

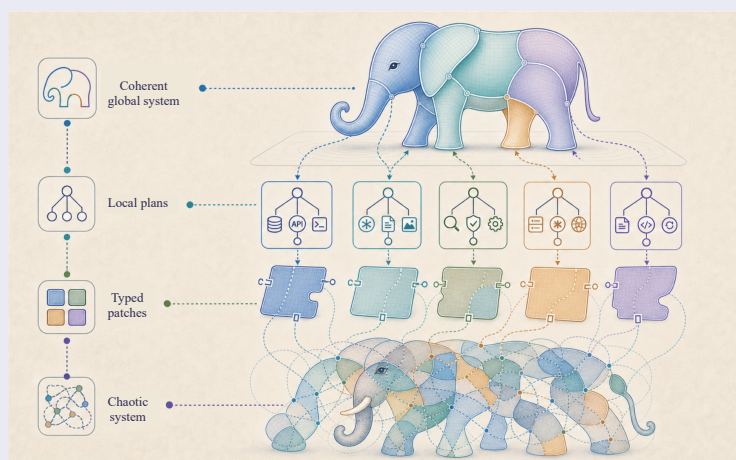
Calculus of intelligence: A topos-monadic framework for agentic workflows

Yang Yuan^{1,2,✉}, Andrew Chi-Chih Yao^{1,2}

 **Cite this article:** Yuan Y, Yao AC-C. *iFuture* 2026, 1:9710001. <https://doi.org/10.26599/IF.2026.9710001>

ABSTRACT: In this paper, we propose “Calculus of Intelligence” (COIN), a mathematical framework that formalizes agentic tasks as typed free-monadic task spaces inside a Grothendieck topos, and studies them as a calculus: differentiation via task-space decomposition, differentials as the resulting local task spaces, and integration via monadic composition together with sheaf-theoretic compatibility of declared overlaps. COIN treats a task as a valid-plan space rather than as a single output: the user intent is progressively elaborated into a typed task space, and each valid decomposition presents a sound subspace of plans obtainable from local solutions whose declared overlaps agree and whose composition is certified.

KEYWORDS: Grothendieck topos; calculus of intelligence (COIN); agentic workflow



1 Introduction

Current trends in AI are defined by a clear trajectory: expanding model scale and data volume to tackle increasingly complex problem spaces [1–4]. However, this trajectory faces a fundamental limit. Theoretical results suggest that a pretrained model’s capabilities are bounded by the structural information inherent in its training data and the algorithms used to extract it [5]. Empirically, we are observing a deceleration in scaling laws [6, 7], indicating that we may be approaching the asymptotic boundaries of single-model capabilities.

If the capabilities of a single model are bounded, addressing increasingly complex problem spaces calls for a shift from a singular model to a composed system. Just as classical calculus studies complex global structures through local approximations and integration, we propose the “Calculus of Intelligence” (COIN), which studies complex agentic tasks through typed decomposition, local planning and certified composition.

The mathematical foundation of COIN is a Grothendieck topos of typed contexts and local views. This topos provides the internal logic in which task specifications are interpreted and the

geometric setting in which task spaces are represented as subspaces of typed free-monadic plan spaces^①. Within this semantic universe, COIN separates the agentic lifecycle into three structural operations:

1. **Differentiation via task-space decomposition:** The system should support recursive decomposition of complex tasks — including coding, proving, or planning — into smaller local task spaces whose specifications are easier to analyze and solve. In COIN, a decomposition acts on a task space: it records the local context of each sub-task, the interfaces shared between sub-tasks, and the consistency checks required for integration. Such a decomposition need not cover every possible plan in the original task space. Rather, it presents a sound subspace of plans obtainable by integrating compatible local task spaces. Thus, differentiation becomes a checkable decomposition of task space rather than an informal prompting heuristic.

2. **The differential as a local task space:** The system must resolve the context management paradox: simply passing global history to a sub-task introduces noise, but clipping it makes a sub-task ambiguous. COIN addresses this by treating each sub-task as a logical “differential”: a local task space obtained from the

¹Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing 100084, China. ²Shanghai QiZhi Institute, Shanghai 200438, China.

✉ Corresponding author. E-mail: yuanyang@tsinghua.edu.cn

Received: April 16, 2026; Revised: July 1, 2026; Accepted: July 16, 2026

© The Author(s) 2026. This is an open access article under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0, <http://creativecommons.org/licenses/by/4.0/>).

^①Throughout the paper, we use the word “task” for two equivalent readings. Logically, it is a specification: it states which plans are acceptable. Geometrically, it is the space of all plans satisfying that specification. These are two views of the same object, not two different objects. When the spatial aspect is important, we may say “task space” or “local task space”; otherwise, we simply say “task” for readability.

decomposition of the original task space. This local task space is not a fragment of raw context, but a self-contained task with its own local specification, local input-output interface, assumptions about the external environment, and declared requirements toward neighboring task spaces.

