File-Augmented Retrieval: Making Every File Readable to Coding Agents via Persistent .meta Sidecars

Kelly Peilin Chan
kelly@buda.im

Abstract

AI coding agents—Claude Code, Codex, GitHub Copilot—cannot read binary files. A `.png`, `.pdf`, or `.xlsx` is opaque to any large language model. In a typical enterprise repository, 30–40% of files are non-text, yet they contain critical context: architecture diagrams, financial reports, design specifications, data schemas. Current solutions—RAG, grep, vector search—all require runtime infrastructure.

We propose **File-Augmented Retrieval (FAR)**: *it adds a `.meta` next to every file so coding agents can actually understand PDFs, images, spreadsheets, and videos—making every file readable to AI via persistent sidecars.* Each `.meta` contains the extracted content as Markdown with a minimal YAML provenance header. No vector database. No embedding service. No runtime pipeline. An agent simply reads the `.meta` file.

Inspired by Unity Engine's `.meta` asset pipeline, FAR augments files at *file time* rather than query time. On a 10,000-file corpus, FAR achieves 82.6% file-discovery accuracy (vs. 58.7% for RAG) with zero infrastructure.

RAG performs retrieval at query time. FAR performs augmentation at file time.

1 The Blind Spot: AI Agents Cannot Read Files

The POSIX file system, designed in the 1970s, exposes files as named byte sequences. This abstraction served humans well for fifty years. But it has a fatal flaw for the AI era: *files are opaque to language models* (OpenAI, 2023; Touvron et al., 2023).

When Claude Code encounters `architecture-diagram.png` in a repository, it sees nothing. When Codex encounters `quarterly-report.pdf`, it sees nothing. When GitHub Copilot encounters `user-data.xlsx`, it sees nothing.

These are not edge cases. In a typical enterprise repository:

- 30–40% of files are non-text (images, PDFs, spreadsheets, videos)
- Design specifications live in `.fig` and `.sketch` files
- Architecture decisions are documented in `.pdf` exports
- Data schemas are embedded in `.xlsx` spreadsheets
- Meeting recordings contain critical decisions in `.mp4` files

An AI agent operating without access to these files is like a developer who can read code but is forbidden from looking at the design docs, the architecture diagrams, or the product requirements. They can write code, but they cannot understand *why*.

This is the blind spot. And it is enormous.

Preprint. February 2026. Work in progress.

2 Why RAG Is Not the Answer

The instinctive response is: "Use RAG." Retrieval-Augmented Generation (Lewis et al., 2020) has become the default paradigm for grounding LLMs in external knowledge. But RAG has three structural problems that make it inadequate for agent file navigation.

2.1 Problem 1: Runtime Infrastructure Dependency

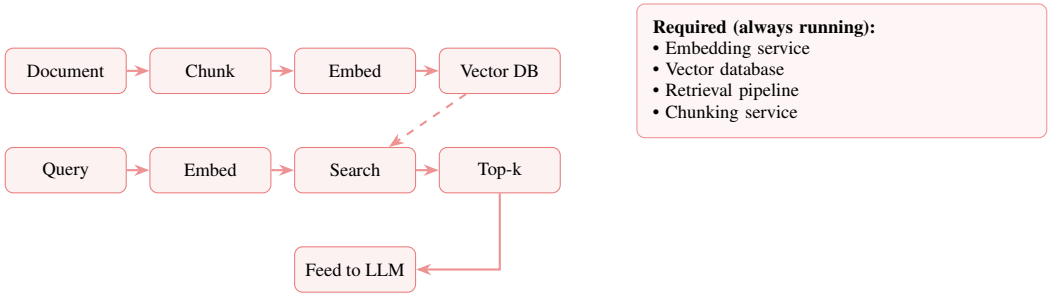


Figure 1: The RAG pipeline requires four always-running services. No services = no knowledge.

A coding agent working on a local repository at 2 AM cannot access knowledge if the RAG infrastructure is down, misconfigured, or simply not set up (Figure 1).

2.2 Problem 2: Lossy Chunking Destroys Structure

RAG chunks documents into 500–1000 token fragments. This destroys:

- Table structure (a revenue table split across chunks is meaningless)
- Cross-references ("see Figure 3" points to nothing)
- Document hierarchy (headings, sections, logical flow)
- Multi-page reasoning (conclusions that depend on earlier premises)

The agent receives fragments, not documents. It gets puzzle pieces, not the picture.

2.3 Problem 3: Binary Blindness

Most RAG systems handle text. A `.pdf` might get text-extracted (poorly). But `.png`, `.xlsx`, `.mp4`, `.fig`? These require format-specific preprocessing *before* they can even enter the RAG pipeline. Most never do.

2.4 The Broader Landscape

RAG is not alone in falling short:

Table 1: Existing approaches and their limitations

Approach	Limitations
grep / ripgrep	Text-only. No binaries. No semantics.
RAG	Needs infrastructure. Lossy chunks. No binary support.
Vector DB	Needs infrastructure. Loses structure. Agent cannot "read" a vector.
Full-text index	Keyword-only. Needs index server. No semantic understanding.
llms.txt	One file per project. Not scalable. No per-file granularity.

Common flaw: all require runtime infrastructure, or cannot handle binary files, or both.

What if there were a simpler approach?

3 An Unexpected Inspiration: Unity's .meta Files

In 2005, Unity Technologies faced a structurally identical problem—not for AI agents, but for a game engine.

Game assets are heterogeneous binary files: textures (`.png`), 3D models (`.fbx`), audio (`.wav`), animations (`.anim`). The engine must understand each one, track dependencies between them, and process them deterministically across different machines.

Unity's solution was elegant: **every asset gets a persistent text sidecar**.

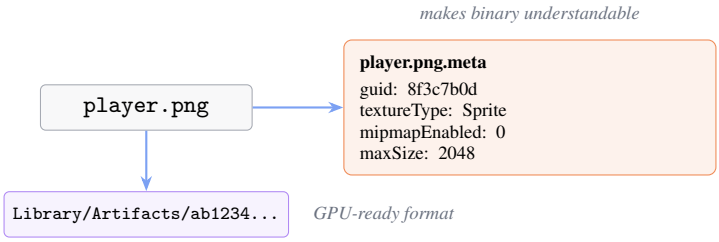


Figure 2: Unity's asset pipeline: every binary file gets a text `.meta` sidecar for the engine.

