Enabling Recovery and Adaptation in LangGraph via Bayesian Control

Angshul Majumdar

Abstract

Execution frameworks such as LangGraph represent agent workflows as deterministic control-flow graphs over language model calls and external tools. While this abstraction enables modular orchestration, routing decisions are typically fixed by explicit conditionals. Once a branch is taken, alternative execution paths are discarded unless manually reintroduced through handcrafted recovery logic.

We propose probabilistic and Bayesian extensions of LangGraph-style execution. In the probabilistic formulation, routing is defined as sampling from a state-dependent distribution over candidate nodes rather than selecting a single deterministic branch. Log-probabilities are computed from signals such as model confidence, evidence strength, or structured metadata, allowing uncertainty to be represented explicitly at the control layer. Deterministic routing arises as a degenerate zero-entropy special case.

In the Bayesian formulation, tool reliability is modeled using conjugate Beta priors updated from observed success indicators. Posterior expectations are incorporated into routing distributions, enabling adaptive reallocation of probability mass toward consistently reliable tools across sessions.

We provide a formal analysis establishing strict expressivity separation between deterministic and probabilistic routing, conditions under which stochastic routing improves expected correctness, and asymptotic optimality of posterior-mean-based tool selection. Controlled demonstrations illustrate recovery from suboptimal tool choices and evidence-dependent fallback behavior. The framework remains fully compatible with existing LangGraph infrastructure while extending it with principled stochastic and Bayesian control.

1. Introduction

Agentic execution frameworks such as LangChain and LangGraph have popularized the construction of tool-augmented large language model (LLM) systems using explicit control-flow graphs. In these systems, nodes represent LLM calls or external tools, and edges encode deterministic routing logic. While this paradigm enables composability and modularity, routing decisions are typically hard-coded or rule-based. Once a branch is taken, the system does not maintain uncertainty over alternative execution paths.

Recent work on language model agents highlights the importance of structured reasoning and tool use [1], as well as the role of external search and retrieval [2]. Framework-level abstractions such as LangChain and LangGraph formalize these patterns into programmable graphs. However, the routing layer remains fundamentally deterministic: execution proceeds along a single committed path determined by explicit conditionals.

This deterministic commitment introduces two structural limitations. First, the system cannot represent uncertainty over competing tools or reasoning strategies. Second, it cannot revise earlier decisions unless explicit recovery logic is handcrafted. In contrast, probabilistic graphical models represent distributions over latent decisions and update beliefs as evidence accumulates [3]. Bayesian methods further allow principled adaptation by updating posterior distributions over model parameters.

In this work, we introduce probabilistic and Bayesian extensions of LangGraph-style execution. Instead of selecting a single next node via fixed control logic, we define routing as sampling from a state-dependent distribution over candidate nodes. Log-probabilities are computed from state features such as LLM confidence scores, evidence measures from tools, or accumulated reliability statistics. Execution traces therefore reflect both sampled actions and the probability mass assigned to alternatives.

We further introduce a Bayesian reliability model over tools. Each tool is assigned a latent reliability parameter with a conjugate Beta prior. Observed binary success indicators update posterior parameters online. Routing probabilities incorporate posterior means, enabling gradual adaptation across sessions. Tools that consistently produce reliable outputs gain higher posterior mass, while unreliable tools are down-weighted.

We do not evaluate the system using benchmark metrics. Instead, we provide controlled demonstrations illustrating qualitative behavioural differences between deterministic, probabilistic, and Bayesian execution. These include

recovery from initially selected tools, evidence-dependent fallback in real API scenarios, and session-level reliability adaptation.

The contribution of this paper is conceptual and architectural. We formalize execution graphs as stochastic control processes over nodes, demonstrate that deterministic routing is a special case of zero-entropy policies, and provide a concrete implementation compatible with existing LangGraph infrastructure. All theoretical results establishing structural advantages of probabilistic and Bayesian routing are presented in Appendix A.

2. Deterministic LangGraph as a Degenerate Inference Procedure

Modern LLM agent frameworks such as LangChain and LangGraph [4, 5] implement execution as a directed control graph: nodes correspond to language model calls or external tools, and edges encode allowable transitions. At each step, the system selects a single next action (tool or LLM invocation), executes it, appends the resulting observation to the state, and proceeds. Closely related ideas also appear in the broader literature on sequential decision-making and planning under uncertainty, where policies map states to actions, and trajectory distributions provide a natural language for analysis [6, 7].

Although typically presented as procedural orchestration, deterministic orchestration can be interpreted as a decision process over latent trajectories. In this section, we formalise this view and show that standard deterministic execution corresponds to a zero-entropy (or "zero-temperature") limit of a probabilistic trajectory model. This result is deliberately lightweight: it provides the conceptual bridge needed to justify our probabilistic and Bayesian extensions, while leaving all non-trivial theorems to Appendix A.

2.1. Execution as a Directed Control Graph

Let x denote the user query (and any static context such as system instructions). Let s_t denote the agent state at step t, which includes the full conversation history up to step t and all tool outputs observed so far. A LangGraph-style agent is defined by:

- an action set $\mathcal{A}$, consisting of tool invocations and LLM calls;
- a transition mechanism producing an observation o_t given (s_t, a_t);
- a deterministic policy π_{det} that selects $a_t = \pi_{\text{det}}(s_t)$.