A differential is therefore not yet a concrete plan. It is the local space in which valid local plans may be found. The role of decomposition is to make this space small enough and explicit enough that a local solver can reason inside it without needing the entire global context, while still preserving the information required for later integration.

3. Integration via sheaf compatibility and monadic composition: Decomposing a task into local task spaces is useful only if their solutions can be integrated into a valid plan for the parent task. COIN performs this integration through three ingredients. First, compatibility is checked on the interfaces, assumptions, contracts, and other data shared by the local task spaces, and the sheaf condition glues compatible local declarations into a coherent global description. Second, monadic composition connects the solved local plans along their typed interfaces and combines them into a candidate plan for the parent task. Finally, sheaf compatibility and monadic composition ensure only that the candidate plan is coherently and correctly assembled; they do not by themselves imply that it satisfies the parent specification. An integration soundness certificate is therefore required to prove that the composed candidate belongs to the parent task space.

If a framework satisfies these three conditions, it offers a principled way to organize increasingly complex agentic workflows. But how do we construct such a framework?

In this paper, we present the concrete realization of COIN. Unlike ad-hoc engineering heuristics, the COIN framework is mathematically constructed upon a typed task algebra (Section 5), and satisfies formal versions of the three conditions for certified decompositions (Theorem 5.1). This suggests several system-level implications:

1. Dependency-guided parallelism: COIN turns parallel execution into a checkable property of the task structure rather than an unconditional guarantee. Because each local task space records its local context, interfaces, and declared dependencies, a scheduler can identify branches whose typed dependencies are certified to be independent and whose declared external contracts are compatible. Such branches may be executed concurrently under the external consistency assumptions declared by the task.

2. Localized verification: Safety in complex systems is hindered by the difficulty of tracing errors. Because the COIN task structure records explicit dependencies, verification can be localized to the relevant branch or interface rather than requiring a monolithic scan of the whole system.

3. Model specialization: When a certified decomposition is available, COIN organizes a general task space into smaller local task spaces. Some of these local spaces may be decomposed further, while others may be simple enough for a specialized model, solver, or tool to produce a concrete local plan. Because each sub-task operates on a limited typed context, this makes specialized, smaller models more practical, especially when those models are fine-tuned or distilled for well-defined local roles.

Remark. Taken together, these implications shift the design of agentic systems from monolithic prompting toward structured intelligent pipelines. COIN separates roles that are often conflated in current agent design: the topos supplies the internal logic and local geometry in which task specifications determine task spaces; the free monad supplies an inspectable typed representation of plan structure; sheaf compatibility checks agreement on declared

overlaps; monadic composition records typed handoffs between local plans; integration soundness certificate ensures that the assembled candidate remains inside the parent task space; and an external realization layer implements leaf contracts under the assumptions declared by the task.

The categorical machinery behind COIN is substantial, but its operational content is simple: decompose a complex task, solve the resulting local tasks, and recompose the local plans with explicit certificates. For this reason, the main body emphasizes intuition, architecture, and concrete case studies (Section 6), while using only the notation needed to state the framework. The appendix contains the formal categorical construction, definitions, and proof details, including the typed context site $(\mathcal{C}, J)$, the free-monadic plan language, task objects, gluing and contract conditions, and the proof of Theorem 5.1.

2 Differentiation via task-space decomposition

In the Calculus of Intelligence, the first operation is differentiation: replacing a complex task by a family of smaller local tasks whose interfaces, dependencies, and consistency conditions are explicit. This is not merely a syntactic split of a prompt. A vague natural-language instruction can always be broken into bullet points, but such a split may lose information, duplicate assumptions about external state, or hide dependencies that later reappear as integration failures.

COIN avoids this failure by requiring each decomposition to expose the structure needed for later integration. A local task must specify what information it receives, what output it is expected to produce, which assumptions or external contracts its eventual realization may rely on, and which interfaces it shares with neighboring local tasks. In the main text, we treat this as the practical content of differentiation: a valid decomposition is one that preserves the relevant information, propagates the relevant assumptions, and makes dependencies visible. The categorical machinery, developed formally in the appendix, explains why this structure has a local logic: task specifications can be restricted to local views, overlap declarations can be checked, and compatible local data can be used by a certified integration diagram.

2.1 From vague objectives to typed tasks

A task becomes suitable for COIN only after its relevant context has been made explicit. For example, the instruction “implement authentication” is too vague to differentiate directly. It does not say what counts as a user, what typed inputs and outputs are expected, or which external services are assumed to realize credential storage and session creation.