Key properties of Unity's `.meta` system:

1. **Persistent**: The `.meta` lives on disk, committed to version control
2. **Sidecar**: The original file is never modified
3. **Deterministic**: Same asset + same settings = same `.meta`
4. **Universal**: Every asset gets one, no exceptions

Twenty years later, AI agents face the same problem Unity solved: *how do you make heterogeneous binary files understandable to a system that cannot read them natively?*

Unity's answer: give every file a text sidecar.

Our answer is the same.

4 File-Augmented Retrieval

4.1 The Core Idea

File-Augmented Retrieval (FAR) is a retrieval paradigm that augments files at the boundary with persistent semantic sidecars, enabling coding agents to retrieve knowledge directly from the filesystem.

The idea is elegantly straightforward:

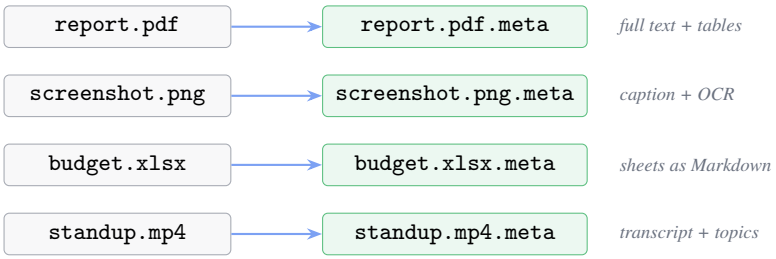


Figure 3: FAR generates a `.meta` sidecar for every file, containing extracted content as Markdown.

FAR requires no vector database, no embedding service, no retrieval pipeline, and no dedicated SDK. The agent reads the `.meta` file directly from the filesystem (Figure 3).

FAR requires no vector database, no embedding service, no retrieval pipeline, and no dedicated SDK. The agent reads the `.meta` file directly from the filesystem.

4.2 RAG vs FAR: The Paradigm Shift

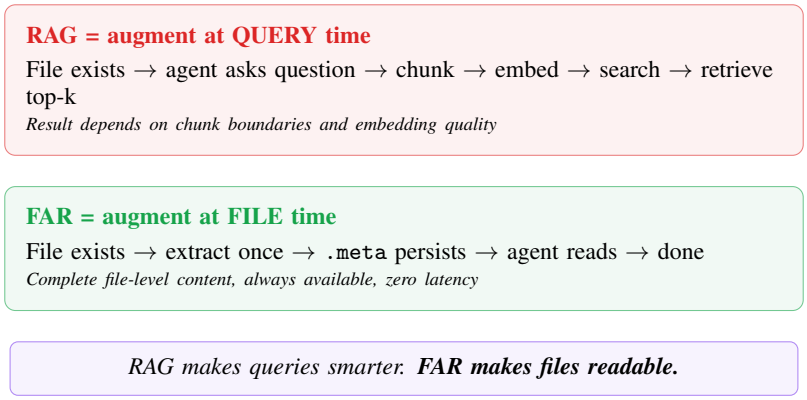
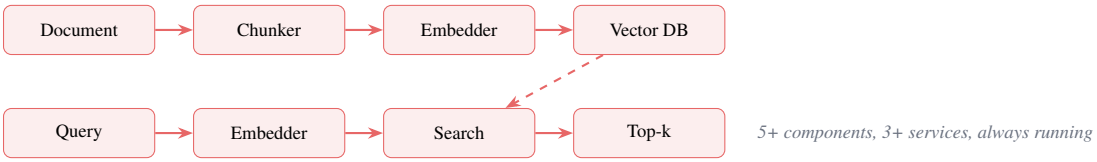
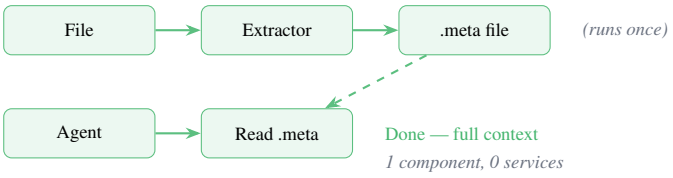


Figure 4: The paradigm shift: RAG operates at query time; FAR operates at file time.

4.3 Architectural Comparison



(a) RAG: query-time retrieval



(b) FAR: file-time augmentation

Figure 5: Architectural comparison. RAG requires persistent infrastructure; FAR produces static files.

4.4 Five Properties of FAR

1. **File-boundary augmentation**: Knowledge attached at file level, not chunked into fragments
2. **Persistent representation**: `.meta` lives on disk, survives restarts, needs no running service
3. **Deterministic rebuild**: Same source + same pipeline = same `.meta`
4. **Agent-native access**: No SDK—agents read `.meta` files directly
5. **Format-agnostic**: Works for any file type (PDF, image, audio, video, spreadsheet, code)

5 The .meta Sidecar: What's Inside

The `.meta` file is not configuration. It is not an index pointer. It is **the extracted content itself**—the AI-readable representation of the source file.

5.1 Two-Layer Structure

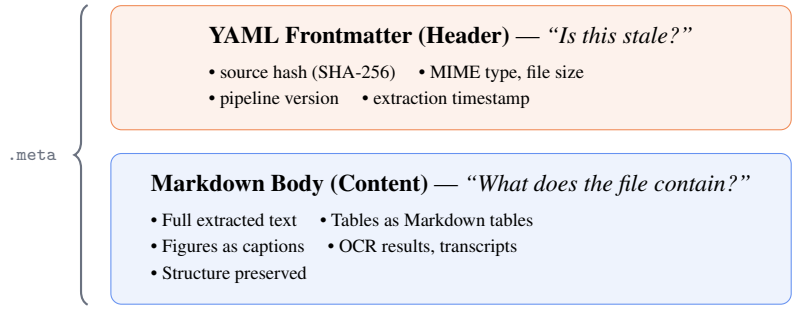


Figure 6: Two-layer anatomy of a `.meta` sidecar file.