The execution induces a single trajectory

$$\tau \;=\; (a_{1:T}, o_{1:T}), \tag{2.1}$$

where T is the (possibly data-dependent) stopping time. Deterministic orchestration corresponds to producing one such τ by committing to one action at each step and discarding all alternatives.

In practice, π_{det} is implemented by heuristics: tool routers, JSON parsers, hard thresholds, or a single LLM call whose output is interpreted as the next action. When tool outputs are noisy, APIs fail, or the LLM produces malformed structured outputs, this commitment can cause irreversible errors (e.g., selecting an unreliable tool early and never reconsidering).

2.2. Implicit Scoring and the MAP View

Deterministic orchestration can be recast as selecting a trajectory that minimises an implicit cost (or maximises an implicit score). Concretely, let $C(\tau; x)$ be a real-valued functional measuring the "badness" of a complete trajectory τ for query x. The exact form of C is not required here; it can encode, for example, invalid tool calls, hallucinated answers, contradictions with retrieved evidence, or failures of structured output constraints. Deterministic execution corresponds to selecting a single trajectory, typically in a greedy manner, that approximates

$$\tau^\star \in \arg\min_{\tau \in \mathcal{T}(x)} C(\tau; x), \tag{2.2}$$

where $\mathcal{T}(x)$ is the set of feasible trajectories induced by the control graph and stopping rules.

Equation (2.2) is an optimisation statement, but it is equally natural to interpret it probabilistically. Define an unnormalised trajectory density

$$\tilde{p}_\beta(\tau \mid x) \;\propto\; \exp\Big(-\beta\, C(\tau; x)\Big), \tag{2.3}$$

with inverse-temperature $\beta > 0$. Then the maximum a posteriori (MAP) trajectory under (2.3) is precisely a minimiser of $C(\tau; x)$. This provides a clean interpretation:

> Deterministic orchestration corresponds to selecting a point estimate (MAP) trajectory under an implicit energy model over trajectories.

This framing is standard in probabilistic modelling and statistical physics, where $\beta \to \infty$ yields concentration on minimisers (the "zero-temperature" limit) [7].

2.3. Degenerate Posterior Limit

We now state and prove the key bridge result: deterministic execution is the degenerate limit of the probabilistic trajectory model (2.3). This is the only self-contained formal proof we place in the main paper; all results establishing strict advantages (recovery guarantees, regret, posterior concentration, etc.) are deferred to Appendix A.

Proposition 2.1 (Deterministic execution as a zero-temperature (MAP) limit)**.** *Fix a query x and let $\mathcal{T}(x)$ denote the finite or countable set of feasible trajectories. Assume there exists a unique minimiser*

$$\tau^\star \in \arg\min_{\tau \in \mathcal{T}(x)} C(\tau; x). \tag{2.4}$$

Define the normalised distribution

$$p_\beta(\tau \mid x) \;=\; \frac{\exp\Big(-\beta\, C(\tau; x)\Big)}{Z_\beta(x)}, \qquad Z_\beta(x) \;=\; \sum_{\tau \in \mathcal{T}(x)} \exp\Big(-\beta\, C(\tau; x)\Big). \tag{2.5}$$

Then as $\beta \to \infty$,

$$p_\beta(\tau^\star \mid x) \to 1 \qquad \textit{and} \qquad p_\beta(\tau \mid x) \to 0 \quad \forall \tau \neq \tau^\star. \tag{2.6}$$

Equivalently, $p_\beta(\cdot \mid x)$ converges weakly to the point mass $\delta_{\tau^\star}$.

Proof. Let $\Delta(\tau) = C(\tau; x) - C(\tau^\star; x)$. By uniqueness, $\Delta(\tau) > 0$ for all $\tau \neq \tau^\star$. Consider the ratio

$$\frac{p_\beta(\tau \mid x)}{p_\beta(\tau^\star \mid x)} \;=\; \exp\Big(-\beta(C(\tau; x) - C(\tau^\star; x))\Big) \;=\; \exp\Big(-\beta\, \Delta(\tau)\Big). \tag{2.7}$$

Since $\Delta(\tau) > 0$, we have $\exp(-\beta\, \Delta(\tau)) \to 0$ as $\beta \to \infty$, hence $p_\beta(\tau \mid x)/p_\beta(\tau^\star \mid x) \to 0$ for all $\tau \neq \tau^\star$.

