Design, Potential and Realization in Agentic Systems

AGENTIC SYSTEMS | SYSTEM DESIGN | AUTHORITY | RELIABILITY

ABSTRACT

Existing work usually treats model and tool selection, access control, resource allocation, workflow composition, and evaluation as separate design problems. This separation leaves no common object relating a system's complete pre-execution specification to both its permitted and realized behavior. We introduce a typed Design object that combines models, tools, data access, authority, resources, composition, and control policies. For each task, its contracts define Potential as the set of admitted traces, while its models and policies induce a probability law over realized traces. The paper makes four linked contributions:

- First, we give typed execution semantics in which compatible component traces compose invariantly under the declared coherence conditions, and executed events remain within previously granted authority under the enforcement and transfer assumptions.
- Second, we define reliability-constrained selection using Potential-minimality and Pareto efficiency. The former rules out a reliable Design when another reliable Design admits strictly fewer traces, whereas the latter rules out a reliable Design dominated across cost, latency, exposure, risk, and overhead. Neither criterion implies the other.
- Third, we establish that passive traces cannot identify a permission-minimal reliable repair when unexercised permission extensions induce the same observed law, which makes support necessary for logged evaluation.
- Fourth, for direct evaluation over a finite safe catalog, we derive simultaneous confidence intervals and a sound classification rule that removes Designs shown to be unreliable or dominated while retaining unresolved candidates.

1 Introduction

Consider an invoice-payment system that checks an invoice against purchase records, obtains any required approval, and initiates payment. One agent may propose the payment, another approve it, and a third execute it. Each operates with its own model, tools, permissions, and resource limits. These choices determine not only whether the system pays correctly, but also which actions it can take, which information can cross component boundaries, what a run can cost, and which actions remain possible even when no observed run exercises them. Existing methods often treat these design choices separately: model routing balances quality and cost, access control limits authority, workflow design selects and connects components, and tracing records realized behavior. Each view is useful, but each describes only one consequence of the system's configuration.

We call the complete configuration a *Design*. For each component, a Design specifies its model, tools, data access, authority, resource limits, and enforced contracts, together with how the components are connected. The contracts determine which executions the system admits. The models and policies determine the probabilities of those executions. This separates what the system may do, what it does in a given run, and which strategies can complete a task. A permission may be unused in one run yet essential to another successful strategy, or available without being necessary. Passive traces cannot identify the effect of a permission they never exercise: when two permission configurations make every observed trace equally likely, the data provide no basis for distinguishing them. A Bayesian analysis can prefer one only through its prior or through modeled behavior beyond those observations.

For a task q drawn from a task distribution Q, a Design θ induces two related objects. Its enforced contracts define its admitted *Potential*, $\mathrm{Pot}(\theta \mid q)$, as the set of traces permitted from that task's inputs, initial State, and initial authority. Its models and policies induce a probability law $P_\theta(H \mid q)$ over the realized trace H. This law assigns probability only to admitted traces. A realized measurement map Φ^R, where R denotes realized rather than potential quantities, converts each trace H into its observed outcomes. Their expectations over tasks and traces yield the Design's success probability $p_{\mathrm{succ}}(\theta)$ and objective vector $z(\theta)$:

$$Q \longrightarrow \theta \longrightarrow \{\text{component contracts}, \mathrm{Pot}(\theta \mid q)\} \longrightarrow P_\theta(H \mid q) \longrightarrow \Phi^R(H) \xrightarrow{\mathbb{E}_{q,H}} \{p_{\mathrm{succ}}(\theta), z(\theta)\} \tag{1}$$

We seek Designs that meet the task-specific success criterion with the required probability, exclude unnecessary traces from their Potential, and balance cost, latency, exposure, risk, and overhead. The construction applies recursively to Tools, model-governed Agents, and Systems of Agents. A Tool performs a primitive operation without using a Model, an Agent combines a Model with an iterative policy and callable Tools, and a System composes Agents or other Systems. The same definitions can therefore describe both a single model-driven Agent and a nested System of cooperating Agents.

The formalism connects the traces a Design admits to the probabilities of the traces it realizes. We use this connection to establish when components produce the same combined trace regardless of how their connections are grouped. We also establish when every executed action remains within the authority reachable from the Design's initial permissions and from permissions that admitted actions can create or delegate. We then identify reliable Designs that admit no unnecessary traces or are not outperformed across the measured objectives. Finally, we show why passive traces cannot identify the effect of unexercised permissions and, for a finite catalog, give a conservative rule for removing unreliable or inferior Designs together with a bound on the required number of runs.

These results integrate several established lines of work. Agent-governance research studies formal enforcement, recursive delegation, and conjunctive capability closure [24, 22, 51]. Routing and architecture-search research optimizes the selection and organization of models and teams [7, 38, 21, 44]. We claim neither priority over these areas nor that they are interchangeable. The contribution is a typed Design object that connects admitted behavior, realized execution, reliability, minimality, Pareto selection, and the limits of trace-based inference. A worked invoice system carries these concepts through one finite catalog of candidate Designs. A reproducible, fixed-seed simulation generates the reported intervals from declared probability models. The results illustrate the formalism and do not constitute evidence from a deployed system.

1.1 Notation

Capitalized Task, Tool, Model, Agent, System, Design, Potential, Run, State, and Surface denote formal types. Superscripts P and R distinguish potential from realized quantities. P_θ denotes a probability law, while lowercase p denotes a scalar probability. The table collects recurring notation; each symbol is still defined locally before its first substantive use.

q, Q	Task and Task law	θ	Design
c, C	component and catalog	$H, \mathcal{H}$	trace and trace space
$\mathrm{Pot}(\theta \mid q)$	admitted global traces	$P_\theta(H \mid q)$	induced trace law
F_f^P, F_f^R	potential/realized projection	$z(\theta)$	objective vector
α, δ	success threshold/tolerance	μ_q, Φ^R	score/realized tuple
$s, \mathcal{S}$	State and State space	S_b^R	surfaced data at boundary b
$X, \widehat{X}$	abstract/executed authority	G_S	System dependency graph
$\mathcal{E}_{\mathrm{evt}}, \mathcal{L}_{\mathrm{lbl}}$	events / authority labels	ℓ	event-to-label map
$\mathfrak{L}, \mathcal{L}_X$	authority lattice / labels in X	Δ_e	event authority increment
$\Lambda_{\mathrm{exec}}, \Lambda_{\mathrm{stat}}$	authority expansions	$\mathrm{cl}_{\mathrm{stat}}$	static authority closure
$\mathcal{D}_{\mathrm{haz}}(\theta)$	Design hazard catalog	W	counterfactual kernel world
Δ_{cls}	classification-separation margin	α_{FW}	family-wise error level
$A \Rightarrow b$	validated Horn authority rule	$\mathcal{U}_Q$	Task-law uncertainty set
Ψ_G	group-success functional	$\rho(m, \kappa)$	Model-cost function

2 Typed execution model

2.1 State and events

Let $\mathcal{T}$ be a set of data types and let $\mathcal{V}_t$ be the value space of type $t \in \mathcal{T}$. Each object identity $o \in O$ has a declared type $\tau(o) \in \mathcal{T}$, and $\mathcal{V} = \bigsqcup_{t \in \mathcal{T}} \mathcal{V}_t$ is the disjoint union of the typed value spaces. A State is a type-respecting partial map

$$s : O \rightharpoonup \mathcal{V}, \qquad o \in \mathrm{dom}(s) \Longrightarrow s(o) \in \mathcal{V}_{\tau(o)}$$

Thus s assigns a current value to each object that exists in that State. The partial-map arrow indicates that an identity in O may not yet exist and may therefore have no value in s. The State space $\mathcal{S}$ contains all such maps allowed by the application schema.

Let C be the catalog of callable components. Each $c \in C$ has an input space $\mathcal{X}_c$ and an output space $\mathcal{Y}_c$. Let $\mathcal{B}$ be the set of communication boundaries, $\mathcal{P}_b$ the payload space at boundary $b \in \mathcal{B}$, and $\mathcal{R}$ the set of resource types. Each resource $r \in \mathcal{R}$ has a nonnegative amount space $\mathcal{A}_r$ carrying its declared unit, such as tokens, seconds, or currency.

Event occurrences require stable identities because two events may have the same type and arguments. Let $\mathcal{I}$ be a countable set of event identifiers and let $\mathrm{Fin}(A)$ denote the finite subsets of a set A. A provenance record $\nu \in \mathrm{Fin}(O \sqcup \mathcal{I})$ names the objects and earlier event occurrences that contributed to a surfaced payload.

The event-label space $\mathcal{E}_{\text{evt}}$ is the disjoint union generated by the following constructor signatures:

$$\begin{aligned}
\text{Read} &: \bigsqcup_{o \in \mathcal{O}} \{o\} \longrightarrow \mathcal{E}_{\text{evt}} \\
\text{Modify} &: \bigsqcup_{o \in \mathcal{O}} \{o\} \times \mathcal{V}_{\tau(o)} \times \mathcal{V}_{\tau(o)} \longrightarrow \mathcal{E}_{\text{evt}} \\
\text{Create, Delete} &: \bigsqcup_{o \in \mathcal{O}} \{o\} \times \mathcal{V}_{\tau(o)} \longrightarrow \mathcal{E}_{\text{evt}} \qquad (2) \\
\text{Call} &: \bigsqcup_{c \in \mathcal{C}} \{c\} \times \mathcal{X}_c \longrightarrow \mathcal{E}_{\text{evt}} \\
\text{Return} &: \bigsqcup_{c \in \mathcal{C}} \{c\} \times \mathcal{Y}_c \longrightarrow \mathcal{E}_{\text{evt}} \\
\text{Surface} &: \bigsqcup_{b \in \mathcal{B}} \{b\} \times \mathcal{P}_b \times \text{Fin}(\mathcal{O} \sqcup \mathcal{I}) \longrightarrow \mathcal{E}_{\text{evt}} \\
\text{Consume} &: \bigsqcup_{r \in \mathcal{R}} \{r\} \times \mathcal{A}_r \longrightarrow \mathcal{E}_{\text{evt}} \qquad (3)
\end{aligned}$$

Because this is a disjoint union, the constructor and its argument types are part of every label. For example, $\text{Modify}(o, v_{\text{old}}, v_{\text{new}})$ requires $v_{\text{old}}, v_{\text{new}} \in \mathcal{V}_{\tau(o)}$, while $\text{Call}(c, x)$ requires $x \in \mathcal{X}_c$ and $\text{Return}(c, y)$ requires $y \in \mathcal{Y}_c$.

The control-event type set is denoted by $\mathcal{E}_{\text{ctl}}$:

$$\mathcal{E}_{\text{ctl}} = \{\text{Start}, \text{Stop}, \text{Failure}, \text{Denial}, \text{Approval}, \text{Delegation}\}$$

Each $k \in \mathcal{E}_{\text{ctl}}$ has an application-declared payload space $\mathcal{Z}_k$, and the control labels form $\bigsqcup_{k \in \mathcal{E}_{\text{ctl}}} \{k\} \times \mathcal{Z}_k$. The full space $\mathcal{E}_{\text{evt}}$ is the disjoint union of these control labels and the State, interface, Surface, and resource labels defined above. A payload-free event uses a singleton $\mathcal{Z}_k$, in which case its argument is suppressed. This convention permits, for example, a Denial event to identify the rejected proposal and a Delegation event to identify its grant without imposing one application-specific schema on every Design.

An event occurrence is a pair $(i, e) \in \mathcal{I} \times \mathcal{E}_{\text{evt}}$ with a unique identifier i and typed label e. For a Surface label $\text{Surface}(b, p, v)$, every event identifier in v must resolve to one unique causal predecessor. Surface is distinct from Read: an object may be read without its value crossing a component boundary.

The four State-event constructors have the following transition semantics, where $s[o \mapsto v]$ updates or inserts o and $s \setminus \{o\}$ removes it:

$$\begin{aligned}
s \xrightarrow{\text{Read}(o)} s &\iff o \in \text{dom}(s) \\
s \xrightarrow{\text{Modify}(o, v_{\text{old}}, v_{\text{new}})} s' &\iff s(o) = v_{\text{old}} \text{ and } s' = s[o \mapsto v_{\text{new}}] \\
s \xrightarrow{\text{Create}(o, v)} s' &\iff o \notin \text{dom}(s) \text{ and } s' = s[o \mapsto v] \\
s \xrightarrow{\text{Delete}(o, v)} s' &\iff s(o) = v \text{ and } s' = s \setminus \{o\} \qquad (4)
\end{aligned}$$

The stored old or deleted value supports audit, reversibility analysis, and effect provenance. At this layer, Call, Return, Surface, Consume, and control events leave the modeled State unchanged. Their interface, resource, authority, and application effects enter through the contracts defined below.

Definition 1 (Trace)**.** A trace $H = (E_H, \prec_H)$ contains a finite set $E_H \subseteq \mathcal{I} \times \mathcal{E}_{\text{evt}}$ of event occurrences with unique identifiers and a strict partial order $\prec_H$ on E_H. The order records component-local execution, call-and-return causality, and declared precedence constraints. Events not ordered relative to one another may occur concurrently. The trace is well formed only when its State-event restrictions satisfy the transition semantics above and every provenance reference points to a causal predecessor.

A Run is one realized execution, represented by its sampled trace, outputs, and Task score.

Definition 2 (Compatible trace composition)**.** Let $A \sqcup B$ denote disjoint union, which preserves the source identity of elements even when their labels coincide. For traces $H_i = (E_i, \prec_i)$ whose event identifiers have been consistently renamed, define

$$H_2 \circ H_1 = (E_1 \sqcup E_2, \mathrm{tc}(\prec_1 \cup \prec_2 \cup \prec_I)) \tag{5}$$

The relation $\prec_I$ contains the interface edges declared by the Design. These edges connect matching caller and callee events, enforce call-and-return order, and record declared data dependencies. Caller and callee interface events remain separate typed events. Composition is defined only when:

1. Caller and callee events match in type and label.
2. The combined order contains no causal cycle.
3. Concurrent State effects satisfy the Design's conflict rule.

The operator tc takes the transitive closure of the combined order. Thus, if one event precedes a second and the second precedes a third, the closure also records that the first precedes the third. A conflict rule may, for example, require classical transaction serializability [40].

Assumption 1 (Coherent finite compatibility)**.** The Design declares one compatibility predicate for every finite family of traces. Every subfamily of a compatible family is compatible and inherits the same event matching, interface edges, and interpretation of State effects. An incompatible family is rejected rather than assigned a meaning that depends on how its traces are grouped.

For three traces H_1, H_2, H_3, simultaneous composition first forms the disjoint union of all three event sets and inserts every declared interface edge. It then checks types, State conflicts, and acyclicity for the complete group before taking the transitive closure. The three traces are compatible exactly when this simultaneous construction is valid.