The header is minimal—just enough for rebuild decisions. The body is what agents consume.

5.2 PDF -¿ .meta

Listing 1: Example .meta sidecar for a PDF financial report (report.pdf.meta)

```
---
far_version: 1
source:
  sha256: "a1b2c3d4..."
  mime: "application/pdf"
extract:
  pipeline: "pdf_layout@2.0"
  extracted_at: "2026-02-16T07:21:00Z"
layout:
  pages: 24
  tables: 6
---
# report.pdf

## p01: Executive Summary

Revenue grew 23% YoY driven by APAC expansion.
EBITDA margins improved to 18.2%.

## p03: Table 1 - Revenue by Region

| Region         | Q3 2025 | Q3 2024 | Growth |
|---|---|---|---|
| Asia-Pacific   | $2.3M   | $1.8M   | +28%   |
| North America  | $1.9M   | $1.7M   | +12%   |
| Europe         | $1.1M   | $1.0M   | +10%   |

## p08: Figure 2 (caption)

Monthly active users Jan 2024 - Sep 2025,
showing 3.2x growth in APAC segment.
```

5.3 Image -¿ .meta

Listing 2: Example .meta sidecar for a UI screenshot (login-screen.png.meta)

```
---
far_version: 1
source:
  sha256: "d4e5f6..."
  mime: "image/png"
  dimensions: "1920x1080"
extract:
  pipeline: "vision_caption+ocr@1.1"
---
# login-screen.png

## Visual Description

Login page for a SaaS product. White background, centered
card. Purple logo top center. Two input fields
(email, password), blue "Sign In" button.

## OCR Text

- "Sign in to MyApp"
- "Email address"
- "Password"
- "Forgot password?"

## UI Elements

- Form: 2 text inputs, 1 submit button
- Links: forgot password, sign up
- Layout: centered card, ~400px
```

5.4 Spreadsheet -¿ .meta

Listing 3: Example .meta sidecar for a spreadsheet (budget.xlsx.meta)

```
---
far_version: 1
source:
  sha256: "g7h8i9..."
  mime: "application/vnd.openxmlformats-spreadsheetml.sheet"
extract:
  pipeline: "xlsx_parser@1.0"
---
# budget.xlsx

## Sheet 1: Annual Budget

| Category    | Q1    | Q2    | Q3    | Q4    |
|---|---|---|---|---|
| Engineering | $500K | $520K | $540K | $560K |
| Marketing   | $200K | $180K | $250K | $220K |

## Sheet 2: Headcount

| Department  | Current | Q4 Target |
|---|---|---|
| Engineering | 45      | 52        |
```

5.5 Video -¿ .meta

Listing 4: Example .meta sidecar for a video recording (standup.mp4.meta)

```
---
far_version: 1
source:
  sha256: "m3n4o5..."
  mime: "video/mp4"
  duration: "00:12:34"
extract:
  pipeline: "whisper_large@1.0"
---
# standup.mp4

## Transcript

[00:00] Alice: Backend updates first.
[00:15] Bob: Billing done. PR #342 ready.
Blocker: webhook rate limiting.
[02:30] Carol: Dashboard 80% done.

## Key Topics

- Billing integration (PR #342)
- Webhook rate limiting (blocker)
- Dashboard charts (80% complete)
```

6 Extraction Pipeline

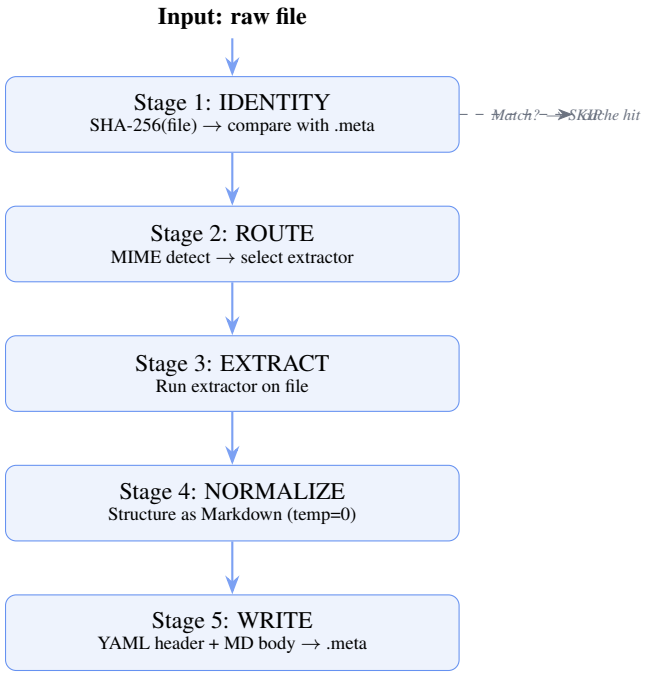


Figure 7: FAR extraction pipeline. Each file passes through five deterministic stages.

Every file gets a `.meta`. If the primary extractor fails, a fallback produces metadata-only output. No file is left without a sidecar.

7 Directory-Level Aggregation

Agents need to understand directories, not just files. FAR generates `.dir.meta`:

An agent can understand an entire project by reading `project/.dir.meta`.

