PROFORMA GLOBAL RESEARCH

Dynamic Context Assembly

Matt Rollings, Founder and Principal, Proforma Global

Whitepaper · 2026-05-18

Executive Summary

Dynamic context assembly (DCA) constructs the model's input window per request from a structured substrate: the underlying data model and the supporting structures that make selective composition possible. Its purpose is to prevent attention degradation: as a context window fills, reasoning quality drops regardless of the model's stated token limit. The assembly step composes the smallest set of inputs sufficient for each reasoning step, deterministically, from the components the substrate defines. This paper defines the practice, the failure it addresses, what a workable substrate has to solve, and where the boundary of the discipline sits. The design concepts and best practices behind each component are treated in the papers that follow.

---

1. The Failure

Language models degrade as their context windows fill. A million-token window accepts a million tokens and reasons worse the closer the prompt gets to the limit. Relevant content gets lost in the middle while irrelevant content competes for attention and pulls outputs toward whichever fragment reads as salient.

Retrieval-augmented generation does not solve this. Chunking a corpus and pulling top-k similarity matches still fills the window with content of unknown authority and uncertain applicability. The model does the filtering work that should have happened before the call.

Enterprise agents compound the problem: internal and ambiguous vocabulary; data with structure, freshness, and permission constraints the model cannot infer; rules the output must conform to. Each addition is material the model needs and volume that degrades attention.

Dynamic context assembly composes the smallest set of inputs sufficient for the specific reasoning step in front of the model. The substrate is what makes "sufficient" defensible.

2. What It Is

Dynamic context assembly constructs the data model the language model consumes when responding to a request. The data model is built per request, composed from a substrate the organization maintains for this purpose, and contains the smallest set of inputs sufficient for the specific reasoning step the model is about to perform.

Earlier practice attempted the same problem one prompt at a time. A human author drafted instructions, examples, and constraints into a static prompt, refined the prompt against observed outputs, and accepted the result as a fixed artifact. The practice was called prompt engineering and remains the dominant approach in production agent deployments. It produces brittle artifacts that resist measurement, accumulate without coherence, and degrade as the agent's scope expands beyond what the original author had in mind.

Retrieval-augmented generation attempted to make the prompt dynamic by appending retrieved fragments at request time. The mechanism works for narrow factual lookup against a single corpus. It does not survive enterprise scale, because retrieved fragments arrive without authority, freshness, permission, or topology, and the model is left to filter and reason simultaneously over inputs of unknown trust.

Dynamic context assembly replaces both. The organization builds a structured data model whose components are designed to be selectively composed. An assembly step runs between every request and every model invocation; it composes the appropriate slice of the data model for that request, records the composition, and invokes the model against a context window built for the question at hand. The composition is per request. No two requests see the same window.

Most of the difficulty of DCA is not in the assembly step. The assembly step is mechanical once the data model is right. The discipline is hard because the data model has to be designed correctly, and the data model is what the rest of this paper concerns.

3. What the Strategy Has to Solve

Organizations that attempt enterprise agent deployment without first addressing their data architecture produce a consistent set of outcomes regardless of vendor, model, or framework: agents that are convincing in demos and unreliable on workloads, that look promising in isolation and fail in combination, that the business stops trusting before the program reaches scale. The pattern is consistent enough to be diagnostic. The cause is not the model and not the orchestration layer. It is the absence of a substrate the model can reason against.

A workable substrate has to solve several problems the organization typically has not solved at the level the model requires.

Resolution of ambiguous vocabulary. Business terms carry meanings that depend on context the user never articulates. In finance, the construct of Net Income has substantively different meaning depending on whether the data is pre- or post-elimination, actual or plan or forecast, before or after topsides, this version or that one. A human reader resolves the ambiguity from context. A model cannot, unless the substrate has resolved the term before the model is invoked.

Reduction of data volume to the slice the request needs. Models do not benefit from being shown the full corpus; they are degraded by it. The substrate has to know which slice serves which kind of question and produce that slice on demand. The selection has to be deterministic, because non-deterministic context produces non-reproducible reasoning, and an enterprise will not trust an agent whose answers shift when its inputs shift invisibly.

Attribution composed alongside the data, not bolted on. Every value the model sees should arrive with its source, its freshness, its lineage, and the permission boundary that governs who may see it. None of these properties can be inferred from the values themselves. They have to be present at the moment of the call, from a substrate that maintained them from the day the data was first ingested. Organizations that attempt to retrofit attribution after a substrate has accumulated content without it discover the work is roughly the scope of the original build.

