# Modeling Agent Customization and Optimization: A Category Theoretic Framework for Multi-Agent Workflows

J. Michael Constans, Gemini Pro 3.1.7, and Claude Opus 4.7

***Abstract*—As large language model (LLM) agents become increasingly modular, the orchestration of their components—such as discrete skills, model providers, and external data servers—presents a complex optimization problem. In this paper, we introduce a category theoretic framework for constructing and optimizing complex agent workflows. We define a strict monoidal category where objects are typed data streams and morphisms represent parameterized agent capabilities. By implementing this framework in Haskell using Generalized Algebraic Data Types (GADTs), we provide compile-time guarantees of well-formedness for agent topologies. Furthermore, we formalize the search for optimal agent configurations as a Bayesian Optimization problem over this categorical space, employing Pareto optimization to balance resource constraints against solution quality.**

***Index Terms*—Applied Category Theory, Large Language Models, Bayesian Optimization, Pareto Optimization, Monoidal Categories, Haskell.**

## I. Introduction

The proliferation of agentic frameworks has enabled the construction of complex systems where multiple language models and tools interact to solve graduated tasks. However, discovering the optimal combination of these components under finite resource constraints (e.g., token limits and API latency) remains an open challenge.

In this work, we cast the formulation of agent architectures into the language of Applied Category Theory. Drawing inspiration from Waites' plumbing calculus for agent coordination, we define a copy-discard category that enforces structural boundaries on data flows. We then layer a Sequential Decision-Making framework over this category to iteratively evaluate and optimize these structures using Bayesian Optimization (BO).

## II. Categorical Semantics of Agent Workflows

To ensure structural integrity, we define a category $\mathcal{C}$ where:

- **Objects** ($\mathrm{Ob}(\mathcal{C})$) are specific types of information states, such as Prompt, Context, Code, and TestResult.
- **Morphisms** ($\mathrm{Hom}(A, B)$) are computational processes mapping an input state to an output state.

In our Haskell implementation, this category is realized via the `AgentGraph a b` Generalized Algebraic Data Type (GADT).

### A. Monoidal Structure and Routing

Agents rarely operate in a purely linear sequence. To support parallelism and scatter-gather patterns, $\mathcal{C}$ is endowed with a strict monoidal structure, expressed via the tensor product $\otimes$ (implemented as `Par` or `***` in Haskell).

Furthermore, our category requires explicit structural routing morphisms, making it a copy-discard category:

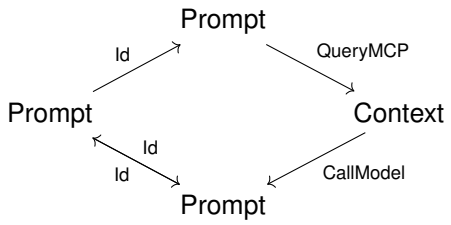
$$A \xrightarrow{\Delta} A \otimes A \qquad A \xrightarrow{\lozenge} I \tag{1}$$

Here, $\Delta$ (Copy) duplicates a stream, and $\lozenge$ (Drop) discards it, terminating the flow.

### B. Composition of Morphisms

A complex agent workflow is built by composing simpler morphisms. For instance, consider an agent that duplicates a user prompt, uses one copy to fetch external context via a Model Context Protocol (MCP) server, and passes both the prompt and the new context to a generative model.

This topology is represented by the following commutative diagram:




(2)

In the DSL, this exact structure is guaranteed by the compiler:

```
standardPiAgent :: AgentGraph Prompt TestResult
standardPiAgent =
    Copy
    >>> (Id *** QueryMCP "github-pr-server")
```

J. M. Constans is with Feature Software, LLC. E-mail: mikeco@constans.dev.

Gemini Pro 3.1 is a large language model developed by Google (model identifier `gemini-3.1-pro-preview`). The main body of this paper was prepared through interactive collaboration with that model: the human author directed the scope, theorems, and exposition; the model drafted prose, formalised statements, and produced the `tikz-cd` diagrams under continuous human review.

Claude Opus 4.7 is a large language model developed by Anthropic (model identifier `claude-opus-4-7`). The appendix on user-harness feedback as a lens was prepared through interactive collaboration with that model: the human author directed the scope and the canonical mapping; the model identified the parametric-lens correspondence in the categorical cybernetics literature, drafted the prose, and formalised the typed-substitution claim under continuous human review.

```
>>> CallModel "claude-3-7-sonnet"
>>> ApplySkill "linter-skill"
>>> RunTests
```

### C. Advanced Control: Choices, Loops, and Parameterization

While directed acyclic topologies cover basic use cases, robust autonomous agents require discrete routing decisions and feedback loops (e.g., self-correction). We extend $\mathcal{C}$ with *Coproducts* (sum types) to model discrete conditional branching (`ArrowChoice`), and *Traces* to model feedback loops (`ArrowLoop`).