Lemma 1 (Grouping does not change compatible composition)**.** *Under Assumption 1, suppose each interface event has a unique match across three well-typed traces and their simultaneous composition is valid. Both binary groupings are then defined, and*

$$(H_3 \circ H_2) \circ H_1 \cong H_3 \circ (H_2 \circ H_1)$$

Here $\cong$ means that the two results contain the same labelled event structure up to an isomorphism that preserves event identifiers and types.

Proof. Assumption 1 permits every two-trace restriction required by either grouping. Both groupings retain the same typed events and insert the same Design-declared edges. Because the complete compatibility check has already accepted their common interpretation of State effects, each grouping takes the transitive closure of the same acyclic relation. That closure is unique. ∎

These conditions permit recursive composition. The same argument shows that every grouping of a compatible finite collection of traces produces an isomorphic result.

Remark 1 (Measurable trace space)**.** Assume that every primitive event attribute ranges over a standard Borel space. Each event type inherits the corresponding product σ-algebra, and the complete event space is their measurable disjoint union. Finite labelled partial orders then inherit a measurable structure denoted by $\mathscr{H}$.
A component transition kernel maps into

$$(\mathcal{Y}_c \times \mathcal{S} \times \mathcal{H}, \mathscr{Y}_c \otimes \mathscr{S} \otimes \mathscr{H})$$

where $\mathcal{Y}_c$ is the component's output space. The success, exposure, risk, and resource measurements used later are measurable on this space. Well-formed traces form a measurable subset, and traces that differ only by a consistent renaming of event identifiers represent the same execution.

The realized measurement map introduced in the Introduction returns the tuple

$$\Phi^R(H) = \big(F^R(H), B^R(H), C^R(H), L(H), E^R(H), K^R(H), O^R(H), Y(H)\big) \tag{6}$$

The coordinates record:

- $F^R(H)$: realized effect footprints
- $B^R(H)$: realized resource consumption
- $C^R(H)$: realized monetary or deployment cost
- $L(H)$: realized latency
- $E^R(H)$: realized exposure
- $K^R(H)$: realized harm or risk
- $O^R(H)$: realized coordination overhead
- $Y(H)$: designated outputs

The output $Y(H)$ contains the final component returns and recipient-facing Surface payloads designated by the Design. Coordination overhead $O^R(H)$ measures the Call, Return, Surface, and Consume events attributable to decomposition, orchestration, communication, verification, and aggregation. An application may include the corresponding resource and latency costs in $B^R(H)$ and $L(H)$, but it must not count them again in $O^R(H)$.

Realized effect footprints are projections that select events of a given type from H. For example,

$$R^R(H) = \mathrm{proj}_{\mathrm{Read}}(H), \qquad S_b^R(H) = \mathrm{proj}_{\mathrm{Surface}(b,\cdot,\cdot)}(H) \tag{7}$$

Thus $R^R(H)$ contains the realized Read events, while $S_b^R(H)$ contains the payloads surfaced across boundary b. Relevant boundaries include Tool-to-Agent, Agent-to-System, and System-to-recipient communication. State accessed by Tools remains part of the State and data model rather than forming another architectural level.

Proposition 1 (Typed footprint projection)**.** *Every realized effect footprint is obtained by selecting one event type from H, and every Surface projection retains its communication boundary and provenance. For compatible traces,*

$$\mathrm{proj}_f(E_{H_2 \circ H_1}) = \mathrm{proj}_f(E_{H_1}) \cup \mathrm{proj}_f(E_{H_2})$$

This equality concerns which events belong to each footprint. It does not claim that an order-sensitive projection remains unchanged when events or dependencies are hidden.

Proof. Every event retains one type and its associated attributes, so selecting events by type defines each footprint. Compatible composition preserves the events of both traces through disjoint union. Type selection therefore distributes over the composed event set, while the additional order edges leave event membership unchanged. ∎

2.2 Tasks and successful strategies

Definition 3 (Task)**.** A Task is a tuple

$$q = (x_q, s_{0,q}, \Gamma_q, \mu_q)$$

where x_q is the Task input, $s_{0,q} \in \mathcal{S}$ is its initial State, Γ_q is its constraint set, and $\mu_q : \mathcal{H} \to [0,1]$ is a measurable success score.

The measurable Task space is denoted by $\mathcal{Q}$, and a Task law Q is a probability measure on that space. Because the initial State is part of q, the law Q determines the deployment distribution of both inputs and initial States.

Assumption 2 (Valid root invocation)**.** For a fixed Design, let $\mathcal{Q}_0 \subseteq \mathcal{Q}$ contain the Tasks that satisfy the root component's invocation precondition. The deployment law satisfies

$$Q(\mathcal{Q}_0) = 1$$

This assumption restricts the deployment population to Tasks for which the Design is defined. It does not assert that the Design completes those Tasks successfully.

Definition 4 (Successful trace set)**.** For a declared threshold $\alpha \in [0,1]$, the successful traces for Task q are

$$\mathcal{H}_q^\alpha = \{H \in \mathcal{H} : \mu_q(H) \geq \alpha\} \tag{8}$$

Success is therefore a property of the complete trace relative to a Task, not of one fixed sequence of actions. Two traces in $\mathcal{H}_q^\alpha$ may complete the same Task with different Models, Tools, permissions, or resources. The formalism treats them as distinct successful strategies rather than assuming one necessary footprint.

2.3 Callable components and contracts

For each component $c \in C$, let $\mathcal{H}_c \subseteq \mathcal{H}$ be its trace space with measurable sets $\mathscr{H}_c$. A call begins with an input $x \in \mathcal{X}_c$ and State $s \in \mathcal{S}$, then returns an output $y \in \mathcal{Y}_c$, a resulting State $s' \in \mathcal{S}$, and a component trace $H_c \in \mathcal{H}_c$. Let $\mathcal{R}_{\text{hard}} \subseteq \mathcal{R}$ contain the resource types whose limits the monitor enforces on every trace. Section 2.4 defines the remaining stochastic resource types.

Definition 5 (Callable component)**.** A callable component is

$$c = (\mathcal{X}_c, \mathcal{Y}_c, \text{Pre}_c, K_c, \text{Cap}_c, \text{Auth}_c, B_c^{\max}, \text{Inv}_c, M_c) \tag{9}$$

where

$$\begin{aligned} \text{Pre}_c &\subseteq \mathcal{X}_c \times \mathcal{S} \\ K_c(\cdot \mid x, s) &\text{ is a probability kernel on } \mathcal{Y}_c \times \mathcal{S} \times \mathcal{H}_c \\ \text{Cap}(c \mid x, s), \text{Auth}(c \mid x, s) &\in \mathscr{H}_c \\ B_c^{\max} &= (B_{c,r}^{\max})_{r \in \mathcal{R}_{\text{hard}}} \\ \text{Inv}_c &\in \mathscr{H}_c \end{aligned}$$

and M_c is the runtime monitor that enforces the component contract.

The precondition Pre_c specifies valid input–State pairs. $\text{Cap}(c \mid x, s)$ contains the traces the component can structurally produce, while $\text{Auth}(c \mid x, s)$ contains those permitted by its authority contract. The hard limits $B_c^{\max}$ bound enforceable resource consumption. The invariant Inv_c covers schema and type validity, interface protocol, provenance, call-and-return matching, preconditions, postconditions, and application-specific safety properties.

Assumption 3 (Declared capability supports execution)**.** For every $(x, s) \in \text{Pre}_c$,

$$K_c\{H_c \in \text{Cap}(c \mid x, s) \mid x, s\} = 1$$

Thus the declared structural capability contains the support of the component's trace kernel. It may also contain admitted traces to which the kernel assigns probability zero.

For each hard resource $r \in \mathcal{R}_{\text{hard}}$, let $B_r^R : \mathcal{H}_c \to \mathcal{A}_r$ measure the amount consumed in a realized component trace.

Definition 6 (Contract trace sets)**.** The hard-budget and invariant trace sets are

$$\text{Budget}(c) = \{H \in \mathcal{H}_c : B_r^R(H) \le B_{c,r}^{\max} \text{ for every } r \in \mathcal{R}_{\text{hard}}\} \tag{10}$$

$$\text{Valid}(c) = \text{Inv}_c \tag{11}$$

Stochastic resource limits constrain the trace law rather than membership in $\text{Budget}(c)$. Section 2.4 distinguishes the two forms of resource constraint.

Definition 7 (Component Potential)**.** For a valid call $(x, s) \in \text{Pre}_c$, the component's admitted Potential is

$$\text{Pot}(c \mid x, s) = \text{Cap}(c \mid x, s) \cap \text{Auth}(c \mid x, s) \cap \text{Budget}(c) \cap \text{Valid}(c) \subseteq \mathcal{H}_c \tag{12}$$

This intersection separates four questions: whether a trace is structurally possible, authorized, within hard resource limits, and valid under the remaining contract. The paper optimizes their enforced intersection without treating the four questions as interchangeable. Effect signatures and stateful contracts provide foundations for this construction [31, 47]. Recent Agent-governance work also separates capability from permission [57], while Progent provides programmable privilege control at Tool boundaries [50].

Assumption 4 (Safety and rejection closure)**.** For every valid (x, s), the sets $\mathrm{Cap}(c \mid x, s)$, $\mathrm{Auth}(c \mid x, s)$, and Inv_c contain the empty trace and are closed under prefixes. Hard resource consumption is nonnegative and nondecreasing along a trace. If $h \in \mathrm{Pot}(c \mid x, s)$ and M_c rejects the next proposed event, appending the corresponding Denial event remains in all three sets and in $\mathrm{Budget}(c)$. The monitor either charges no hard resource for this audit event or reserves enough hard budget to emit it.

Proposition 2 (Prefix closure of component Potential)**.** *Under Assumption 4,* $\mathrm{Pot}(c \mid x, s)$ *contains the empty trace and is closed under prefixes, including traces truncated by the monitor.*

Proof. The capability, authority, and invariant sets are prefix-closed by assumption. Nonnegative, nondecreasing consumption makes the hard-budget set prefix-closed. Their intersection is therefore prefix-closed, and the rejection clause keeps the final Denial event of a monitor-truncated trace inside every constituent set. ∎

Potential specifies admissibility and safety. Successful completion is a liveness property evaluated separately through the Task score μ_q and the required success probability.

Assumption 5 (Boundary enforcement)**.** At every callable-component boundary, M_c enforces $\mathrm{Auth}(c \mid x, s)$, $\mathrm{Budget}(c)$, and Inv_c. When a proposed event violates one of these sets, the monitor emits the admitted Denial event and terminates the component trace before the rejected event executes.

This enforcement pattern includes Designs that isolate trusted control flow from untrusted Model outputs [11].

Proposition 3 (Almost-sure local admissibility)**.** *Under Assumptions 3, 4, and 5, the trace of a primitive Tool satisfies*

$$K_c\{H_c \in \mathrm{Pot}(c \mid x, s) \mid x, s\} = 1$$

for every valid call $(x, s) \in \mathrm{Pre}_c$.

Proof. Assumption 3 places the trace in the capability set almost surely. Boundary enforcement places it in the authority, budget, and invariant sets, including the monitor-truncated case. The trace therefore belongs to their intersection almost surely. ∎

Assumption 6 (Grant-before-use causality)**.** Every event that uses delegated, created, or minted authority has a precedence edge from the event that grants that authority. Initial grants come from the Task's initial State. The Design's conflict or isolation rule determines one invocation State from each causal past, using $s_{0,q}$ at the empty prefix.

2.4 Resources

The resource set is partitioned into hard and stochastic dimensions:

$$\mathcal{R} = \mathcal{R}_{\mathrm{hard}} \sqcup \mathcal{R}_{\mathrm{stoch}}$$

The disjoint union means that each resource dimension uses exactly one constraint type. A hard dimension represents a limit that the monitor can enforce on every run, while a stochastic dimension represents a probabilistic service or risk requirement.

For $r \in \mathcal{R}_{\mathrm{hard}}$, the component contract requires

$$B_r^R(H_c) \le B_{c,r}^{\max} \quad \text{almost surely} \tag{13}$$

Here $B_r^R(H_c)$ is the realized amount consumed and $B_{c,r}^{\max}$ is the enforced upper limit for component c.

For $r \in \mathcal{R}_{\mathrm{stoch}}$, the contract may instead impose

$$\mathbb{P}\{B_r^R(H) > b_r \mid q, \theta\} \le \varepsilon_r \tag{14}$$

The threshold b_r is the declared resource level and $\varepsilon_r \in [0, 1]$ is the largest permitted probability of exceeding it. Unlike a hard limit, this condition is evaluated under the Design's task-conditioned trace law and does not remove individual traces from Potential.

Provision and forecast consumption are separate quantities. The provision B_r^{prov} records the amount reserved before execution, while $\widehat{\mathbb{E}}[B_r^R \mid q, \theta]$ is the pre-execution forecast of average realized consumption. Neither quantity is the hard limit $B_{c,r}^{\max}$ or the stochastic threshold b_r, and the realized trace may consume less than the provision or differ from its forecast.

2.5 Tools

Definition 8 (Tool)**.** A Tool $\tau \in C$ is a primitive callable component. Its execution does not invoke another component through the component interface.

The Tool kernel may be deterministic or stochastic and may contain internal steps, failure, State changes, external dependence, or asynchronous events. Primitive means that these effects occur within one callable-component boundary, not that the Tool's implementation is simple.

The realized Tool footprint collects five event-type projections:

$$F^R_\tau(H_\tau) = \big(F^R_{\tau,\mathrm{Read}}(H_\tau), F^R_{\tau,\mathrm{Modify}}(H_\tau), F^R_{\tau,\mathrm{Create}}(H_\tau), F^R_{\tau,\mathrm{Delete}}(H_\tau), (F^R_{\tau,\mathrm{Surface}(b)}(H_\tau))_{b\in\mathcal{B}}\big) \tag{15}$$

Each coordinate is the corresponding event projection, for example $F^R_{\tau,\mathrm{Read}}(H_\tau) = \mathrm{proj}_{\mathrm{Read}}(H_\tau)$. The Surface family is indexed by b so that the footprint retains which communication boundary each payload crossed. Realized cost remains the separate measurement $C^R(H_\tau)$, and the Tool's admitted footprints are bounded by $\mathrm{Pot}(\tau \mid x, s)$.

Preconditions and transition semantics belong to the Tool contract rather than to annotations outside it. Effectful and stateful contracts are established foundations [31, 47]; the use made here is their integration into the common Design, Potential, and realization model.

2.6 Models and task-conditioned capability

Definition 9 (Model configuration)**.** A Model configuration is

$$m = (\mathrm{id}_m, \iota_m, \chi_m, \zeta_m, \rho_m)$$

where id_m is the Model identity, ι_m its inference configuration, χ_m its context policy, ζ_m its decoding and action policy, and ρ_m its resource and cost profile.