A typed version of the same task may instead specify objects such as

UserID, PasswordHash, SessionToken,
LoginRequest, AuthResult,

together with assumptions about the external environment, such as a user database realizing a declared schema, a session service satisfying a stated consistency contract, or an identity provider implementing a specified authentication interface. The type objects do not solve the task by themselves, and the external services are not objects of the base category. Rather, the typed objects and declared assumptions make the task boundary visible enough to support later decomposition and integration.

2.2 What counts as a certified decomposition?

A COIN decomposition is more than a collection of sub-tasks. At each internal node, it specifies three components: a family of local

task objects, an integration rule, and a soundness certificate.

- **Local task objects.** Each child is a typed task with its own input-output interface, local specification, and declared external assumptions. The local contexts must collectively retain the typed information required by the parent specification. Information shared by multiple children must be exposed explicitly as boundary data rather than hidden or duplicated independently.

- **Integration rule.** The integration rule specifies how compatible child plans are assembled into a candidate plan for the parent task. It records the typed handoffs between children and the compatibility conditions imposed on shared interfaces, contracts, boundary data, and external assumptions. Monadic composition determines how the local plans are connected, while sheaf compatibility governs agreement on their declared overlaps.

- **Soundness certificate.** Local validity and compatibility do not by themselves imply that the assembled candidate satisfies the parent specification. The soundness certificate proves that every compatible family of valid child plans admitted by the integration rule produces a valid plan for the parent task.

Applying this structure recursively produces a certified decomposition tree. The leaf tasks need not be solved when the tree is constructed; once valid plans are supplied for them, the certified integration rules compose those plans bottom-up into a valid plan for the root task.

3 Subtasks in certified decompositions

Differentiation gives us local tasks, but it does not by itself solve them. In COIN, a sub-task is treated as a local task within a certified decomposition: solving it compatibly with the neighboring sub-tasks contributes to solving the parent task.

Thus a decomposition has a conditional meaning. If a parent task T is decomposed into local tasks $T_1, \dots, T_n$, the decomposition should be read as a certified statement of the following form:

compatible solutions of $T_1, \dots, T_n \implies$ a solution of T (1)

Notice that a certified decomposition should not solve residual global constraints after the children have been solved. When a parent task is decomposed, each parent-level constraint must be translated at decomposition time into local constraints for the children. A child task is then solved only relative to its own local context and the assumptions exposed on its boundary. The integration step checks the declared overlaps and handoffs; it does not ask independently solved children to renegotiate an unresolved global condition.

For example, a global budget constraint should not remain as a post-hoc condition

$$\text{cost}(p_1) + \dots + \text{cost}(p_n) \leq B \quad (2)$$

to be solved only after the child plans have been selected. A valid decomposition should instead allocate local budgets $B_1, \dots, B_n$ with $\sum_i B_i \leq B$, and assign the local condition $\text{cost}(p_i) \leq B_i$ to each child. Then the global budget follows from the decomposition certificate rather than from a later global search.

This is similar to reducing a theorem to lemmas. When a proof is decomposed into lemmas, the lemmas may not yet be proven. Nevertheless, the decomposition is meaningful if the mathematician can certify that the lemmas, once proven and used with the stated interfaces, imply the original theorem. Likewise, in COIN, a sub-task receives its meaning from its position in a certified decomposition of the parent task.

3.1 Task interfaces before solutions

After decomposition, the parent task does not need to inspect how a sub-task will eventually be solved. It only needs the declared boundary of that sub-task. For example, an authentication task may be decomposed into local tasks such as defining the user schema, verifying credentials, issuing a session token, logging authentication events, and exposing the login endpoint. At the time of decomposition, none of these local tasks needs to contain code. The decomposition is still meaningful if it states the required interfaces and assumptions: which schema the credential verifier assumes, what authenticated identity it must produce, what the token issuer requires, and which session-service contract the relevant leaves rely on.

For the credential-verification task, for instance, no concrete plan needs to be selected at decomposition time. COIN does not yet need to know whether it will be realized by a database query, a password-hash library, or an external identity provider. What matters at the parent level is its declared goal: given the appropriate request and user-record assumptions, it must produce either an authenticated identity or a specified failure result, while respecting the declared security constraints and external consistency assumptions.

3.2 Representing plans with free monads

At the interface level, a local task may be described by a simple function type

$$I \rightarrow O \quad (3)$$

where I is the input type and O is the expected output type. This interface specifies what a solver consumes and produces, but it does not reveal how the solution is assembled from smaller tasks. The decomposition tree records this additional structure, including the local task objects, their certified integration rules, and the relationships between parent and child tasks.

