

PROFILE

LINKEDIN

AGENTIC AI SYSTEMS

DF CODEX V1.1

Engineering Persistent AI Knowledge Through Vector-Based Semantic Retrieval

Publication ID: 202610-AASPUB-0006
Publisher: Agentic AI Systems
Publication Date: 21 October 2026
Category: Artificial Intelligence | Software Engineering | Agentic Systems
Classification: Technical Research & Engineering Publication
Project Status: V1 Operational Acceptance Complete | V1.1 Development in Progress

Agentic AI Systems - Intelligence. Automation. Advantage.

Abstract

Modern artificial intelligence systems can analyse source code, identify defects, generate recommendations and automate engineering workflows. However, a fundamental limitation remains: the ability to retain, retrieve and reuse previously acquired engineering knowledge across independent analysis sessions.

Without a persistent knowledge retrieval architecture, AI systems may repeatedly investigate problems that have already been analysed, consuming computational resources and limiting the accumulation of institutional knowledge.

Agentic AI Systems is developing DF Codex, an incremental AI-assisted source-code analysis system integrated with DF Memory Ontology Edition, a governed knowledge repository.

DF Codex V1 has completed operational acceptance testing across its tested scope, including Smart Archive snapshot and manifest verification, incremental execution, concurrency controls, checkpoint integrity, interruption recovery and scheduled operation.

DF Codex V1.1 extends this foundation by introducing vector-based semantic retrieval. This capability is designed to allow engineering knowledge to be discovered through contextual meaning rather than depending exclusively on exact keywords or identifiers.

The initial semantic embedding engine has been deployed and successfully tested on local hardware. Persistent vector indexing, retrieval APIs and automated integration with DF Codex remain under development.

1. Introduction

Large language models have substantially improved the automation of software engineering tasks. Their ability to interpret source code, explain complex systems and identify potential defects creates opportunities for more autonomous engineering operations.

Yet generating an intelligent response and maintaining a reliable engineering knowledge base are fundamentally different capabilities.

An AI model can analyse a repository today without necessarily having access to the evidence, decisions or findings generated during previous analyses.

This creates several operational challenges:

- Repeated analysis of previously investigated issues.
- Difficulty locating historical engineering decisions.
- Increased inference and computational overhead.
- Fragmented technical knowledge across projects.
- Limited traceability between AI recommendations and historical evidence.
- Reduced continuity between independent agent execution sessions.

DF Codex addresses these challenges through a staged engineering architecture combining incremental analysis, governed memory and semantic knowledge retrieval.

2. DF Codex V1 - Operational Foundation

DF Codex V1 establishes the underlying source-code analysis and execution framework. Its architecture incorporates incremental repository scanning, execution-state management, archival checkpoints, concurrency protection and recovery mechanisms.

2.1 Incremental Analysis

Rather than automatically reprocessing an entire repository during every scheduled execution, DF Codex maintains state information to identify whether additional processing is required. This approach is designed to reduce redundant analysis and improve operational efficiency.

2.2 Smart Archive - Intelligent Snapshot and Evidence Management

A core component of DF Codex V1 is its Smart Archive, implemented by smart_archive.py. It captures or discovers repository source snapshots and records associated manifests. These archived records provide the historical reference points used by the snapshot resolver and incremental planner to determine whether a repository has changed.

Smart Archive is therefore more than a document repository: it is an evidence and snapshot-management component within the analysis pipeline. Its manifest records support comparison against prior checkpoints, enabling an unchanged repository to follow the UP_TO_DATE path without unnecessary semantic inference or checkpoint advancement.

The verified processing sequence is: repository scan - Smart Archive snapshot/manifest - snapshot resolution and checkpoint verification - incremental planning - conditional semantic analysis - validated checkpoint advancement. The archive stores snapshots and manifests separately from incremental state and per-run analysis outputs.

Operational acceptance evidence included verification of four archive manifests against their canonical hashes and repository identities. Recorded checkpoint references to the last and previous successful manifests were also checked for existence and matching digests. These tests support integrity claims for the inspected records, not for every possible archive artifact or failure scenario.