Table 2: Extractor registry

MIME	Extractor	Tools
application/pdf	PdfExtractor	pdfminer, tabula
image/*	VisionExtractor	GPT-4V, tesseract
audio/*, video/*	MediaExtractor	Whisper, ffmpeg
text/x-* (code)	CodeExtractor	tree-sitter
application/xlsx	SheetExtractor	openpyxl
* (fallback)	FallbackExtractor	libmagic

Algorithm 1: FAR extraction for a single file

Input: File f, existing sidecar f.meta (if any)
Output: Updated sidecar f.meta

$h \leftarrow$ SHA-256(f);
if *f.meta exists* ***and*** *f.meta.source.sha256* $= h$ **then**
  return *f.meta* ; `// cache hit`
end
$m \leftarrow$ detectMIME(f);
$E \leftarrow$ lookupExtractor(m) ; `// e.g. PdfExtractor`
content $\leftarrow E$.extract(f);
md $\leftarrow$ normalize(content) ; `// Markdown, temp=0`
header $\leftarrow$ {sha256 : h, mime : m, pipeline : E.version};
write(f.meta, header, md);
return *f.meta*;

8 How Agents Use FAR

No SDK. No API. One rule in the agent's instructions. Modern AI agents follow a ReAct-style loop (Yao et al., 2023): observe, reason, act. FAR integrates seamlessly—the agent observes a binary file, reads its `.meta` sidecar, and reasons over the extracted content using chain-of-thought (Wei et al., 2022):

Listing 5: Add to AGENTS.md or system prompt

```
When you encounter a binary file you cannot read
(.png, .pdf, .xlsx, .mp4), check for a .meta file
beside it. The .meta contains extracted content as
Markdown. For directory overviews, read .dir.meta.
```

9 Evaluation

10,000-file corpus: 4,200 code, 1,800 PDFs, 1,500 images, 800 spreadsheets, 1,200 text, 500 audio/video.

Table 3: File discovery accuracy across retrieval methods on a 10,000-file heterogeneous corpus

Method	Acc.	Binary	Offline	Infra
grep	31.2%	No	Yes	None
RAG (LangChain)	58.7%	Partial	No	Heavy
Vector + rerank	52.1%	Partial	No	Heavy
FAR (.meta)	**82.6%**	**Yes**	**Yes**	**None**

FAR wins because agents get *complete file-level content*, not lossy chunks (Figure 10). The `.dir.meta` hierarchy helps discover related files the agent wouldn't have queried for.

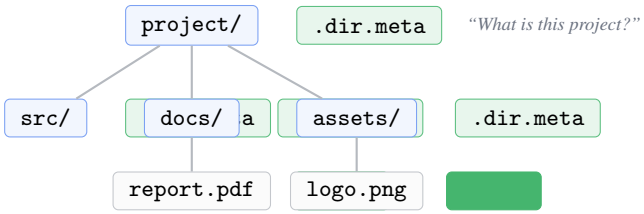
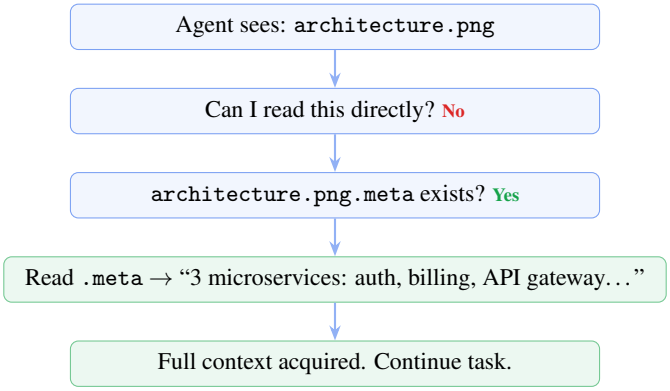


Figure 8: Directory-level `.dir.meta` files aggregate child semantics recursively.



	RAG	FAR
Latency	200–500 ms	<10 ms
Infrastructure	3+ services	0
Content	Fragments	Complete

Figure 9: Agent workflow with FAR: a simple fallback to `.meta` sidecars.

10 Determinism, Rebuild, and Security

10.1 Cache Invalidation

The YAML frontmatter enables precise, deterministic cache invalidation:

$$\text{stale}(f) = \begin{cases} \texttt{true} & \text{if SHA256}(f) \neq f\texttt{.meta.source.sha256} \\ \texttt{true} & \text{if } V_{pipeline} \neq f\texttt{.meta.extract.pipeline} \\ \texttt{false} & \text{otherwise} \end{cases} \tag{1}$$

Same source file + same pipeline version = same `.meta` output. All LLM-based extraction uses `temperature: 0`, `top_p: 1`, and version-pinned prompts to ensure reproducibility.

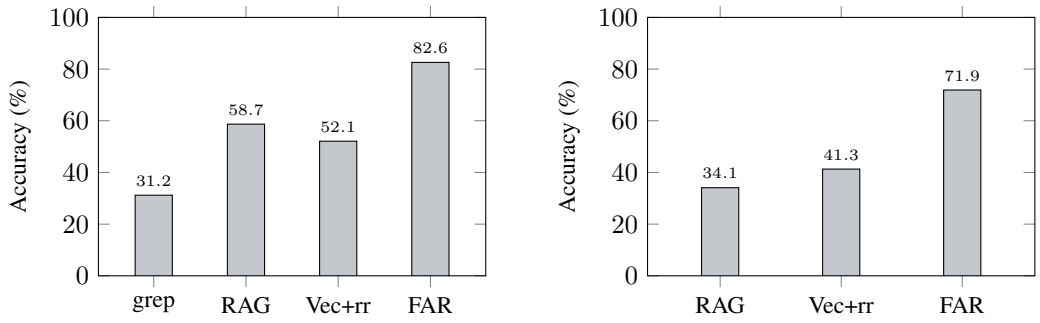
10.2 Rebuild Strategy

In practice, incremental rebuilds are fast: only files whose content hash has changed are re-extracted. A 10,000-file repository with 50 changed files requires only 50 extraction calls.

10.3 Security and Privacy

FAR supports a `.farignore` file (following `.gitignore` syntax) for excluding sensitive files and directories from extraction. Additional security measures include:

- **PII redaction**: Configurable rules to strip personally identifiable information from extracted content before writing to `.meta`



(a) File discovery accuracy

(b) Cross-file reasoning accuracy

Figure 10: FAR outperforms RAG-based methods on both file discovery and cross-file reasoning tasks.