Next write the normaliser as

$$Z_\beta(x) = \exp\Big(-\beta C(\tau^\star; x)\Big) \left(1 + \sum_{\tau \neq \tau^\star} \exp\Big(-\beta\, \Delta(\tau)\Big)\right). \tag{2.8}$$

Therefore,

$$p_\beta(\tau^\star \mid x) = \frac{1}{1 + \sum_{\tau \neq \tau^\star} \exp(-\beta\, \Delta(\tau))}. \tag{2.9}$$

Each term in the sum tends to 0 as $\beta \to \infty$, and the sum is over a countable set with nonnegative terms, so the denominator tends to 1. Hence $p_\beta(\tau^\star \mid x) \to 1$ and consequently $p_\beta(\tau \mid x) \to 0$ for all $\tau \neq \tau^\star$. □

Proposition 2.1 justifies viewing deterministic LangGraph-style execution as a degenerate (zero-entropy) special case of our probabilistic formulation. Importantly, the proposition does not claim that deterministic execution is optimal in practice: real systems operate with misspecified costs, noisy tools, and partial observability. The purpose of the probabilistic and Bayesian extensions introduced in Sections 3–4 is precisely to handle such uncertainty by maintaining and updating uncertainty over trajectories and tool reliabilities, rather than collapsing immediately to a single committed path.

3. Probabilistic LangGraph

Section 2 recast deterministic LangGraph-style execution as selecting a single maximum a posteriori trajectory under an implicit energy model over feasible trajectories. We now retain a full distribution over trajectories and treat orchestration as inference under that model.

3.1. Trajectory Distribution

For a query x, let $\mathcal{T}(x)$ denote the set of feasible trajectories induced by the control graph and stopping rules. We use the trajectory representation introduced in Section 2.1:

$$\tau = (a_{1:T}, o_{1:T}), \qquad a_t \in \mathcal{A}, \tag{3.1}$$

where a_t is the action at step t (tool invocation or LLM call) and o_t is the resulting observation (tool output or model output). The agent state s_t is the interaction history up to time t, and evolves deterministically via

$$s_{t+1} = \mathrm{Update}(s_t, a_t, o_t), \tag{3.2}$$

where Update corresponds to appending (a_t, o_t) to the history as in Section 2.1.

Each feasible trajectory is assigned a real-valued cost $C(\tau; x)$ measuring trajectory quality for the query x. The probabilistic formulation assigns a Boltzmann-type distribution over trajectories:

$$p_\beta(\tau \mid x) = \frac{\exp\Big(-\beta\, C(\tau; x)\Big)}{Z_\beta(x)}, \qquad Z_\beta(x) = \sum_{\tau' \in \mathcal{T}(x)} \exp\Big(-\beta\, C(\tau'; x)\Big), \tag{3.3}$$

with inverse-temperature $\beta > 0$. For finite β, probability mass is distributed across multiple trajectories. As $\beta \to \infty$, the distribution concentrates on the minimum-cost trajectory, recovering deterministic execution as the degenerate limit established in Proposition 2.1.

3.2. Sequential Decomposition and Prefix Weights

In agent execution, costs are naturally accumulated step-by-step. We assume the trajectory cost decomposes additively as

$$C(\tau; x) = \sum_{t=1}^{T} c_t(s_t, a_t, o_t; x), \tag{3.4}$$

where c_t is a per-step cost term that may encode, for example, invalid tool calls, failures of structured output constraints, inconsistency with retrieved evidence, or other task-specific penalties.

For $t \leq T$, define the trajectory prefix

$$\tau_{1:t} = (a_{1:t}, o_{1:t}). \tag{3.5}$$

Using (3.4), the unnormalised trajectory density factorises as

$$p_\beta(\tau \mid x) \propto \prod_{t=1}^{T} \exp\Big(-\beta\, c_t(s_t, a_t, o_t; x)\Big). \tag{3.6}$$

This factorisation induces a natural sequential weight update for prefixes. Let $w_t(\tau_{1:t})$ denote the unnormalised weight of a prefix. Then

$$w_t(\tau_{1:t}) = w_{t-1}(\tau_{1:t-1}) \exp\Big(-\beta\, c_t(s_t, a_t, o_t; x)\Big), \qquad w_0 = 1. \tag{3.7}$$

Equation (3.7) is the operational mechanism behind probabilistic orchestration: rather than committing to a single continuation, the controller carries a

set of candidate prefixes whose relative weights are updated after each tool or LLM observation.

3.3. Particle Approximation of the Trajectory Posterior

The space $\mathcal{T}(x)$ typically grows combinatorially with depth, making exact computation of (3.3) infeasible. We therefore approximate $p_\beta(\tau \mid x)$ with a particle system.

Let $\{\tau_{1:t}^{(i)}, w_t^{(i)}\}_{i=1}^N$ denote N weighted prefixes at time t, where

$$\tau_{1:t}^{(i)} = (a_{1:t}^{(i)}, o_{1:t}^{(i)}). \tag{3.8}$$

A generic particle update consists of:

1. **Propose and execute:** For each particle i, propose an admissible next action $a_{t+1}^{(i)}$ consistent with the control graph, execute it, and observe $o_{t+1}^{(i)}$. Extend the prefix to $\tau_{1:t+1}^{(i)}$ using (3.8).

2. **Weight update:** Update weights using the incremental factor from (3.7):

$$w_{t+1}^{(i)} = w_t^{(i)} \exp\Big(-\beta\, c_{t+1}(s_{t+1}^{(i)}, a_{t+1}^{(i)}, o_{t+1}^{(i)}; x)\Big), \tag{3.9}$$

   with $s_{t+2}^{(i)} = \mathrm{Update}(s_{t+1}^{(i)}, a_{t+1}^{(i)}, o_{t+1}^{(i)})$.