2.3 Execution Integrity

DF Codex V1 includes mechanisms for preventing overlapping execution against the same repository, managing state transitions and preserving checkpoint information. Operational acceptance testing covered concurrency controls, atomic state recovery, interruption scenarios and analyser failure handling.

2.4 Historical Evidence

Smart Archive manifests, analysis state and archival mechanisms preserve operational records that can support subsequent investigation and verification. Historical evidence is particularly important when AI-generated engineering findings must be reviewed, compared or reproduced.

2.5 Operational Acceptance

DF Codex V1 completed its defined operational acceptance tests in October 2026. The acceptance process included recovery testing, scheduled operation and post-reboot service verification. This acceptance establishes a tested operational baseline. It does not establish universal reliability across every possible repository, workload or failure scenario.

3. DF Memory Ontology Edition

DF Memory provides a governed storage layer for information generated or used by AI systems. Its architecture distinguishes between knowledge records, entities, relationships and lifecycle states.

A memory record progresses through a controlled lifecycle: STAGED → APPROVED → COMMITTED.

This separation allows information to be recorded and reviewed before becoming committed knowledge. The existing implementation uses SQLite as its authoritative database and exposes local API endpoints for managing memory and ontology records.

The DF Codex V1 operational acceptance document has been successfully stored, independently hash-verified and committed to DF Memory. This establishes a verified example of a complete engineering document preserved within the governed knowledge repository.

However, persistent storage alone does not provide semantic retrieval. That capability is the primary objective of DF Codex V1.1.

4. Why Vector-Based Semantic Retrieval?

Traditional keyword searches depend on the words appearing in a document or associated metadata. Semantic retrieval uses mathematical representations of text, commonly called embeddings, to estimate relationships between passages and queries.

For example, a stored engineering record might describe: "Atomic checkpoint recovery following interrupted execution."

A subsequent query might ask: "Can the system recover if a scan stops unexpectedly?"

Although the wording differs, the underlying engineering concepts are related. An embedding-based retrieval engine can identify this relationship through vector similarity.

4.1 Embedding Representation

Text is processed by an embedding model to produce a numerical vector. For the initial DF Codex V1.1 implementation, each embedding contains 384 numerical dimensions. These dimensions collectively represent learned characteristics of the text.

4.2 Similarity Search

Query embeddings can be compared against stored document embeddings using cosine similarity. This produces a mathematical similarity score that supports ranking candidate passages. Similarity does not establish factual correctness. Retrieved evidence must still be verified against its original source.

4.3 Contextual Knowledge Reuse

The intended result is a system capable of retrieving relevant historical evidence before generating or evaluating a new engineering conclusion. This can support continuity between analysis sessions without requiring the language model to retain every previous document within its active context window.

5. DF Codex V1.1 - Technical Architecture

The proposed architecture separates four responsibilities. The V1 Smart Archive operates inside the analysis layer and is distinct from the V1.1 vector index: Smart Archive manages source snapshots and manifests; the vector index is intended to support meaning-based retrieval of committed knowledge.

Layer 1 - DF Codex Analysis Engine and Smart Archive. Captures repository snapshots and manifests, resolves historical state, and performs conditional incremental analysis.

Layer 2 - DF Memory Ontology Edition. Maintains authoritative, governed records, including their lifecycle state, identity, provenance and relationships.

Layer 3 - Semantic Embedding and Vector Index. Transforms eligible knowledge passages into numerical embeddings and maintains a searchable index.

Layer 4 - Knowledge Retrieval. Receives engineering questions, identifies relevant stored passages and returns evidence with source attribution.

The vector index is designed as a derived, rebuildable structure. It does not replace the authoritative memory database. Only committed knowledge is intended to be eligible for production retrieval.

6. Local Embedding Engine

The initial implementation uses the following technology stack:

Component	Implementation
Embedding model	all-MiniLM-L6-v2
Model execution	ONNX Runtime 1.31.0
Embedding dimensions	384
Numerical processing	NumPy
Tokenisation	Hugging Face Tokenizers
Processing hardware	AMD Ryzen 9 5900X
Execution mode	Local CPU inference
Python	3.14.4
Intended similarity metric	Cosine similarity
Intended vector persistence	Separate SQLite-backed index

The system is deployed in an isolated Python environment on a local engineering server equipped with an AMD Ryzen 9 5900X processor, approximately 60 GiB of system memory and two NVIDIA RTX 3090 GPUs.