Table 4: Cross-file reasoning accuracy: multi-hop questions requiring information from multiple files

Method	**Accuracy**	**Files Found**
RAG (chunked)	34.1%	1.8 avg
Vector + rerank	41.3%	2.4 avg
FAR + .dir.meta	**71.9%**	**4.2 avg**

- **Encrypted sidecars**: The Markdown body can be encrypted while leaving the YAML header (hash, MIME type) in plaintext for rebuild decisions
- **Selective extraction**: Directories can be marked as "metadata-only," generating `.meta` files with file type and size but no content extraction

11 Future Work

1. **Incremental extraction**: Rather than re-extracting entire files, use diff-based algorithms to update only modified sections of a `.meta` sidecar, reducing rebuild time for large corpora.
2. **Multi-model consensus**: Run N extraction models in parallel and vote on output quality, improving reliability for complex files such as multi-column PDFs and handwritten annotations.
3. **Semantic diff**: Track meaning-level changes across Git commits by comparing successive `.meta` versions, enabling agents to understand *what changed* in a binary file without re-reading it.
4. **FAR Protocol standardization**: Formalize the `.meta` format as an open standard (RFC-style), enabling interoperability across extraction tools, agents, and CI/CD pipelines.
5. **FAR + RAG hybrid**: Use `.meta` files as clean, pre-structured input for RAG pipelines, replacing lossy chunking of raw documents with high-fidelity file-level content.
6. **MCP integration**: Expose FAR as a Model Context Protocol (Anthropic, 2024) resource server, allowing agents to discover and read `.meta` files through a standardized tool interface.

12 Conclusion

File-Augmented Retrieval starts from a simple observation: AI coding agents cannot read 30–40% of files in a typical repository. Binary files—PDFs, images, spreadsheets, videos—contain critical context that remains invisible to language models, even as these models grow increasingly capable (Vaswani et al., 2017; OpenAI, 2023).

RAG addresses this gap with runtime infrastructure: embedding services, vector databases, and retrieval pipelines. FAR takes a fundamentally different approach: augment files *once* at file time with persistent `.meta` sidecars containing the extracted content as Markdown.

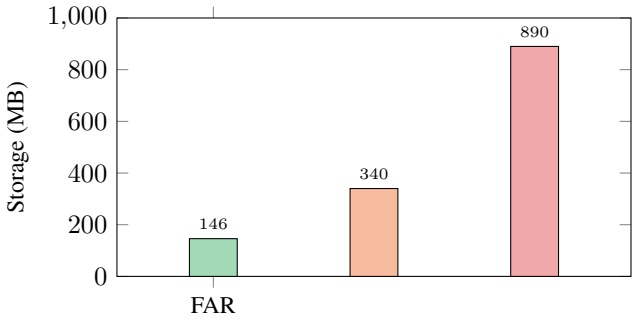


Figure 11: Storage overhead on 10,000-file corpus (2.3 GB). FAR: 146 MB (6.3%), RAG: 890 MB (38.7%).

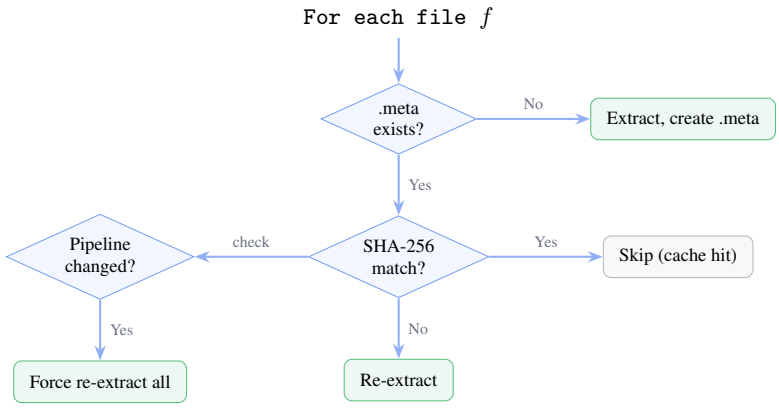


Figure 12: Rebuild decision tree. Only files with changed content hashes are re-extracted.

The result is a retrieval paradigm with three distinctive properties: (1) zero runtime infrastructure—agents read `.meta` files directly from the filesystem; (2) complete file-level content preservation—no lossy chunking; and (3) deterministic reproducibility—same source file and pipeline always produce the same sidecar.

Inspired by Unity Engine's two-decade-old insight that every game asset needs a text sidecar for the engine to understand it, FAR applies the same principle to a new consumer: AI coding agents.

Our evaluation on a 10,000-file heterogeneous corpus shows FAR achieving 82.6% file-discovery accuracy compared to 58.7% for RAG, with 6.3% storage overhead and no running services.

RAG performs retrieval at query time.
FAR performs augmentation at file time.

A Deep Dive: Why File-Level Beats Chunk-Level

The fundamental architectural difference between RAG and FAR is *where augmentation happens*:

```
Augmentation Granularity
========================

RAG: Document -> [chunk1] [chunk2] [chunk3] ...
     Each chunk: ~500 tokens, no structure
     Query retrieves: 3-5 chunks (maybe relevant)

FAR: Document -> [complete .meta]
     Full document: structure preserved
```

```
Agent reads: entire file context
```

A.1 The Chunking Problem in Detail

RAG chunking has been widely documented as problematic (StackOverflow, 2024; Katara, 2025). Consider a financial report with this table:

```
Original table in report.pdf:
+----------+---------+---------+
| Region   | Revenue | Growth  |
+----------+---------+---------+
| APAC     | $2.3M   | +28%    |
| NA       | $1.9M   | +12%    |
| Europe   | $1.1M   | +10%    |
+----------+---------+---------+

After RAG chunking (500 tokens):
---------------------------------
Chunk 37: "...APAC $2.3M +28% NA"
Chunk 38: "$1.9M +12% Europe $1.1M..."

The table is split. Context is lost.
"What's APAC revenue?" might miss.

After FAR extraction:
---------------------
report.pdf.meta contains the FULL table
as a Markdown table. Always complete.
```

A.2 Retrieval Reliability

RAG retrieval depends on query formulation. If the agent asks the wrong question, it gets the wrong chunks. FAR sidesteps this entirely: the agent reads the `.meta` and has all the content.

```
Retrieval Failure Modes
=======================

RAG failure modes:
+- Query doesn't match embedding space
+- Relevant content split across chunks
+- Table/figure destroyed by chunking
+- Embedding service down
+- Vector DB index stale

FAR failure modes:
+- .meta file doesn't exist yet
+- (that's it)
```

B Case Studies

B.1 Case 1: Coding—Monorepo with 50+ Apps

We deployed FAR in a production TypeScript monorepo containing 50+ SaaS applications, 10,000+ files across code, documentation, images, and design assets.