3. **Normalisation:** Form normalised weights

$$\tilde{w}_{t+1}^{(i)} = \frac{w_{t+1}^{(i)}}{\sum_{j=1}^N w_{t+1}^{(j)}}. \tag{3.10}$$

4. **Resampling (optional):** If the weight distribution collapses, resample prefixes according to $\{\tilde{w}_{t+1}^{(i)}\}$ and reset weights to $1/N$.

At termination, the particle system defines the empirical approximation

$$\hat{p}_{\beta,N}(\tau \mid x) = \sum_{i=1}^N \tilde{w}_T^{(i)}\, \delta_{\tau^{(i)}}, \tag{3.11}$$

where $\tau^{(i)} = (a_{1:T}^{(i)}, o_{1:T}^{(i)})$ is the completed trajectory for particle i.

3.4. Action Selection Under Uncertainty

Deterministic LangGraph selects a single action at each step. In the probabilistic formulation, action selection can be derived from marginals induced by the particle approximation. Let $\mathcal{A}_t$ denote the set of admissible actions at state s_t. The induced marginal over the action at time t is estimated by

$$\hat{P}(a_t = a \mid x) = \sum_{i=1}^{N} \tilde{w}_t^{(i)} \mathbf{1}\{a_t^{(i)} = a\}, \qquad a \in \mathcal{A}_t. \tag{3.12}$$

This estimate supports several operational choices, including sampling an action according to (3.12), selecting the maximiser of (3.12), or using a risk-sensitive criterion based on expected continuation cost computed from the particle set. The key distinction from deterministic commitment is that multiple alternatives are retained long enough for subsequent observations to revise their relative plausibility via (3.9). Formal results quantifying correction and recovery properties are provided in Appendix A.

3.5. Deterministic Execution as a Special Case

The probabilistic controller reduces to deterministic orchestration under two simultaneous degenerations: retaining a single particle ($N = 1$) and concentrating the trajectory distribution ($\beta \to \infty$). In this regime, only one prefix is propagated and the exponential weighting collapses to selecting the locally minimal-cost continuation, yielding standard deterministic execution on the same control graph.

Section 4 extends this probabilistic formulation by placing distributions over latent parameters that govern trajectory costs and tool reliabilities, enabling learning across sessions rather than inference within a single run.

4. Bayesian LangGraph

Section 3 introduced a probabilistic controller that maintains a distribution over trajectories for a fixed cost functional $C(\tau; x)$. In this section, we extend the formulation by introducing latent parameters governing trajectory costs and tool reliabilities. The resulting framework maintains uncertainty not only over trajectories but also over the mechanisms that assign costs to trajectories.

4.1. Latent Parameterisation of Trajectory Costs

Recall the per-step cost from Section 3.2:

$$c_t(s_t, a_t, o_t; x), \tag{4.1}$$

and the additive decomposition

$$C(\tau; x) = \sum_{t=1}^{T} c_t(s_t, a_t, o_t; x), \qquad \tau = (a_{1:T}, o_{1:T}). \tag{4.2}$$

In practice, the quantities influencing c_t are not fully known a priori: tools may differ in reliability, structured outputs may fail with different probabilities, and retrieval sources may vary in quality across sessions. We model such variability through a latent parameter vector

$$\theta \in \Theta, \tag{4.3}$$

which governs the per-step costs. We therefore refine the decomposition to

$$C(\tau; x, \theta) = \sum_{t=1}^{T} c_t(s_t, a_t, o_t; x, \theta). \tag{4.4}$$

Conditional on θ, the trajectory distribution becomes

$$p_\beta(\tau \mid x, \theta) = \frac{\exp\Big(-\beta\, C(\tau; x, \theta)\Big)}{Z_\beta(x, \theta)}, \tag{4.5}$$

where

$$Z_\beta(x, \theta) = \sum_{\tau' \in \mathcal{T}(x)} \exp\Big(-\beta\, C(\tau'; x, \theta)\Big). \tag{4.6}$$

4.2. Data Derived from Execution

Bayesian updating requires a data object derived from observed execution. After observing the prefix

$$\tau_{1:t} = (a_{1:t}, o_{1:t}), \tag{4.7}$$

we compute summary events

$$D_t = D(\tau_{1:t}), \tag{4.8}$$

such as validity indicators, consistency checks, tool success flags, or any other diagnostics already reflected in the step costs. At termination we write $D = D_T$.

This definition keeps the Bayesian layer aligned with Sections 2–3: all information used for updating is derived from the same observed quantities (a_t, o_t) and the induced state sequence via $s_{t+1} = \mathrm{Update}(s_t, a_t, o_t)$.