COIN makes the resulting plan structure explicit by representing a local plan as

$$p : I \rightarrow M(O) \quad (4)$$

where M is the free monad generated by the workflow signature. The object $M(O)$ represents pure plan syntax with eventual return type O : it records typed workflow nodes, continuations, and their composition, but it does not execute the external procedures represented by the plan.

A plan for the complete task is constructed bottom-up from a solved decomposition tree. At each leaf ℓ , a local solver or external realization supplies a plan

$$p_\ell : I_\ell \rightarrow M(O_\ell) \quad (5)$$

At an internal node, once plans have been obtained for all of its children, the node's integration rule connects those plans along their typed interfaces and combines them through monadic composition into a plan for the parent task. The associated soundness certificate ensures that the composed plan satisfies the parent specification. Repeating this process recursively produces a free-monadic plan for the root task.

The decomposition tree and the resulting free-monadic plan therefore play different roles. The tree records the task structure and its certification evidence, while the free-monadic plan records the composed workflow syntax. In this sense, the plan resembles an architectural blueprint: it can be inspected, verified, or modified before any external realization changes the world.

4 Certified integration via sheaf compatibility and monadic composition

Integration in COIN has three components. First, monadic composition connects solved local plans through typed handoffs: the output produced by one task becomes the input required by another. Second, sheaf compatibility checks that local tasks agree on the data they share, such as interfaces, assumptions, contracts, boundary variables, and proof requirements. Together, these two components produce a coherently assembled candidate plan for the parent task.

Coherent assembly, however, is not the same as correctness. Monadic composition ensures that the local plans are connected according to their typed interfaces, and sheaf compatibility ensures that their shared declarations agree, but neither condition alone implies that the resulting candidate satisfies the parent specification. An integration soundness certificate is therefore required to prove that every compatible family of valid local plans, when composed according to the chosen integration rule, produces a valid plan for the parent task.

For example, consider an authentication task decomposed into credential verification, token issuance, logging, and endpoint response. Monadic composition connects the authenticated identity produced by credential verification to token issuance, and then connects the resulting session token to the endpoint response. Sheaf compatibility checks that the local tasks agree on shared declarations such as the user schema, session-service contract, and security policy. The integration soundness certificate then proves that the composed workflow satisfies the original authentication specification, relative to its declared external assumptions.

Thus, monadic composition determines how local plans are connected, sheaf compatibility determines whether their shared data are coherent, and the integration soundness certificate establishes that the resulting plan is valid for the parent task.

4.1 Monadic composition records handoffs

Monadic composition records the handoff structure between local tasks. If one local task promises an output of type B , and the next local task requires an input of type B , then the two can be chained by the monadic composition law. In COIN, this operation records how the solution of one local task supplies the value or guarantee needed by the next task.

For example, consider a simple authentication workflow:

LoginRequest $\rightarrow$ UserRecord $\rightarrow$
SessionToken $\rightarrow$ AuthResult

Each arrow may later be realized by an external procedure: querying a table, checking a password hash, writing to a session service, or logging an event. Monadic composition does not execute these actions. It records the typed continuation structure that allows local plans to be chained into a single larger plan once their stated interfaces and assumptions line up.

However, a handoff is not only a type check. Chaining is valid only when two kinds of handoff checks pass:

1. **Structural alignment.** The output shape of one task must match the input shape expected by the next. If the first task produces raw text but the next task expects a typed JSON object, the chain is structurally invalid. The typed interface exposes these mismatches before execution.

2. **Logical entailment.** The post-condition of one task must establish the pre-condition of the next. It is not enough for the data to have the right type; it must also be in the right state. For

example, a later step may require a document value together with a certificate that an external file service has made it writable. A payment step may require an account view that is not only typed as Account, but also verified and sufficiently funded.

4.2 Choosing clean cuts: Overlap complexity

Integration also explains why some decompositions are better than others. A task may admit many valid decompositions, but the cost of checking them depends heavily on the complexity of their overlaps and handoffs.

A poor decomposition creates large, fragile interfaces. Several sub-tasks may rely on a broad implicit state assumption, an ambiguous stateful API contract, a complex database consistency assumption, or a broad implicit context. Even if each local task is simple, integration becomes expensive because many facts must be checked across many overlaps. Such a decomposition may reduce local solving cost while increasing compatibility cost.

A good decomposition creates clean cuts. It keeps local contexts small, makes handoffs explicit, and turns shared information into narrow, typed, stable interfaces. For example, instead of letting several tasks rely on an implicit mutable user-session state, the decomposition may pass a typed SessionID and require session changes to be realized through a declared external session-service contract. This does not remove the dependency. It makes the dependency explicit enough to check at the level of declarations and assumptions.

