

Opus

A Quantitative Framework for Workflow Evaluation

Alan Seroul

Research Affiliate
AppliedAI

Théo Fagnoni

Member of Technical Staff
AppliedAI

Inès Adnani

Member of Technical Staff
AppliedAI

Dana O. Mohamed

Member of Technical Staff
AppliedAI

Phillip Kingston*

Member of Technical Staff
AppliedAI

5 November 2025



This work is licensed under a Creative Commons Attribution-Noncommercial-ShareAlike 4.0 International License
(CC BY-NC-SA 4.0)

Abstract

This paper introduces the Opus Workflow Evaluation Framework, a probabilistic-normative formulation for quantifying Workflow quality and efficiency. It integrates notions of correctness, reliability, and cost into a coherent mathematical model that enables direct comparison, scoring, and optimization of Workflows. The framework combines the Opus Workflow Reward, a probabilistic function estimating expected performance through success likelihood, resource usage, and output gain, with the Opus Workflow Normative Penalties, a set of measurable functions capturing structural and informational quality across Cohesion, Coupling, Observability, and Information Hygiene. It supports automated Workflow assessment, ranking, and optimization within modern automation systems such as Opus¹ and can be integrated into Reinforcement Learning loops to guide Workflow discovery and refinement. In this paper:

1. We introduce the Opus Workflow Reward model that formalizes Workflow success as a probabilistic expectation over costs and outcomes.
2. We define measurable Opus Workflow Normative Penalties capturing structural, semantic, and signal-related properties of Workflows.
3. We propose a unified optimization formulation for identifying and ranking optimal Workflows under joint Reward–Penalty trade-offs.

* Corresponding author: phillip.kingston@opus.com

¹ Opus: opus.com

1 Introduction

Evaluating the quality and efficiency of work has been a core problem in automation since its inception. From early industrial systems to modern AI-driven pipelines, the need to evaluate how effectively tasks are executed, both by humans and machines, remains central to productivity, accountability, and optimization. Today, as organizations increasingly rely on Large Language Models (LLMs) and agents to plan, execute, and refine operational processes, the question of how to formally evaluate these processes becomes critical.

Existing business frameworks, such as Business Process Management (BPM), process querying and mining, and related methods, provide tools for analyzing how work is structured and performed, yet they were not designed for probabilistic, AI-generated processes that exhibit uncertainty, continuous adaptation, and multi-agent interactions. NLP metrics such as BLEU, ROUGE, and BERTScore quantify textual overlap but are inadequate for assessing the structural and operational quality of deployed Workflows.

To address this gap, we propose the Opus Workflow Evaluation Framework, which integrates probabilistic modeling with normative penalties to evaluate Workflows as stochastic, resource-bounded processes. The framework extends concepts from operations research, software engineering, and BPM into a quantitative formulation suitable for modern AI system evaluation at scale. It provides a principled basis for measuring, comparing, and optimizing Workflows according to their expected performance and structural quality, establishing Workflow evaluation as a measurable and computationally grounded discipline.

Definitions Our methodology is based on the following concepts, a subset of which has been previously introduced in *Opus: A Large Workflow Model for Complex Workflow Generation* by Fagnoni et al. [1], *Opus: A Workflow Intention Framework for Complex Workflow Generation* by Kingston et al. [2], and *Opus: A Prompt Intention Framework for Complex Workflow Generation* by Fagnoni et al. [3]:

Workflow: A Workflow is a Directed Acyclic Graph (DAG) of Task, Workflow Input, and Workflow Output nodes, with edges capturing data and execution dependencies. Workflows define ordered sequences of operations that transform inputs into target outputs. The same algorithm may be represented by multiple Workflows at varying levels of abstraction, granularity, or descriptive detail.

Workflow Input: Workflow Inputs are represented by dedicated input nodes in a Workflow. Each input node accepts a single, explicit data structure, ensuring clarity in how data is processed and transformed. Every input node is associated with a correctness probability, capturing the likelihood that the provided document conforms to the expected information and format. This accounts for cases where the input of one Workflow is produced by the output of another, and supports resilience against malformed data.

Workflow Output: Workflow Outputs are represented by dedicated output nodes in a Workflow. Each output node emits a single, explicit data structure, ensuring interpretability and a clear mapping between process and outcome. Every output node is associated with a correctness probability, derived from the overall information and dependencies within the Workflow.

Workflow Task: An atomic unit of work within a Workflow, performing a specific function with defined input and output schemas, objectives, timing constraints, and success criteria. Tasks follow a single-responsibility principle (defined later in the paper), support automation or manual intervention, and maintain contextual awareness of dependencies. Tasks are auditable by humans or AI agents against their definition.

Task cumulative resources: Resources permanently consumed by the execution of a Workflow Task, accumulating over the course of a Workflow execution. Examples include monetary cost, energy usage, persistent storage (e.g. hard drive space), raw materials, mobile data on capped plans, and labor hours.

Task releasable resources: Resources temporarily allocated during the execution of a Workflow Task, held for its duration and released upon completion. Examples include RAM, CPU or GPU cycles, active threads, API quota, bandwidth, manpower, etc.

Workflow gain: The measurable value produced when a Workflow executes successfully. It represents the benefit, typically monetary, obtained by automating a process execution that would otherwise require longer execution time and higher per-hour costs. It may also capture benefits such as time savings, reduced risk, or improved accuracy, expressed on a consistent value scale.

2 Background

A historical challenge: measuring the quality of work The evaluation of work quality has long been a central concern in industrial and organizational systems. Following the industrial revolution, Taylor’s theory of scientific management sought to turn labor efficiency into a measurable science, focusing on standardization, time-motion analysis, and waste elimination. As economies evolved through mass production, digitalization, and globalization, the challenge of measuring and improving work quality spread across fields such as operations research and software engineering. Business Process Management (BPM) emerged as a discipline defining process models as structured representations of organizational Workflows and introducing analytical frameworks for their evaluation [4–7]. In parallel, process querying and mining techniques [8–10] enabled the comparison of real executions with their modeled intent, bridging the gap between theory and practice. These developments established a foundation for assessing structure and correctness but did not provide means to quantify expected performance, uncertainty, or value, which are essential in modern AI-driven contexts.

Frameworks and derivations from software engineering In BPM literature, Workflows are viewed as structured models evaluated through principles of structural soundness and semantic clarity. The ISO 9000 framework defines quality as “the degree to which a set of inherent characteristics fulfills requirements”, aligning naturally with the evaluation of process models. From software engineering, principles such as the Single Responsibility Principle (SRP), Cohesion, and Coupling offer parallel insights: atomicity, modularity, and maintainability are as relevant to Workflow graphs as to code. Vanderfeesten et al. [4] map software-quality metrics to process models, while Petri net properties such as soundness, liveness, and safeness [10] provide tools for ensuring correctness. Together, these foundations establish a rigorous structural basis for Workflow evaluation.

Limitations of current approaches Despite these advances, classical BPM and software metrics largely assume deterministic execution. Metrics such as control-flow complexity [7], modularity indices [4], or SEQUAL-style quality dimensions [6] cannot express probabilistic dependencies, cumulative cost propagation, or stochastic reliability. Similarly, semantic evaluation methods (BLEU, ROUGE, METEOR, and BERTScore) assess textual similarity but not operational fidelity. These tools therefore fail to capture whether a deployed Workflow performs efficiently, reliably, or intelligibly in practice.

Bridging BPM and probabilistic evaluation Recent work in AI-assisted automation reframes Workflows as DAGs of stochastic Tasks, each with measurable probabilities of success, durations, and resource vectors. Evaluating such systems requires unifying the structural clarity of BPM with stochastic modeling. This intersection calls for a framework capable of quantifying expected performance under uncertainty while retaining interpretability and normative alignment.

