YANZI

Intent Infrastructure for AI-Assisted Development

Deterministic Context Composition

An operational pattern for stable AI execution context · yanzi.sh

Overview

AI-assisted software development introduces a new operational challenge:

> *Preserving stable execution context across long-running workflows.*

As projects grow, context becomes fragmented across:

- prompts
- documents
- conversations
- rules
- release notes
- workflows
- architecture decisions
- operational conventions

Most AI workflows today assemble this context manually, inconsistently, transiently, emotionally, and from memory. This creates drift, contradictory instructions, unstable outputs, repeated explanations, forgotten constraints, and operational inconsistency.

Deterministic Context Composition is a workflow pattern designed to reduce that instability.

Important Clarification

Yanzi is **NOT**:

- an AI framework
- an orchestration engine
- an autonomous agent platform
- a workflow runtime
- a deployment system
- a release management system

Yanzi does **not**:

- execute workflows
- coordinate agents
- interpret operational intent
- enforce process models
- manage approvals
- implement runtime orchestration

Yanzi is intentionally:

- ubiquitous
- agnostic
- unopinionated
- filesystem-friendly
- composable
- deterministic

> *Yanzi provides primitives. Humans and external workflows determine how those primitives are used.*

The Problem

AI systems operate almost entirely on context. The quality and stability of execution depends heavily on what context is provided, how it is assembled, how consistently it is reused, and whether operational constraints remain stable.

In many environments:

- prompts evolve ad hoc
- operational rules drift
- architecture decisions disappear into chat history
- workflow constraints are inconsistently applied
- release expectations become tribal knowledge

This produces unstable execution behavior over time.

Deterministic Context Composition

Deterministic Context Composition is the practice of assembling AI execution context from explicit, composable, version-controlled artifacts using stable loading patterns.

The goal is not automation. The goal is:

- consistency
- recoverability
- operational clarity
- reproducibility
- explicitness

The composition process itself remains human-controlled, workflow-defined, and operationally transparent.

Why Determinism Matters

When context assembly becomes deterministic:

- operational expectations stabilize
- workflows become repeatable
- architectural constraints persist
- AI execution becomes less volatile
- recovery becomes easier
- onboarding becomes simpler
- drift becomes observable

> *Determinism reduces hidden state.*

Yanzi's Role

Yanzi facilitates Deterministic Context Composition through:

- explicit artifacts
- append-only history
- checkpoints
- deterministic retrieval
- exportable operational state
- portable workflow memory

Yanzi stores and retrieves artifacts. It does not interpret them. This distinction is critical.

Yanzi intentionally avoids:

- orchestration semantics
- workflow enforcement
- role enforcement
- runtime dependency injection
- hidden operational behavior

Example Composition Pattern

One possible composition model may look like:

```
1. system-level operational rules
2. role-specific operational guidance
3. workflow definitions
4. project context
5. current phase context
6. current task artifacts
```

The exact composition strategy is intentionally external to Yanzi. Different organizations may structure artifacts differently, load context differently, define different operational models, or use different workflow semantics. Yanzi remains neutral.

Filesystem-Native Operational Context

One practical pattern is maintaining operational artifacts directly in the repository:

```
.yanzi/
```

This directory may contain:

- governance documents
- workflow definitions
- role packs
- release policies
- QA expectations
- operational templates

These artifacts are explicit, inspectable, version-controlled, portable, and composable.

> *These are repository-local conventions — not canonical Yanzi runtime behavior.*

Human Governance

Deterministic Context Composition does not eliminate human judgment. Humans still determine:

- what artifacts matter
- what workflows exist
- how context is assembled
- how operational decisions are made
- how releases are approved

> *The system preserves operational memory. It does not replace operational responsibility.*

Why This Matters

As AI-assisted engineering systems become larger and longer-lived, context management becomes increasingly important. Without explicit operational memory, workflows drift, decisions disappear, prompts fragment, and execution becomes inconsistent.

Deterministic Context Composition attempts to reduce that instability by making operational context:

- explicit
- portable
- version-controlled
- composable
- recoverable

Yanzi provides primitives that make those patterns easier to implement without imposing a specific operational model.

Design Philosophy

Yanzi intentionally prefers:

- explicit artifacts over hidden state
- deterministic retrieval over inference
- composability over orchestration
- portability over platform coupling

- operational transparency over automation magic

The result is a system that remains simple, inspectable, adaptable, and workflow-agnostic — while still supporting sophisticated operational patterns externally.

Closing Thoughts

Deterministic Context Composition is not a product feature. It is an operational pattern.

Yanzi facilitates the pattern by making operational memory explicit, stable, composable, and recoverable — without imposing assumptions about workflows, organizations, AI systems, release processes, or orchestration models.

> *That separation is intentional.*