4.3. Prior and Posterior over Parameters

We place a prior distribution $\pi(\theta)$ over Θ. The observations D provide information about θ through a likelihood $L(D \mid \theta)$, yielding the posterior

$$\pi(\theta \mid D) \propto \pi(\theta)\, L(D \mid \theta). \tag{4.9}$$

The Bayesian trajectory distribution marginalises over parameter uncertainty:

$$p_\beta(\tau \mid x, D) = \int_\Theta p_\beta(\tau \mid x, \theta)\, \pi(\theta \mid D)\, d\theta. \tag{4.10}$$

Equation (4.10) replaces the fixed-parameter model of Section 3 with a mixture over conditional trajectory distributions weighted by posterior belief over θ.

4.4. Example: Tool Reliability Model

To make the construction concrete, consider a finite set of tools indexed by $j \in \{1, \ldots, J\}$. Associate to each tool a reliability parameter $\theta_j \in [0, 1]$, and let $\theta = (\theta_1, \ldots, \theta_J)$. At step t, if $a_t = j$, we compute a binary success indicator as a deterministic function of the observed interaction:

$$y_t = g(s_t, a_t, o_t) \in \{0, 1\}, \tag{4.11}$$

where g encodes the success criterion used by the system, such as parse validity, verification success, or consistency with retrieved evidence.

Given the selected tool $a_t = j$, we model y_t as a Bernoulli observation with parameter θ_j:

$$\Pr(y_t = 1 \mid a_t = j, \theta) = \theta_j, \qquad \Pr(y_t = 0 \mid a_t = j, \theta) = 1 - \theta_j. \tag{4.12}$$

Let D include the collection of such indicators across steps. The likelihood

for the indicator sequence can be written as

$$L(D \mid \theta) = \prod_{t=1}^{T} \theta_{a_t}^{y_t} \left(1 - \theta_{a_t}\right)^{1-y_t}. \tag{4.13}$$

With independent Beta priors

$$\theta_j \sim \mathrm{Beta}(\alpha_j, \beta_j), \tag{4.14}$$

the posterior remains in the Beta family:

$$\theta_j \mid D \sim \mathrm{Beta}\left(\alpha_j + \sum_{t=1}^{T} \mathbf{1}\{a_t = j\} y_t,\ \beta_j + \sum_{t=1}^{T} \mathbf{1}\{a_t = j\}(1 - y_t)\right). \tag{4.15}$$

The posterior mean is

$$\mathbb{E}[\theta_j \mid D] = \frac{\alpha_j + \sum_{t=1}^{T} \mathbf{1}\{a_t = j\} y_t}{\alpha_j + \beta_j + \sum_{t=1}^{T} \mathbf{1}\{a_t = j\}}. \tag{4.16}$$

The reliability parameters can influence costs through $c_t(\cdot; x, \theta)$. One concrete and compatible construction is

$$c_t(s_t, a_t, o_t; x, \theta) = c_t^{\mathrm{base}}(s_t, a_t, o_t; x) + \lambda\, \mathbf{1}\{a_t = j\} \left(-\log \theta_j\right), \tag{4.17}$$

where $\lambda \geq 0$ controls the strength of the reliability penalty and $j = a_t$. This construction increases the cost of selecting tools inferred to be unreliable.

4.5. Joint Inference over Trajectories and Parameters

Bayesian LangGraph maintains uncertainty at two levels:

1. Trajectory uncertainty within a session, represented by weighted prefixes $\{\tau_{1:t}^{(i)}, w_t^{(i)}\}$ from Section 3.3.
2. Parameter uncertainty across sessions, represented by the posterior $\pi(\theta \mid D)$ in (4.9).

Two operational approximations are particularly natural:

- Plug-in approximation: replace θ in (4.4) by a point estimate such as the posterior mean (4.16), yielding a probabilistic controller of the form in Section 3.

- Mixture approximation: draw $\theta^{(k)} \sim \pi(\theta \mid D)$ and propagate particles under $p_\beta(\tau \mid x, \theta^{(k)})$, thereby approximating the marginal distribution (4.10) by Monte Carlo integration over θ.

Both strategies preserve the control-graph structure of Section 2 while enabling adaptation based on observed execution. Deterministic orchestration uses a fixed implicit cost model and does not update beliefs about tool reliability. Bayesian LangGraph explicitly models, updates, and exploits such uncertainty. Formal results on posterior concentration and long-run preference for more reliable tools are provided in Appendix A. Appendix B details the implementation of these update rules and their integration with the particle system.

5. Demonstrations and Observed Behaviour

We do not report benchmark metrics. Instead, we document controlled demonstrations implemented in the accompanying project files, and analyse their execution traces. The goal is to exhibit behavioural differences between deterministic LangGraph execution and the proposed probabilistic and Bayesian variants.

5.1. Minimal Probabilistic Router

The first demonstration instantiates a minimal probabilistic control loop over three nodes A, B, and C. Each node modifies a scalar state variable x, and routing decisions are sampled from log-probabilities depending on x.

The key observation from the execution trace is that:

- Routing decisions are stochastic but reproducible under fixed seeds.
- The distribution over next nodes evolves as a function of state.
- The system terminates when the router emits the special token `__end__`.

Unlike deterministic LangGraph, where routing is fixed by conditional edges, here routing is explicitly sampled from a normalized distribution derived via log-sum-exp stabilization. The trace logs include both sampled actions and the underlying probability mass assigned to each alternative.