CPU inference was selected for the initial implementation to avoid competing with existing GPU workloads. The deployment does not require PyTorch for embedding inference.

7. Initial Semantic Validation Results

The first embedding-engine test was completed successfully on 21 October 2026. Three sentences were processed to verify model loading, tokenisation, embedding dimensions, mean pooling, vector normalisation and basic semantic discrimination.

The test compared a source-code analysis statement against a related engineering statement and an unrelated weather statement.

7.1 Results

Measurement	Result
Embedding dimensions	384
Related sentence similarity	0.6665
Unrelated sentence similarity	0.1861
Related/unrelated score ratio	Approximately 3.58x
CPU inference	PASS
Tokeniser validation	PASS
Vector normalisation	PASS
Basic semantic ranking	PASS

The related engineering sentences received a substantially higher similarity score than the unrelated sentence. This result demonstrates that the deployed embedding engine can distinguish these examples according to semantic similarity.

It does not establish production retrieval accuracy, comprehensive relevance performance or measured improvements in engineering productivity. Those outcomes require additional evaluation against a representative knowledge collection.

8. Incremental Vector Indexing

The next development stage is the creation of a persistent vector index. The proposed indexing workflow is:

- Identify committed records within DF Memory.
- Retrieve their authoritative content and metadata.
- Divide long documents into meaningful passages.
- Generate embeddings for each passage.
- Store embeddings alongside references to their originating memory records.
- Detect content changes using cryptographic hashes.
- Re-embed only records or passages requiring updates.
- Return ranked passages with verifiable source attribution.

This approach is intended to avoid unnecessary repeated embedding computation. The index must also support reconstruction from authoritative committed records following corruption or data loss.

At the time of publication, this indexing capability remains in development and has not completed acceptance testing.

9. Knowledge Governance and Evidence Integrity

Semantic retrieval introduces additional requirements beyond similarity ranking. A retrieved passage must be traceable to the knowledge record from which it originated.

DF Codex V1.1 is therefore being designed to retain provenance information including:

- Memory identifier and associated entity identifier.
- Authoritative source reference.
- Document or content hash.
- Knowledge lifecycle state.
- Passage location or chunk reference.
- Embedding model and indexing version.

These controls are intended to support repeatability, auditability and evidence-based engineering decisions. Importantly, a high similarity score must not be interpreted as proof that a retrieved statement is correct. The original evidence remains authoritative.

10. Engineering Benefits and Expected Outcomes

The potential benefits of combining incremental analysis with semantic retrieval include reduced repeated investigation, faster access to historical engineering evidence and improved continuity between AI agent sessions.

The architecture also creates opportunities for cross-project knowledge reuse. For example, a previously resolved configuration issue may be relevant to another repository even when filenames, project names or descriptive terminology differ.

However, these benefits are engineering objectives rather than demonstrated production outcomes. Future evaluation will need to measure retrieval precision, relevance, indexing latency, query latency, resource consumption and the effect of retrieved context on downstream analysis quality.

11. Conclusion

DF Codex represents an engineering approach to combining incremental AI-assisted code analysis with governed, persistent knowledge.

Version 1 establishes the tested operational analysis foundation, including Smart Archive snapshot and manifest handling. Version 1.1 introduces the infrastructure required for semantic knowledge retrieval, enabling the future possibility of identifying relevant historical engineering evidence through contextual meaning.



PROFILE

LINKEDIN

AGENTIC AI SYSTEMS

Initial local embedding tests have confirmed successful execution of a 384-dimensional ONNX-based semantic model and basic similarity discrimination. The remaining engineering work is to build, integrate and validate persistent retrieval against authoritative DF Memory records.

The longer-term objective is not simply to build an AI system that can analyse code. It is to develop an AI engineering system capable of accumulating, retrieving and applying verified knowledge across successive operations.

DF Codex V1 analyses. DF Memory preserves. DF Codex V1.1 is being developed to retrieve and reuse.