For a required capability or complexity level κ, the function $\rho(m, \kappa)$ evaluates the resource and cost profile of Model configuration m. The symbol κ is used here because c already denotes a callable component.

Let η collect every Agent choice other than its Model, including its Tools, policies, contract, and resource envelope. The task-conditioned success probability is

$$p_{\mathrm{succ}}(m, q, \eta) = \mathbb{P}\{H \in \mathcal{H}^\alpha_q \mid m, q, \eta\} \tag{16}$$

Model capability is therefore relational: it depends on the Task and on the Agent configuration in which the Model operates. One Model may have a higher success probability on some Tasks and a lower probability on others, so the formalism does not assume a global ranking of Models.

The relative difficulty of Task q for fixed (m, η) is

$$d(q; m, \eta) = -\operatorname{logit} p_{\mathrm{succ}}(m, q, \eta) \tag{17}$$

where $\operatorname{logit}(p) = \log(p/(1-p))$. The value is $+\infty$ when success probability is zero and $-\infty$ when it is one. Because d depends on m and η, it is not an intrinsic scalar property of the Task. A later appendix uses a scalar-margin specialization only for conditional architecture illustrations.

Cost-aware Model routing and cascades address related assignment problems [7, 38]. Here Model assignment is one coordinate of the joint Design choice.

Assumption 7 (Common Model contract)**.** Candidate Models substituted within one Agent share its externally declared component contract: admitted action and output events, context and interface bounds, capability trace envelope, authorization, invariants, hard resource envelope, and authority labels. A substitution may change the transition kernel and realized stochastic costs but not these contract fields.

A Model outside the common contract changes the Agent contract and may change its Potential. The conditional architecture calculations exclude that case rather than treating it as a Model-only substitution.

2.7 Agents, Systems, and generation

Definition 10 (Agent). An Agent is a Model-governed callable component

$$a = (m_a, T_a, \mathrm{gen}_a, \mathrm{val}_a, \pi_a, \sigma_a, B_a^{\max}) \tag{18}$$

where m_a is its Model, $T_a \subseteq C$ its callable Tool set, gen_a an optional generator, val_a its validator, π_a its action policy, σ_a its stopping policy, and $B_a^{\max}$ its resource envelope.

The resource envelope $B_a^{\max}$ instantiates the hard limits of the general callable-component contract. It is not a second independent budget.

Definition 11 (System). A System is

$$S = (A_S, G_S, \Pi_S, \Sigma_S, B_S^{\max}) \tag{19}$$

where $A_S \subseteq C$ contains its callable Agents or nested Systems, G_S is their dependency graph, Π_S is the delegation and scheduling policy, Σ_S is the verification, aggregation, and stopping policy, and $B_S^{\max}$ is the System resource envelope.

A System may designate one member $a_0 \in A_S$ as an orchestration Agent. That Agent uses its own Model and Tools while calling specialist Agents as components. The definition also permits peer communication or a non-Agent controller, so the presence of an orchestration Agent does not give the System an additional intrinsic Model.

Agents and Systems may generate components before those components enter the callable catalog. Let $\widetilde{C}$ be the candidate-component space and let $h \in \mathcal{H}$ be the prior trace available to generation and validation. The three stages are typed as

$$\mathrm{gen} : \mathcal{H} \to \widetilde{C}, \qquad \mathrm{val} : \widetilde{C} \times \mathcal{H} \to C \cup \{\bot\}, \qquad \mathrm{Adm} : C \times \mathcal{H} \to \{0, 1\} \tag{20}$$

The generator maps h to candidate $\widetilde{c}$. The validator maps $(\widetilde{c}, h)$ either to a typed component c or to the rejection symbol $\bot$. Even a validated component enters C and becomes callable only when $\mathrm{Adm}(c, h) = 1$.

Generated source code, compositions of existing Tools, parameterized contracts, and generated Agents belong to distinct validation classes. Each class declares its own validator and admission conditions rather than inheriting validity from the fact that some generator produced it.

3 Composition and system-level Potential

Components c_1 and c_2 compose when an output of c_1 satisfies the input type and invocation precondition of c_2. Their Call, Return, and Surface events combine under Definition 2. Other quantities require laws appropriate to their meaning:

Quantity	Composition law
State effects	ordered State transitions with a declared conflict rule
Resources	dimension-specific sum, maximum, peak, or stochastic functional
Latency	critical path plus coordination and queuing
Authority	typed reachability through monotone closure
Success	Task score after verification and aggregation
Risk	a general functional of Potential and component interactions

The table does not impose one algebra on all Design quantities. It records why trace composition alone is insufficient: State is ordered, resources depend on units, latency depends on concurrency, authority propagates through reachability, and risk may be non-additive.

3.1 Typed authority closure

Let $\mathcal{L}_{\mathrm{lbl}}$ be the finite catalog of abstract event labels. A label records the contract-relevant class of an event, such as a State region and operation, a callable component, a credential class, a delegation type, an approval authority, or a Surface boundary. The map

$$\ell : \mathcal{E}_{\mathrm{evt}} \to \mathcal{L}_{\mathrm{lbl}}$$

assigns one label to each event label from Section 2.1. For an event occurrence (i, e), write $\ell(i, e) = \ell(e)$ because its identifier i does not change its abstract label.

Let $\mathcal{A}$ be the finite catalog of typed authority facts. An authority fact records a permission, credential, delegation edge, or comparable fact from which component or event reachability can follow. Event labels describe what an event does, while authority facts describe why that event may become reachable.

Definition 12 (Authority state)**.** An authority state is

$$X = (C_X, \mathcal{L}_X, \mathcal{A}_X) \in \mathfrak{L} = 2^C \times 2^{\mathcal{L}_{\text{lbl}}} \times 2^{\mathcal{A}} \tag{21}$$

The order $X \preceq X'$ is componentwise set inclusion, and the join $X \vee X'$ is componentwise union.

Thus C_X contains reachable components, $\mathcal{L}_X$ contains reachable event labels, and $\mathcal{A}_X$ contains accumulated authority facts. The catalogs are finite, so the authority lattice $\mathfrak{L}$ is finite. The closure results below use the classical monotone-lattice construction [53].

The validated authority catalog may contain a conjunctive rule

$$A \Rightarrow b, \qquad A = \{a_1, \ldots, a_k\} \subseteq \mathcal{A}, \quad b \in \mathcal{A}$$

The rule adds authority fact b only after every antecedent in A is present. These Horn-style rules represent AND dependencies rather than treating each antecedent as sufficient on its own. Their authorization lineage includes Delegation Logic, RT, and SecPAL [32, 33, 5], and their least-model computation is classical [13].

Definition 13 (Executed authority expansion)**.** For $X \in \mathfrak{L}$, $\Lambda_{\text{exec}}(X)$ is X together with:

1. each component in C callable through an authority fact in $\mathcal{A}_X$;
2. each label $\ell(e)$ admitted by $\text{Pot}(c \mid x, s)$ for some valid call of a component c in the expanded reachable-component set;
3. each consequent of a validated conjunctive rule whose antecedents are in the expanded authority-fact set.

The executed expansion uses only authority facts already accumulated from the initial State or executed events. It may expose the admitted labels of a reachable component, but it does not add a credential, delegation, component, or authority fact merely because some future event could create it.

Definition 14 (Static authority expansion)**.** For $X \in \mathfrak{L}$, $\Lambda_{\text{stat}}(X)$ applies every rule of Λ_{exec} and also adds authority facts, credentials, and components that an admitted Create or Delegation event can produce after validation and admission.

Both operators only add coordinates, so they are inflationary and monotone, and $\Lambda_{\text{exec}}(X) \preceq \Lambda_{\text{stat}}(X)$ for every X. They adapt compositional authorization, Agent information-flow, and classical rights-propagation analysis [22, 24, 19, 34]. The paper uses them as Design coordinates rather than claiming authority closure as a new standalone construction.

Definition 15 (Event authority transfer)**.** Each validated event occurrence e has a constant lattice increment $\Delta_e \in \mathfrak{L}$. Its authority transfer is

$$T_e(X) = X \vee \Delta_e$$

The increment depends on the typed event label and validated payload, not on X.

An ordinary event has an empty increment. A Delegation, credential-minting Create, or admitted generation event may add the authority fact or component recorded by its validated payload.

Lemma 2 (Commuting event transfers)**.** *The event transfers are monotone and commute.*

Proof. Lattice join is monotone, associative, and commutative. Hence

$$T_e(T_f(X)) = X \vee \Delta_f \vee \Delta_e = T_f(T_e(X))$$

Definition 16 (Accumulated prefix authority)**.** Let $X^0_{\theta,q}$ contain the root components and initial authority facts declared by Design θ for Task q. For a finite downward-closed executed prefix h and an event e maximal in $h \cup \{e\}$, define

$$X_{\theta,q}(\varnothing) = X^0_{\theta,q}, \qquad X_{\theta,q}(h \cup \{e\}) = T_e(X_{\theta,q}(h)) \tag{22}$$

A prefix is downward-closed when it contains every causal predecessor of each event it contains. Lemma 2 makes $X_{\theta,q}(h)$ independent of the linear order chosen to process concurrent events [35].

Definition 17 (Executed authority closure)**.** The authority reachable after prefix h is

$$\widehat{X}_{\theta,q}(h) = \mathrm{cl}_{\mathrm{exec}}(X_{\theta,q}(h)) = \bigvee_{n\geq 0} \Lambda^n_{\mathrm{exec}}(X_{\theta,q}(h)) \tag{23}$$

The join is the least authority state containing every iterate. Because the iterates form an increasing sequence, it equals their coordinatewise union. A component added by T_e was executed or created in the prefix, while one added only by $\mathrm{cl}_{\mathrm{exec}}$ is callable from authority already accumulated in that prefix.

The authority semantics is monotone and does not model revocation or expiring credentials. Those mechanisms require time-indexed authority or separate grant and revoke transitions because removing authority would break the increasing sequence used by both closures.

Definition 18 (Static authority closure)**.** The static authority closure above X is

$$\mathrm{cl}_{\mathrm{stat}}(X) = \bigvee_{n\geq 0} \Lambda^n_{\mathrm{stat}}(X) \tag{24}$$

Proposition 4 (Finite static authority closure)**.** *Iteration from X reaches $\mathrm{cl}_{\mathrm{stat}}(X)$ in at most the height of $\mathfrak{L}$. The union of initially assigned authority coordinates equals this closure only when that union is already a fixed point of Λ_{stat}.*

Proof. Inflationarity and monotonicity produce an ascending chain. Every ascending chain in the finite lattice $\mathfrak{L}$ stabilizes within its height. The stable state is a fixed point, and induction places every iterate below every fixed point above X. ∎

Assumption 8 (Static transfer domination)**.** Every admitted event transfer is represented in the validated static catalogs:

$$T_e(X) \preceq \Lambda_{\mathrm{stat}}(X)$$

for every abstract authority state X.

Lemma 3 (Executed authority is statically bounded)**.** *Under Assumption 8, every downward-closed executed prefix h satisfies*

$$\widehat{X}_{\theta,q}(h) \preceq \mathrm{cl}_{\mathrm{stat}}(X^0_{\theta,q})$$

Proof. Induct over a linear extension of h. The initial state is below its static closure. Assumption 8 bounds each event transfer, while $\Lambda_{\mathrm{exec}}(X) \preceq \Lambda_{\mathrm{stat}}(X)$ bounds each reachability step. Monotonicity preserves the bound after every transfer and closure. ∎

Remark 2 (Authority closure is an overapproximation)**.** Static authority closure can forget event order, invocation preconditions, history-dependent authorization, correlations, and mutually exclusive branches. It therefore bounds reachable authority but does not define the exact contract-admissible trace set or the support of the execution law.

The executed closure also admits every label available under some valid call of a reachable component, even when the actual call in h does not admit that label. Global Potential removes this call-uniform overapproximation by checking the contract of each actual invocation. If a Design omits a conjunctive rule, both closures are relative to the remaining declared abstraction and do not cover that unmodeled AND dependency. This is the boundary identified by non-compositional capability-closure results [51].

Proposition 5 (Executed-event authority containment). *Suppose Assumptions 5, 6, 4, and 8 hold, component boundaries are contract-compatible, and every generated component or credential is validated and admitted into its declared catalog. If $X^0_{\theta,q}$ contains every root component, initial authority fact, and environment-issued root-invocation label declared by the Design, then every executed event e satisfies*

$$\ell(e) \in \mathcal{L}_{\widehat{X}_{\theta,q}(H_{\prec e})}$$

where $H_{\prec e}$ is the causal-past prefix of e.

Proof. Induct over a topological order of the finite trace. Initial authority licenses each root call, and local contracts admit ordinary primitive effects. For a Call, authority in the causal-past state makes the callee reachable and adds its admitted labels. A Delegation transfer adds delegated authority only after the Delegation event. A credential-minting Create or admitted generation event similarly adds its validated result only after that event. A conjunctive rule fires only when its final antecedent enters the accumulated authority state.

Assumption 6 places every grant before its use. Ordinary events have empty authority increments. A rejected proposal does not execute; the monitor emits the admitted Denial event and truncates the trace under Assumptions 5 and 4. Compatible interfaces preserve the event matching and causal order, so the containment invariant holds after every prefix. ∎

Proposition 6 (Initial authority union can be incomplete). *The union of initially assigned authority coordinates can be strictly smaller than* $\mathrm{cl}_{\mathrm{stat}}(X^0_{\theta,q})$.

Proof. Let Tool τ_1 be admitted to create a validated credential granting call authority to Tool τ_2, and let τ_2 admit a Modify label absent from τ_1. The initial authority state contains neither τ_2 nor that label. Static expansion adds the credential, then τ_2, then its Modify label. ∎

Before the credential-creation event, executed authority contains neither τ_2 nor its Modify label. Static authority closure contains both because it includes the validated future branch. The bound in Lemma 3 can therefore be strict.

Figure 1 shows a common Agent-orchestrated System.