The Opus quantitative approach The Opus Workflow Evaluation Framework addresses this challenge by formalizing Workflows as measurable stochastic systems. Each Task is modeled as a random variable over a measurable space, while Workflow performance and quality are expressed through expectation-based Reward and Normative Penalty functions. The Reward function quantifies the expected value of a Workflow given its success probabilities, resource costs, and gains; the Normative Penalties extend classical principles into continuous, measurable forms (Cohesion, Coupling, Observability, and Information Hygiene). Collectively, they establish a quantitative basis for comparing, ranking, and optimizing Workflows under uncertainty.

3 Opus Workflow Evaluation Framework

3.1 Opus Workflow Formalism

Workflow Task We model the execution of a Task as a random variable $X : \Omega \rightarrow \Omega'$, where Ω belongs to a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ and Ω' is a measurable space. The sample space Ω represents the universe of all possible initial conditions for the Task. Note that both Ω and Ω' depend on the considered Task X . It is defined as the Cartesian product $\Omega = I \times S$, where:

I is the Task input space. Each element $i \in I$ is a tuple where an input payload is paired with a boolean flag indicating its validity.

S is the set of all possible random seeds influencing the Task’s stochastic behavior during its execution.

An element $\omega = (i, s) \in \Omega$ thus represents a single, deterministic execution instance of the Task with a given input i and a given seed s .

The target space Ω' represents the space of all possible outcomes of the Task. It is defined as the Cartesian product $\Omega' = O \times R$, where:

O is the Task’s output space. Each element $o \in O$ is a tuple of generated outputs, each accompanied by a boolean indicating its validity.

$R \subseteq \mathbb{R}_+^{n+m+1}$ is a vector space representing the resources consumed during execution with $n, m \in \mathbb{N}$.

The application of the Task X to a specific initial condition $\omega = (i, s)$ yields a deterministic outcome $X(\omega) = (o, r)$, which consists of the output data o and the consumed resources r . The probability measure \mathbb{P} on Ω models the likelihood of encountering specific inputs and seeds, allowing us to reason about the probability distribution of outcomes in Ω' . This probabilistic formalization is essential for analyzing a Task's success probability, performance, and overall quality.

Workflow We define a Workflow by a pair (G, Φ) , where:

$G = (V, E)$ is a DAG that encodes the dependency structure of the process. Nodes $v \in V$ represent either Workflow Inputs, Workflow Outputs or Workflow Tasks, and edges $e = (u \rightarrow v) \in E$ represent the execution flow and the data dependencies between them.

The node set V is partitioned into three disjoint subsets: $V = V_{\text{in}} \sqcup V_{\text{task}} \sqcup V_{\text{out}}$, where:

V_{in} : Workflow Input nodes, which have no incoming edges. These nodes typically receive external data and serve as entry points into the Workflow.

V_{out} : Workflow Output nodes, which have no outgoing edges. These nodes produce the final outcomes of the Workflow.

V_{task} : Workflow Task nodes, which are internal and have both incoming and outgoing edges. These nodes perform intermediate transformations or decision making.

Each Workflow Input node $v \in V_{\text{in}}$ is associated with:

$$\phi_{\text{in}} : \begin{cases} V_{\text{in}} & \rightarrow [0, 1] \times \mathcal{T} \\ v & \rightarrow \phi_{\text{in}}(v) = (\pi_v, \tau_v) \end{cases}$$

where π_v is the initial correctness probability, and $\tau_v \in \mathcal{T}$ denotes the type of data expected at node v . The type space \mathcal{T} can be defined as a set of structured data schemas or type signatures (e.g. tuples, lists, or user-defined types with domain and codomain constraints).

Each Workflow Task node $v \in V_{\text{task}}$ is associated with:

$$\phi_{\text{task}} : \begin{cases} V_{\text{task}} & \rightarrow [0, 1] \times [0, 1] \times \mathbb{R}_+^m \times \mathbb{R}_+ \times \mathbb{R}_+^n \times \mathcal{I}, \quad n, m \in \mathbb{N} \\ v & \rightarrow \phi_{\text{task}}(v) = (p_v, q_v, r_v^{(g)}, d_v, r_v^{(r)}, \iota_v) \end{cases}$$

where:

p_v : probability of success given that all parent Tasks succeed.

q_v : probability of success given that at least one parent Task fails.

$r_v^{(g)} \in \mathbb{R}_+^m$: cumulative resource vector.

$d_v \in \mathbb{R}_+$: execution duration of the Task.

$r_v^{(r)} \in \mathbb{R}_+^n$: releasable resource vector.

$\iota_v \in \mathcal{I}$: Task implementation (a function), represented either as an effective probabilistic Turing machine or as its realization in a specific programming language.

We assume that the Workflows under comparison share an identical set of Workflow Inputs and Workflow Outputs. This ensures that differences in performance can be attributed solely to variations in their internal structure, without requiring additional domain-specific assumptions.

We assume that for a given Task, its execution duration and resource consumption remain constant across its executions. This assumption yields a discrete timeline for Workflow execution, where each Task is either active or inactive, simplifying the evaluation of cumulative resource usage. In practice, however, this assumption often fails, e.g. in OCR Tasks where both duration and resource demand scale with document size. A better formulation can model duration and resources as functions of input characteristics. Nevertheless, data-driven estimation methods, based on historical execution traces and large-scale empirical measurements, can approximate these relationships with sufficient accuracy to preserve the validity of the framework under realistic conditions.

3.2 Opus Workflow Reward

3.2.1 Workflow resource consumption

Building on the Task-level resource definitions above, we measure Workflow-level resource consumption across three dimensions: permanently consumed cumulative resources that accumulate during execution, total execution time determined by the critical path of dependent Tasks, and releasable resources that must be provisioned concurrently at peak demand. This breakdown focuses on the main cost drivers of Workflow execution and provides a structured basis for evaluation and comparison.

Let V denote the node set of a Workflow $W = (G, \Phi)$, let $\tilde{\mathcal{P}}$ be the set of all root-to-leaf paths in G . We define the following Workflow-level resource consumption measures:

1. Cumulative resources $R^{(g)}$: the total amount of permanently consumed resources throughout the Workflow,

$$R^{(g)}(W) = \sum_{v \in V_{\text{task}}} r_v^{(g)} \quad (1)$$

2. Execution duration d : the worst-case (longest path) execution time, assuming sequential dependencies,

$$d(W) = \max_{P \in \tilde{\mathcal{P}}} \sum_{v \in P \cap V_{\text{task}}} d_v \quad (2)$$

3. Releasable resources $R^{(r)}$: the peak concurrent demand of Workflow Task releasable resources at any time t , with V_t denoting the set of possible active Tasks at t . We assume an ASAP schedule where each Task begins immediately after all predecessors have been completed,

$$R^{(r)}(W) = \max_t \sum_{v \in V_t \cap V_{\text{task}}} r_v^{(r)} \quad (3)$$

More details are given in the Appendix.

3.2.2 Success Probability

Let v be a node with parents $\{u_1, \dots, u_k\}$. For each node v , we define the event

$$T_v := \{\text{"node } v \text{ produced a correct output"}\}$$

Let $p_v = \mathbb{P}(T_v | \bigcap_{j=1}^k T_{u_j})$ and $q_v = \mathbb{P}(T_v | \bigcup_{j=1}^k \bar{T}_{u_j})$.

We assume the following:

- (i) Parent correctness events are independent: $\mathbb{P}(\bigcap_{j=1}^k T_{u_j}) = \prod_{j=1}^k \mathbb{P}(T_{u_j})$.
- (ii) Output nodes introduce no additional error modes. They act as logical conjunctions of their parents, which is equivalent to setting $p_v = 1$ and $q_v = 0$.