Publication and Attribution

Agentic AI Systems
Intelligence. Automation. Advantage.
Website: https://www.agenticaisystem.com.au/
Technical Publications: https://agenticaisystem.com.au/publications/

Publication Reference: 202610-AASPUB-0006
Document Title: DF Codex V1.1 - Engineering Persistent AI Knowledge Through Vector-Based Semantic Retrieval
Published: October 2026

© 2026 Agentic AI Systems. All rights reserved.

Technical disclosure: This publication describes an independently developed engineering implementation and its reported local validation results. The use of established techniques including embeddings, cosine similarity, ONNX inference and incremental indexing does not imply invention of those underlying techniques. Performance and operational claims are limited to the tests described. Future capabilities are identified as development objectives rather than completed features.

CONTROLLED UPDATE | 14 NOVEMBER 2026

DF CODEX V1.1 - LIVING ENGINEERING RECORD

Publication ID: 202610-AASPUB-0006 **Update ID:** U001 **Audit:** CP-DFCODEX-AUDIT-001 through 004

Document policy: The original 21 October 2026 publication (pages 1-8) remains intact. This dated supplement adds findings as at 14 November 2026. Historical statements retain their original time scope. No live-node changes were made during the audit.

A. Update and change register

ID / date	Change added	Evidence / classification
BASE | 21 Oct 2026	Original V1.1 publication: V1 acceptance, semantic retrieval design and initial embedding validation.	Original pages 1-8; historical baseline
U001 | 14 Nov 2026	Added current operating status, pipeline schematics, software stack, proven vs unverified functions, gaps, proposals and acceptance roadmap.	Read-only AI3090 audit CP-DFCODEX-AUDIT-001 to 004
NEXT | pending	Future updates must append an ID, date, evidence, changed sections and verification status.	Not yet performed

B. Status terminology and scope

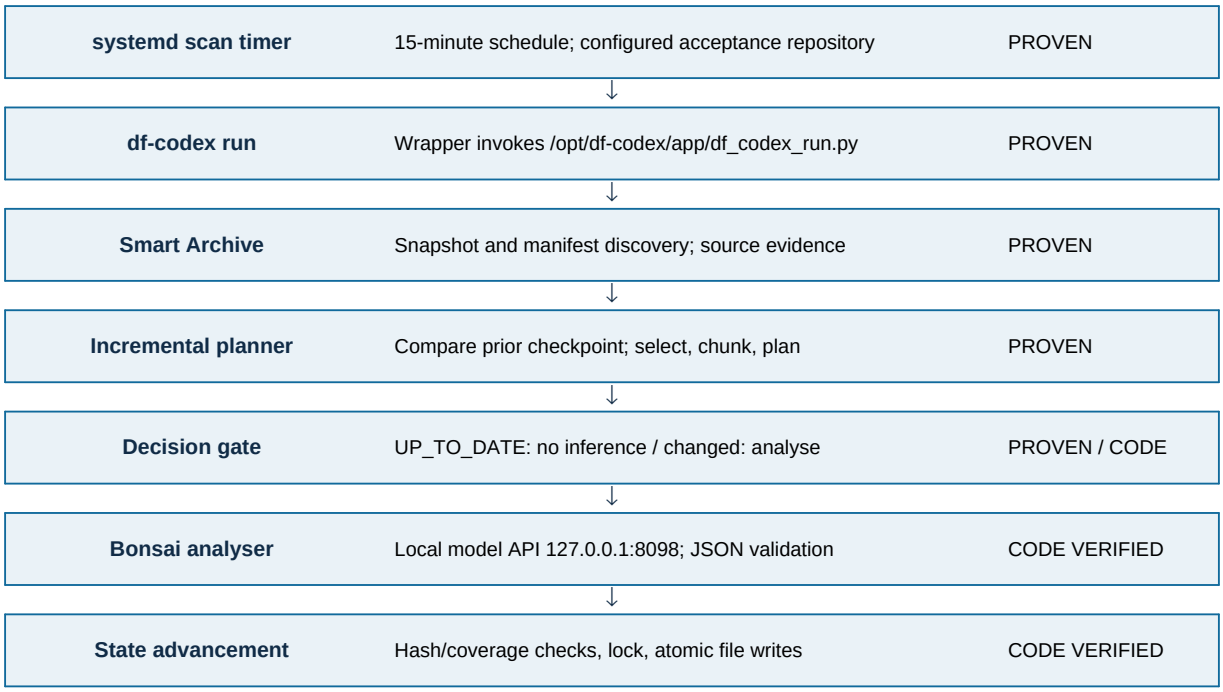
Label	Meaning
PROVEN (scoped)	Observed in logs, audit outputs or acceptance evidence within stated test scope.
IMPLEMENTED / UNVERIFIED	Code or service exists; end-to-end runtime behavior not demonstrated.
PARTIAL / GAP	Some implementation exists but an important behavior is absent or stops safely.
PROPOSED	Future design; not a deployed or accepted capability.