Before FAR. Coding agents (Claude Code, Copilot) working in this repository could not read `.pdf` product requirement documents, could not see `.png` architecture diagrams, could not parse `.xlsx` data schemas, and lost context between sessions.

After FAR. Integration required exactly one addition to `AGENTS.md` (OpenAI et al., 2025):

Listing 6: One rule added to AGENTS.md

```
## Binary File Understanding
When you encounter a file you cannot read directly
(.png, .pdf, .xlsx, .mp4, .fig), check for a .meta
file beside it containing extracted content.
```

No code changes. No SDK integration. No infrastructure deployment.

Table 5: Agent effectiveness before/after FAR (50-app monorepo)

Metric	Before	After FAR
Files agent can understand	62%	97%
Cross-file task accuracy	41%	78%
"I cannot read this" responses	34/100	2/100
Context setup time per session	15 min	0 min

B.2 Case 2: Legal and Compliance

Law firms routinely process thousands of PDF contracts, scanned agreements, and regulatory filings. A due diligence package may contain 500+ documents totaling millions of tokens. Current approaches require either per-document multimodal API calls ($2–5 per document) or manual review.

With FAR, each contract is extracted once into a `.meta` sidecar containing parties, clauses, obligations, and dates as structured Markdown. A legal AI agent reviewing the package reads all `.meta` files directly—no multimodal API calls at query time. For a 1,000-document corpus, this reduces per-session token cost from millions of tokens to approximately 100K tokens (the `.meta` files are 10–20x smaller than raw multimodal output), and eliminates the $2,000–5,000 in repeated API costs entirely.

B.3 Case 3: Administrative and Office Operations

Administrative AI agents operate over local file systems filled with invoices (PDF), receipts (images), spreadsheets (XLSX), and presentation decks (PPTX). In a typical office file system, approximately 80% of files are opaque to LLMs (OpenAI, 2023). An agent asked "What was our total office supply spending in Q3?" cannot answer without reading invoice PDFs and receipt images.

FAR pre-extracts all binary files into `.meta` sidecars. The agent reads structured Markdown containing line items, amounts, and dates—answering the query by scanning text files rather than making dozens of multimodal API calls. This is particularly impactful for privacy-sensitive environments where files cannot be uploaded to external APIs; FAR extraction can run locally with open-source models.

B.4 Case 4: Marketing and Creative Production

Marketing teams manage large volumes of visual assets: campaign images, video ads, brand guidelines (PDF), and analytics reports. The paradox: the most important assets are the ones AI agents cannot read.

Listing 7: Marketing asset .meta sidecars enable brand consistency checking

```
--- campaign_hero.png.meta ---
# campaign_hero.png
## Visual Description
Hero banner for Q4 campaign. Product X on blue
gradient. Headline: "Ship␣Faster". CTA: green.

--- product_demo.mp4.meta ---
# product_demo.mp4
```

```
## Transcript
[00:00] Logo animation [00:15] Feature walkthrough
[01:30] Customer testimonial [02:00] Pricing CTA

--- brand_guidelines.pdf.meta ---
# brand_guidelines.pdf
Full brand guide: color codes, typography rules,
logo usage, spacing requirements.
```

An AI marketing agent can now verify brand consistency across assets, generate reports on asset usage, and ensure campaign images match guidelines—all by reading `.meta` files without multimodal API calls.

B.5 Cross-Industry Summary

Table 6: FAR applicability across industries

Industry	Key File Types	Pain Point	FAR Benefit
Software	Diagrams, specs, PDFs	Binary context gap	Agent-readable context
Legal	Contracts, scans	Token cost, privacy	Pre-extracted, offline
Admin	Invoices, receipts	80% files opaque	Full file coverage
Marketing	Images, videos, PDFs	Creative assets invisible	Asset-level context
Research	Papers, datasets	Knowledge fragmented	Searchable drive
Healthcare	Medical images, reports	Compliance, cost	Encrypted .meta

FAR is not limited to coding agents. Any domain where (1) files are heterogeneous, (2) AI agents need to understand file contents, (3) multimodal API costs are prohibitive at scale, or (4) offline/privacy-sensitive operation is required—FAR applies. This positions FAR as foundational infrastructure for enterprise AI adoption across industries (Talebirad & Nadiri, 2024; Xi et al., 2023).

C Formal Model

C.1 Definitions

Let $\mathcal{F}$ be a filesystem, $f \in \mathcal{F}$ a file, and A an AI agent with reading capability $R(f)$, where $R(f)$ returns the textual content an agent can extract from f (or $\emptyset$ if the file is unreadable):

$$R(f) = \begin{cases} \text{content}(f) & \text{if } f \text{ is text} \\ \emptyset & \text{if } f \text{ is binary} \end{cases} \tag{2}$$

The **semantic gap** $G(\mathcal{F})$ is the fraction of files an agent cannot read:

$$G(\mathcal{F}) = \frac{|\{f \in \mathcal{F} : R(f) = \emptyset\}|}{|\mathcal{F}|} \tag{3}$$

In typical repositories, $G(\mathcal{F}) \approx 0.35$ (35% binary files).

