, 60.5–80.3%; two-proportion z test, $p = 0.25$).

This experiment has limited power. With 50 hold-out runs the confidence intervals are about ±10 to ±13 points wide, so it can reveal only gross deviations of the model, and the non-significant difference in q between routes does not establish independence. It covers one task, one model and one judge (the same model), and the main-path probability drifted during the experiment. It nevertheless shows that the quantities of the model can be estimated from execution logs of a real agent with the method of Section 3.8, and that the resulting closed-form predictions are consistent with the observed completion rates.

4.11 Limitations of This Evaluation and Future Work

First, the success probabilities p and p′ used in Sections 4.1–4.9 are assumed values set for this study (80% / 70% / 90%), not measured values from actual operational logs. Section 4.10 estimates them from the logs of a real agent, but for a single task, a single model (which also served as the faithfulness judge) and 100 runs, so that the hold-out comparison has 95% confidence intervals of about ±10 to ±13 points and can reveal only gross deviations; moreover, the success probability of the main path drifted during the experiment. Validation on other tasks, other models, independent judges and longer operation remains future work.

Second, the probability models of Equations 9–13 assume that the outcomes of the alternative strategies are statistically independent. Section 4.8 quantified the effect of a correlation between one primary and one alternative path and showed that the conditional estimator of Equation 15 absorbs it; common-cause failures that affect three or more

strategies or several stages at once (for example, when several sites depend on the same CDN or API provider) are not covered, and correction methods such as the beta-factor method are left for future work.

Third, this experiment does not include a direct comparison with existing production-oriented orchestration frameworks such as Claude Agent SDK or LangGraph. While the mathematical models of Equations 9–13 themselves are a general contribution not tied to any particular implementation, comparison on non-probabilistic metrics such as implementation cost and execution speed is left for future work.

5 Conclusion and Future Work

The novelty of this study lies, first, in integrating a probabilistic model of task completion into the existing DAG- and State-Machine-based execution control of AI agents (Section 2.3), so that the completion probability of non-linear workflows involving conditional branching, parallel execution and alternative paths can be computed; second, in generalizing the task completion probability P(Goal) into closed-form equations (Equations 9–11) by drawing on knowledge from reliability engineering (Billinton and Allan), in contrast to existing frameworks such as LangGraph and the LLM Compiler that remain limited to execution-control engineering, and theoretically formulating this all the way through to the redundant-path allocation problem (Equations 12 and 13); and third, in verifying those closed-form expressions against Monte Carlo simulation at scales of 10,000 to 100,000 trials: the error between theoretical and measured values was at most 0.18 points across the 24 parameter-sweep combinations of Table 3 and below 1 point (at most 0.28 points) under all conditions, within the sampling error of the simulation. What this establishes is that, under the assumption that task outcomes are independent, the analytical model and the numerical simulation agree, so that the completion probability of a plan can be computed in closed form instead of by simulation; it does not by itself show that real AI agents complete more tasks. Sections 4.7–4.10 examine how far the conclusions carry over when retries are available, when failures are correlated, when costs are non-uniform, and when the probabilities are estimated from a real agent.

Regarding effectiveness, in the simulated scenario of Section 4.1 the proposed model reached an achievement rate of 84.4% (Success: 7,163 cases + Saved by Alt-Path: 1,277 cases), compared with 72.7% for the conventional linear model. Furthermore, through the parameter sweep (Section 4.4), the parallel-branching-plus-AND-join structure (Section 4.5), and the redundancy allocation experiment (Section 4.6), we showed that the closed-form model reproduces the simulation not only under a single evaluation condition but across a wide parameter range and general DAG structures. In particular, we demonstrated that, under cost constraints, preferentially placing a redundant path on the bottleneck stage is about 3.9 times as effective as placing it on the most reliable stage (Section 4.6), in line with Section 3.7; Section 4.9 showed that this rule reaches 96.8–

99.4% of the exact optimum on average, and that a cost-weighted version of Equation 13 reaches the optimum in all instances with uniform costs. Section 4.7 further showed that an alternative path pays off most when failures of the primary route are persistent, whereas independent failures are handled equally well by retries, and Section 4.8 showed how much correlated failures reduce the gain. Finally, with a real LLM agent (Section 4.10), completion rates predicted from probabilities estimated on half of 100 runs fell within the 95% confidence intervals of the rates observed on the other half (DAG agent: 67.9% predicted, 62.0% observed), and the alternative path raised the observed completion rate from 11.0% to 66.0%, although the small sample allows only a coarse test.