C. Current operating conclusion

DF Codex V1 scheduled incremental scanning is active and functioning on its configured acceptance repository. A successful 14 Nov 2026 scan returned UP_TO_DATE / NO CHANGE in 0.171 seconds. This is evidence for the no-change path only; it does not demonstrate a fresh full inference cycle on that date. Hierarchical file/component/worktree synthesis is implemented in separate modules but is not evidenced as a completed, integrated scheduled pipeline.

The FOCUS repository is **retired historical test evidence**, not an active development target. The historical 29 Sep 2026 synthesis plan was structurally consistent: 54 chunk references, 27 file nodes, 10 component IDs and zero missing referenced chunk files. It does not prove synthesis execution.

D. Architecture schematic - deployed V1 processing



D1. V1 operational boundaries

The scheduler currently targets the acceptance repository, not all five registered repositories. Repository registration is not equivalent to scheduled monitoring. The analyser has an HTTP timeout, but the parent runner subprocess invocations have no configured timeout. State update uses per-repository locking and atomic replacement of individual files; history and current-state writes are not one transactional unit.

D2. Historical acceptance and audit evidence

Reference	Observed result
21 Oct 2026 V1 acceptance	Accepted within defined test scope: snapshot/manifest, incremental execution, concurrency, checkpoint and recovery tests.
14 Nov 2026 scheduler	df-codex-model.service and df-codex-scan.timer active and enabled; scanner finished successfully.
14 Nov 2026 no-change run	UP_TO_DATE, work_required=false, queue/bundle null; 0.171 s.
Historical synthesis plan	54 chunks / 27 files / 10 components / 1 worktree; referenced chunk files all present.

E. Synthesis and knowledge architecture

E1. Hierarchical synthesis - current versus intended

Chunk ingestion	Chunk analysis and qualification artifacts	MODULE EXISTS
↓		
File synthesis	Direct or grouped REDUCE processing	CODE EXISTS
↓		
Component synthesis	Aggregates file evidence; oversized inputs stop	PARTIAL
↓		
Worktree synthesis	Aggregates component evidence; oversized inputs stop	PARTIAL
↓		
Integrated orchestration	Schedule, resume, qualify and publish hierarchy	PROPOSED

File-level reduction is implemented in code using grouped reduction. Component and worktree planners label oversized payloads REDUCE_REQUIRED, but their execute paths stop instead of reducing them. No references connecting these synthesis scripts to the active V1 runner were found in the audited Python, shell and systemd paths. Separate manual invocation remains possible; completed outputs were not found in the searched artifact roots.

E2. Knowledge and retrieval - original V1.1 design

DF Memory ontology	Governed authoritative records; STAGED / APPROVED / COMMITTED	BASELINE
↓		
Embedding model	all-MiniLM-L6-v2; ONNX CPU; 384 dimensions	TESTED 21 OCT
↓		
Vector index	Derived, rebuildable SQLite-backed index	SEPARATE POC
↓		
Retrieval API / grounding	Committed-only retrieval with provenance and citation	NOT VERIFIED

The original publication reports a local embedding smoke test: related similarity 0.6665 versus unrelated 0.1861, with CPU inference, tokenisation and normalisation passing. A separate vector database was observed during the later audit; this does **not** establish that DF Codex production indexing or retrieval is integrated. Persistent retrieval accuracy and lifecycle gating remain unverified.

E3. Evidence rules

Every derived analysis or retrieval artifact should bind to repository ID, snapshot and manifest hash, source chunk/content hash, model identity, schema version, run ID and qualified lifecycle state. Retrieval similarity is a ranking signal, not proof of correctness.