A feedback loop where a component $D$ is fed back to the input is modeled categorically via a trace operator:

$$\begin{array}{ccc} B & \xrightarrow{\text{loop}} & C \\ {\scriptstyle \text{id}_B \otimes \text{trace}}\uparrow & & \uparrow{\scriptstyle \text{id}_C \otimes \text{trace}} \\ B \otimes D & \xrightarrow{f} & C \otimes D \end{array} \tag{3}$$

Furthermore, to optimize an agent, we must separate the *hyperparameter space* (e.g., model temperature) from the *data flow* (the prompt). Following Hanks et al. [3], we construct the parameterized category $\mathsf{Para}(\mathcal{C})$. A parameterized morphism is one where the computational process is contingent on an external parameter configuration $P$:

$$f : P \otimes A \to B \tag{4}$$

In our framework, this allows the Bayesian Optimizer to iteratively tune the parameter space $P$ without altering the underlying topological validity of the graph from $A \to B$.

## III. Case Studies in Complex Agent Modeling

To demonstrate the expressiveness of the $\mathcal{C}$ category and its Haskell implementation, we mapped two real-world, complex agentic architectures to our framework.

### A. Claude Code: Orchestration and Dreaming

The March 2026 Anthropic Claude Code leak revealed a complex "lead-agent-plus-subagents" pattern, where a primary coordinator spawns isolated sub-agents in parallel. In our DSL, this is natively modeled using the monoidal tensor product $\otimes$ alongside $\Delta$ (Copy) to duplicate the context safely.

Furthermore, the leak highlighted a "Dreaming" routine for memory consolidation, which periodically compacts older conversation context. Because this involves taking a modified context and feeding it back into the agent loop, it cannot be modeled by a simple DAG. We use the trace operator (`ArrowLoop`) to mathematically model this cyclic dependency without deadlock.

### B. OpenClaw: Embedded Runtimes and Event Taps

OpenClaw is an open-source workflow implemented directly on top of the Pi agent harness. Instead of executing the agent as an isolated subprocess, OpenClaw imports the agent session natively to inject custom tool pipelines and subscribe to real-time events for block chunking.

Our framework models this embedded paradigm elegantly. The custom policy filtering is modeled as a sequence of context-modifying skills. More importantly, the event subscription is modeled mathematically as a parallel process tapping the execution stream. We use $\Delta$ to copy the data stream, run the agent loop on the primary branch, and run the event subscriber on the secondary branch. The subscriber's output is then explicitly discarded via $\Diamond$ (Drop), ensuring the stream's structural integrity is maintained.

## IV. Optimization Over Categories

Once the structural space of valid agent graphs is defined, we address the challenge of finding the optimal graph configuration. We frame this as a combinatorial Bayesian Optimization problem.

### A. Pareto Frontier Evaluation

Each execution of a specific `AgentGraph` yields a set of `Metrics`, mapping to a multidimensional objective space (e.g., Token Cost vs. Quality). Because we cannot exhaustively evaluate all valid graphs, we rely on Pareto optimization.

Let $\Phi \in \Sigma_{\mathcal{C}}(S)$ represent a specific agent wiring. A configuration $\Phi_1$ Pareto-dominates $\Phi_2$ if it performs strictly better on at least one metric without degrading any others. The optimization seeks the Pareto Frontier—the set of non-dominated morphisms.

### B. Surrogate Modeling and Acquisition

In continuous Bayesian Optimization, a Gaussian Process (GP) is often used as a surrogate model. Because our search space consists of discrete categorical diagrams, we utilize a graph-based surrogate model to predict the performance of untested configurations.

The map $(c, p) \mapsto \mathsf{Metrics}$ from configuration–problem pairs to observed metrics is non-deterministic: model sampling, harness timing, and validation race conditions inject noise, and the noise level itself varies with the configuration — some configurations are intrinsically more variable than others. We therefore model the surrogate as a *heteroscedastic* GP, predicting both the conditional mean and an input-dependent variance:

$$\mathsf{predictPerformance} : \mathsf{History} \to \mathsf{Hom}(A, B) \to (\mathbb{E}[\mathsf{Metrics}],\ \mathrm{Var}[\mathsf{Metr} \tag{5}$$

Below a threshold of accumulated trials — the surrogate's bootstrap regime — variance estimates from sparse data are unreliable, and acquisition reverts to pure exploration over the slot space (Latin-hypercube or uniform-random). Above the threshold, the acquisition function uses both predicted mean and variance to balance the exploration of novel agent topological structures against the exploitation of known, high-performing configurations: high-variance promising regions are sampled to reduce uncertainty, while high-variance unpromising regions are skipped.

## V. Conclusion

By lifting agent orchestration into a strictly typed monoidal category, we decouple the structural validation of multi-agent systems from their execution. This abstraction provides a rigorous, mathematically sound foundation for deploying Sequential Decision-Making and Bayesian Optimization algorithms to autonomously design and refine the next generation of software engineering agents.

## Appendix A
## User-Harness Feedback as a Lens

The Para($\mathcal{C}$) construction of Section II-C separates the parameter space $P$ from the underlying data flow $A \to B$, but it remains silent on *how* parameters are determined or refined during a trial. In the practitioner literature on coding-agent harnesses, Bockeler partitions the user-facing wrapper around a coding agent — the *user harness* — into two regulation mechanisms [5]: *guides* (feedforward; applied before the agent acts: project conventions, system prompts, bootstrap scripts, CodeMod tools) and *sensors* (feedback; applied after the agent acts: pre-commit hooks, test suites, static analyzers, AI code review). We observe that the guide/sensor pair has the structure of a *parametric lens* in the sense of categorical cybernetics [6].