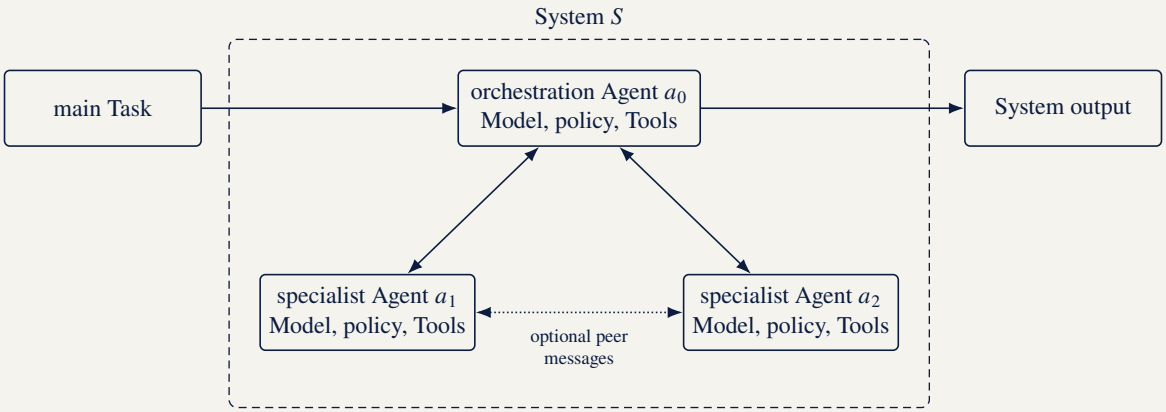

Figure 1. Structural illustration. One orchestration Agent coordinates two specialist Agents. Each Model belongs to an Agent; the dashed System boundary composes the three Agents.

4 Sufficient and efficient Designs

Definition 19 (Design)**.** Let Θ be the Design space. A Design $\theta \in \Theta$ fixes the component contracts, Model and Tool assignments, authority, resource envelopes, decomposition, scheduling, verification, aggregation, and stopping rules of an agentic system.

For each Task q, the Design combines its component kernels with its scheduling and stopping rules to induce the conditional trace law $P_\theta(\cdot \mid q)$.

Assumption 9 (Constructed trace law)**.** The scheduling and stopping rules are measurable kernels. Starting from $s_{0,q}$, they alternate with selected component kernels and terminate almost surely after finitely many events. Compatible trace composition pushes the resulting sequential kernel law forward to $P_\theta(\cdot \mid q)$.

Assumption 10 (Measurable Design semantics)**.** For every supported Task, Global Potential is a measurable subset of the trace space. The Task score $(q, H) \mapsto \mu_q(H)$, every stochastic objective, and every selected finite footprint envelope are measurable in their displayed arguments.

One sufficient implementation condition is that component contract graphs, prefix-safety monitors, event labels, authority transfers, trace composition, scheduling kernels, stopping kernels, and Task scores are measurable on the standard-Borel spaces introduced in Part 2. An application using an analytic footprint envelope may instead declare an outer-measure convention.

For an event occurrence $e \in E_H$, let

$$H_{\prec e} = \big(\{e' \in E_H : e' \prec_H e\}, \prec_H |_{\{e' \in E_H : e' \prec_H e\}}\big)$$

be its causal-past prefix. For the jth invocation of Tool τ, let $H|_{\tau,j}$ be the subtrace from its matched Call through Return, including the causally enclosed events attributed to that invocation. The index j distinguishes retries, recursion, and concurrent calls to the same Tool.

Let $\mathrm{Safe}_{\theta,q}(H)$ mean that H satisfies the Design's prefix-closed scheduling, interface, stopping, and composite-component safety rules. These rules include every Agent and System authority envelope and hard resource limit, evaluated at every prefix of that component's invocation subtrace.

Definition 20 (Global contract trace sets)**.** For Design θ and Task q, define

$$\mathcal{H}^{\mathrm{auth}}_{\theta,q} = \{H \in \mathcal{H} : \ell(e) \in \mathcal{L}_{\widehat{X}_{\theta,q}(H_{\prec e})} \text{ for every } e \in E_H\} \tag{25}$$

$$\mathcal{H}^{\mathrm{local}}_{\theta,q} = \{H \in \mathcal{H} : H|_{\tau,j} \in \mathrm{Pot}(\tau \mid x_{\tau,j}, s_{\tau,j}) \text{ for every Tool invocation } (\tau, j)\} \tag{26}$$

$$\mathcal{H}^{\mathrm{safe}}_{\theta,q} = \{H \in \mathcal{H} : \mathrm{Safe}_{\theta,q}(H)\} \tag{27}$$

The authority set checks each event against the authority reachable from its causal past, so a later grant cannot authorize an earlier event. The local set checks the actual input and invocation State of every Tool call. Under Assumption 6, $(x_{\tau,j}, s_{\tau,j})$ is determined by the causal past of invocation (τ, j).

Definition 21 (Global Potential)**.** Global Potential is the set-valued map

$$\mathrm{Pot} : \Theta \times Q \longrightarrow 2^{\mathcal{H}}$$

defined by

$$\mathrm{Pot}(\theta \mid q) = \mathcal{H}^{\mathrm{auth}}_{\theta,q} \cap \mathcal{H}^{\mathrm{local}}_{\theta,q} \cap \mathcal{H}^{\mathrm{safe}}_{\theta,q} \tag{28}$$

Thus $\mathrm{Pot}(\theta \mid q)$ is the set of traces admitted by Design θ for fixed Task q. It is not a probability. When the Task is fixed by context, the paper may abbreviate it as $\mathrm{Pot}(\theta)$.

Proposition 7 (Prefix closure of Global Potential). *Under Assumptions 4, 6, and 1,* $\mathrm{Pot}(\theta \mid q)$ *contains the empty trace and is closed under prefixes.*

Proof. Authority is checked against each event's causal past, local Tool Potential is prefix-closed by Proposition 2, and $\mathrm{Safe}_{\theta,q}$ is prefix-closed by declaration. Truncation preserves all three conditions, including a final admitted Denial event. ∎

Definition 22 (Potential footprint). For event coordinate f, the Potential footprint is

$$F_f^P(\theta, q) = \bigcup_{H \in \mathrm{Pot}(\theta \mid q)} \mathrm{proj}_f(H) \tag{29}$$

Global Potential is a denotational specification, not a general decision procedure. An application supplies the monitor language represented by $\mathrm{Safe}_{\theta,q}$. Finite-state contracts, monitors, catalogs, and Horn rules make membership computable in one important implementation class, but the unrestricted formalism does not guarantee decidability or synthesize the monitor.

Corollary 1 (Almost-sure global admissibility). *Under Assumptions 2, 9, 10, 3, 5, 6, 4, 8, and 1,*

$$P_\theta\{H \in \mathrm{Pot}(\theta \mid q) \mid q\} = 1$$

for every $q \in \mathcal{Q}_0$.

Proof. Assumption 10 makes the membership event measurable. Proposition 5 supplies prefix-wise authority containment, while Proposition 3 supplies local Tool admissibility. Boundary enforcement supplies the composite authority and hard-budget conditions. Each finite trace contains finitely many invocations, and the remaining interface, scheduling, and stopping conditions hold by construction of P_θ. ∎

The conditional trace law is therefore concentrated on Global Potential. Global Potential may still contain traces with probability zero, so it need not equal the support of that law.

Corollary 2 (Model-law invariance). *If Designs* θ_m *and* $\theta_{m'}$ *differ only by Models satisfying Assumption 7, then*

$$\mathrm{Pot}(\theta_m \mid q) = \mathrm{Pot}(\theta_{m'} \mid q)$$

for every fixed Task q, although their conditional trace laws may differ.

Proof. The common Model contract leaves every component contract, initial authority state, authority operator, and Global Potential membership condition unchanged. The Model substitution may still change probabilities within that common trace set. ∎

4.1 Reliability and feasible Designs

Definition 23 (Design reliability). The average success probability of Design θ is

$$p_{\mathrm{succ}}(\theta) = \mathbb{P}_{q \sim Q,\, H \sim P_\theta(\cdot \mid q)}\{H \in \mathcal{H}_q^\alpha\}$$

For tolerated failure probability $\delta \in [0, 1]$, the average feasible set is

$$\mathcal{F} = \{\theta \in \Theta : p_{\mathrm{succ}}(\theta) \geq 1 - \delta\} \tag{30}$$

Thus reliability means meeting the Task-specific success threshold with the required probability. It does not mean that every trace in Potential succeeds.

Definition 24 (Group reliability). For a measurable Task group $\mathcal{Q}_g \subseteq \mathcal{Q}$ with $Q(\mathcal{Q}_g) > 0$, group reliability requires

$$\mathbb{P}_{q \sim Q,\, H \sim P_\theta(\cdot \mid q)}\{H \in \mathcal{H}_q^\alpha \mid q \in \mathcal{Q}_g\} \geq 1 - \delta_g$$

for every declared group g.

Definition 25 (Robust reliability)**.** For a declared set $\mathcal{U}_Q$ of plausible Task laws, robust reliability requires

$$\inf_{Q' \in \mathcal{U}_Q} \mathbb{P}_{q \sim Q',\, H \sim P_\theta(\cdot|q)}\{H \in \mathcal{H}_q^\alpha\} \geq 1 - \delta$$

The deployment policy states whether feasibility uses average, group, robust, or combined reliability constraints. The paper uses $\mathcal{F}$ for whichever declared rule is active.

4.2 Objectives and minimality

Definition 26 (Oriented Design objective vector)**.** One complete deployment objective is

$$z(\theta) = \big(C^P(\theta), \mathbb{E}[C^R(H)], \mathbb{E}[L(H)], E^P(\theta), \mathbb{E}[E^R(H)], K^P(\theta), \mathbb{E}[K^R(H)], \mathbb{E}[O^R(H)], 1 - u(\theta)\big) \tag{31}$$

Every coordinate is oriented so that smaller is better.

Unless stated otherwise, each expectation is over $q \sim Q$ and $H \sim P_\theta(\cdot \mid q)$. The Potential quantities C^P, E^P, and K^P are declared Design-level provision, exposure, and risk functionals. The realized coordinates measure cost, latency, exposure, harm, and coordination overhead from H. An application may replace an expectation with a declared quantile or tail probability.

A contract-known coordinate needs no sampling interval, although its specification or model may still be uncertain. For example, a declared model-based Potential risk can take the form

$$K^P(\theta) = \sum_{d \in \mathcal{D}_{\text{haz}}(\theta)} w(d)\, p_{\text{misuse}}(d \mid \theta)\, I(d)$$

where $\mathcal{D}_{\text{haz}}(\theta)$ is the finite hazard catalog, $w(d)$ a sensitivity weight, p_{misuse} a modeled probability, and $I(d)$ a normalized impact. Estimated probabilities and impacts retain their model uncertainty.

The utilization $u(\theta)$ states its footprint coordinates and weights. The vector uses $1 - u(\theta)$ so its orientation matches the other coordinates. Utilization diagnoses overprovision rather than rewarding activity: increasing realized access or consumption remains charged through realized exposure, cost, and harm.

Definition 27 (Trace-Potential order)**.** Write $\theta' \preceq_{\text{tr}} \theta$ when

$$\text{Pot}(\theta' \mid q) \subseteq \text{Pot}(\theta \mid q)$$

for Q-almost every q. Write $\theta' \prec_{\text{tr}} \theta$ when the inclusion is strict on a measurable Task set of positive Q-measure.

Definition 28 (Coordinate-Potential order)**.** For selected event coordinates $\mathcal{J}$, write $\theta' \preceq_{\mathcal{J}} \theta$ when, for Q-almost every q,

$$F_f^P(\theta', q) \subseteq F_f^P(\theta, q) \quad \text{for every } f \in \mathcal{J}$$

and θ' has no broader callable authority or selected resource envelope. The order is strict when one Task-indexed inclusion is strict on a positive-measure Task set or one non-Task-indexed authority or resource relation is strict.

Definition 29 (Potential-minimal Design)**.** A Design $\theta \in \mathcal{F}$ is trace-Potential minimal when no $\theta' \in \mathcal{F}$ satisfies $\theta' \prec_{\text{tr}} \theta$. It is $\mathcal{J}$-Potential minimal when no feasible Design is strictly smaller under the coordinate-Potential order.

Permission, resource, and joint Potential minimality select the corresponding coordinates in $\mathcal{J}$. Unqualified formal results use the stronger trace-Potential order.

Call an event a wrapper-only audit event when it is a Denial emitted by a component monitor and has no causal successor. Full trace inclusion retains these events, so monitors with different audit verbosity may be incomparable even when they admit the same operational authority. A deployment concerned only with authority contraction can delete wrapper-only audit events before applying the trace-Potential order. The full order retains them when observability itself matters.

This formulation extends least-privilege reasoning across stochastic component levels without claiming priority over classical least privilege or separation of privilege [46].

Definition 30 (Pareto-efficient Design)**.** A Design $\theta \in \mathcal{F}$ is Pareto efficient for z when no $\theta' \in \mathcal{F}$ satisfies $z(\theta') \preceq z(\theta)$ with at least one strict coordinate.

For any $\mathcal{G} \subseteq \mathcal{F}$, define

$$\mathrm{MinPot}_{\mathrm{tr}}(\mathcal{G}) = \{\theta \in \mathcal{G} : \text{no } \theta' \in \mathcal{G} \text{ satisfies } \theta' \prec_{\mathrm{tr}} \theta\}$$
$$\mathrm{Pareto}_z(\mathcal{G}) = \{\theta \in \mathcal{G} : \text{no } \theta' \in \mathcal{G} \text{ strictly Pareto dominates } \theta\}$$

The subscripts identify the order and objective used by each set-valued operator.

Proposition 8 (Potential minimality and Pareto efficiency are independent)**.** *Potential minimality does not imply Pareto efficiency, and Pareto efficiency does not imply Potential minimality.*

Proof. Consider one Task and two prefix-closed Potentials $\mathrm{Pot}(\theta_1) = \{h_\varnothing, h_1\}$ and $\mathrm{Pot}(\theta_2) = \{h_\varnothing, h_2\}$, where h_1 and h_2 are distinct successful single-event traces and neither admits a successful strict contraction. Both Designs are Potential minimal. Assigning $z(\theta_1) = (2, 2)$ and $z(\theta_2) = (1, 1)$ makes θ_1 Pareto dominated.

Conversely, let $\mathrm{Pot}(\theta_3) = \{h_\varnothing, h_1\} \subsetneq \mathrm{Pot}(\theta_4) = \{h_\varnothing, h_1, h_2\}$ and assign $z(\theta_3) = (2, 1)$ and $z(\theta_4) = (1, 2)$. Both Designs are feasible and neither objective vector dominates the other, so θ_4 is Pareto efficient. Its strict contraction to θ_3 shows that it is not Potential minimal. ∎

4.3 Supervisory-control specialization

Fix one Task, an event alphabet Σ with uncontrollable subalphabet $\Sigma_u \subseteq \Sigma$, a prefix-closed plant language $L \subseteq \Sigma^*$, and a required prefix-closed specification $E \subseteq L$. Let each available Design select a prefix-closed language K satisfying

$$K \subseteq L, \qquad K\Sigma_u \cap L \subseteq K$$

and use a constant objective vector. Index Tasks by words $w \in E$, choose a full-support probability law on the countable language E, and let the Task score equal one exactly when the realized trace produces its required word. Set $\alpha = 1$ and $\delta = 0$. For Design K, let the trace law produce w on Task w when $w \in K$ and a failing trace otherwise. Reliability one is then equivalent to $E \subseteq K$.