Node success probability Workflow Input nodes: correctness is given by π_v ,

$$\mathbb{P}(T_v) = \pi_v$$

Workflow Task nodes: by the law of total probability and the independence assumption (i),

$$\mathbb{P}(T_v) = q_v + (p_v - q_v) \prod_{j=1}^k \mathbb{P}(T_{u_j})$$

This models a two-phase behavior: if all parents are correct, the node succeeds with probability p_v ; otherwise, it succeeds with a degraded probability q_v .

Workflow Output nodes: correctness is the conjunction of parent correctness,

$$\mathbb{P}(T_v) = \prod_{j=1}^k \mathbb{P}(T_{u_j})$$

Workflow success probability We define the Workflow success event as the simultaneous correctness of all Workflow Output nodes:

$$T_W := \bigcap_{v \in V_{\text{out}}} T_v$$

Accordingly, the Workflow success probability is

$$\mathbb{P}(W) := \mathbb{P}(T_W) = \mathbb{P}(\bigcap_{v \in V_{\text{out}}} T_v)$$

Exact evaluation requires the joint distribution of output correctness events. Assuming independence among $\{T_v, v \in V_{\text{out}}\}$, this reduces to

$$\mathbb{P}(W) = \prod_{v \in V_{\text{out}}} \mathbb{P}(T_v) \tag{4}$$

$\mathbb{P}(W)$ quantifies the likelihood that the Workflow produces entirely correct outputs given its inputs.

3.2.3 Reward

We define a scalar cost function that aggregates cumulative, temporal, and releasable resources:

$$C(W) = \langle w^{(g)}, R^{(g)}(W) \rangle + w^{(d)} \cdot d(W) + \langle w^{(r)}, R^{(r)}(W) \rangle \quad (5)$$

where $w^{(g)} \in \mathbb{R}^m$, $w^{(d)} \in \mathbb{R}$, and $w^{(r)} \in \mathbb{R}^n$ are weights encoding the relative importance of each resource type.

For each Workflow Output node $v \in V_{\text{out}}$, we define the random variable

$$G_v = \begin{cases} g_v & \text{with probability } \mathbb{P}(T_v) \\ 0 & \text{with probability } 1 - \mathbb{P}(T_v) \end{cases}$$

representing the gain realized if a Workflow Output is correct.

The net benefit of executing the Workflow W is then

$$B_W = \sum_{v \in V_{\text{out}}} G_v - C(W) \quad (6)$$

Fixed costs $C(W)$ are always incurred, while benefits g_v are realized only when the corresponding Workflow Output succeed. We assume that the total $C(W)$ is fixed for each Workflow execution, accounting for all Tasks defined in the Workflow, even those that may not run if execution halts prematurely. A more refined formulation could model per-Task costs as random variables, incurring cost only for Tasks that are actually executed.

We define the expected Reward of W :

$$\begin{aligned} \mathcal{R}(W) &= \mathbb{E}[B_W] \\ &= \sum_{v \in V_{\text{out}}} \mathbb{P}(T_v) \cdot g_v - C(W) \end{aligned} \quad (7)$$

$\mathcal{R}(W)$ quantifies the expected net value of a Workflow execution, combining output-specific gains weighted by their success probabilities and offset by execution costs. Positive Reward indicates that the Workflow is expected to generate value; negative Reward indicates expected loss. The weights $(w^{(g)}, w^{(d)}, w^{(r)})$ can be set from empirical measurements, domain-specific heuristics, or market-based resource pricing, enabling comparisons across heterogeneous Workflows in operationally meaningful units.

3.3 Opus Workflow Normative Penalties

We evaluate nodes and Workflows along four dimensions, referred to as the Opus Workflow Normative Penalties. Cohesion (Ch) penalizes nodes that conflate multiple responsibilities, reflecting the dispersion of functions within a node. Coupling (Cp) penalizes dependencies among nodes, capturing structural and semantic links that can propagate failures or complicate evolution. Observability (Ob) penalizes opacity and incompleteness in runtime signals, measuring the difficulty of inferring the actual state of execution from logs, metrics, and traces. Information Hygiene (Ih) penalizes the emission of irrelevant, redundant, or privacy-sensitive signals, ensuring that the information produced is concise, relevant, and actionable.

From these dimensions we derive two higher-level evaluative penalties, which we postulate as normative requirements for Workflow quality. The Cohesive Independence Penalty (CIP) combines Cohesion and Coupling, ensuring that each node is atomic in purpose and minimally dependent on others. The Signal Integrity Penalty (SIP) combines Observability and Information Hygiene, ensuring that runtime signals are trustworthy, relevant, necessary, and sufficient for monitoring, debugging, and auditing. Both penalties are measurable and continuous: they provide criteria by which Workflows can be systematically compared and optimized, with 0 representing the optimal penalty-free state and 1 representing the worst-case scenario.

Let $W \in \mathcal{W}$ denote a Workflow with $|V_{\text{task}}| = n$ Workflow Task nodes.

Cohesion Penalty (Ch) Cohesion denotes the degree to which each node in a Workflow performs a single, well-defined function. A cohesive node is singular and atomic; its function can be specified without ambiguity or the mixing of unrelated concerns.

A low Cohesion Penalty provides clarity of purpose and supports maintainability. A high Cohesion Penalty results in overloaded nodes that conflate responsibilities, creating ambiguity and complicating evolution.

$$\text{For a Task, } \text{Ch} : \begin{cases} V_{\text{task}} & \rightarrow [0, 1] \\ v & \mapsto \text{Ch}(v) \end{cases}, \quad \text{for a Workflow, } \text{Ch} : \begin{cases} \mathcal{W} & \rightarrow [0, 1] \\ W & \mapsto \left(\frac{1}{n} \sum_{v \in V_{\text{task}}} \text{Ch}(v)^2 \right)^{1/2} \end{cases}$$

Coupling Penalty (Cp) Coupling measures the degree of dependence among nodes. It reflects both structural connections and semantic links such as shared data, state, or control.

A low Coupling Penalty supports modularity and substitution: nodes can be altered, replaced, or reused without destabilizing the Workflow as a whole. A high Coupling Penalty produces brittle systems, where failures or modifications propagate widely.

$$\text{For a Task, } \text{Cp} : \begin{cases} V_{\text{task}} & \rightarrow [0, 1] \\ v & \mapsto \text{Cp}(v) \end{cases}, \quad \text{for a Workflow, } \text{Cp} : \begin{cases} \mathcal{W} & \rightarrow [0, 1] \\ W & \mapsto \left(\frac{1}{n} \sum_{v \in V_{\text{task}}} \text{Cp}(v)^2 \right)^{1/2} \end{cases}$$

Observability Penalty (Ob) Observability measures whether runtime signals (outputs, logs, traces, metrics) are sufficient and whether they faithfully reflect the Workflow's actual execution state. It characterizes the alignment between exposed information and underlying execution.

A low Observability Penalty indicates that signals are complete and accurate enough to support monitoring, diagnosis, and decision-making. A high Observability Penalty indicates that information is missing to establish the true execution state.

$$\text{For a Task, } \text{Ob} : \begin{cases} V_{\text{task}} & \rightarrow [0, 1] \\ v & \mapsto \text{Ob}(v) \end{cases}, \quad \text{for a Workflow, } \text{Ob} : \begin{cases} \mathcal{W} & \rightarrow [0, 1] \\ W & \mapsto \left(\frac{1}{n} \sum_{v \in V_{\text{task}}} \text{Ob}(v)^2 \right)^{1/2} \end{cases}$$

Information Hygiene Penalty (Ih) Information Hygiene evaluates whether runtime signals are necessary and restricted to what is relevant for understanding the Workflow's execution. It penalizes the emission of irrelevant, redundant, or privacy-sensitive information.