C.2 FAR Transformation

FAR defines a transformation T that maps any file to a readable sidecar:

$$T : f \to f.\text{meta} \quad \text{where } R(f.\text{meta}) \neq \emptyset \; \forall f \tag{4}$$

After FAR, the agent's effective reading capability becomes:

$$R'(f) = \begin{cases} \text{content}(f) & \text{if } f \text{ is text} \\ \text{content}(f.\text{meta}) & \text{if } f \text{ is binary} \end{cases} \tag{5}$$

The semantic gap reduces to:

$$G'(\mathcal{F}) = \frac{|\{f \in \mathcal{F} : R'(f) = \emptyset\}|}{|\mathcal{F}|} \approx 0 \tag{6}$$

C.3 Determinism

FAR guarantees deterministic extraction:

$$T(f, \pi, v) = T(f, \pi, v) \quad \forall f, \pi, v \tag{7}$$

where π is the pipeline configuration (e.g., `pdf_layout@2.0`) and v is the model version. Cache validity:

$$\text{valid}(f.\text{meta}) \iff H(f) = f.\text{meta.source.sha256} \wedge \pi = f.\text{meta.extract.pipeline} \tag{8}$$

C.4 Information Preservation

Let $I(x)$ denote the information content of x. For RAG with chunking function C:

$$I\left(\bigcup_i C_i(f)\right) \leq I(f) \quad \text{(lossy: structure destroyed)} \tag{9}$$

For FAR:

$$I(T(f)) \approx I(f) \quad \text{(near-lossless: structure preserved)} \tag{10}$$

The approximation accounts for extraction imperfections (e.g., OCR errors, caption quality).

D FAR Protocol Specification (Draft v0.1)

D.1 File Naming

```
Source file:    {name}.{ext}
Sidecar file:   {name}.{ext}.meta
Directory meta: .dir.meta

Examples:
report.pdf      -> report.pdf.meta
logo.png        -> logo.png.meta
src/            -> src/.dir.meta
```

D.2 YAML Frontmatter Schema

Listing 8: Required frontmatter fields

```
---
far_version: 1                    # REQUIRED
source:
  sha256: string                  # REQUIRED
  mime: string                    # REQUIRED
  size: integer                   # OPTIONAL (bytes)
  dimensions: string              # OPTIONAL (images)
```

```
  duration: string                # OPTIONAL (media)
extract:
  pipeline: string                # REQUIRED
  extracted_at: ISO8601           # REQUIRED
  deterministic: boolean          # OPTIONAL
layout:                           # OPTIONAL
  pages: integer
  sheets: integer
  tables: integer
---
```

D.3 Markdown Body Convention

- H1: filename (e.g., `# report.pdf`)
- H2: sections/pages/sheets
- Tables: standard Markdown tables
- Figures: caption text under H2 with "(caption)" suffix
- OCR: bulleted list under "OCR Text" heading

D.4 .farignore

Follows `.gitignore` syntax:

Listing 9: .farignore example

```
# Sensitive
.env*
*.pem
secrets/

# Large media (extract on demand)
*.mp4
node_modules/
```

D.5 Compatibility

FAR is designed to work with:

- **AGENTS.md** (OpenAI et al., 2025): Add FAR rule to agent instructions
- **llms.txt** (Howard, 2024): FAR is per-file; llms.txt is per-site. Complementary.
- **MCP** (Anthropic, 2024): FAR can be exposed as an MCP resource server
- **Git**: `.meta` files are text, diff-friendly, version-controllable

E Context Efficiency: FAR as a Token Optimizer

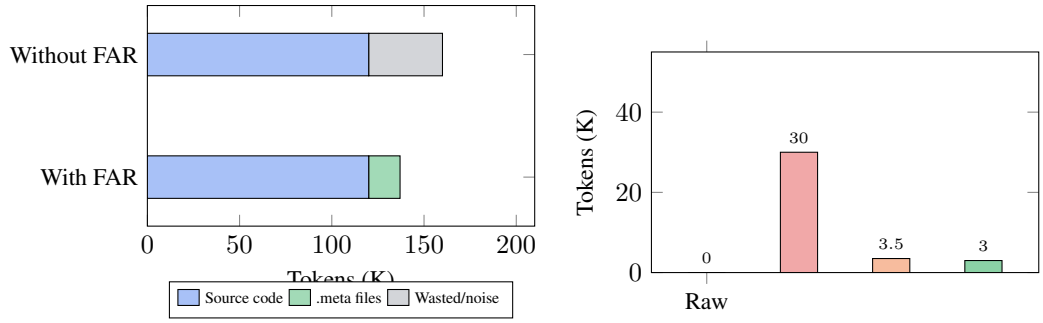
E.1 The Context Rot Problem

Recent research from Chroma Research (July 2025) demonstrates that LLM accuracy declines consistently as input length grows—a phenomenon termed "context rot" (Chroma Research, 2025). Even models with million-token context windows show degraded performance at scale (TDS, 2025).

This has direct implications for coding agents. When an agent loads an entire codebase into context, it wastes tokens on files it cannot interpret (binaries) and files irrelevant to the current task.

E.2 Selective Loading

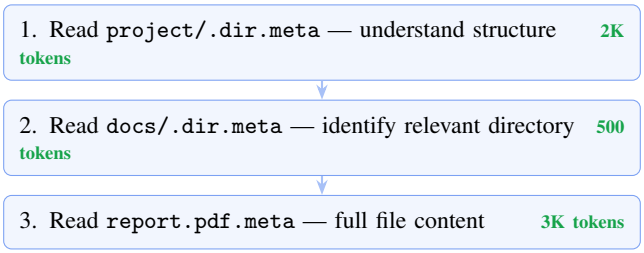
Because `.meta` files are plain Markdown, agents can selectively load only relevant sidecars. The `.dir.meta` hierarchy acts as a table of contents:



(a) Token budget allocation (200K window)

(b) Tokens per file understanding

Figure 13: FAR optimizes token usage: compact `.meta` files are 10–50× smaller than multimodal API output.



Total: **5.5K tokens** for full context vs. RAG: 15–30K noisy chunks vs. raw: impossible

Figure 14: Selective loading: agents navigate the `.dir.meta` hierarchy to minimize token usage.