Proposition 9 (Supervisory-control specialization)**.** *In this specialization, the trace-Potential-minimal feasible Designs are exactly the catalog-represented inclusion-minimal prefix-closed controllable superlanguages of E [43, 29].*

Proof. Each Design is identified with its admitted language. The displayed condition is controllability for prefix-closed K, reliability is equivalent to $E \subseteq K$, and the trace-Potential order is language inclusion. Feasible strict Potential contraction therefore coincides with strict contraction among the represented prefix-closed controllable superlanguages of E. ∎

Corollary 3 (Full-class supervisory-control uniqueness)**.** *If the Design space contains every prefix-closed controllable language K satisfying $E \subseteq K \subseteq L$, then one trace-Potential-minimal Design exists and admits the infimal prefix-closed controllable superlanguage of E [29].*

Proof. The full class is closed under arbitrary intersections and therefore has one infimal member. Proposition 9 identifies that member with the unique trace-Potential-minimal Design. ∎

Corollary 4 (Effective finite-state synthesis)**.** *In the deterministic finite-state full-class specialization, the Lafortune–Chen construction effectively synthesizes the unique trace-Potential-minimal Design [29].*

Proposition 10 (Existence in a finite catalog)**.** *If $\mathcal{F}$ is finite and nonempty, then $\mathrm{MinPot}_{\mathrm{tr}}(\mathcal{F})$ and $\mathrm{Pareto}_z(\mathcal{F})$ are nonempty. No two members of either set strictly order one another under its defining relation.*

Proof. Starting from any feasible Design, follow strict Potential contractions or strict Pareto dominations. Finiteness forces each chain to terminate at a minimal element. Two minimal elements cannot strictly order one another. ∎

Remark 3 (Scalarization)**.** A positive weighted sum returns supported Pareto-efficient Designs but may miss efficient Designs in a nonconvex attainable set. The weights express a deployment policy rather than a universal definition of optimality [18].

4.4 Slack and utilization

Let w_f be a declared nonnegative weight on sets of events in coordinate f, with $w_f(F_f^P(\theta,q)) < \infty$ for Q-almost every q. Define

$$g_f(H,\theta,q) = w_f\big(F_f^P(\theta,q) \setminus F_f^R(H)\big) \tag{32}$$

$$\bar{g}_f(\theta) = \mathbb{E}_{q,H}[g_f(H,\theta,q)] \tag{33}$$

$$g_f^{\mathrm{cov}}\big(\theta;(q_j,H_j)_{j=1}^n\big) = w_f\left(\bigcup_{j=1}^n F_f^P(\theta,q_j) \setminus \bigcup_{j=1}^n F_f^R(H_j)\right) \tag{34}$$

$$u_f(\theta) = \mathbb{E}_{q,H}\left[\frac{w_f(F_f^R(H))}{w_f(F_f^P(\theta,q))}\right] \tag{35}$$

The quantities are Run slack g_f, expected slack $\bar{g}_f$, sample coverage slack g_f^{cov}, and utilization u_f. When both weights in the utilization ratio are zero, the ratio is defined as one.

For declared weights $\lambda_f \geq 0$ with $\sum_f \lambda_f = 1$, aggregate utilization is

$$u(\theta) = \sum_f \lambda_f u_f(\theta)$$

This definition uses an expectation of per-Task ratios, not a ratio of aggregate expectations. The weight w_f must state whether it measures distinct objects, regions, authority facts, Tools, boundaries, sensitivity, or another operational footprint. Different choices define different quantities and must not share one name without qualification.

Run slack, expected slack, sample coverage slack, and utilization answer different questions. None establishes that unused Potential can be removed without reducing reliability.

Set union does not preserve multiplicity or magnitude for quantities such as tokens, data volume, retries, or repeated effects. For a declared nonnegative measurable trace functional m_f^R, define the finite measurable Potential envelope and quantity slack by

$$m_f^P(\theta,q) = \sup_{H \in \mathrm{Pot}(\theta|q)} m_f^R(H), \qquad g_f^{\mathrm{qty}}(H,\theta,q) = m_f^P(\theta,q) - m_f^R(H) \tag{36}$$

Set footprints record which effects are possible, while scalar envelopes retain their multiplicity and magnitude. A resource-like quantity may instead remain in the Potential and realized resource coordinates. Quantity slack enters $z(\theta)$ only when the deployment policy includes it explicitly.

Remark 4 (Slack is not a removability test)**.** Run and expected slack quantify overprovision, utilization normalizes realized use, and sample coverage slack identifies Potential not observed in a sample. None supplies the counterfactual evidence needed to conclude that the unobserved Potential can be removed safely.

Activity-based coverage analysis is established deployment practice [2]. A later measurement section distinguishes observational slack from intervention, structural guarantees, and replay evidence.

5 Nested optimization

The same feasible-set and minimality definitions apply at the Tool, Agent, and System levels. Each level selects only its own Design fields, but its objectives and constraints retain the notation introduced in Part 4.

5.1 Tool-contract problem

Let $\mathcal{K}_\tau^{\mathrm{svc}}$ be the declared service class of admissible transition kernels for Tool τ, and let $\Theta_\tau \subseteq \Theta$ contain the candidate Tool-level Designs. The feasible Tool set and its Pareto-efficient subset are

$$\begin{aligned} \mathcal{F}_\tau &= \{\theta_\tau \in \Theta_\tau : K_\tau^{\theta_\tau} \in \mathcal{K}_\tau^{\mathrm{svc}},\ K_\tau^{\theta_\tau}\{H_\tau \in \mathrm{Pot}(\tau \mid x,s) \mid x,s\} = 1 \text{ for every } (x,s) \in \mathrm{Pre}_\tau\} \\ \mathrm{Sol}_\tau^{\mathrm{eff}} &= \mathrm{Pareto}_{z_\tau}(\mathcal{F}_\tau) \end{aligned} \tag{37}$$

Here $K_\tau^{\theta_\tau}$ is the Tool transition kernel selected by θ_τ, and z_τ is the restriction of z to declared Tool-level coordinates. The admissibility condition is guaranteed by the contract assumptions in Part 2, but it remains explicit in the feasible set so a candidate kernel that violates its declared Potential is excluded.

5.2 Agent problem

Let $\Theta_a \subseteq \Theta$ contain the candidate Agent-level Designs. Each $\theta_a \in \Theta_a$ selects the Model, Tool set, optional generation class, authority, context policy, resource envelope, action policy, and stopping policy of Agent a. For Agent Task law Q_a and tolerated failure probability δ_a, define

$$\begin{aligned} \mathcal{F}_a &= \left\{\theta_a \in \Theta_a : \mathbb{P}_{q\sim Q_a,\, H\sim P_{\theta_a}(\cdot|q)}\{H \in \mathcal{H}_q^a\} \geq 1-\delta_a\right\} \\ \mathrm{Sol}_a^{\mathrm{eff}} &= \mathrm{Pareto}_{z_a}(\mathcal{F}_a) \end{aligned} \tag{38}$$

The vector z_a contains the Agent-level coordinates selected from z. It does not rename those quantities or change their orientation.

5.3 System and global problems

Let $\Theta_S \subseteq \Theta$ contain the candidate System-level Designs. Each $\theta_S \in \Theta_S$ selects the callable components A_S, dependency graph G_S, component Model and Tool fields, authority, System resource envelope $B_S^{\max}$, delegation and scheduling policy Π_S, and verification, aggregation, and stopping policy Σ_S. These are the fields already defined for a System, now treated as joint decision variables.

For the global feasible set $\mathcal{F}$, the two solution sets remain

$$\mathrm{MinPot}_{\mathrm{tr}}(\mathcal{F}), \qquad \mathrm{Pareto}_z(\mathcal{F}) \tag{39}$$

The first returns feasible Designs with no strict trace-Potential contraction, while the second returns feasible Designs with no strict Pareto improvement.

A Tool or Agent can be locally Pareto efficient while belonging to a globally inefficient System. Composition can duplicate context and authority, correlate failures, or add communication, verification, and aggregation overhead. Existing work automates Agent and team Design search [21, 49, 44]; the object defined here is the typed feasible frontier shared across all three levels.

5.4 Mathematical action of each Design field

Each Design field changes the admitted trace set, the conditional trace law, or the aggregation of realized measurements:

Design field	Mathematical action	Consequence
Model m	changes $P_\theta(H \mid q)$ under a common contract	changes the probabilities of admitted strategies; a contract-changing Model also changes $\mathrm{Pot}(\theta \mid q)$
Tool set T_a	changes the callable component set	changes available strategies and authority reachability
Data access and authority	contracts or expands $\mathrm{Pot}(\theta \mid q)$	trades information and action availability against exposure and harm
Resource envelope $B^{\max}$	contracts or expands admitted resource paths	tightening can add exhaustion or Denial outcomes; relaxing can admit longer paths
Dependency graph G_S	changes Task decomposition and trace composition	can reduce component difficulty while adding coordination and correlated failure
Scheduling policy Π_S	changes the trace partial order	changes critical-path latency without necessarily changing total work
Verification policy Σ_S	adds verification, retry, acceptance, and Denial traces	trades detectable error reduction against cost, latency, and overhead

For a continuous Design space, a reliability level set may provide local geometric intuition. A finite Design catalog instead uses the feasible-set definitions directly. The conditional scalar response laws in the later appendix illustrate particular architectures but are not assumptions of the selection or measurement results.

6 Trace-based measurement, estimation, and limits

A trace can record the Task, Design, component versions, Model and Tool calls, typed effects, Surface provenance, resources, Denial events, retries, failures, latency, termination, score, and logging propensity. Protected payloads may remain as hashes or access-controlled references, and the trace store requires its own access controls [54]. These records support measurement of executed behavior. They do not by themselves reveal what an unexecuted Design would have done.

Definition 31 (Passive observation record)**.** Fix an admitted-permission set A, and let $\theta(A)$ be the resulting Design. Define the measurement map

$$\psi_A : \mathcal{Q} \times \mathcal{H} \times [0,1] \longrightarrow \mathcal{Y}_A$$

and the passive observation record

$$Y_A = \psi_A\big(q, H, \mu_q(H)\big). \tag{40}$$

The record Y_A retains the executed trace H, its score and failure indicator, timing, and observed Denial events. Its measurable space is $(\mathcal{Y}_A, \mathscr{Y}_A)$, where $\mathscr{Y}_A$ is generated by those retained fields.

The map ψ_A states what the analyst can inspect. In particular, Y_A does not name a counterfactual repair that the Design never attempted.

To compare such counterfactuals, fix a baseline permission set A. A world is a family of Task-conditioned trace kernels

$$W = \big\{P^W_{A'}(\,\cdot \mid q) : A' \supseteq A\big\}, \tag{41}$$

one for every permission extension A'. Let $\pi_0(A' \mid q)$ be the probability that the logging policy assigns Design $\theta(A')$ to Task q, and define its support by

$$\mathcal{S}_0 = \big\{A' : \mathbb{P}_{q\sim Q}[\pi_0(A' \mid q) > 0] > 0\big\}. \tag{42}$$

Two worlds are observationally equivalent under π_0 when they induce the same law of $Y_{A'}$ for every $A' \in \mathcal{S}_0$.

Proposition 11 (Passive-trace non-identifiability)**.** *Complete passive observation under $\theta(A)$, including failures and Denial events, does not in general identify a permission-minimal reliable repair.*

Proof. Hold every non-permission Design field fixed, and choose incomparable permission labels $a, a' \notin A$. Under the baseline Design $\theta(A)$, let both worlds assign mass $1-\varepsilon$ to the same successful trace and mass ε to the same failed trace H_{fail}. The failed trace has the same score, timing, and Denial events in both worlds.

In world W_a, admitting a replaces H_{fail} with a successful continuation, while admitting a' preserves the failure. In world $W_{a'}$, reverse these two extension kernels. If the logging support is $\mathcal{S}_0 = \{A\}$, the worlds induce the same passive observation law. For any reliability target above $1-\varepsilon$, however, their permission-minimal repairs are respectively $A \cup \{a\}$ and $A \cup \{a'\}$. An estimator of the common passive law therefore cannot identify both repairs. ∎

The failed trace shows that the baseline lacks a successful strategy, but it does not identify which unattempted permission supplies one. Identification requires observed attempts, safe interventions, randomized grants, or structural assumptions. This is the positivity limitation for withheld permissions: behavior outside logging support is not determined by behavior inside it [45, 17].

Proposition 12 (Replay scope)**.** *Let H be a recorded trace whose exogenous events and random values are fixed. Let θ' remove only authority unused in H. Replay under θ' reproduces H if the removed authority cannot change context assembly, Model output, Tool output, State reads, scheduling, stopping, or randomness along the recorded path. This guarantee applies to H, not to the Task law.*

Proof. Under the stated invariance, induction over the trace order preserves every next event. Without the invariance, unused authority can change an earlier retrieval set, which can change the Model context and its next-action distribution. The recorded continuation may then have probability zero even though H did not exercise the removed authority. ∎

6.1 Logged-policy evaluation

Let $\pi(\theta \mid q)$ be a candidate Design-assignment policy and $\pi_0(\theta \mid q)$ the logging policy. Ordinary off-policy evaluation requires the support condition

$$\pi(\theta \mid q) > 0 \quad \Longrightarrow \quad \pi_0(\theta \mid q) > 0. \tag{43}$$

The condition ensures that every Design used by π is represented under π_0 for the same Task. It is an established requirement for off-policy inference, not a new result of this paper [41, 45, 52].

The statistical model above assigns one Design per Task. Changing Design fields during a trace instead requires sequential support and a sequential off-policy estimator. Even with support, logged-policy evaluation needs a valid estimator, such as an importance-weighted or doubly robust estimator, together with simultaneous confidence intervals for its estimands [15]. The direct-evaluation intervals below do not apply to reused logged outcomes.

Full Designs form a combinatorial space, so evaluation should use a restricted safe catalog and a factorized Design representation. A structured response model or Bayesian-optimization surrogate can share information across related Designs [48]. This connects Design selection to stochastic Pareto-front and constrained Pareto-set identification [4, 59, 58, 42, 27].

6.2 Safe direct evaluation

Let $\Theta_0 \subseteq \Theta$ be a finite safe catalog with $D = |\Theta_0|$. Each Design $\theta \in \Theta_0$ receives $n_\theta \geq 1$ independent evaluation Tasks from the declared deployment Task law. For evaluation i, let $S_{\theta i} \in \{0, 1\}$ indicate success and let $Z_{\theta ij} \in [0, 1]$ be the observed value of stochastic objective j, for $j \in \{1, \ldots, J\}$. Denote their sample means by $\widehat{p}_\theta$ and $\widehat{z}_{\theta j}$.