A low Information Hygiene Penalty means that signals are concise, meaningful, and privacy-preserving. A high Information Hygiene Penalty means that signals are excessive, noisy or low-value, burdening operators and automated systems while obscuring critical insights.

$$\text{For a Task, } \text{Ih} : \begin{cases} V_{\text{task}} & \rightarrow [0, 1] \\ v & \mapsto \text{Ih}(v) \end{cases}, \quad \text{for a Workflow, } \text{Ih} : \begin{cases} \mathcal{W} & \rightarrow [0, 1] \\ W & \mapsto \left(\frac{1}{n} \sum_{v \in V_{\text{task}}} \text{Ih}(v)^2 \right)^{1/2} \end{cases}$$

Note that $\text{Ch}(v)$, $\text{Cp}(v)$, $\text{Ob}(v)$ and $\text{Ih}(v)$ implicitly depend on the Workflow.

Cohesive Independence Penalty (CIP) CIP integrates Cohesion and Coupling Penalties. A Workflow achieves low CIP when its nodes each hold a clear, atomic responsibility (low Ch) and remain minimally dependent on others (low Cp). CIP establishes structural modularity and functional separation as essential conditions of Workflow design.

Cohesion and Coupling are treated as complementary quantities of structural responsibility. To enable further factorization, we suppose $\forall v \in V_{\text{task}}, \text{Ch}(v) + \text{Cp}(v) = 1$.

$$\text{CIP} : \begin{cases} \mathcal{W} & \rightarrow [0, 1] \\ W & \mapsto (\alpha_{\text{Ch}} \cdot \text{Ch}(W)^2 + \alpha_{\text{Cp}} \cdot \text{Cp}(W)^2)^{1/2}, \\ & \alpha_{\text{Ch}}, \alpha_{\text{Cp}} \in \mathbb{R}_+, \alpha_{\text{Ch}} + \alpha_{\text{Cp}} = 1 \end{cases}$$

$$\forall W \in \mathcal{W},$$

$$\text{CIP}(W)^2 = \alpha_{\text{Ch}} \cdot (1 - \alpha_{\text{Ch}}) + \frac{1}{n} \sum_{v \in V_{\text{task}}} (\alpha_{\text{Ch}} - \text{Cp}(v))^2 \quad (8)$$

Signal Integrity Penalty (SIP) SIP integrates Observability and Information Hygiene Penalties. A Workflow achieves low SIP when its runtime signals are both sufficient, accurate (low Ob) and necessary (low Ih) to reflect the underlying execution state. SIP establishes transparency and reliability as essential conditions for trustworthy Workflow execution.

Observability and Information Hygiene are treated as complementary quantities of signal integrity. To enable further factorization, we suppose $\forall v \in V_{\text{task}}, \text{Ob}(v) + \text{Ih}(v) = 1$.

$$\text{SIP} : \begin{cases} \mathcal{W} \rightarrow [0, 1] \\ W \mapsto (\alpha_{\text{Ob}} \cdot \text{Ob}(W)^2 + \alpha_{\text{Ih}} \cdot \text{Ih}(W)^2)^{1/2}, \\ \alpha_{\text{Ob}}, \alpha_{\text{Ih}} \in \mathbb{R}_+, \alpha_{\text{Ob}} + \alpha_{\text{Ih}} = 1 \end{cases}$$

$$\text{SIP}(W)^2 = \alpha_{\text{Ob}} \cdot (1 - \alpha_{\text{Ob}}) + \frac{1}{n} \sum_{v \in V_{\text{task}}} (\alpha_{\text{Ob}} - \text{Ih}(v))^2 \quad (9)$$

CIP and SIP together provide a measurable and normative foundation for Workflow evaluation. They define continuous criteria for assessing quality, enabling systematic comparison and principled optimization.

Opus Workflow Penalty We introduce

$$\forall W \in \mathcal{W}, \quad \mathcal{L}(W)^2 = \gamma_s \cdot \text{CIP}(W)^2 + \gamma_d \cdot \text{SIP}(W)^2$$

where $\gamma_s, \gamma_d \in [0, 1]$ are user-defined trade-off parameters.

Posing $\gamma_s + \gamma_d = 1$ without loss of generality, we have:

$$\mathcal{L}(W)^2 = \gamma_s \cdot \text{CIP}(W)^2 + (1 - \gamma_s) \cdot \text{SIP}(W)^2 \quad (10)$$

By construction, $\mathcal{L}(W)$ is bounded between 0 and 1:

$\mathcal{L}(W) = 0$ if the Workflow is perfectly minimized against all Penalty terms,

$\mathcal{L}(W) = 1$ if the Workflow deviates maximally in all dimensions.

These relationships are visualized in Figure 1, which represents how the four normative penalties (Ch, Cp, Ob, Ih) combine into the higher-level CIP and SIP penalties.

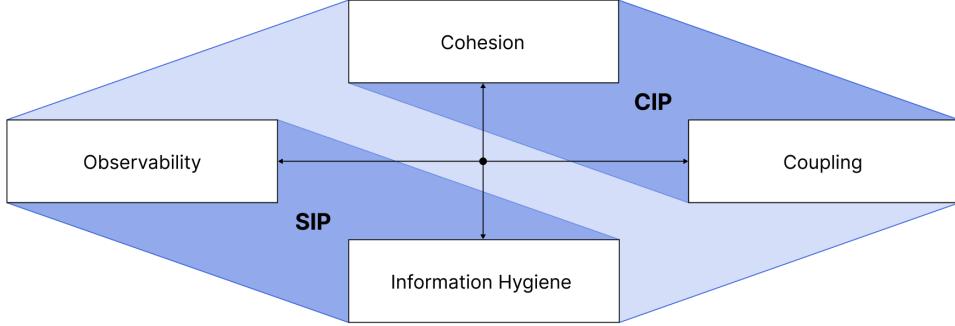


Figure 1: Opus Workflow Normative Penalties

3.4 Framework application

Finding optimal Workflows One strategy for identifying optimal Workflows is a two-stage procedure: first maximize the Reward \mathcal{R} to obtain efficient candidates, then, among these, minimize the Penalty \mathcal{L} to select the best formulations.

Formally, this process is expressed as the two-step optimization problem:

$$\mathcal{W}^{**} = \underset{W \in \mathcal{W}^*}{\operatorname{argmin}} \mathcal{L}(W) \quad (11)$$

where

$$\mathcal{W}^* = \underset{W \in \mathcal{W}_c}{\operatorname{argmax}} \mathcal{R}(W) \quad (12)$$

\mathcal{W}_c denotes the subset of feasible Workflows satisfying business and technical constraints (e.g. cost limits, manpower, bandwidth, reliability, and input/output specifications). These constraints may also include user-defined restrictions on performance or resources. Equation (11) highlights the trade-off between maximizing Reward (raw performance) and minimizing Penalty (interpretability, complexity, maintainability, and structural quality). In this setting, the Reward reflects how well the current Workflow performs, while the Penalty measures its potential for further improvement.

As with most optimization problems, the existence of a global optimum cannot be guaranteed, nor can any general algorithm ensure its discovery. Moreover, explicitly characterizing \mathcal{W}_c is challenging. In practice, a common approach is to construct a finite candidate set $\mathcal{W}'_c \subset \mathcal{W}_c$ expected to perform well, then iteratively refine this set using heuristics, domain knowledge, or algorithmic exploration techniques.