COIN therefore treats decomposition quality as an interface-design problem. The goal is not to create as many sub-tasks as possible. Excessive decomposition can increase the number of overlaps and make integration harder. The goal is to find cuts that reduce global reasoning into manageable local checks while preserving the information needed for correctness.

5 The COIN framework

The previous sections described the three operations of COIN: task decomposition, free-monadic plan representation, and certified integration. We now assemble these operations into a concrete framework. In this framework, a decomposition tree records how the original task has been reduced to local tasks, what interfaces and compatibility checks connect them, and why solving the leaf tasks would solve the root task.

5.1 Typed contexts and operation signatures

COIN fixes a typed environment. This environment specifies which type objects may appear in a task, which interfaces and assumptions may be declared, and which structural plan-forming operations are available inside the plan language.

Atomic types represent primitive domain entities, such as UserID, BankBalance, SessionID, or LoginRequest. Composite types are built from these primitives using classes and enumerations. For example, a LoginRequest may be a class containing a UserID and a password field. External resources require a different treatment. A database, file system, session store, or tool runtime is not an object of the base type category, and its internal state is not managed by COIN. Such systems enter the framework only through declared assumptions and leaf contracts: for example, a database may be assumed to realize a schema and a consistency condition, and a session service may be assumed to implement a specified interface.

In COIN, the operation signature belongs to the typed workflow language. It records structural operations for forming plans, sequencing typed continuations, returning typed values, delegating local tasks, branching, joining, asserting local

certificates, and packaging local results. It does not contain external actions such as ReadFile, QueryDatabase, model calls, or proof-checker invocations as primitive internal operations. Those actions may later realize a decomposition-tree leaf contract in an external runtime, but their implementation is not part of the free-monadic syntax.

5.2 Certified decomposition trees: Structure is the solution

A certified decomposition tree is a structured certificate explaining why compatible solutions of its leaf tasks can be composed into a solution of the root task. It is distinct from the free-monadic plan to which it compiles. Each internal node records a certified decomposition, including the declared handoffs, overlaps, and integration evidence, while each leaf specifies a local task to be solved. Once valid leaf plans are supplied, the tree compiles bottom-up into a complete plan for the root task.

For a complex objective such as “build an app,” the COIN framework therefore contains both the decomposition tree—showing how the task is divided into frontend, backend, API, authentication, deployment, and testing tasks—and, after the leaf tasks have been solved, the compiled free-monadic plan. The tree carries the certification evidence, whereas the compiled plan records the resulting workflow syntax.

If a leaf plan is invalid, the error is localized to that leaf. If two branches disagree on a shared interface, the error is localized to their overlap. If a sequential handoff fails, the error is localized to the corresponding pair of steps. COIN therefore turns global failure into localized, structured verification requirements.

5.3 The main soundness theorem

Our main theorem below states the following principle: if every leaf task has a local plan satisfying its declared contract, and every internal node carries the required handoff certificates, overlap compatibility certificates and integration soundness certificate, then the root task is solved.

Theorem 5.1 (Soundness of certified COIN decompositions (informal)). *Suppose a global task has been turned into a typed specification of acceptable plans, including any assumptions about the external environment under which the task is to be interpreted. Suppose the task is decomposed into a finite certified decomposition tree. Each leaf is assigned a local plan satisfying its local specification, and each internal node records how the plans of its children are meant to be integrated.*

Assume that the decomposition carries the following certificates:

1. **Leaf validity:** *each leaf plan satisfies the local specification assigned to that leaf.*
2. **Spatial compatibility:** *whenever sibling sub-tasks share an interface, state predicate, proof requirement, external contract, or other declared overlap, their local declarations agree on that shared part.*
3. **Sequential compatibility:** *whenever one sub-task depends on another, the earlier task produces the type of output and the guarantee required by the later task.*
4. **Integration soundness certificate:** *each internal node carries evidence that the recomposed child plans satisfy the specification of the parent task.*

Then bottom-up integration produces a single global plan for the root task, and this global plan satisfies the original task specification, conditional on the declared external assumptions.

The theorem should be read as a conditional bottom-up soundness principle. COIN does not claim that every informal prompt admits a valid decomposition, nor that arbitrary local outputs can always be assembled, nor that the framework

internally proves the consistency of external systems such as databases or tool runtimes. Rather, it states that if a task is equipped with a finite decomposition tree, and if every leaf and every internal integration step carries the required certificates, then the recursively assembled root plan satisfies the original specification under the assumptions declared at the task boundary. In this precise sense, COIN makes agentic workflows compositional.

6 Concrete examples

We now illustrate how COIN applies across several domains. The examples are not meant to exhaust the formalism, but to show the same pattern in different settings: a vague objective is first converted into a typed task, the task is organized into a certified decomposition tree of local tasks, and integration checks the handoffs and overlaps required by the decomposition. When the remaining leaf tasks are solved or externally realized, the same decomposition structure yields a concrete plan for the original task.