Let $\alpha_{\mathrm{FW}} \in (0, 1)$ be the allowed probability that any interval in the catalog fails to cover its mean. Define the Design-specific radius

$$r_\theta = \sqrt{\frac{\log(2D(J+1)/\alpha_{\mathrm{FW}})}{2n_\theta}}. \tag{44}$$

Contract-exact coordinates have radius zero. A model-estimated coordinate uses this radius only when evaluated on independent bounded losses. Otherwise it requires its own validated simultaneous interval.

Proposition 13 (Finite-catalog simultaneous coverage). *With probability at least* $1 - \alpha_{\mathrm{FW}}$, *every interval*

$$[\widehat{p}_\theta - r_\theta, \widehat{p}_\theta + r_\theta] \quad \text{and} \quad [\widehat{z}_{\theta j} - r_\theta, \widehat{z}_{\theta j} + r_\theta],$$

clipped to $[0, 1]$, *contains its reliability or stochastic objective mean for every* $\theta \in \Theta_0$ *and* $j \in \{1, \ldots, J\}$.

Proof. Hoeffding's inequality bounds each two-sided failure probability by $\alpha_{\mathrm{FW}}/(D(J+1))$ [20]. A union bound over the $D(J+1)$ means gives the stated simultaneous coverage. ∎

Corollary 5 (Gap-dependent classification). *Suppose every reliability comparison with* $1 - \delta$ *and every pairwise objective comparison required for the true catalog classification has a gap greater than* $\Delta_{\mathrm{cls}} > 0$. *If every Design receives*

$$n_\theta \geq \frac{8}{\Delta_{\mathrm{cls}}^2} \log\left(\frac{2D(J+1)}{\alpha_{\mathrm{FW}}}\right) \tag{45}$$

independent evaluations, all required comparisons resolve with probability at least $1 - \alpha_{\mathrm{FW}}$.

Proof. The bound gives $r_\theta \leq \Delta_{\mathrm{cls}}/4$. A reliability comparison separates one interval from the fixed threshold. A pairwise objective comparison separates two intervals and uses their combined half-width. The stated bound covers both cases on the simultaneous coverage event. ∎

Adaptive evaluation rounds may allocate error budgets $\alpha_{\mathrm{FW},t} > 0$ satisfying $\sum_t \alpha_{\mathrm{FW},t} \leq \alpha_{\mathrm{FW}}$, provided each round uses fresh outcomes. Reusing adaptively selected outcomes requires time-uniform confidence sequences or another sequentially valid construction.

Definition 32 (Empirical Design classification). Let $[\underline{p}_\theta, \overline{p}_\theta]$ cover the reliability of Design θ, and let $[\underline{z}_{\theta j}, \overline{z}_{\theta j}]$ cover each minimized objective $z_j(\theta)$. Under one simultaneous coverage event:

1. θ is definitely infeasible if $\overline{p}_\theta < 1 - \delta$;
2. θ is definitely feasible if $\underline{p}_\theta \geq 1 - \delta$;
3. the feasibility of θ is unresolved otherwise;
4. a definitely feasible Design θ' definitely dominates a definitely feasible Design θ if $\overline{z}_{\theta' j} \leq \underline{z}_{\theta j}$ for every j, with a strict inequality for at least one j;
5. a definitely feasible Design is definitely dominated if a definitely feasible competitor definitely dominates it, and is a feasible frontier candidate otherwise.

A feasible frontier candidate has not been eliminated by the intervals. The label does not certify Pareto efficiency. Overlapping objective intervals leave the relevant dominance relation unresolved.

Proposition 14 (Classification soundness). *If the intervals cover simultaneously with probability at least* $1 - \alpha_{\mathrm{FW}}$*, no truly feasible Pareto-efficient Design is classified as definitely infeasible or definitely dominated with probability at least* $1 - \alpha_{\mathrm{FW}}$.

Proof. On the simultaneous coverage event, definite infeasibility implies reliability below $1 - \delta$. Definite dominance supplies a truly feasible competitor whose objective vector weakly improves every coordinate and strictly improves at least one. A truly feasible Pareto-efficient Design therefore receives neither label. ∎

The resulting procedure versions the Tasks and Design fields, declares the estimands and safe catalog, collects protected traces with logging propensities, constructs simultaneous intervals, and classifies the candidate Designs. Safe interventions target unresolved comparisons. A material change in the Task law or a versioned Design field invalidates the corresponding evaluation and requires revalidation.

7 Worked invoice catalog

The invoice catalog instantiates the definitions with a fixed-seed synthetic simulation. It does not report measurements from a deployed System. Most Tasks are ordinary invoices, while an ε fraction requires additional evidence. A successful trace matches the invoice to a purchase order, obtains approval above the payment threshold, and executes a payment whose amount and recipient match the approved proposal. The final effects are a System-to-recipient Surface event and the corresponding payment-State transition.

The catalog uses eight authority labels. Labels r_I, r_P, r_E, and r_T permit Read events on the invoice, purchase-order, exceptional-evidence, and signed-attestation regions. Label w_R permits a Modify event that writes the payment proposal. Labels c_A and c_X permit calls to the approval and execution components, and a_P permits payment approval. The validated Horn rule

$$\{a_P, w_R\} \Rightarrow c_X \tag{46}$$

derives execution-call authority only after the approval and proposal authority facts are both present.

Interface matching is unique, and payment-State transitions are serialized. The resulting compatibility predicate satisfies Definition 2. Authority transfers use Definition 15, and Λ_{stat} includes the rule in Equation 46. Each simulated Task produces one finite outcome, so the catalog satisfies static transfer domination and the constructed trace law.

The five candidate Designs are:

Design	Contract-admissible structure
θ_B	one broad Agent holds all eight authority labels
θ_S	proposal, approval, and execution Agents divide the required paths
θ_L	the separated Design without r_E or another exceptional repair
θ_V	θ_S with an independent proposal verifier and retry rule
θ_U	the separated Design without r_P, evaluated near the reliability threshold

Design θ_U uses signed supplier attestations as an alternative evidence path. This path supports some Tasks without purchase-order access. The attestation and purchase-order paths are incomparable under coordinate-wise authority inclusion.

The authority matrix follows directly from the five contracts:

	r_I	r_P	r_E	r_T	w_R	c_A	a_P	c_X
θ_B	1	1	1	1	1	1	1	1
θ_S	1	1	1	0	1	1	1	derived
θ_L	1	1	0	0	1	1	1	derived
θ_V	1	1	1	0	1	1	1	derived
θ_U	1	0	0	1	1	1	1	derived

7.1 Simulation protocol

The reliability target is 0.95. The simulator draws $n_\theta = 10{,}000$ independent bounded outcomes per Design from the declared Bernoulli laws, using seed 20260802. With $D = 5$, $J = 2$, and $\alpha_{\mathrm{FW}} = 0.05$, Equation 44 gives the common sampling radius 0.0179 for reliability, normalized cost loss, and normalized latency loss. Exposure and authority are contract-exact and therefore have radius zero.

The Bernoulli means used by the simulator are:

	reliability	normalized cost loss	normalized latency loss
θ_B	0.975	0.740	0.460
θ_S	0.976	0.580	0.400
θ_L	0.895	0.505	0.365
θ_V	0.990	0.610	0.510
θ_U	0.945	0.460	0.340

The cost and latency outcomes are normalized pedagogical losses, not monetary or timing measurements. All four objective coordinates below are minimized.

The fixed-seed estimates and simultaneous intervals are:

	reliability	cost	latency	exposure	authority
θ_B	$[0.960, 0.996]$	$[0.721, 0.757]$	$[0.447, 0.483]$	0.82	0.90
θ_S	$[0.959, 0.995]$	$[0.568, 0.604]$	$[0.391, 0.427]$	0.45	0.46
θ_L	$[0.874, 0.910]$	$[0.494, 0.530]$	$[0.346, 0.382]$	0.34	0.35
θ_V	$[0.972, 1.000]$	$[0.596, 0.631]$	$[0.504, 0.540]$	0.47	0.48
θ_U	$[0.927, 0.963]$	$[0.437, 0.473]$	$[0.332, 0.368]$	0.31	0.32

Design θ_L is definitely infeasible because its reliability interval lies below 0.95. The feasibility of θ_U is unresolved because its interval crosses the threshold. Its true reliability gap is 0.005. Setting $\Delta_{\text{cls}} = 0.0049$ in Corollary 5 gives the conservative requirement $n_\theta \geq 2{,}131{,}422$ independent evaluations per Design to resolve all comparisons with gaps above that value.

Design θ_S is definitely feasible and definitely dominates θ_B in every objective. Designs θ_S and θ_V remain feasible frontier candidates because their cost intervals overlap. Verification improves reliability but increases latency, exposure, and authority. The smaller Potentials of θ_L and θ_U do not compensate for insufficient or unresolved reliability because reliability is a constraint on the feasible set.

The failures of θ_L also illustrate Proposition 11. One world can require r_E, while an observationally equivalent world requires an incomparable permission to call a certified manual-review component. Their passive laws under θ_L coincide, but their permission-minimal repairs differ. Thus the contracts and authority closure determine Potential, simulation estimates the trace law, the intervals classify Designs, and logging support limits the repairs identified from passive traces.

8 Related work and boundaries

8.1 Imported foundations

Effect signatures, stateful contracts, and assume–guarantee composition supply the contractual basis for the component semantics [31, 47, 8, 6]. Labelled partial orders and commuting independent actions supply the execution model [30, 35]. Probabilistic trace semantics also derives trace laws from controller architecture and memory bounds [28]. Here the trace law is paired with a pre-execution Design-selection problem.

Proposition 9 uses the supervisory-control order on controllable sublanguages [43, 29, 14]. The stochastic formulation replaces exact specification containment with a reliability constraint, then adds Pareto selection and trace-identification limits. The static authority expansion uses the standard propagation structure of propositional Horn satisfiability [13].

Least privilege, separation of privilege and duty, lattice-based information flow, and capability security provide the security foundations [46, 12, 9, 36]. The paper does not claim priority for smallest-sufficient authority. It places authority selection in the same stochastic Design problem as Model and Tool selection, resource envelopes, decomposition, and verification.

Agent-governance research studies capability and permission separation, compositional delegation, privilege control, and capability closure [57, 22, 24, 51, 50, 11, 16]. Activity-based permission refinement is deployed practice, and permission and role mining are established research areas [2, 37]. These results inform the authority coordinates and closure operators used here.

Model routing, Agent architecture search, heterogeneous team optimization, trace observability, and off-policy evaluation address other Design coordinates [7, 38, 21, 49, 44, 25, 56, 54, 52]. Constrained Pareto-identification methods supply the statistical selection context [58, 27, 42]. The present object connects these coordinates through one Global Potential, one induced trace law, and one reliability-constrained frontier.

Contemporaneous work studies admission invariants, Agent contracts, and risk-aware causal gating for capability minimization [17, 55, 23]. Trace-based assurance combines step contracts, replay, fault injection, and governance metrics [39], while protocol-oriented models specify host Agents and Task lifecycles in temporal logic [1]. These works develop particular enforcement, assurance, or minimization mechanisms. The distinction here is the joint treatment of Potential, the trace law, the feasible frontier, and the boundary of passive identification.

8.2 Comparison by formal role

Table 1 compares which parts of that chain each work family makes explicit. "Partial" indicates that a family treats the object without making it the central formal variable. A dash does not claim that every work in the family omits it.

Work family	contract/ execution	Pot./ Real.	authority/ delegation	Model/ Tool/team	reliability	Pareto	trace identification
Effect and contract systems [31, 47, 6]	explicit	partial	partial	–	partial	–	–
HRU / take-grant [19, 34]	–	–	explicit	–	–	–	–
Supervisory control [43, 29, 14]	explicit	explicit	partial	–	explicit	partial	–
Progent / CaMeL [50, 11]	explicit	partial	explicit	partial	partial	–	partial
SEAgent [24]	explicit	partial	explicit	explicit	partial	–	partial
Overlay / capability closure [22, 51]	explicit	partial	explicit	–	explicit	–	–
Agent/team search [21, 49, 44, 56]	–	–	partial	explicit	partial	explicit	partial
Constrained Pareto identification [58, 27, 42]	–	–	–	generic Designs	explicit	explicit	–
Trace provenance / OPE [54, 15]	partial	partial	partial	partial	partial	–	explicit
This paper	typed	joint	recursive	joint	feasible	frontier	non-identifiability

Table 1. Formal roles emphasized by the closest work families.

The comparison does not treat the listed foundations as interchangeable. It locates the paper's contribution in their connection: Potential-minimality and Pareto efficiency become comparable without becoming the same order, and the selected Design remains subject to what its trace evidence can identify.

8.3 Boundaries

The formalism does not assume a universal scalar measure of Model capability or Task complexity. It does not require additive risk, universal response curves, perfect enforcement, or a unique optimal Design. It also does not infer unattempted repairs from passive traces without additional assumptions or interventions.

The formalism specifies what a valid instantiation must declare, but it does not solve ontology construction, Model training, sandbox implementation, infrastructure scheduling, organizational governance, or legal policy. Extracting contracts, authority labels, Horn rules, and calibrated trace kernels from a deployed System is a separate engineering and validation problem. The synthetic invoice catalog does not measure that cost.

9 Conclusion

A Design specifies the contracts, Models, Tools, authority, resources, decomposition, scheduling, verification, and stopping rules of an agentic System. These fields define its Potential: the traces admitted for a Task before any particular trace is executed. The induced trace law then describes how the System realizes that Potential across the Task law. Keeping these objects separate prevents observed behavior from being mistaken for the full behavior admitted by the Design.

This separation supports two complementary selection criteria. Potential-minimality excludes a reliable Design when another reliable Design admits strictly fewer traces. Pareto efficiency excludes a reliable Design when another improves at least one objective without worsening the others. Because one criterion compares admitted trace sets and the other compares objective vectors, a Design can satisfy either criterion without satisfying the other. Typed composition extends both criteria from Tools to Agents and Systems, while executed authority closure ensures that each executed event remains licensed by authority accumulated along its trace prefix.

These selection criteria can be evaluated from trace evidence only for Designs represented in the observed data. In particular, passive traces cannot identify the effect of an unexercised permission when different permission extensions induce the same observed law. Logged evaluation therefore requires support, and direct evaluation requires uncertainty control. For a finite safe catalog, simultaneous intervals conservatively identify Designs that are unreliable or dominated and retain the unresolved candidates. The resulting Design problem is to select a reliable System with no unsupported contraction or objective improvement, while stating explicitly which conclusions follow from contracts, executed traces, statistical evidence, or additional modeling assumptions.

This paper holds the Task law and success conditions fixed while selecting the Design. The next problem arises when a System decomposes a main Task into subtasks assigned to existing or newly spawned Agents. Optimizing this decomposition together with each Agent's Design could make local Potential and authority match the narrower subtask, reducing per-Agent exposure and failure impact. These local reductions must be evaluated against the coordination, shared State, and cross-Agent authority introduced by the decomposition. The resulting problem is to preserve reliable completion of the main Task while minimizing the Potential and exposure of the composed System.