Ranking a set of Workflows While the framework permits comparisons between arbitrary Workflows, we recommend restricting comparisons to those sharing the same Workflow Inputs and Outputs and addressing the same business objective. This constraint ensures that compared Workflows operate within an equivalent process scope, making their evaluation and ranking both meaningful and practically relevant. Therefore, it should be enforced when constructing the candidate set \mathcal{W}_c .

The reliability of ranking depends critically on consistent estimation of underlying parameters. For the comparison to be valid, resource costs and success probabilities must be assigned consistently across identical nodes in different Workflows. Otherwise, rankings may reflect parameter inconsistencies rather than genuine structural differences.

Within a consistently estimated candidate set, Workflows can be ordered by a strict lexicographic preference: given two Workflows W_1 and W_2 , we write

$$W_1 \succ W_2 \iff \mathcal{R}(W_1) > \mathcal{R}(W_2) \quad \text{or} \quad (\mathcal{R}(W_1) = \mathcal{R}(W_2) \text{ and } \mathcal{L}(W_1) < \mathcal{L}(W_2)) \quad (13)$$

This rule reflects our optimization philosophy: Reward prevails, with Penalty acting as a tie-breaker.

Beyond ordering, it is also useful to quantify performance gaps. We define the distance between two Workflows W_1 and W_2 as

$$d(W_1, W_2) = |\mathcal{R}(W_1) - \mathcal{R}(W_2)| \quad (14)$$

which measures the absolute difference in Reward, consistent with the lexicographic order where Penalty is only considered when Rewards are equal.

A Reinforcement Learning Perspective The terms Reward and Penalty are deliberately borrowed from the Reinforcement Learning (RL) paradigm. This analogy highlights the inherently iterative nature of Workflow optimization, which can be described as a feedback-driven cycle:

1. A candidate Workflow is proposed for a specific process.
2. The candidate is assessed by estimating its resource costs, success probabilities, and expected gains.
3. A Reward is computed, serving as a feedback signal that quantifies performance and encourages the discovery of superior alternatives.
4. A Penalty is computed, guiding exploration toward well-structured Workflows that improve debugging, maintainability, and iterative refinement.

In this perspective, the organization or system designing Workflows acts as the Agent, while the Opus Workflow Evaluation Framework provides the environmental signals through which outcomes are measured. The State is defined by the Reward and Penalty values associated with a candidate Workflow, the Action corresponds to proposing a new Workflow, and the Policy denotes the strategy or generative mechanism guiding the creation of such candidates.

4 Case study

To illustrate the framework in practice, we present a simple use case. This example benchmarks a set of candidate Workflows designed to automatically classify customer complaint emails.

Workflow Inputs The process requires the following inputs:

Customer Full Name (text): the customer's complete name as registered in their profile, including both given and family names.

Customer Email File (email file): the raw email received from the customer, including headers and message body.

Customer Account Number (text): a unique customer account identifier, validated for existence and activity.

Workflow Output The expected output from the process is:

Support Ticket Record (support ticket JSON file): a structured ticket containing all required customer details and request information.

Data Characteristics We assume the following values for data and processing:

Average email length: 100 tokens.

Average prompt length: 100 tokens.

Average response length: 50 tokens.

Average LLM call duration: 1000 ms.

Average Python script execution duration: 100 ms.

Cold start latency (serverless warm-up): 800 ms.

LLM costs:

Small model: input $\$0.40 / 1M$ tokens ($\$4.0 \times 10^{-5}$ for 100 tokens), output $\$1.60 / 1M$ tokens ($\$8.0 \times 10^{-5}$ for 50 tokens).

Large model: input $\$2.00 / 1M$ tokens ($\$2.0 \times 10^{-4}$ for 100 tokens), output $\$8.00 / 1M$ tokens ($\$4.0 \times 10^{-4}$ for 50 tokens).

Customer Lifetime Value (CLV) The business value depends on the customer segment:

Small customer: \$30

Medium customer: \$360

Enterprise customer: \$16,000

Impact of Ticket Classification The quality of classification affects customer satisfaction:

Well classified: the customer receives a relevant response quickly.

Misclassified: three possible outcomes occur,

- No answer (5% probability, 30% churn).
- Delayed answer due to email redirection (80% probability, 1% churn).
- Uninformed answer (15% probability, 5% churn).

Expected Loss Estimation The expected financial loss is expressed as:

$$\begin{aligned} g &= \text{CLV} \times (0.05 \times 0.3 + 0.8 \times 0.01 + 0.15 \times 0.05) \\ &= \text{CLV} \times 0.0305 \end{aligned}$$

Thus:

$$\begin{aligned} g_s &= 30 \times 0.0305 \approx \$0.92 \quad (\text{for small customer}) \\ g_m &= 360 \times 0.0305 \approx \$10.98 \quad (\text{for medium customer}) \\ g_e &= 16,000 \times 0.0305 \approx \$488 \quad (\text{for enterprise customer}) \end{aligned}$$

We interpret these quantities as the monetary value of avoiding the expected loss. We focus here on the B2C setting with a small customer (\$30 CLV).

Weights We assign:

$$\begin{aligned} w^{(g)} &= 1 \quad (\text{all costs expressed in dollars}) \\ w^{(d)} &= 2.1 \times 10^{-12} \quad (\text{serverless function execution cost per ms}) \end{aligned}$$

Default Parameter Values Unless otherwise specified, the default values for p , q and cost are $p = 1$, $q = 0$ and cost = 0. This simplified setting corresponds to a purely deterministic node with no stochastic uncertainty. For instance, retrieving an element from a dictionary always succeeds if the key exists and always fails otherwise. Cost is cumulative and expressed in dollars (\$). Duration (d) is expressed in milliseconds (ms).

4.1 Workflow 1

Candidate W_1 is shown in Figure 2. The Workflow Tasks are:

Extract Email Subject: $p = 0.95, d = 1500 + 800, \text{cost} = 1.6 \times 10^{-4}$

Extract Email Body: $d = 50 + 800$

Identify Request Type from Email: $p = 0.9, d = 1500, \text{cost} = 1.6 \times 10^{-4}$

Validate Customer Account: $d = 200 + 800$

Review - Identify Request Type from Email: $p = 0.95, q = 0.7, d = 1500, \text{cost} = 8.0 \times 10^{-4}$

Review - Validate Customer Account: $p = 0.99, q = 0.7, d = 500, \text{cost} = 7.0 \times 10^{-4}$

Retrieve Customer Name: $d = 200 + 800$

Assemble Support Ticket: $d = 10$

Review - Assemble Support Ticket: $p = 0.95, q = 0.7, d = 800, \text{cost} = 7.0 \times 10^{-4}$

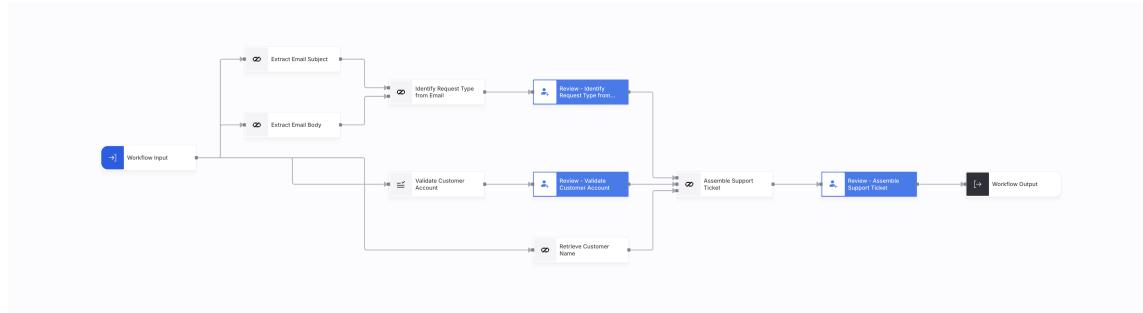


Figure 2: Workflow 1 - Extracting and classifying email content before ticket assembly.