6.1 Mathematics: Automated theorem proving

Modern automated theorem proving, for example in Lean or Coq, faces a combinatorial explosion. A deep theorem may require many intermediate lemmas, case splits, induction branches, and tactic steps. A monolithic LLM often loses the proof-state context or hallucinates non-existent lemmas when generating long proofs.

The COIN view. We model a proof task as the construction of a proof plan from an initial hypothesis context to the goal theorem accepted by the proof assistant kernel.

- **The task:**

- I : The type of the initial hypothesis context (assumptions).
- O : The type of the goal theorem.
- Φ : the requirement that the generated proof plan is accepted by the proof checker.

- **Differentiation:** Instead of asking the model to solve the theorem in one step, COIN decomposes the proof into intermediate lemmas, case splits, induction branches, and smaller proof states. Each sub-task has a local goal and a restricted hypothesis context.

- **Integration:** Integration checks both proof-state handoffs and overlaps. Handoff checks ensure that each tactic step or lemma application produces the proof state required by the next step. Overlap checks ensure that branches cover the relevant cases and agree on shared assumptions. If every remaining proof goal is solved, the overlaps are compatible, and the composed proof is accepted by the kernel, then the theorem is certified.

In this setting, the proof assistant kernel provides a strong final verifier. COIN organizes the search so that failures can be localized to a lemma, case branch, or tactic handoff.

6.2 Software engineering: Reliable code generation

LLM-based coding agents often struggle with repo-level consistency. They may generate a correct-looking function in one file that calls a deprecated API in another, or introduce circular dependencies that break the build.

The COIN view. We treat a code-generation task as a typed planning problem over a repository context.

- **The task:**

- I : the typed interaction context of the system, such as user events, API requests, form submissions, webhook events, background-job triggers, declared external assumptions, service contracts, and runtime interfaces;

- O : the typed observable outcome, such as rendered UI state, API response, job result, notification outcome, or another externally visible system result;

- Φ : the semantic and system requirements, such as system functionality, API-contract compatibility, type-correct outputs, security policies, external-assumption compatibility, and complexity bounds.

- **Differentiation:** A composite task such as “implement the authentication system” is decomposed into typed local tasks: define the user schema, implement password verification, issue session tokens, expose the login route, and update tests. Each sub-task receives a restricted local context and explicit interfaces to the others.

- **Integration:** Integration checks both handoffs and overlaps. Handoff checks ensure that the output of one coding task supplies the input and guarantee needed by the next. Overlap checks ensure that shared interfaces agree across files and modules. If each local patch satisfies its local contract, the shared declarations agree, and the handoffs align, the generated system is certified with respect to the declared checks and specified requirements.

COIN changes debugging from an unstructured global search into a dependency-guided traversal. If the build fails, the system can inspect the relevant task branch or interface, identify which local plan violated its specification, and regenerate only that part.

6.3 Organizational planning: Resource-constrained coordination

Large organizations face a different version of the same compositionality problem. A strategic objective is decomposed into departmental plans, but local teams may optimize for incompatible metrics, compete for the same resources, or assume dependencies that other teams do not satisfy.

The COIN view. We model organizational planning as a resource-constrained decomposition problem. The goal is not to replace human judgment, but to make dependencies, resource usage, and consistency requirements explicit.

- **The task:**

- I : global resources (total budget, total headcount).
- O : the target organizational state, such as a launched product, achieved quarterly revenue target, or completed hiring state.
- Φ : constraints such as budget limits, compliance requirements, dependency ordering.

- **Differentiation:** The global objective is decomposed into department-level or phase-level tasks. For example, a product launch may be split into engineering, marketing, sales, legal, and operations plans, each with its own resources, deliverables, and dependencies.

- **Integration:** Integration checks both handoffs and overlaps. Handoff checks ensure that phase dependencies are respected: for example, marketing should not launch before the product milestone and legal approval are complete. Overlap checks ensure that shared resources and constraints agree: departmental budgets must not exceed the global budget, headcount allocations must be consistent, and shared deliverables must have compatible assumptions. If local proposals satisfy their local constraints and the shared resource and dependency checks pass, the global plan is well formed with respect to the specified organizational constraints.

This example is less formal than theorem proving or software verification, but it illustrates the same structural idea. Resource conflicts and dependency failures can be represented as failed consistency checks at the planning stage, making local autonomy more compatible with global coordination.

7 Related work

COIN and CIC. The calculus of inductive constructions (CIC) [8] provides the type-theoretic foundation for proof assistants such as Coq [9] and Lean [10]. CIC realizes the Curry–Howard perspective in a static proof environment: propositions are represented as types, proofs as terms, and correctness is checked by a kernel. COIN is inspired by the same proof/program perspective, but applies it to typed agentic plans whose leaves may be solved by local procedures or realized by external runtimes.