On the other hand, as noted in Section 4.11, the real-agent validation (Section 4.10) is limited to one task and one model, common-cause failures across three or more strategies or several stages remain unmodeled, and a comparison of implementation cost with existing frameworks remains unverified. Building a mechanism to feed back cases where the proposed model results in Failure (planning failures or the occurrence of circular references) to a human, or to use them for re-planning, is also an important challenge for improving the autonomy of AI agents going forward. Through these further validations, we hope to establish, in a more convincing way, the effectiveness of task-generation models based on DAG and State-Machine in real-world operation.

Declarations

Funding: This research was funded by the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan through the Supporting Pioneering Research through AI for 1,000 Discovery Challenges Program (SPReAD), grant number JPMXP1726256770.

Competing interests. The author declares no competing interests.

Ethics approval and consent to participate. Not applicable. This study did not involve human participants, human data, or animal subjects; it is a mathematical modeling, Monte Carlo simulation and software-agent study.

Consent for publication. Not applicable.

Data availability. No secondary dataset was used in this study. The data of the Monte Carlo simulations (Sections 4.1–4.9) were generated by the author's code from the parameter values stated in the manuscript. The real-agent experiment (Section 4.10) processed news articles retrieved at run time from two publicly accessible RSS feeds, which can be opened without registration: the NHK economy news feed, https://www3.nhk.or.jp/rss/news/cat5.xml, and the Google News RSS search feed queried with each of the 20 economic topics of the experiment; for example, the feed for the topic "prices" is

https://news.google.com/rss/search?q=%E7%89%A9%E4%BE%A1&hl=ja&gl=JP&ceid=JP:ja (the other topics are obtained by replacing the q parameter). These articles are copyrighted by their publishers and change over time, so they are not redistributed. The simulation code, the simulation outputs and the execution logs of the real-agent experiment generated during the current study are available from the corresponding author on reasonable request.

Code availability. The reference implementation (dag_state_machine_model.py), the script that generates every number of Sections 4.2–4.9 (experiments.py) and the real-agent harness (real_agent_experiment.py) are available from the corresponding author on reasonable request.

Use of generative AI. The research question, the completion-probability model (Equations 1–16), the design of the evaluation scenario and the original manuscript are the author's work. In the present revision, the author decided which points to address, the additional experiments to run (comparison with retries, correlated failures, exact allocation, real-agent validation) and the conclusions to draw from them. Under the author's direction, Claude Opus 5.5 (Anthropic), used through Claude Code, assisted by (i) drafting the revised English text, (ii) implementing the experiment scripts specified by the author, and (iii) executing these scripts. All numbers in the tables are generated by the author's scripts, and all AI-assisted text and code were reviewed, verified and approved by the author, who takes full responsibility for the content. The NVIDIA Nemotron-Nano-9B-v2-Japanese model in Section 4.10 is the object of the real-agent experiment and was not used to write or edit the manuscript; likewise, gpt-oss-120b (Section 3.2) belongs to the implementation example only. In preparing the original manuscript, Claude Opus 5.5 (Anthropic) was used only to assist with the evaluation experiments and to translate the author's Japanese text into English; the author reviewed and verified all of its output.

Author contributions. A.O. conceived the study, developed the mathematical model, designed and implemented the simulations, analyzed the results, and wrote the manuscript. The author read and approved the final manuscript.

Clinical trial number: not applicable.

References

[1] Futran Solutions. Choosing the right AI use cases: linking AI to decisions that matter [Internet]. 2026 [cited 2026 Mar 10]. Available from: https://futransolutions.com/blog/choosing-the-right-ai-use-cases-linking-ai-to-decisions-that-matter-2026/

[2] Lu C, Lu C, Lange RT, Foerster J, Clune J, Ha D. The AI Scientist: towards fully automated open-ended scientific discovery. arXiv:2408.06292 [Preprint]. 2024. Available from: https://doi.org/10.48550/arXiv.2408.06292

[3] Yamada Y, Lange RT, Lu C, Hu S, Lu C, Foerster J, Clune J, Ha D. The AI Scientist-v2: workshop-level automated scientific discovery via agentic tree search. arXiv:2504.08066 [Preprint]. 2025. Available from: https://doi.org/10.48550/arXiv.2504.08066