A Illustrative parametric regimes

The main results do not require scalar Model strength, intrinsic Task complexity, or the response laws below. This appendix imposes those assumptions only to show how particular Design tradeoffs can be calculated.

Assumption 11 (Scalar margin model)**.** Each Model m has scalar strength $\nu(m)$, each Task q has scalar complexity $\kappa(q)$, and $\phi : \mathbb{R} \to [0, 1]$ is increasing. With every other Agent field η fixed,

$$p_{\mathrm{succ}}(m, q, \eta) = \phi\big(\nu(m) - \kappa(q)\big). \tag{47}$$

Under the logistic link, Definition 9 gives $d(q; m, \eta) = \kappa(q) - \nu(m)$. Thus the difficulty value remains a Task–Model margin within this specialization, not a universal Task ranking.

A.1 Model selection and data access

For fixed Task q and remaining Agent fields η, the feasible Model set is

$$\mathcal{M}_F(q, \eta) = \{m : p_{\mathrm{succ}}(m, q, \eta) \geq 1 - \delta\}. \tag{48}$$

The selected Models are the nondominated resource and cost profiles in $\mathcal{M}_F(q, \eta)$. Since the set depends on q and η, Models need not have the same order across Tasks.

To illustrate the local relation between Model strength and access, let $v = V_q(A)$ be the success-relevant information value of admitted data region A, and set

$$p(\nu, v) = \mathrm{logit}^{-1}(1.2\nu + 0.9v - 5). \tag{49}$$

Within this response law, greater Model strength reduces the data value required at a fixed reliability level, while greater data value can compensate for lower Model strength.

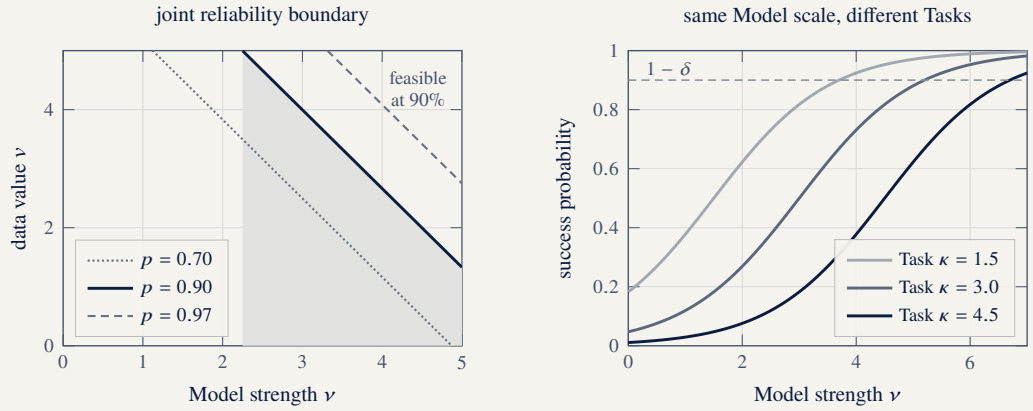

Figure 2. Analytical illustration using arbitrary constants. Left: reliability contours over Model strength and data value. Right: the same increase in Model strength has a different effect at each Task complexity.

For admitted data region $A \subseteq \mathcal{D}$, let $V_q(A)$ be its success-relevant information value and let $K(A)$ be its potential exposure or risk. Under a declared scalar policy, define

$$J_q(A) = C_{\mathrm{fail}}[1 - \phi(V_q(A))] + \lambda_K K(A). \tag{50}$$

For candidate region $d \in \mathcal{D} \setminus A$, write $\Delta_d f(A) = f(A \cup \{d\}) - f(A)$. Adding d improves this scalar objective exactly when

$$C_{\mathrm{fail}} \Delta_d \phi(V_q(A)) > \lambda_K \Delta_d K(A). \tag{51}$$

The general Design problem retains the objective vector. This marginal rule applies only after selecting the displayed scalar policy.

A.2 Resource allocation and Potential utilization

For a scalar bottleneck budget $b \geq 0$, suppose

$$p_R(b) = p_0 + \Delta p(1 - e^{-kb}), \qquad J_R(b) = cb + C_{\text{fail}}[1 - p_R(b)], \tag{52}$$

where $k, c > 0$, $p_0 \in [0, 1]$, and $0 \leq \Delta p \leq 1 - p_0$. An interior minimizer exists when $C_{\text{fail}}\Delta pk > c$ and equals

$$b^* = k^{-1} \log(C_{\text{fail}}\Delta pk/c). \tag{53}$$

When $C_{\text{fail}}\Delta pk \leq c$, the minimizer is $b^* = 0$. This result belongs to the displayed scalar resource law rather than the general resource definition.

Figure 3 separates reliability from utilization. Its left panel uses

$$p_A(v) = 0.15 + 0.83(1 - e^{-0.7v}), \qquad K(v) = (v/5)^{1.4}, \tag{54}$$

whose 0.90 reliability boundary is $v_{\min} = 3.34$. Its right panel lets $W \sim \text{Exp}(1)$ denote the Potential required by a successful strategy. Then $\mathbb{P}(W \leq x) = 1 - e^{-x}$ and average utilization is $\mathbb{E}[\min(W, x)]/x = (1 - e^{-x})/x$.

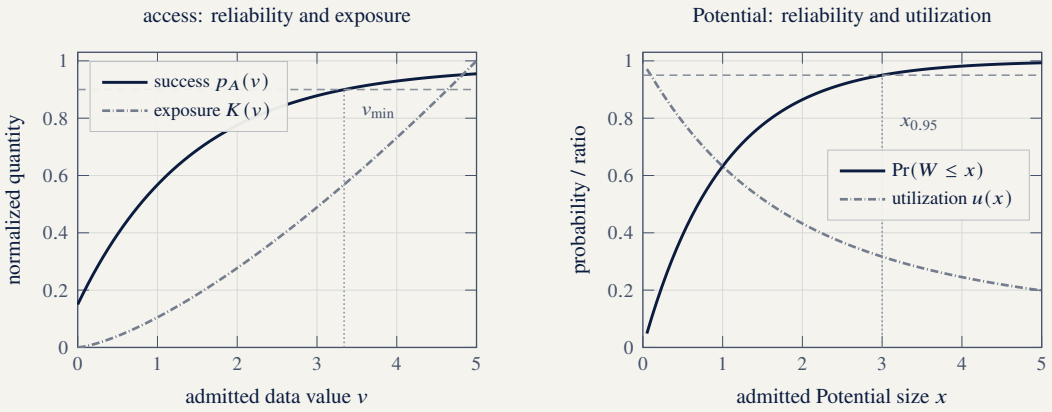

Figure 3. Analytical illustration using arbitrary constants. Left: added data value raises both reliability and exposure. Right: low average utilization can coexist with a large reliable Potential because rare Tasks require more admitted behavior.

A.3 Bounded Tool selection

Let $n \in \{0, \ldots, N\}$ be the number of available Tools. Suppose their coverage value is $v(n) = v_0 + a(1 - e^{-kn})$ and their selection burden is $d(n) = \beta n(n - 1)/2$. The bounded success law is

$$p_T(n) = \text{logit}^{-1}\big(v(n) - d(n)\big). \tag{55}$$

Because the logistic link is increasing, an interior maximizer of the continuous relaxation satisfies

$$ake^{-kn} = \beta(n - \tfrac{1}{2}). \tag{56}$$

Here β summarizes semantic ambiguity, context pressure, and selector limits. If $\beta = 0$, added Tools do not reduce success under this law.

Lemma 4 (Integer Tool optimum). *For $a, k > 0$ and $\beta > 0$, the logit argument in Equation 55 is strictly concave. If its continuous maximizer $\widetilde{n}$ lies in $(0, N)$, an integer maximizer belongs to $\{\lfloor\widetilde{n}\rfloor, \lceil\widetilde{n}\rceil\}$. A maximizer above N selects N.*

Proof. The second derivative is $-ak^2e^{-kn} - \beta < 0$. The logistic link is strictly increasing, so it preserves the maximizer. Strict concavity makes the logit argument increase before its unique maximizer and decrease afterward, leaving only the adjacent integers to compare. ∎

A.4 Decomposition and heterogeneous Models

Under Assumption 11, decompose Task q into the System graph $G_S = (Q_S, E_S)$ and assign Model m_i to subtask q_i. Define

$$p_i = \phi\big(\nu(m_i) - \kappa(q_i)\big), \qquad C_i = \rho\big(m_i, \kappa(q_i)\big). \tag{57}$$

Let C_G be the decomposition, communication, and aggregation cost. Let p_G^{coord} be coordination reliability, and let $\Sigma \in \mathfrak{S}$ encode dependence. For the measurable group-success functional $\Psi_G : [0,1]^{n+1} \times \mathfrak{S} \to [0,1]$,

$$C_{\text{multi}} = \sum_i C_i + C_G, \qquad p_{\text{multi}} = \Psi_G(p_1, \ldots, p_n, p_G^{\text{coord}}, \Sigma). \tag{58}$$

If the subtasks are conditionally independent and jointly necessary, then $p_{\text{multi}} = p_G^{\text{coord}} \prod_i p_i$. The heterogeneous Design improves both cost and reliability over monolithic pair (m_0, q) when

$$\sum_i \rho\big(m_i, \kappa(q_i)\big) + C_G < \rho\big(m_0, \kappa(q)\big), \qquad \Psi_G \geq p_{\text{succ}}(m_0, q, \eta_0). \tag{59}$$

A further specialization makes the cost threshold explicit. Let $\phi = \text{logit}^{-1}$, let n independent necessary subtasks each reduce complexity from κ to $\kappa - \Delta$, and let Model cost at strength ν be $e^{\gamma\nu}$ for $\gamma > 0$. For System reliability target $p^* \in (0,1)$ and coordination reliability $p_c > p^*$, equal subtask reliability requires

$$\bar{p}_n = (p^*/p_c)^{1/n}, \qquad \nu_i = \kappa - \Delta + \text{logit}(\bar{p}_n). \tag{60}$$

The specialized System is cheaper than the monolithic Model exactly when

$$n e^{\gamma[\kappa - \Delta + \text{logit}(\bar{p}_n)]} + C_G < e^{\gamma[\kappa + \text{logit}(p^*)]}. \tag{61}$$

Proposition 15 (Specialist decomposition threshold)**.** *Under the preceding logistic reliability, exponential cost, independence, necessity, and equal-allocation assumptions, define $C_0 = e^{\gamma[\kappa + \text{logit}(p^*)]}$. If $C_G < C_0$, specialization is cheaper exactly when*

$$\Delta > \Delta_{\min} = \text{logit}(\bar{p}_n) - \text{logit}(p^*) + \frac{\log n}{\gamma} - \frac{1}{\gamma} \log\left(1 - \frac{C_G}{C_0}\right). \tag{62}$$

Proof. Subtract C_G from both sides of Equation 61, take logarithms, and isolate Δ. The logarithm is defined exactly when $C_G < C_0$. ∎

The threshold shows that complexity reduction must offset both duplicated Models and the higher subtask reliability required by a conjunctive System. If a common-cause event fails every subtask with probability ξ, and that event is independent of coordination and residual subtask failures, then

$$p_{\text{multi}} = (1 - \xi) p_c \prod_i p_i, \qquad \bar{p}_n = \left(\frac{p^*}{(1-\xi)p_c}\right)^{1/n}. \tag{63}$$

This regime is feasible only when $(1 - \xi)p_c > p^*$.

A.5 Parallelism and State conflicts

For dependency graph G_S, define critical-path latency by

$$L_{\text{par}} = \max_{p \in \text{Paths}(G_S)} \sum_{i \in p} L_i + L_{\text{coord}}. \tag{64}$$

This abstraction places the complete coordination latency L_{coord} on the critical path [26]. Nodes without a precedence edge may execute concurrently only when their State changes are disjoint, commutative, transactionally isolated, or explicitly resolved. Cost, total compute, throughput, peak resources, and reliability remain separate objective coordinates.

For n equal ready subtasks with latency L and coordination latency $L_{\text{coord}}(n) = c_0 + c_1(n-1)$, the speedup is

$$S(n) = \frac{nL}{L + c_0 + c_1(n-1)}. \tag{65}$$

Coordination bounds this speedup even though ideal speedup grows as n, following the same bottleneck principle as Amdahl's law [3].

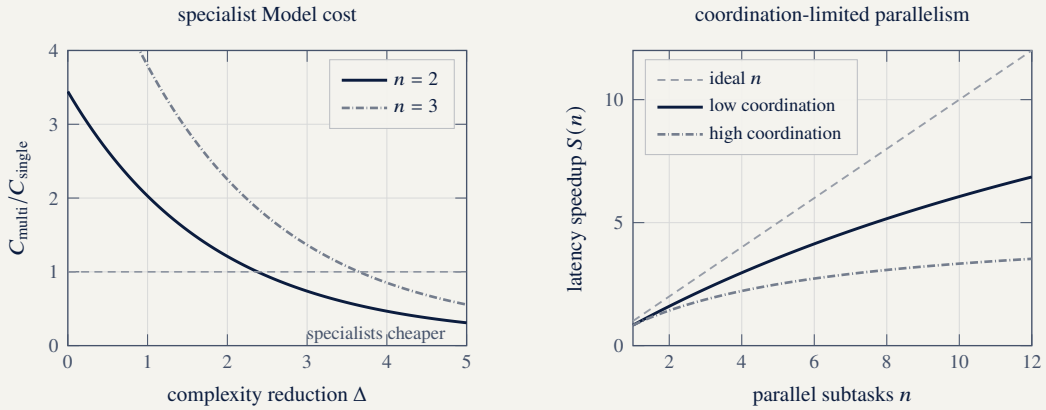

Figure 4. Analytical illustration using arbitrary constants. Left: specialist Models become cheaper after sufficient Task-complexity reduction. Right: coordination keeps parallel speedup below its ideal value. Parameters are $\gamma = 0.55$, $\kappa = 5$, $p^* = 0.90$, $p_c = 0.98$, $C_G \in \{5, 8\}$, monolithic cost 52.4, $L = 1$, $c_0 = 0.2$, and $c_1 \in \{0.05, 0.20\}$.

A.6 Verification, generation, and authority separation

Under a declared scalar policy, a verifier that reduces failure probability by Δf_v improves the objective when

$$C_{\text{fail}} \Delta f_v > C_v + \lambda_L L_v + \lambda_K K_v. \tag{66}$$

Here C_v, L_v, and K_v are the verifier's incremental cost, latency, and risk or exposure. Without this scalar policy, the verified Design is another point on the objective frontier [10].