F. Software and infrastructure register

Component	Observed / documented stack	Evidence status
Node	AI3090; AMD Ryzen 9 5900X, ~60 GiB RAM, 2 x RTX 3090 (original publication)	Documented 21 Oct
Runtime	Python scripts under /opt/df-codex/app; /usr/local/bin/df-codex wrapper; systemd timer/service	Audited 14 Nov
V1 model	Bonsai-27B-Q1_0; CPU inference via local HTTP 127.0.0.1:8098	Code/service audited
Memory API	DF Memory ontology service on local port 8096; SQLite authoritative storage	Observed / documented
Embedding	all-MiniLM-L6-v2, ONNX Runtime 1.31.0, 384-d, NumPy, HF Tokenizers, Python 3.14.4	Original 21 Oct test
Vector store	Separate SQLite vector proof of concept: /var/lib/df-memory-vector/vectors.db	Not proven integrated
Evidence storage	/var/lib/df-codex/smart-archive, incremental-state, runs and evidence	Audited paths
Validation	JSON Schema Draft202012Validator in synthesis; SHA256, strict JSON and coverage gates	Code audited

G. Functioning and readiness register

Capability	Classification	Basis / limitation
Scheduled no-change scan	PROVEN (scoped)	14 Nov journal: success, UP_TO_DATE, 0.171 s.
V1 change analysis	ACCEPTED HISTORICALLY	Oct V1 acceptance; not re-executed in this read-only audit.
State integrity and locks	IMPLEMENTED; TESTED IN SCOPE	SHA checks, flock, atomic per-file writes; two-file crash window.
File reduction	IMPLEMENTED / UNVERIFIED	Grouping/reduction code; no completed file artifact found.
Component/worktree reduction	PARTIAL / GAP	REDUCE_REQUIRED blocks oversized execution.
Full synthesis orchestration	NOT ESTABLISHED	No active-runner linkage found.
Production semantic retrieval	NOT VERIFIED	Embedding smoke test and separate vector POC only.
Multi-repository production	NOT VERIFIED	Only acceptance repo scheduled in audited timer.

H. Development proposals and acceptance gates

The following are proposals only. No remediation was executed during the read-only audit.

Priority	Proposed update	Acceptance evidence required
P0	Build component and worktree hierarchical reduction with bounded grouping and qualified child provenance.	Oversized real component/worktree completes without truncation; hashes and coverage reconcile.
P0	Implement one resumable stage orchestrator: chunk -> file -> component -> worktree, with stage locks and explicit skip/fail states.	End-to-end run yields all qualified artifacts; safe restart resumes without duplication.
P0	Add subprocess deadlines, bounded retries and failure reporting without disrupting existing services.	Injected hang/failure exits predictably; checkpoints never advance on failure.
P1	Close state-history/current-state crash window via recovery journal or transactional approach.	Fault injection at each write boundary recovers consistently.
P1	Validate evidence schema, model outputs and unsupported-positive-claim negation cases.	Regression tests for invalid JSON, length finish, negation and provenance.
P1	Connect committed DF Memory records to rebuildable vector index and cited retrieval API.	Lifecycle gating, change detection, rebuild and retrieval benchmark pass.
P2	Expand opt-in scheduler registry coverage and production telemetry.	Representative repositories, resource metrics and non-overlap tests pass.

I. Suggested next release gates

Gate 1: Preserve current V1 acceptance baseline and reproduce tests. **Gate 2:** Complete reduction and orchestrator with fixtures. **Gate 3:** Test recovery, failure injection and resource ceilings. **Gate 4:** Prove governed vector retrieval on committed records. **Gate 5:** Pilot multiple representative repositories before declaring production-wide readiness.

J. Update procedure for future entries

For each subsequent revision append: update ID (U002, U003...), UTC date, author/owner, affected sections, change description, checkpoint reference, evidence path or log, classification before/after, verification result, and outstanding risks. Never silently convert a proposal into a proven claim. Preserve previous dated entries and the original publication.

Current audit boundary: CP-DFCODEX-AUDIT-001 to 004 complete; audit read-only; no software, configuration or service changes made.