Prompt-structured reasoning. Chain-of-thought prompting [11] showed that explicitly generating intermediate reasoning steps can improve LLM performance on complex tasks. Later methods such as Tree-of-thought and Graph-of-thought [12, 13] further structure the search space of reasoning traces. These approaches organize reasoning at the level of prompt trajectories or search graphs. COIN targets a different layer: it asks what additional typed structure, local specifications, external-realization boundaries, overlap checks, and integration certificates are needed for local reasoning traces to be assembled into a globally valid plan.

Autonomous and tool-using agents. Recent agent systems move beyond single-prompt reasoning toward tool use, code execution, and long-horizon planning, including code-generation and software-engineering agents such as Codex-like systems [14], Manus [15], and Claude Code [16]. A persistent challenge in such systems is the compositionality gap: locally plausible outputs may disagree on shared interfaces, hidden external assumptions, or sequential assumptions. COIN formalizes this failure mode as a lack of integration certificates.

Topos theory and subtoposes. Recent work by Caramello and Lafforgue develops a general topos-theoretic account of the relation between first-order logic, Grothendieck topologies, and subtoposes. In particular, they study how provability problems in first-order logic can be translated into generation problems for Grothendieck topologies, and they analyze operations on subtoposes such as union, intersection, difference, pullback, and pushforward [17]. COIN is adjacent to this program but operates at a different level. Caramello–Lafforgue study the local computability of logic and topology generation, especially through subtopos refinement. COIN fixes an operational topos of typed contexts and works internally: a task is represented as a specification over a free-monadic plan space, a certified decomposition organizes the task into local tasks, and the free monad records how their plans compose. Thus, COIN uses topos-theoretic locality as an operational calculus for agentic task organization rather than as a general theory of subtopos refinement.

Category theory in AI. Several recent works have explored category-theoretic views of intelligence and foundation models. Yuan [5] characterizes capabilities of foundation models via the Yoneda lemma; Yuan [18] proposes a presheaf-based framework for general intelligence; and Yuan [19] studies hierarchical concept decomposition. More applied systems, such as the Coq prover of Lu et al. [20] and the backend verification framework of Xu et al. [21], demonstrate the practical value of typed decomposition and structured context in theorem proving and software verification. COIN builds on this line but adds an explicit certified-integration layer, with sheaf compatibility checks agreement on declared overlaps.

8 Conclusions

The trajectory of artificial intelligence has long been dominated by

the pursuit of scale. In this paper, we present a complementary direction: the pursuit of structure. The Calculus of intelligence (COIN) formalizes agentic workflows as a typed calculus of decomposition, local planning, and certified integration. A task is treated not as an informal prompt, but as a specification over a space of possible plans; solving the task means constructing a plan together with the structural evidence needed to justify its local validity and global integration.

This framework does not claim that structure alone replaces model scale, nor that arbitrary informal prompts automatically admit valid decompositions. Rather, COIN suggests that the reliability of composed agentic systems depends not only on the capability of individual models, but also on the algebraic organization of their tasks, contexts, external assumptions, and interfaces. In this sense, intelligence in composed systems should be studied not only through model parameters and training data, but also through the calculus of decomposition and integration that makes local reasoning globally coherent.

Declaration of competing interest

The authors have no competing interests to declare that are relevant to the content of this article.

Electronic Supplementary Material

Supplementary material is available in the online version of this article at <https://doi.org/10.26599/IF.2026.9710001>.