4.2 Workflow 2

Candidate W_2 is shown in Figure 3. The Workflow Tasks are:

Extract and Identify Request Type from Email: $p = 0.9, d = 1500 + 800, \text{cost} = 1.6 \times 10^{-4}$

Review - Extract and Identify Request Type from Email: $p = 0.95, q = 0.7, d = 1500, \text{cost} = 8.0 \times 10^{-4}$

Validate Customer Account: $d = 200 + 800$

Retrieve Customer Name: $d = 200 + 800$

Assemble Support Ticket: $d = 10$

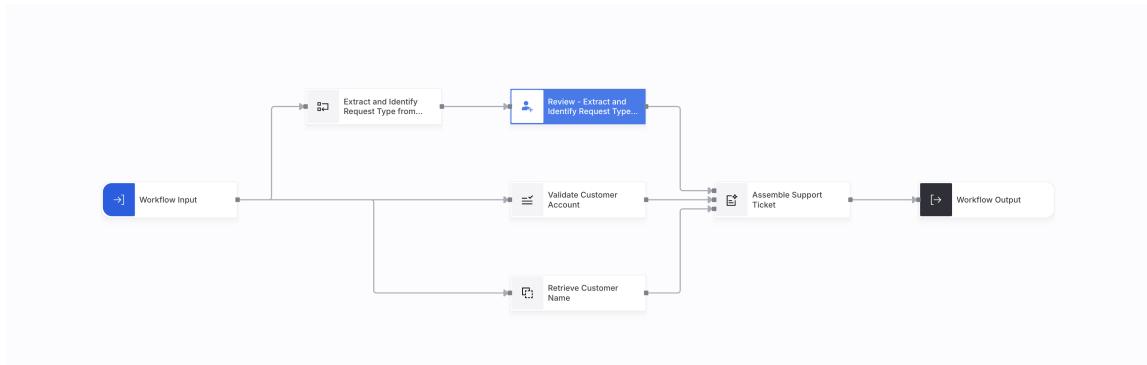


Figure 3: Workflow 2 - Compact design with fewer reviews.

4.3 Workflow 3

Candidate W_3 is shown in Figure 4. The Workflow Tasks are:

Extract and Identify Request Type from Email: $p = 0.9 \times 0.95 + (1 - 0.9) \times 0.7 = 0.925$,
 $d = 1500 + 1500 + 800$, cost = 9.6×10^{-4}

Validate Customer Account: $d = 200 + 800$

Retrieve Customer Name: $d = 200 + 800$

Assemble Support Ticket: $d = 10$

This Workflow merges the nodes *Extract and Identify Request Type from Email* and *Review of Extract and Identify Request Type from Email*.

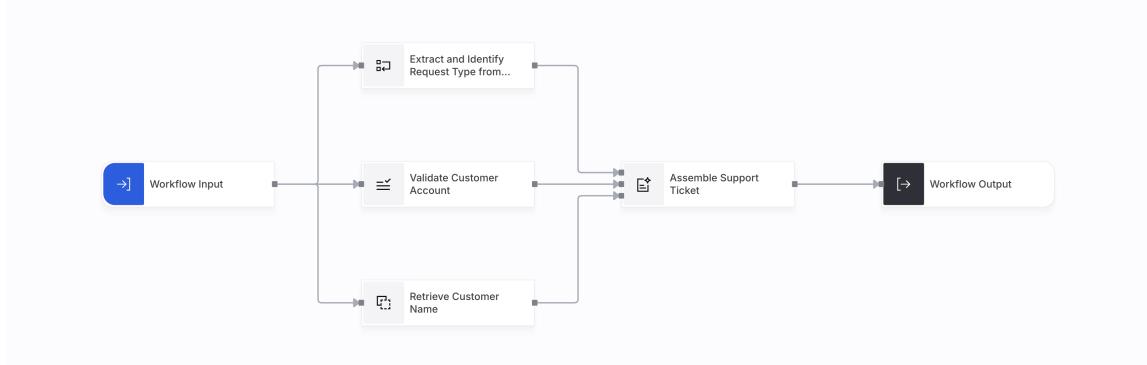


Figure 4: Workflow 3 - Merged review.

4.4 Analysis

Benchmark	Workflow 1	Workflow 2	Workflow 3
Cost (\$)	2.52×10^{-3}	9.6×10^{-4}	9.6×10^{-4}
Max duration (ms)	6110.0	3810.0	3810.0
Success probability	0.9262	0.9250	0.9250
Reward \mathcal{R} (\$)	0.8495	0.8500	0.8500
CIP	0.2525	0.2520	0.2570
SIP	0.2500	0.2573	0.2848
Penalty \mathcal{L}	0.2513	0.2547	0.2713

Table 1: Benchmark evaluation for the three candidate Workflows.

Table 1 summarizes the metrics. The expected Reward values $\mathcal{R}(W_1) = \$0.8495$ and $\mathcal{R}(W_2) = \$0.8500$ establish that $W_2 \succ W_1$. W_2 and W_3 achieve identical Reward scores. However, their topologies differ: in W_3 , merging the review with the action prevents independent evaluation of the review’s contribution. Observability is therefore reduced (because only output information is displayed), and the Penalty increases. For these reasons, Workflow 2 emerges as the most balanced option: lower cost and shorter runtime, with high probability of success preserved. Finally, note that our model prioritizes shorter execution times over higher reliability. If reliability were valued more highly, no corrective term would be required in the equations: adjusting the gain parameters, especially the CLV, would naturally shift the optimization. While serverless costs dominate in low-value B2C cases, for high-value enterprise scenarios, reliability becomes paramount.

5 Conclusion

In this paper, we introduced the Opus Workflow Evaluation Framework, a unified probabilistic-normative system for measuring and optimizing the quality of AI-driven Workflows. The framework formalizes the Opus Workflow Reward, a probabilistic expectation over success, cost, and gain, and the Opus Workflow Normative Penalties, a continuous set of structural quality measures grounded in Cohesion, Coupling, Observability, and Information Hygiene. Together, these components transform Workflow assessment into a quantitative, optimization-oriented process.

Empirical evaluations confirm that the framework is directly applicable to live Workflow builder systems, where it enables the consistent evaluation, ranking, and refinement of Workflows in production environments. By quantifying performance and structure within a unified formulation, the framework supports objective comparison and optimization across heterogeneous processes. Future work will extend this foundation toward Reinforcement Learning settings, where the Opus Reward and Normative Penalties will serve as feedback signals guiding autonomous Workflow discovery and continuous improvement. This progression will position the Opus Workflow Evaluation Framework as a cornerstone for self-optimizing Workflow automation systems.

References

- [1] Fagnoni, T., Mesbah, B., Altin, M., and Kingston, P. (2024). Opus: A Large Workflow Model for Complex Workflow Generation.
- [2] Kingston, P., Fagnoni, T., and Altin, M. (2025). Opus: A Workflow Intention Framework for Complex Workflow Generation.
- [3] Fagnoni, T., Altin, M., Tuning, A., Mohamed, D., Adnani, I. and Kingston, P. (2025). Opus: A Prompt Intention Framework for Complex Workflow Generation.
- [4] Vanderfeesten, I., Cardoso, J., Mendling, J., Reijers, H., and Van der Aalst, W. (2014). Quality Metrics for Business Process Models.
- [5] Dumas, M., La Rosa, M., Mendling, J., and Reijers, H. (2018). Fundamentals of Business Process Management.
- [6] Krogstie, J. (2016). Quality in Business Process Modeling.
- [7] Cardoso, J. (2005). Control-flow Complexity Measurement of Processes and Weyuker's Properties.
- [8] Polyvyanyy, A., ter Hofstede, A. H. M., La Rosa, M., Ouyang, C., and Pika, A. (2024). Process Query Language: Design, Implementation, and Evaluation.
- [9] Karunaratne, A., Polyvyanyy, A., and Moffat, A. (2024). The Role of Log Representativeness in Estimating Generalization in Process Mining.
- [10] Van der Aalst, W. (2002). Workflow Management: Models, Methods and Systems.
- [11] Weske, M. (2007). Business Process Management: Concepts, Languages, Architectures.
- [12] Scheer, A. W. (1999). ARIS - Business Process Modeling.
- [13] Van der Aalst, W. (2016). Process Mining.