5.2. LLM Confidence-Based Tool Switching

The second demonstration integrates a real LLM (via Colab Gemini) and a local deterministic tool.

The control flow is:

$$\texttt{LLM} \to \texttt{(FINAL or TOOL)} \to \texttt{FINAL}.$$

The LLM is prompted to return structured JSON:

- answer,
- confidence in $[0, 1]$,
- rationale.

The router computes

$$\log p(\texttt{FINAL}) = \log(\epsilon + \text{confidence}), \quad \log p(\texttt{TOOL}) = \log(\epsilon + (1 - \text{confidence})),$$

and samples accordingly.

Observed behaviour:

- For high-confidence answers, the graph often terminates immediately.
- For low-confidence answers, the system routes to the tool node.
- Execution traces explicitly show the sampled branch and associated probabilities.

Deterministic LangGraph lacks this mechanism: once the LLM node is executed, subsequent routing is predetermined. In contrast, the probabilistic version maintains a distribution over possible next actions conditioned on model-reported uncertainty.

5.3. Dual Search Engines: Fast vs Verified

A more structured demonstration compares two search tools:

- `FastSearchCache`: unverified, potentially stale.
- `PyPI_JSON`: verified authoritative source.

The deterministic baseline graph commits to the fast tool and does not reconsider alternative tools after receiving an answer.

The probabilistic graph:

1. Calls the fast tool.
2. Uses the LLM to produce a draft version, confidence, and a Boolean `should_verify`.
3. Routes probabilistically between `VERIFIED` and `FINAL`.

When the initial tool result is unverified and the query concerns "latest version", probability mass shifts toward the verified tool.

The execution trace reveals:

- An explicit routing step where probability mass is redistributed.
- A correction step in which the verified API is called.
- A final answer sourced from the authoritative endpoint.

This demonstrates recovery from an initially selected tool. Deterministic LangGraph, by design, does not revise earlier tool commitments without explicit handcrafted control logic.

5.4. Pathological Real API Case: DuckDuckGo Instant Answer

A further demonstration uses a real free API: DuckDuckGo Instant Answer.

The deterministic graph:

- Calls API1 (DuckDuckGo).
- Computes an evidence score from returned tokens.
- If evidence is insufficient, the LLM reports insufficient information.

The probabilistic graph:

- Calls API1.
- Computes a smooth evidence score $e \in [0, 1]$.
- Routes probabilistically between answering directly and consulting structured fallback providers (API2, API3).

When the evidence score is low, probability mass shifts toward fallback providers. The resulting answer includes structured month recommendations and budget considerations, with explicit citation tokens indicating source provenance.

The trace shows:

- Evidence-dependent routing probabilities.
- Sequential consultation of additional providers.
- Final answer grounded in multiple structured sources.

This behaviour emerges from probabilistic control rather than hard-coded failure detection.

5.5. Bayesian Reliability Updating

In `bayesian_langgraph.py` and `bayesian_rag.py`, tool reliability is treated as a latent parameter θ_j with Beta prior:

$$\theta_j \sim \text{Beta}(\alpha_j, \beta_j).$$

Binary success indicators y_t update the posterior:

$$\alpha_j \leftarrow \alpha_j + y_t, \quad \beta_j \leftarrow \beta_j + (1 - y_t).$$

Routing probabilities incorporate posterior means:

$$\mathbb{E}[\theta_j] = \frac{\alpha_j}{\alpha_j + \beta_j}.$$

Observed behaviour across repeated sessions:

- Tools with consistent success accumulate higher posterior mass.
- The router increasingly favours reliable tools.
- Poorly performing tools are gradually down-weighted.

Unlike deterministic graphs, this adaptation persists across runs and reflects accumulated evidence rather than fixed control logic.

6. Implementation Details

6.1. Project Structure

The implementation consists of three principal modules:

- `probabilistic_langgraph.py`
- `bayesian_langgraph.py`
- `bayesian_rag.py`

The probabilistic layer wraps a standard LangGraph `StateGraph` and introduces:

- A stochastic router node.
- Log-probability normalization using log-sum-exp.
- Explicit sampling from categorical distributions.
- Execution trace recording including probability mass.

6.2. State Representation

Execution state is represented as a structured object containing:

- user data,
- routing metadata,
- trace logs,
- optional random seed.

Routing decisions are stored in state fields and recorded in the trace with both sampled action and probability vector.

6.3. Numerical Stability

All routing decisions are computed in log space:

$$\tilde{p}_k = \exp(\ell_k - \log \sum_j \exp(\ell_j)).$$

This avoids underflow when combining multiple additive log terms, especially in multi-step compositions.

6.4. LLM Integration

LLM calls use Colab-integrated Gemini access via `google.colab.ai`. Structured outputs are enforced via JSON prompts. When necessary, regex extraction is applied to isolate valid JSON objects.

Confidence scores returned by the LLM are clipped to $[0, 1]$ and incorporated into routing log-probabilities.

6.5. Evidence Scoring

In the DuckDuckGo demonstration, evidence is quantified by:

$$e = \tanh\left(\frac{n}{200}\right),$$

where n is the number of non-trivial tokens extracted from the API response. This produces a smooth scalar gate in $[0, 1]$ used in routing decisions.