A lens $\binom{S}{T} \to \binom{A}{B}$ consists of a forward view $v : S \to A$ and a backward update $u : S \otimes B \to T$. Mapping this to the user harness around a parameterized agent morphism $f : P \otimes A \to B$:

- A **guide** is the forward leg: from the harness state (problem statement, project conventions, prior trial history) it derives the parameter configuration and any context shaping the agent's input before $f$ runs.
- A **sensor** is the backward leg: from the agent's output $B$ and the surrounding state, it produces a signal that closes the loop — either to the agent itself (self-correction, inner loop) or to the optimizer (Bayesian update on $P$, outer loop).

This perspective recasts a user-harness configuration as a stack of parametric lenses composed over the agent's parameterized morphism, rather than a flat collection of items. The optimization of Section IV thus operates implicitly over *lensed morphisms*: parameterized agent graphs equipped with a guide/sensor envelope.

### A. Typed Items and the Substitution Principle

Each user-harness item additionally carries a type classification [5]: *computational* (deterministic, fast, low-variance: linters, type checkers, structural analyzers) or *inferential* (semantic, slow, higher-variance: LLM-as-judge, semantic code review). The $2 \times 2$ of (role, type) $\in \{\text{guide, sensor}\} \times \{\text{computational, inferential}\}$ gives each item a position in a typed catalog.

Given the lens framing, the following claim is structural rather than empirical: *a substitution that replaces an inferential item with a computational item of the same lens shape — same forward and backward types — is strictly Pareto-dominant* on cost, latency, and variance, with the loop's compositional structure preserved. The Bayesian optimizer of Section IV-B can therefore exploit a catalog of known-equivalent type-preserving substitutions ahead of any random or surrogate-driven move; these substitutions need not be rediscovered statistically.

### B. Partial and Asynchronous Feedback

The strict lens shape assumes the backward leg always returns. Two features of real harnesses violate this:

- **Subjective sensors** (human or LLM-judge ratings) arrive asynchronously or not at all relative to the trial closing.
- **Telemetry-based sensors** emit zero, one, or many events per trial.

The accurate optic is therefore not a strict lens but an *affine traversal* over the sensor channel, capturing zero-or-more feedback semantics. The lens is the right first approximation for the optimization formulation; refinement to the traversal case is a target for future work.

### C. Trial Outcomes as a Sum Type

Beyond sensor-channel partiality, the trial itself does not always yield metrics. We classify each closed trial by its *outcome*:

$$\mathsf{Outcome} \;=\; \mathsf{Completed}(\mathsf{Metrics}) + \mathsf{BoundaryViolation}(\mathsf{Metrics}) + \mathsf{E} \tag{6}$$

where Completed corresponds to natural termination, BoundaryViolation to a configured boundary being crossed (timeout, cost cap), and ErrorEscalated to a transient or persistent failure that exhausted the harness's retry budget. Boundary-violated trials still record the dimensions on which the boundary was crossed, which is informative: they teach the surrogate where the cost cliff lives in feature space and bias future acquisition away from it. Error-escalated trials are preserved for asynchronous human classification but contribute no metric value.

The Bayesian surrogate of Section IV therefore operates not on the full outcome space but on its metric-bearing projection

$$\pi : \mathsf{Outcome} \to \mathsf{Maybe}(\mathsf{Metrics}), \qquad \pi(\mathsf{Completed}(m)) = \pi(\mathsf{Bounda} \tag{7}$$

so the optimization map's effective domain is the disjoint union of completed and boundary-violated trials. Boundary violations are signal, not loss — they are first-class data points for the surrogate. Error-escalated trials sit outside the surrogate's input until human classification reduces them to one of the metric-bearing branches or rejects the trial.

## References


[1] W. Waites, "A Typed Language for Agent Coordination," *The n-Category Café*, Mar. 2026. [Online]. Available: https://golem.ph.utexas.edu/category/2026/03/a_typed_language_for_agent_coordination.html

[2] R. R. Lam and K. E. Willcox, "Lookahead Bayesian Optimization with Inequality Constraints," in *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[3] T. Hanks, B. She, E. Patterson, M. Hale, M. Klawonn, and J. Fairbanks, "Modeling Model Predictive Control: A Category Theoretic Framework for Multistage Control Problems," 2024, arXiv:2305.03820v2.

[4] M. Marcolli, "Pareto Optimization in Categories," 2022, unpublished.

[5] B. Bockeler, "Harness Engineering for Coding Agents," *martinfowler.com*, 2026. [Online]. Available: https://martinfowler.com/articles/harness-engineering.html
[6] M. Capucci, B. Gavranović, J. Hedges, and E. F. Rischel, "Towards Foundations of Categorical Cybernetics," 2021, arXiv:2105.06332.