References

- [1] Comanici, G.; Bieber, E.; Schaeckermann, M.; Pasupat, I.; Sachdeva, N.; Dhillon, I.; Blistein, M.; Ram, O.; Zhang, D.; Rosen, E.; et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv: 2507.06261*, 2025.
- [2] OpenAI; Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; et al. GPT-4 technical report. *arXiv preprint arXiv: 2303.08774*, 2023.
- [3] Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; et al. Qwen3 technical report. *arXiv preprint arXiv: 2505.09388*, 2025.
- [4] DeepSeek-AI; Liu, A.; Mei, A.; Lin, B.; Xue, B.; Wang, B.; Xu, B.; Wu, B.; Zhang, B.; Lin, C.; et al. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv: 2512.02556*, 2025.
- [5] Yuan, Y. On the power of foundation models. *arXiv preprint arXiv: 2211.16327*, 2022.
- [6] Villalobos, P.; Ho, A.; Sevilla, J.; Besiroglu, T.; Heim, L.; Hobbhahn, M. Will we run out of data? limits of LLM scaling based on human-generated data. In: Proceedings of the 41st International Conference on Machine Learning, 49523–49544, 2024.
- [7] Muennighoff, N.; Rush, A.; Barak, B.; Le Scao, T.; Tazi, N.; Piktus, A.; Pyysalo, S.; Wolf, T.; Raffel, C. Scaling data-constrained language models. In: Proceedings of the Conference on Advances in Neural Information Processing Systems, 50358–50376, 2023.
- [8] Bertot, Y.; Castéran, P. *Interactive Theorem Proving and Program Development: Coq/Art the Calculus of Inductive Constructions*. Berlin, Germany: Springer Publishing Company, Incorporated, 2010.
- [9] Huet, G.; Kahn, G.; Paulin-Mohring, C. The Coq proof assistant: A tutorial. Technical Report. 1997.
- [10] de Moura, L.; Ullrich, S. The Lean 4 theorem prover and programming language. In: Proceedings of the 28th International Conference on Automated Deduction, 625–635, 2021.
- [11] Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In: Proceedings of the Conference on Advances in Neural Information Processing Systems, 24824–24837, 2022.
- [12] Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. In: Proceedings of the Conference on Advances in Neural Information Processing Systems, 11809–11822, 2023.
- [13] Besta, M.; Blach, N.; Kubicek, A.; Gerstenberger, R.; Podstawski, M.; Gianinazzi, L.; Gajda, J.; Lehmann, T.; Niewiadomski, H.; Nyczyk, P.; et al. Graph of thoughts: Solving elaborate problems with large language models. In: Proceedings of the AAAI Conference on Artificial Intelligence, 17682–17690, 2024.
- [14] Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; H. Pond'e de Oliveira Pinto; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. Evaluating large language models trained on code. *arXiv preprint arXiv: 2107.03374*, 2021.
- [15] Manus. Manus. 2026. Available at <https://manus.im/app>
- [16] Anthropic. Claude Code. 2026. Available at <https://claude.com/product/claude-code>
- [17] Caramello, O.; Lafforgue, L. Generation of Grothendieck topologies, provability and operations on subtoposes. *arXiv preprint arXiv: 2508.21134*, 2025. (in French)
- [18] Yuan, Y. A categorical framework of general intelligence. *arXiv preprint arXiv: 2303.04571*, 2023.
- [19] Yuan, Y. Succinct representations for concepts. *arXiv preprint arXiv: 2303.00446*, 2023.
- [20] Lu, Y.; Yang, H.; Wang, X.; Zhang, G.; Li, B.; Fu, C.; Li, C.; Yuan, Y.; Yao, A. C. Clarifying before reasoning: A Coq prover with structural context. *arXiv preprint arXiv: 2507.02541*, 2025.
- [21] Xu, K.; Luo, Y.; Yuan, Y.; Yao, A. C.-C. Towards automated formal verification of backend systems with LLMs. *arXiv preprint arXiv: 2506.10998*, 2025.
- [22] Mac Lane, S. *Categories for the Working Mathematician*. New York, USA: Springer, 2013.
- [23] Riehl, E. *Category Theory in Context*. Mineola, USA: Courier Dover Publications, 2017.
- [24] Adámek, J.; Herrlich, H.; Strecker, G. E. *Abstract and Concrete Categories*. New York, USA: John Wiley & Sons, 1990.
- [25] Kashiwara, M.; Schapira, P. *Categories and Sheaves*. Berlin, Germany: Springer, 2006.



Yang Yuan is an associate professor at the Institute for Interdisciplinary Information Sciences (IIIS) and the College of Artificial Intelligence (CAI) at Tsinghua University. He graduated from the Department of Computer Science at Peking University in 2012 and received his Ph.D. in computer science from Cornell University in 2018. From 2018 to 2019, he was a postdoctoral associate at the Massachusetts Institute of Technology (MIT).

His primary research interests include AI + Healthcare, AI Theory, and Applied Category Theory. He has published over 30 papers in top-tier computer science and artificial intelligence conferences such as NeurIPS, ICLR, and ICML.



Andrew Chi-Chih Yao is a renowned computer scientist, recipient of the Turing Award in 2000 and the Kyoto Prize in 2021. Born in Shanghai, China, he received his Ph.D. in physics from Harvard in 1972, and his Ph.D. in computer science from the University of Illinois in 1975. He served on the faculty of MIT, Stanford, UC Berkeley and Princeton University before joining Tsinghua University in 2004. There he created an elite undergraduate CS program (Yao Class) in 2005, and established the Institute for Interdisciplinary Information Sciences (IIIS) and the Center for Quantum Information (CQI) in 2011. Yao is a member of the Chinese Academy of Sciences, a foreign member of the US National Academy of Sciences and the American Academy of Arts and Sciences.