6.6. Bayesian Updating

For each tool j, we maintain (α_j, β_j) parameters. Updates occur online after observing a binary success indicator.

Posterior means influence routing through additive log terms. No closed-form variational approximation is required; the update is exact under the Beta-Bernoulli conjugate model.

6.7. Reproducibility

Random seeds are stored in state and used to initialize per-state random number generators. This ensures that stochastic routing is reproducible when desired.

6.8. Compatibility

The probabilistic and Bayesian layers do not modify LangGraph internals. Instead, they compile a standard graph with:

- A router node.
- Conditional edges based on sampled state.
- Standard node functions.

Thus the system remains compatible with existing LangGraph tooling while extending it with stochastic control

The complete implementation of the probabilistic and Bayesian extensions described in this paper, together with all demonstration scripts discussed in Section 5, is publicly available at https://github.com/AngshulMajumdar/Bayesian-Probabilistic-LangGraph. The repository contains the core probabilistic routing layer, the Bayesian reliability updating module, and reproducible demo workflows illustrating recovery from early tool commitments, evidence-dependent fallback, and session-level adaptation. The code is designed to remain fully compatible with standard LangGraph constructs while introducing stochastic and Bayesian control at the routing layer.

7. Conclusion

We have presented probabilistic and Bayesian extensions of LangGraph-style execution frameworks. The central idea is to replace deterministic routing with distributions over candidate next nodes, allowing the system to represent uncertainty and revise earlier commitments.

The probabilistic formulation introduces stochastic control at the routing layer. Instead of fixed conditional edges, the next action is sampled from a normalized log-probability distribution derived from state-dependent signals such as LLM confidence or evidence strength. Deterministic routing appears as a degenerate special case in which all probability mass is concentrated on a single branch.

The Bayesian formulation augments this mechanism with adaptive reliability estimation. Tool performance is modeled using conjugate Beta priors updated from observed binary success indicators. Routing probabilities incorporate posterior expectations, yielding gradual reallocation of probability mass toward consistently reliable tools.

Through controlled demonstrations, we illustrated three structural differences relative to deterministic execution: the ability to represent uncertainty over actions, the ability to recover from initially chosen tools, and the ability to adapt across sessions via posterior updating. These behaviours emerge from the probabilistic control formulation rather than handcrafted exception logic.

All theoretical guarantees and structural comparisons are developed in Appendix A. Implementation details are provided in Appendix B. The framework remains fully compatible with existing LangGraph infrastructure while

extending it with stochastic and Bayesian control.

We view probabilistic execution as a natural next step in the evolution of agent frameworks, bridging deterministic orchestration and probabilistic decision-making.

References

[1] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, React: Synergizing reasoning and acting in language models, arXiv preprint arXiv:2210.03629 (2023).

[2] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, et al., Webgpt: Browser-assisted question-answering with human feedback, arXiv preprint arXiv:2112.09332 (2021).

[3] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.

[4] H. Chase, Langchain, https://github.com/langchain-ai/langchain, accessed 2026-02-27 (2022).

[5] LangChain AI, Langgraph, https://github.com/langchain-ai/langgraph, accessed 2026-02-27 (2024).

[6] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, 2nd Edition, MIT Press, 2018.

[7] Z. Ghahramani, Probabilistic machine learning and artificial intelligence, Nature 521 (7553) (2015) 452–459.

Appendix A. Theoretical Analysis

Appendix A.1. Setup and Assumptions

Let $\mathcal{N}$ be a finite set of nodes. At each step t, the system is in state $s_t \in \mathcal{S}$ and selects an action (next node) $a_t \in \mathcal{N}$.

We distinguish:

- Deterministic routing policies $\pi_{\text{det}} : \mathcal{S} \to \mathcal{N}$.
- Probabilistic routing policies $\pi_{\text{prob}} : \mathcal{S} \to \Delta(\mathcal{N})$.

For Bayesian tool analysis, we assume:

1. There are K tools indexed by $j \in \{1, \ldots, K\}$.
2. Each tool j has an unknown reliability parameter $\theta_j^* \in (0, 1)$.
3. When tool j is selected, a binary success indicator $y_t \sim \text{Bernoulli}(\theta_j^*)$ is observed.
4. Conditioned on the chosen tool sequence, observations are independent.

Each tool has prior

$$\theta_j \sim \text{Beta}(\alpha_{j,0}, \beta_{j,0}),$$

and posteriors are updated via conjugacy.

Routing may depend on posterior means

$$\mu_j(t) = \mathbb{E}[\theta_j \mid D_t] = \frac{\alpha_{j,0} + k_j(t)}{\alpha_{j,0} + \beta_{j,0} + n_j(t)},$$

where $k_j(t)$ is number of observed successes and $n_j(t)$ number of times tool j was selected up to time t.