[4] AutoGPT [Internet]. Significant Gravitas; [cited 2026 Mar]. Available from: https://github.com/Significant-Gravitas/AutoGPT

[5] BabyAGI [Internet]. Nakajima Y; [cited 2026 Mar]. Available from: https://github.com/yoheinakajima/babyagi

[6] MetaGPT [Internet]. Geekan; [cited 2026 Mar]. Available from: https://github.com/geekan/MetaGPT

[7] LangGraph. arXiv:2605.14612 [Preprint]. 2026 [cited 2026 Aug]. Available from: https://doi.org/10.48550/arXiv.2605.14612

[8] LLM Compiler. In: Agent design patterns [Internet]. [cited 2026 Aug]. Available from: https://agent-patterns.readthedocs.io/en/stable/patterns/llm-compiler.html

[9] Anthropic. Agent SDK overview [Internet]. Claude Agent SDK documentation; [cited 2026 Sep]. Available from: https://code.claude.com/docs/en/agent-sdk/overview

[10] Sarker Y, Ullah MR, Molla M, Joty S. GraphBit: a graph-based agentic framework for non-linear agent orchestration. arXiv:2605.13848 [Preprint]. 2026. Available from: https://doi.org/10.48550/arXiv.2605.13848

[11] Bang-Jensen J, Gutin GZ. Digraphs: theory, algorithms and applications. 2nd ed. London: Springer; 2009. (Springer Monographs in Mathematics).

[12] Hopcroft JE, Motwani R, Ullman JD. Introduction to automata theory, languages, and computation. 3rd ed. Boston: Addison-Wesley; 2006. ISBN 0-321-45536-3.

[13] Billinton R, Allan RN. Reliability evaluation of engineering systems: concepts and techniques. 2nd ed. New York: Plenum Press; 1992. Available from: https://link.springer.com/book/10.1007/978-1-4615-7728-7

[14] Fishman GS. Monte Carlo: concepts, algorithms, and applications. New York: Springer-Verlag; 1996. Available from: https://link.springer.com/book/10.1007/978-1-4757-2553-7

[15] Google. Deep Research Max: a step change for autonomous research agents [Internet]. The Keyword (Google blog); 2026 Apr 21 [cited 2026 Oct 29]. Available from: https://blog.google/innovation-and-ai/models-and-research/gemini-models/next-generation-gemini-deep-research/

[16] Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K, Cao Y. ReAct: synergizing reasoning and acting in language models. In: Proceedings of the International Conference on Learning Representations (ICLR 2023); 2023. Available from: https://doi.org/10.48550/arXiv.2210.03629

[17] Madaan A, Tandon N, Gupta P, Hallinan S, Gao L, Wiegreffe S, et al. Self-Refine: iterative refinement with self-feedback. In: Advances in Neural Information Processing Systems 36 (NeurIPS 2023); 2023. Available from: https://doi.org/10.48550/arXiv.2303.17651

[18] Wei J, Wang X, Schuurmans D, Bosma M, Ichter B, Xia F, Chi EH, Le QV, Zhou D. Chain-of-thought prompting elicits reasoning in large language models. In: Advances in Neural Information Processing Systems 35 (NeurIPS 2022); 2022. Available from: https://doi.org/10.48550/arXiv.2201.11903

[19] Anthropic. How we built our multi-agent research system [Internet]. 2025 [cited 2026 Sep 3]. Available from: https://www.anthropic.com/engineering/multi-agent-research-system

[20] Zhang S, Yuan C, Guo R, Yu X, Xu R, Chen Z, Li Z, Yang Z, Guan S, Tang Z, Hu S, Zhang L, Chen R, Wang H. EvoFSM: controllable self-evolution for deep research with finite state machines. arXiv:2601.09465 [Preprint]. 2026. Available from: https://doi.org/10.48550/arXiv.2601.09465

[21] gpt-oss-120b-MXFP4-Q8 [Internet]. Hugging Face; [cited 2026]. Available from: https://huggingface.co/mlx-community/gpt-oss-120b-MXFP4-Q8

[22] Chern MS. On the computational complexity of reliability redundancy allocation in a series system. Oper Res Lett. 1992;11(5):309–315.

[23] Ibaraki T, Katoh N. Resource allocation problems: algorithmic approaches. Cambridge (MA): MIT Press; 1988. (Foundations of Computing Series).