Navigability of the relationships between concepts. A reference to one entity often implies relationships to others (calculation dependencies, dimension members, hierarchical parents, derived versions) that the model needs in order to reason but cannot discover from the entity in isolation. The substrate has to make this topology explicit and queryable so the assembly step

can compose the relevant neighborhood, not just the named entity. Failure here is the most common silent failure we see in production: the agent returns a number that looks correct and is wrong in a way the user cannot detect from the answer.

Precedence rules the organization decides once and applies consistently. Which source wins when two systems disagree about the same fact. Which definition applies when the user's role implies one context and the workflow implies another. Which version of the data model the agent is reasoning against today versus a month from now. These are organizational decisions, not technical ones, and the substrate enforces whatever the organization has decided. An organization that has not decided cannot build a substrate; it can only build the appearance of one.

We have not seen an enterprise solve all of these at once. We have seen enterprises solve enough of them to make agents reliable for one class of work and then extend the substrate from that foothold. The starting point matters less than the discipline of treating the substrate as the load-bearing artifact, and the model as the consumer of what the substrate produces.

---

4. Semantic Layers

Humans understand information contextually. Models process all information simultaneously. They do not process information the way humans do, and never will. To achieve interpretability and actionability at the agent level, semantic layers should be designed as both multi-dimensional and hierarchical, where context is established through inheritance across and within layers.

Semantic layers exist to convert ambiguous business vocabulary into structured queries before the model is invoked, so the model never has to guess what the user meant when it produces its answer.

In the implementations we have seen, organizations still largely lack the underlying understanding of how to organize semantic layers in a way the model can usefully process. Our view, is that the hierarchy and the multidimensional relationships the semantic layers carry constitute the data model the model needs in order to reason.

Each organization and each implementer will have a view on:

1. What the data model should look like
2. The relationships between semantic layers

3. Inheritance within and across semantic layers
4. The technological mechanism by which the data model is stored and transmitted

No one view will be right or wrong. Their correctness depends on the workload the agent is being built to serve. What is consistent across engagements is that without a comprehensive and articulable design strategy for the semantic complexity, ROI on non-trivial agent work remains strongly negative.

An agent that reasons over resolved values without access to the underlying topology cannot answer questions that depend on structure rather than value, and many of the questions a competent business user asks fall into that category. The failure is silent: the agent returns a number that reads as correct and is wrong in ways the user has no immediate way to detect.

A new form of master data management is emerging from this work. Traditional master data discipline manages the entities: the customer, the account, the cost center, the product. The discipline DCA requires manages the *relationships* among them: calculation dependencies, dimension broadcasting, proportionality structures, hierarchical inheritance, version lineage, and the other topology properties that are not visible in the values themselves. Organizations that build agent programs without recognizing this distinction discover, usually about eighteen months in, that they have been treating a relationship problem as an entity problem.

---

5. Boundary

Dynamic context assembly decides what the model sees. It does not decide what the model does with what it sees, what verifies the output, what executes the action the agent recommends, or what routes the work according to risk tier. Orchestration, verification, deterministic execution, and risk-tier routing remain necessary parts of the broader agent architecture. A well-constructed data model in front of a broken execution layer still produces well-informed wrong actions at scale.

The discipline also depends on substantive data work the organization has to do before assembly produces value. A semantic layer can only resolve terms the organization has actually defined. Inheritance operates over relationships that have been mapped explicitly between business concepts and the underlying data. Attribution requires source records the organization has been disciplined about maintaining. An organization that attempts dynamic context assembly without first performing the underlying data architecture work will produce an expensive abstraction over the same problem it started with.

The remaining papers in this arc describe the data architecture concepts that make DCA work in production: how semantic layers are organized hierarchically and dimensionally to support inheritance; how attribution and provenance are encoded so the agent's conclusions are bounded by what the data supports; how the data model is evolved as business processes and definitions change; the measurement discipline that distinguishes a substrate that is improving the agent's outputs from one that is merely producing more elaborate context; the structural limit at which the language model is no longer required for a class of queries; and the economics that govern scaling as agent workloads expand. Each is a separable concern, and each is required for the foundation defined here to bear production weight.