Appendix A.2. Deterministic Routing as a Special Case

Proposition Appendix A.1. *The set of deterministic routing policies is a strict subset of the set of probabilistic routing policies.*

Proof. Given any deterministic policy π_{det}, define

$$\pi_{\text{prob}}(a \mid s) = \begin{cases} 1 & \text{if } a = \pi_{\text{det}}(s), \\ 0 & \text{otherwise.} \end{cases}$$

This induces identical trajectories. Conversely, a probabilistic policy assigning non-degenerate mass cannot be represented deterministically. Hence strict containment holds. □

Appendix A.3. Irreversibility Under Single-Commitment Deterministic Routing

Consider a state s_0 with two candidate tools T_1 and T_2.

Assume a deterministic policy selects T_1 at s_0 and does not revisit tool selection.

Theorem Appendix A.1. *If $\theta_2^* > \theta_1^*$, then any single-commitment deterministic policy that selects T_1 at s_0 achieves expected success probability θ_1^*, and cannot exceed it.*

Proof. Under single-commitment routing, the selected tool is T_1 with probability one. The observed outcome is Bernoulli with parameter θ_1^*. Therefore expected success equals θ_1^*. No alternative branch is ever executed, hence no correction occurs. □

Now consider a probabilistic routing policy at s_0:

$$\pi(T_1 \mid s_0) = \alpha, \quad \pi(T_2 \mid s_0) = 1 - \alpha.$$

Expected success becomes:

$$J(\alpha) = \alpha\theta_1^* + (1 - \alpha)\theta_2^*.$$

Corollary Appendix A.1. *If $\theta_2^* > \theta_1^*$, then for all $\alpha < 1$,*

$$J(\alpha) > \theta_1^*.$$

Proof. Since $\theta_2^* > \theta_1^*$,

$$\alpha\theta_1^* + (1 - \alpha)\theta_2^* = \theta_1^* + (1 - \alpha)(\theta_2^* - \theta_1^*) > \theta_1^*.$$

□

Thus probabilistic routing strictly dominates single-commitment deterministic routing whenever tool reliabilities differ.

Appendix A.4. Evidence-Gated Routing Improvement

Let $e \in [0, 1]$ denote an evidence score derived from a tool output.
Assume:

- Conditional correctness probability of the initial tool output is p_e, increasing in e.
- A fallback tool has correctness probability p_f.

Consider routing:

$$\pi(\text{final} \mid e) = e, \quad \pi(\text{fallback} \mid e) = 1 - e.$$

Expected success:

$$J(e) = ep_e + (1 - e)p_f.$$

Proposition Appendix A.2. *If $p_f > p_e$ for sufficiently small e, then there exists $\varepsilon > 0$ such that for all $e < \varepsilon$,*

$$J(e) > p_e.$$

Proof. For small e,

$$J(e) - p_e = (1 - e)p_f - (1 - e)p_e = (1 - e)(p_f - p_e).$$

If $p_f > p_e$, then this expression is positive for all $e < 1$. □

Hence evidence-dependent routing strictly improves expected correctness in low-evidence regimes.

Appendix A.5. Posterior Consistency

Theorem Appendix A.2. *For each tool j, if it is selected infinitely often and observations are i.i.d. Bernoulli(θ_j^*), then*

$$\mu_j(t) \to \theta_j^* \quad \textit{almost surely.}$$

Proof. By the strong law of large numbers,

$$\frac{k_j(t)}{n_j(t)} \to \theta_j^* \quad \text{almost surely.}$$

The posterior mean can be written as

$$\mu_j(t) = \frac{\alpha_{j,0} + k_j(t)}{\alpha_{j,0} + \beta_{j,0} + n_j(t)}.$$

Dividing numerator and denominator by $n_j(t)$ and taking limits yields convergence to θ_j^*. □

Appendix A.6. Asymptotic Optimal Tool Selection

Assume:

- There exists a unique optimal tool j^* with $\theta_{j^*}^* > \theta_j^*$ for all $j \neq j^*$.

- The routing policy ensures each tool is selected infinitely often.
- After sufficient exploration, routing selects the tool with largest posterior mean.

Theorem Appendix A.3. *Under these assumptions,*

$$\mathbb{P}(a_t = j^*) \to 1 \quad \text{as } t \to \infty.$$

Proof. By posterior consistency,

$$\mu_j(t) \to \theta_j^* \quad \text{almost surely.}$$

Since the optimal tool has strictly larger true reliability, there exists T such that for all $t > T$,

$$\mu_{j^*}(t) > \mu_j(t) \quad \text{for all } j \neq j^*.$$

If routing selects the tool with largest posterior mean after sufficient exploration, it follows that $a_t = j^*$ for all sufficiently large t, except possibly finitely many exploration steps. Hence the selection probability converges to one. □

Appendix A.7. Conclusion of Theoretical Analysis

We have established:

1. Deterministic routing is a strict subset of probabilistic routing.
2. Single-commitment deterministic policies cannot recover from suboptimal tool selection.
3. Probabilistic routing strictly improves expected success when alternative tools differ in reliability.
4. Evidence-dependent routing improves performance in low-evidence regimes.
5. Bayesian posterior updating is consistent under standard assumptions.
6. Posterior-mean-based routing asymptotically selects the optimal tool.

These results formalize structural and asymptotic advantages of probabilistic and Bayesian execution over deterministic routing.