For one retry, let p be baseline success, p_d the probability of detecting a failure, and r the conditional probability that the retry repairs it. If the retry is independent, the verifier has perfect specificity, and correct outputs do not regress, then

$$p' = p + (1 - p)p_d r, \qquad \Delta f_v = (1 - p)p_d r. \tag{67}$$

Equation 66 must also include expected retry cost $(1 - p)p_d C_{\text{retry}}$ unless C_v already includes it.

Generated objects require validation appropriate to their type:

Generated object	minimum validation	characteristic added cost or risk
parameterized contract	schema, bounds, and authorization	parameter misuse
composition of catalog Tools	interface and closure checks	reachability and interaction risk
new executable code	static and dynamic tests plus sandboxing	compilation, escape, and supply-chain risk
new Agent	all preceding checks plus Model, Task, and policy contracts	coordination, delegation, and correlated failure

Generation class g is admitted only after its validator succeeds. Its complete Design θ_g must satisfy the same reliability constraint and remain nondominated against fixed alternatives. Under a scalar policy, any capability gain must appear as greater success or quality, or as lower expected failure loss, before it can be compared with generation, validation, sandbox, execution, and incremental-risk costs.

Authority separation distinguishes the event classes an Agent can cause. For Agent i, let

$$A_i^{\text{read}},\ A_i^{\text{surf}},\ A_i^{\text{write}},\ A_i^{\text{call}},\ A_i^{\text{delegate}},\ A_i^{\text{approve}} \tag{68}$$

denote its authority over Read, Surface, Modify, Call, Delegation, and approval events. Under enforcement and non-escalation, direct State-change impact is bounded by a monotone measure of A_i^{write}. Write separation alone does not bound leakage, harmful calls, persuasion, approval, credential creation, collusion, or correlated failure because those effects follow other authority paths.

If write authority is divided into k equal non-escalating domains, normalized maximum local write impact is $1/k$. Under a scalar loss model with coordination and cross-domain penalty $\omega(k - 1)$,

$$J_{\text{sep}}(k) = k^{-1} + \omega(k - 1), \qquad k^*_{\text{cont}} = \omega^{-1/2}. \tag{69}$$

The interior expression assumes $0 < \omega \leq 1$. For integer k within a declared deployment bound, compare the feasible floor and ceiling of $\omega^{-1/2}$.

These regimes derive conditional results from their displayed response laws. They are analytical illustrations, not additional assumptions of the main results or evidence from an observed System.

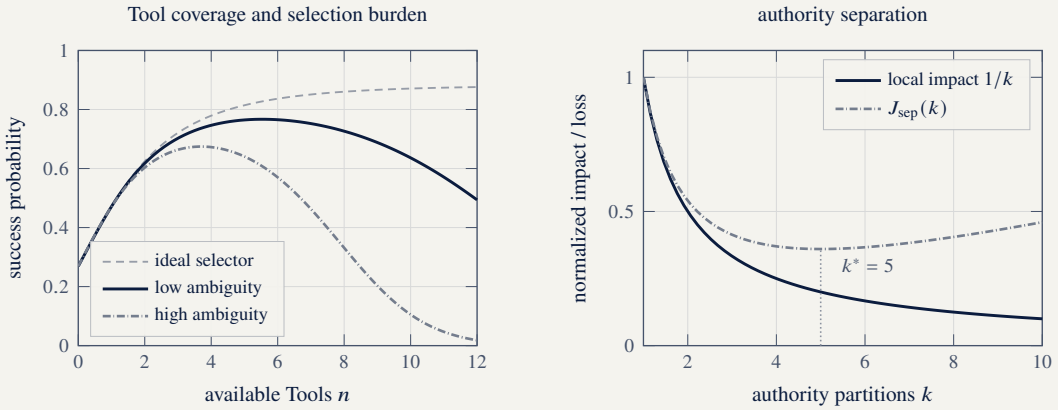

Figure 5. Analytical illustration using arbitrary constants. Left: added Tools improve coverage until selection burden dominates. Right: authority separation reduces local impact until coordination offsets further partitioning. Parameters are $v_0 = -1$, $a = 3$, coverage rate $k = 0.35$, $\beta \in \{0, 0.03, 0.09\}$, and $\omega = 0.04$.

References

[1] Edoardo Allegrini, Ananth Shreekumar, and Z. Berkay Celik. Formalizing the Safety, Security, and Functional Properties of Agentic AI Systems, 2026. arXiv:2510.14133v2.

[2] Amazon Web Services. IAM Access Analyzer Policy Generation. https://docs.aws.amazon.com/IAM/latest/UserGuide/access-analyzer-policy-generation.html, 2026. Documentation accessed 2026-08-01.

[3] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the Spring Joint Computer Conference*, pages 483–485, 1967. doi:10.1145/1465482.1465560.

[4] Peter Auer, Chao-Kai Chiang, Ronald Ortner, and Madalina Drugan. Pareto Front Identification from Stochastic Bandit Feedback. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 939–947, 2016.

[5] Moritz Y. Becker, Cédric Fournet, and Andrew D. Gordon. SecPAL: Design and Semantics of a Decentralized Authorization Language. *Journal of Computer Security*, 18(4):619–665, 2010. doi:10.3233/JCS-2009-0364.

[6] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Thomas A. Henzinger, and Kim G. Larsen. Contracts for System Design. *Foundations and Trends in Electronic Design Automation*, 12(2–3):124–400, 2018. doi:10.1561/1000000053.

[7] Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *Transactions on Machine Learning Research*, 2024.

[8] Chris Chilton, Bengt Jonsson, and Marta Kwiatkowska. Compositional assume-guarantee reasoning for input/output component theories. *Science of Computer Programming*, 91:115–137, 2014. doi:10.1016/j.scico.2013.12.010.

[9] David D. Clark and David R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *IEEE Symposium on Security and Privacy*, pages 184–194, 1987. doi:10.1109/SP.1987.10001.

[10] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems, 2021. arXiv:2110.14168.

[11] Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating Prompt Injections by Design. In *IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2026. arXiv:2503.18813.

[12] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976. doi:10.1145/360051.360056.

[13] William F. Dowling and Jean H. Gallier. Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *Journal of Logic Programming*, 1:267–284, 1984. doi:10.1016/0743-1066(84)90014-1.

[14] Klaus Dräger, Vojtěch Forejt, Marta Kwiatkowska, David Parker, and Mateusz Ujma. Permissive Controller Synthesis for Probabilistic Systems. *Logical Methods in Computer Science*, 11(2), 2015. doi:10.2168/LMCS-11(2:16)2015.

[15] Miroslav Dudík, John Langford, and Lihong Li. Doubly Robust Policy Evaluation and Learning. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1097–1104, 2011.

[16] Marcelo Fernandez. Agent Control Protocol: Admission Control for Agent Actions, 2026. arXiv:2603.18829v10.

[17] Marcelo Fernandez. From Admission to Invariants: Measuring Deviation in Delegated Agent Systems, 2026. arXiv:2604.17517v2.

[18] Arthur M. Geoffrion. Proper Efficiency and the Theory of Vector Maximization. *Journal of Mathematical Analysis and Applications*, 22(3):618–630, 1968. doi:10.1016/0022-247X(68)90201-1.

[19] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976. doi:10.1145/360303.360333.

[20] Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. doi:10.1080/01621459.1963.10500830.

[21] Shengran Hu, Cong Lu, and Jeff Clune. Automated Design of Agentic Systems. In *International Conference on Learning Representations*, 2025. arXiv:2408.08435.

[22] Amjad Ibrahim and Yong Li. Overlaying Governance: A Compositional Authorization Framework for Delegation and Scope in Agentic AI, 2026. arXiv:2606.03518.

[23] Laxmipriya Ganesh Iyer and Rahul Suresh Babu. Capability Minimization as a Safety Primitive: Risk-Aware Causal Gating for Least-Privilege LLM Agents, 2026. arXiv:2606.13884v1.

[24] Zimo Ji, Daoyuan Wu, Wenyuan Jiang, Pingchuan Ma, Zongjie Li, Yudong Gao, Shuai Wang, and Yingjiu Li. Taming Various Privilege Escalation in LLM-Based Agent Systems: A Mandatory Access Control Framework, 2026. arXiv:2601.11893.

[25] Bowen Jin, TJ Collins, Donghan Yu, Mert Cemri, Shenao Zhang, Mengyu Li, Jay Tang, Tian Qin, Zhiyang Xu, Jiarui Lu, Guoli Yin, Jiawei Han, and Zirui Wang. Controlling Performance and Budget of a Centralized Multi-agent LLM System with Reinforcement Learning, 2025. arXiv:2511.02755.

[26] James E. Kelley Jr. and Morgan R. Walker. Critical-Path Planning and Scheduling. In *Proceedings of the Eastern Joint IRE-AIEE-ACM Computer Conference*, pages 160–173, 1959. doi:10.1145/1460299.1460318.

[27] Cyrille Kone, Emilie Kaufmann, and Laura Richert. Constrained Pareto Set Identification with Bandit Feedback. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 31342–31378, 2025.

[28] Roham Koohestani, Ziyou Li, Anton Podkopaev, and Maliheh Izadi. Are Agents Probabilistic Automata? A Trace-Based, Memory-Constrained Theory of Agentic AI, 2026. arXiv:2510.23487v2.

[29] Stéphane Lafortune and Enke Chen. The Infimal Closed Controllable Superlanguage and Its Application in Supervisory Control. *IEEE Transactions on Automatic Control*, 35(4):398–405, 1990. doi:10.1109/9.52291.

[30] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978. doi:10.1145/359545.359563.

[31] Daan Leijen. Koka: Programming with Row-Polymorphic Effect Types. Technical Report MSR-TR-2013-79, Microsoft Research, 2013.

[32] Ninghui Li, Joan Feigenbaum, and Benjamin N. Grosof. A Logic-based Knowledge Representation for Authorization with Delegation. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 162–174, 1999. doi:10.1109/CSFW.1999.779771.

[33] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a Role-based Trust-Management Framework. In *IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

[34] Richard J. Lipton and Lawrence Snyder. A Linear Time Algorithm for Deciding Subject Security. *Journal of the ACM*, 24(3):455–464, 1977. doi:10.1145/322017.322025.

[35] Antoni Mazurkiewicz. Trace Theory. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *Lecture Notes in Computer Science*, pages 278–324. Springer, 1987. doi:10.1007/3-540-17906-2_30.

[36] Mark S. Miller, Ka-Ping Yee, and Jonathan S. Shapiro. Capability Myths Demolished. Technical Report SRL2003-02, Johns Hopkins University, 2003.

[37] Barsha Mitra, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. A Survey of Role Mining. *ACM Computing Surveys*, 48(4):1–37, 2016. doi:10.1145/2871148.

[38] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to Route LLMs with Preference Data, 2025. arXiv:2406.18665.

[39] Ciprian Paduraru, Petru-Liviu Bouruc, and Alin Stefanescu. A Trace-Based Assurance Framework for Agentic AI Orchestration: Contracts, Testing, and Governance, 2026. arXiv:2603.18096v1.

[40] Christos H. Papadimitriou. The Serializability of Concurrent Database Updates. *Journal of the ACM*, 26(4):631–653, 1979. doi:10.1145/322154.322158.

[41] Doina Precup, Richard S. Sutton, and Satinder Singh. Eligibility Traces for Off-Policy Policy Evaluation. In *Proceedings of the 17th International Conference on Machine Learning*, pages 759–766, 2000.

[42] Jixiang Qing, Henry B. Moss, Tom Dhaene, and Ivo Couckuyt. PF^{2}ES: Parallel Feasible Pareto Frontier Entropy Search for Multi-Objective Bayesian Optimization. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pages 2565–2588, 2023.

[43] Peter J. Ramadge and W. Murray Wonham. Supervisory Control of a Class of Discrete Event Processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987. doi:10.1137/0325013.

[44] Antonio Sabbatella. MALBO: Optimizing LLM-Based Multi-Agent Teams via Multi-Objective Bayesian Optimization, 2025. arXiv:2511.11788.

[45] Noveen Sachdeva, Yi Su, and Thorsten Joachims. Off-Policy Bandits with Deficient Support. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 965–975, 2020. doi:10.1145/3394486.3403139.

[46] Jerome H. Saltzer and Michael D. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63:1278–1308, 1975. doi:10.1109/PROC.1975.9939.

[47] Taro Sekiyama and Atsushi Igarashi. Stateful Manifest Contracts. In *POPL*, pages 530–544, 2017. doi:10.1145/3009837.3009875.

[48] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016. doi:10.1109/JPROC.2015.2494218.

[49] Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. AgentSquare: Automatic LLM Agent Search in Modular Design Space. In *International Conference on Learning Representations*, 2025. arXiv:2410.06153.

[50] Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI Agents with Privilege Control, 2025. arXiv:2504.11703.

[51] Cosimo Spera. Safety is Non-Compositional: A Formal Framework for Capability-Based AI Systems, 2026. arXiv:2603.15973.

[52] Hao Sun, Alex J. Chan, Nabeel Seedat, Alihan Hüyük, and Mihaela van der Schaar. When Is Off-Policy Evaluation (Reward Modeling) Useful in Contextual Bandits? A Data-Centric Perspective, 2023. arXiv:2311.14110.

[53] Alfred Tarski. A Lattice-Theoretical Fixpoint Theorem and Its Applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955. doi:10.2140/pjm.1955.5.285.

[54] Yiqi Wang, Jiaqi Zhang, Taotao Cai, Zirui Liu, Qingqiang Sun, Zequn Sun, Zhangkai Wu, Manqing Dong, Mingkai Zheng, Xuefei Yin, and Yanming Zhu. From Agent Traces to Trust: A Survey of Evidence Tracing and Execution Provenance in LLM Agents, 2026. arXiv:2606.04990v4.

[55] Qing Ye and Jing Tan. Agent Contracts: A Formal Framework for Resource-Bounded Autonomous AI Systems, 2026. arXiv:2601.08815v3.

[56] Di Zhao, Longhui Ma, Siwei Wang, Miao Wang, and Yi Kong. SC-MAS: Constructing Cost-Efficient Multi-Agent Systems with Edge-Level Heterogeneous Collaboration, 2026. arXiv:2601.09434.

[57] Haining Zheng, Qian Dong, Rodolfo K. Depena, Jonathan D. Bhatia, Feng Xiao, and Peng Xu. Separating Capability from Permission: A Governance Framework for Agentic AI Autonomy Levels, 2026. arXiv:2607.23438.

[58] Marcela Zuluaga, Andreas Krause, and Markus Püschel. e-PAL: An Active Learning Approach to the Multi-Objective Optimization Problem. *Journal of Machine Learning Research*, 17(104):1–32, 2016.

[59] Marcela Zuluaga, Guillaume Sergent, Andreas Krause, and Markus Püschel. Active Learning for Multi-Objective Optimization. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 462–470, 2013.