A Appendix

A.1 Conditional Workflows

Our current framework does not natively support conditional branching within Workflows. This design decision was made to preserve the simplicity and tractability of the overall system. However, conditional branching is ubiquitous in real-world scenarios, and therefore it is essential to provide a principled way to incorporate such behavior into our model.

Let us assume that a conditional Workflow W can give rise to a finite set of deterministic execution scenarios $\{W_1, W_2, \dots, W_n\}$, depending on the evaluation of internal conditions. Each W_i represents a fully resolved, deterministic Workflow that may be followed in practice.

To account for the uncertainty inherent in the conditional branching, we associate to each scenario W_i a probability $p_i \in [0, 1]$, such that $\sum_{i=1}^n p_i = 1$. These probabilities reflect the likelihood of each branch being taken, and may either be known a priori, learned from data, or inferred from domain knowledge.

Given a Reward function $\mathcal{R}(W)$ that evaluates deterministic Workflows, we define the Reward of the conditional Workflow W as the expected Reward over all its possible realizations:

$$\mathcal{R}(W) = \sum_{i=1}^n p_i \cdot \mathcal{R}(W_i) \quad (15)$$

Since our original Reward function \mathcal{R} is itself expectation-based, this extension provides a natural and elegant generalization of the framework. Moreover, this formulation is easy to implement in practice, and seamlessly integrates with the existing system.

A.2 Computation of Releasable Resource Peaks

ASAP start-times and finish-times (recurrence). Under an ASAP schedule the start time s_v and finish time f_v of each Task v are defined recursively by

$$s_v = \begin{cases} 0 & \text{if } \text{pred}(v) = \emptyset \\ \max_{u \in \text{pred}(v)} f_u & \text{otherwise} \end{cases} \quad (16)$$

$$f_v = s_v + d_v \quad (17)$$

Since W is a DAG and all d_v are finite, the recurrence (16) defines s_v and f_v uniquely for every v .

For any time $t \in \mathbb{R}^+$ we take the standard half-open convention for activity intervals and define

$$V_t = \{v \in V_{\text{task}} : s_v \leq t < f_v\} \quad (18)$$

Equivalently, a Task v is active at t if and only if $t \in [s_v, f_v]$.

By defining a min-heap \mathcal{H} that stores finish events of the form $(f_v, v, r_v^{(r)})$ (sorted by finish time f_v), we can compute $R^{(r)}(W)$ using an event-driven algorithm. At each step, R_{curr} stores the current sum of active releasable resources, and R_{max} stores the componentwise maximum observed so far.

Algorithm 1 Computation of $R^{(r)}(W)$ under an ASAP schedule

```
1: Initialize pred_count[v]  $\leftarrow |\text{pred}(v)|$  for all  $v \in V$ 
2:  $R_{\text{curr}} \leftarrow 0$ ,  $R_{\text{max}} \leftarrow 0$ ,  $\mathcal{H} \leftarrow \emptyset$ 
3: for all  $v$  with  $\text{pred\_count}[v] = 0$  do
4:    $R_{\text{curr}} += r_v^{(r)}$ 
5:   Push  $(f_v = d_v, v, r_v^{(r)})$  into  $\mathcal{H}$ 
6: end for
7:  $R_{\text{max}} \leftarrow R_{\text{curr}}$ 
8: while  $\mathcal{H}$  not empty do
9:    $(t, v, r_v^{(r)}) \leftarrow \text{Pop earliest event from } \mathcal{H}$ 
10:   $R_{\text{curr}} -= r_v^{(r)}$ 
11:  for all  $w \in \text{succ}(v)$  do
12:     $\text{pred\_count}[w] --= 1$ 
13:    if  $\text{pred\_count}[w] = 0$  then
14:       $R_{\text{curr}} += r_w^{(r)}$ 
15:      Push  $(t + d_w, w, r_w^{(r)})$  into  $\mathcal{H}$   $\triangleright f_w = s_w + d_w$  with  $s_w = t$ 
16:    end if
17:  end for
18:   $R_{\text{max}} \leftarrow \max(R_{\text{max}}, R_{\text{curr}})$ 
19: end while
20: return  $R_{\text{max}}$ 
```

A.3 Single Responsibility Principle (SRP): defining a contextual level of atomicity

The conceptual decomposition of Tasks, ranging from high-level responsibilities (e.g. RACI roles) to atomic units of work (e.g. field-level transactions), is inspired by frameworks used in Business Process Management literature. It outlines four conceptual levels of Task decomposition, ranging from high-level roles (Level 1) to atomic actions (Level 4). Within this framework, we present our notion of singular responsibility not as a fixed level of atomicity, but rather as a contextual one. Depending on the Workflow, the actor, and the system’s perspective, the optimal level of decomposition may vary.

For instance, what qualifies as a single-responsibility Task for an AI system (such as “Send email”) may fall under Level 2 (Task level), whereas a software engineer might model this same Task at Level 4, breaking it down into protocol-level operations (e.g. TCP handshake, SMTP commands). Thus, single responsibility is not an absolute metric; it is relative to the granularity that makes sense for the agent performing the Task.

This aligns with Weske’s interpretation of atomicity [11]: “An activity is atomic if it cannot be sensibly subdivided given the process context and stakeholder objectives.” Hence, the goal is not to find the “true atoms” of work universally, but to identify the optimal level of decomposition for a given purpose and context: striking a balance between meaningful action and implementation detail.

This framework aligns with established BPM literature, where hierarchical Task decomposition is well-documented. Scheer's ARIS [12] methodology employs similar multi-level decomposition from organizational roles down to atomic business Tasks, while Van der Aalst [13] describes comparable layered approaches in process mining, distinguishing between high-level business processes and their constituent atomic activities.

Discrete formalization The Single Responsibility Principle can be formalized as follows.

We define the granularity assignment function as:

$$\psi : V \times \mathcal{W} \rightarrow L = \{L_1, L_2, L_3, L_4\} \quad (19)$$

where V is the ensemble of all valid vertices and \mathcal{W} the ensemble of all valid Workflows, and L the set of four classic levels of granularity.

We define L_W^* as the optimal level of granularity of a Workflow, stating that:

A given Task $v \in V$ in the context W has one clear responsibility at level L_W^* ;

Its subdivision to a deeper level than L_W^* would add confusion and not clarity;

Task v is complete (accomplishes its purpose) at level L_W^* .

For a given W in \mathcal{W} , SRP is valid if and only if:

$$\psi(v, W) = L_W^*, \quad \forall v \in V \quad (20)$$

Continuous formalization and intuition While the discrete definition provides structural clarity, it does not capture the continuum of Task granularity observed in real systems. To address this, we introduce a continuous version of the granularity function:

$$\psi_c : V \times \mathcal{W} \rightarrow [0, 1] \quad (21)$$

where $\psi_c(v, W)$ represents the degree of atomicity of a given Task v within Workflow W .

We define λ_W^* as the optimal continuous granularity target for Workflow W , such that:

$$\psi_c(v, W) = \lambda_W^* \quad \forall v \in V \quad (22)$$

Intuitively:

$\lambda_W^* \approx 1$ implies that nodes should be treated as atomic operations (maximal decomposition);

$\lambda_W^* \approx 0$ implies that nodes should be merged or abstracted into a higher-level responsibility;

Intermediate values express contextual balance between decomposition depth and cohesion.

This continuous mapping allows SRP to interact more easily with optimization-based frameworks. It is straightforward to observe from Eq. 8 that, in order to minimize the CIP, each node's Coupling $C_p(v)$ should approach the value of α_{Ch} . Since $C_p(v)$ can be interpreted as an inverse indicator of atomicity (high Coupling implying lower independence) α_{Ch} can therefore be regarded as the target degree of atomicity of the Workflow.

Formally, we may identify:

$$\lambda_W^* = \alpha_{Ch} \quad (23)$$

Hence, the system's structural balance between cohesion and Coupling directly defines its semantic target level of decomposition.

A.4 Workflow Composition

We define two binary operations on Workflows: parallel composition, denoted by $W_1 \parallel W_2$, and sequential composition, denoted by $W_1 \circ W_2$. Each operation produces a new Workflow $W = (G, \Phi)$, preserving the acyclic graph structure and ensuring semantic consistency of Task dependencies.

Let $W_1 = (G_1, \Phi_1)$, $W_2 = (G_2, \Phi_2)$, with $G_i = (V_i, E_i)$ for $i = 1, 2$. We assume that the node sets V_1 and V_2 are disjoint (if not, they are renamed via canonical relabeling).

Parallel Composition The parallel composition corresponds to the concurrent execution of W_1 and W_2 , without introducing additional control flow between them.

$$W_1 \parallel W_2 = (G, \Phi), \quad \text{where} \\ G = (V_1 \cup V_2, E_1 \cup E_2), \quad \Phi = \Phi_1 \cup \Phi_2$$

This operation preserves the internal structure of both Workflows. Workflow Input, Task, and Output nodes are combined:

$$V_{in} = V_{in}^{(1)} \cup V_{in}^{(2)}, \quad V_{task} = V_{task}^{(1)} \cup V_{task}^{(2)}, \quad V_{out} = V_{out}^{(1)} \cup V_{out}^{(2)}$$

The resulting Workflow executes both components independently. No edges are added between V_1 and V_2 .

Sequential Composition The sequential composition connects two Workflows $W_1 = (G_1, \Phi_1)$ and $W_2 = (G_2, \Phi_2)$, such that the outputs of W_1 feed directly into the inputs of W_2 , bypassing any explicit intermediary nodes. This composition is only defined when the following compatibility condition holds:

$$V_{in}^{(2)} \subseteq V_{out}^{(1)}$$

$V_{in}^{(2)} \cap V_{out}^{(1)} = V_{in}^{(2)}$ is the set of interface nodes to be removed. These nodes are entirely eliminated from the composed Workflow, and all connections to or from them are rewired.

The composed Workflow is:

$$W_1 \circ W_2 = (G, \Phi), \quad \text{with } G = (V, E), \quad \Phi = \Phi_1 \cup \Phi_2$$

where:

$$V = (V_1 \cup V_2) \setminus V_{\text{in}}^{(2)}$$

The edge set E is defined as:

$$E = (E_1 \cup E_2) \setminus (E_{\text{in}} \cup E_{\text{out}}) \cup E_{\text{bridge}}$$

with:

$$\begin{aligned} E_{\text{in}} &= \{(u \rightarrow v) \in E_1 \mid v \in V_{\text{in}}^{(2)}\} \\ E_{\text{out}} &= \{(v \rightarrow w) \in E_2 \mid v \in V_{\text{in}}^{(2)}\} \\ E_{\text{bridge}} &= \{(u \rightarrow w) \mid (u \rightarrow v) \in E_1, (v \rightarrow w) \in E_2, v \in V_{\text{in}}^{(2)}\} \end{aligned}$$

In other words:

All nodes in $V_{\text{in}}^{(2)}$ are removed from the graph.

All edges pointing to or from these nodes are removed.

For each pair of edges $(u \rightarrow v) \in E_1, (v \rightarrow w) \in E_2$, where $v \in V_{\text{in}}^{(2)}$, we introduce a new edge $(u \rightarrow w) \in E_{\text{bridge}}$, thereby directly connecting predecessors of v in W_1 to its successors in W_2 .

This rewiring ensures that the data flow is preserved while eliminating redundant intermediate nodes.

This operation strictly preserves the DAG property of the resulting Workflow, assuming both G_1 and G_2 are DAGs and that the rewiring does not introduce backward edges.

Remarks

Sequential composition is not commutative: $\exists W_1, W_2$ such that $W_1 \circ W_2 \neq W_2 \circ W_1$.

This composition preserves acyclicity, assuming that both G_1 and G_2 are DAGs and that the bridge edges respect temporal ordering.

A.5 Cost Properties

We define two levels of Workflow equivalence:

Weak equivalence, denoted $W_1 \sim W_2$, holds if two Workflows share the same Input and Output interfaces. Their internal procedures (Tasks) or execution behavior may differ.

Strong equivalence, denoted $W_1 \equiv W_2$, holds if two Workflows are weakly equivalent and exhibit identical probabilistic behavior when fully decomposed into atomic actions. They are indistinguishable in terms of both structure and outcome distribution at the most granular level of execution.

Let \mathcal{W} denote the set of all valid Workflows, and let $C : \mathcal{W} \rightarrow \mathbb{R}$ be a cost metric that assigns a real number to each Workflow.

Non-triviality Not all Workflows should have the same cost; at least two Workflows must differ in their cost values.

$$\exists W_1, W_2 \in \mathcal{W} \text{ such that } C(W_1) \neq C(W_2)$$

Implementation Sensitivity Two weakly equivalent Workflows can still have different costs due to differences in structure or implementation.

$$\exists W_1, W_2 \in \mathcal{W} \text{ such that } W_1 \sim W_2 \text{ and } C(W_1) \neq C(W_2)$$

Cost Invariance Two strongly equivalent Workflows must have identical costs.

$$\forall W_1, W_2 \in \mathcal{W}, \quad W_1 \equiv W_2 \Rightarrow C(W_1) = C(W_2)$$

Sub-additivity Combining two Workflows may share resources or eliminate redundancies, resulting in a total cost that is lower than the sum of their individual costs.

$$\forall * \in \{\circ, ||\}, \quad \forall W_1, W_2 \in \mathcal{W}, \quad C(W_1 * W_2) \leq C(W_1) + C(W_2)$$

Context Sensitivity Appending the same Workflow to two Workflows with equal cost may yield different overall costs, depending on the structure of the prefix.

$$\forall * \in \{\circ, ||\}, \quad \exists W, V_1, V_2 \in \mathcal{W} \text{ such that } C(V_1) = C(V_2) \text{ and } C(V_1 * W) \neq C(V_2 * W)$$

Order Sensitivity The order in which sub-workflows are composed affects the total cost; re-ordering operations can increase or reduce cost.

$$\exists W_1, W_2 \in \mathcal{W} \text{ such that } C(W_1 \circ W_2) \neq C(W_2 \circ W_1)$$

Parallel Commutativity There is no order in parallelizing sub-workflows, hence parallelizing in one way or another does not affect the total cost.

$$\forall W_1, W_2 \in \mathcal{W}, \quad W_1 || W_2 \equiv W_2 || W_1$$

By Cost Invariance for strongly equivalent Workflows, it follows that:

$$\forall W_1, W_2 \in \mathcal{W}, \quad C(W_1 || W_2) = C(W_2 || W_1)$$