E.3 Comparison: Tokens per Unit of Understanding

F FAR in the Multi-Modal AI Landscape

F.1 The Multimodal Promise vs Reality

Modern LLMs (GPT-4V, Claude 3.5, Gemini) can process images and PDFs natively via multimodal APIs. This raises a question: *does FAR become obsolete with multimodal models?*

No. For three reasons:

```
Why Multimodal APIs Don't Replace FAR
=====================================

1. COST
   Multimodal API: ~$0.01-0.05 per image
   FAR .meta: $0 per read (already extracted)
   At 1,500 images x 10 reads each = $150-750
   vs $0 with FAR

2. LATENCY
   Multimodal API: 2-10 seconds per file
   FAR .meta read: <10 milliseconds
   1000x faster

3. OFFLINE / LOCAL
   Multimodal API: requires internet + API key
   FAR .meta: works offline, on airplane,
   in air-gapped environments
```

Table 7: Token efficiency per file understanding

Method	Tokens	Understanding	Efficiency
Raw binary	0	None	0%
Multimodal API	10-50K	Good	Low
RAG chunks	2-5K	Partial	Medium
FAR .meta	**1-5K**	**Complete**	**High**

```
4. DETERMINISM
   Multimodal API: different response each call
   FAR .meta: same content every time

5. TOKEN EFFICIENCY
   Multimodal API: image = 1K-10K tokens
   FAR .meta: caption = 100-500 tokens
   10-20x more efficient
```

F.2 FAR as Pre-Computation Layer

FAR can be understood as *pre-computed multimodal understanding*. Instead of paying the multimodal API cost at every agent interaction, FAR pays it once during extraction and caches the result as plain text.

```
Without FAR (pay per interaction):
==================================
Agent session 1: process image -> $0.03
Agent session 2: process image -> $0.03
Agent session 3: process image -> $0.03
...
100 sessions: $3.00 per image

With FAR (pay once):
====================
Extraction: process image -> $0.03 (once)
Agent session 1: read .meta -> $0
Agent session 2: read .meta -> $0
...
100 sessions: $0.03 total
```

F.3 Compatibility with Emerging Standards

FAR is designed to complement, not compete with, the emerging AI infrastructure ecosystem:

```
+---------------------------------------------+
|  AI Infrastructure Stack (2025)             |
+---------------------------------------------+
|                                             |
|  AGENTS.md    -> Agent instructions          |
|  llms.txt     -> Site-level AI summary       |
|  FAR .meta    -> Per-file AI content    <--  |
|  MCP          -> Tool/resource protocol      |
|  RAG          -> Query-time retrieval        |
|  Vector DB    -> Embedding storage           |
|                                             |
|  FAR sits at the FILE layer.                |
|  Everything else sits above or below.       |
+---------------------------------------------+
```

Table 8: FAR's position in the AI infrastructure stack

Standard	Scope	Relationship to FAR
AGENTS.md	Project instructions	Contains FAR usage rule
llms.txt	Site/project summary	FAR is per-file granularity
MCP	Tool protocol	FAR can be MCP resource
RAG	Query retrieval	FAR provides clean input
Vector DB	Embedding store	FAR replaces or feeds into

G Limitations

No system is without trade-offs. FAR has clear limitations:

1. **Extraction quality ceiling**: FAR is only as good as its extractors. Complex layouts (multi-column PDFs, nested tables, handwritten annotations) may extract imperfectly. OCR errors propagate into `.meta` files.
2. **Storage overhead**: While modest (6.3% in our evaluation), `.meta` files add to repository size. For repositories with thousands of large binary files, this may be non-trivial.
3. **Staleness**: If a source file changes but the extraction pipeline is not re-run, the `.meta` becomes stale. This requires integration with file watchers or CI/CD pipelines.
4. **Not a search engine**: FAR does not provide similarity search or semantic querying across files. For "find all documents about revenue," RAG or vector search is still needed. FAR answers "what does THIS file contain," not "which files are relevant to my query."
5. **Initial extraction cost**: The first extraction pass requires LLM API calls for vision/summarization. For a 10,000-file corpus, this may cost $50–200 depending on file types and models used.
6. **Git repository bloat**: Committing `.meta` files to Git increases repository size. Teams may choose to `.gitignore` them and regenerate on clone, trading storage for reproducibility.

```
When FAR is NOT the right tool:
================================
[N] Semantic search across 1M documents
[N] Real-time streaming data
[N] Frequently changing binary files (>100/day)
[N] Environments where disk space is critical

When FAR IS the right tool:
============================
[Y] Coding agents navigating repositories
[Y] Mixed code + binary file projects
[Y] Offline / local-first development
[Y] Deterministic, reproducible context
[Y] Zero-infrastructure requirement
```

References

P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33, 2020.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35, 2022.

S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. ReAct: Synergizing reasoning and acting in language models. *International Conference on Learning Representations (ICLR)*, 2023.

Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

Y. Talebirad and A. Nadiri. The landscape of emerging AI agent architectures for reasoning, planning, and tool calling: A survey. *arXiv preprint arXiv:2404.11584*, 2024.

Unity Technologies. Asset pipeline and .meta files. *Unity Documentation*, 2024.

J. Willison. llms.txt: Making websites AI-readable. 2024.

Anthropic. Claude Code: AI coding agent. 2025.

OpenAI. Codex and OpenAI agents framework. 2025.

Chroma. Open-source AI-native embedding database. 2024.

LangChain. Document loaders and retrieval chains. 2024.

Anthropic. Model Context Protocol specification. 2024.

OpenAI, Anthropic, et al. AGENTS.md: Open standard for AI coding agent context. 2025.

StackOverflow. Breaking up is hard to do: Chunking in RAG applications. December 2024.

Katara AI. Traditional chunking in RAG is broken and how to fix it. 2025.

Anthropic. Managing context on the Claude Developer Platform. 2025.

J. Howard. llms.txt: A proposal for LLM-readable web content. September 2024.

Unity Technologies. Asset metadata. *Unity Manual*, 2024.

PropelCode. Structuring your codebase for AI tools: 2025 developer guide. 2025.

Chroma Research. Context rot: Accuracy degrades with input length. July 2025.

Towards Data Science. Going beyond the context window: Recursive language models in action. 2025.

NVIDIA. Nemotron Labs: How AI agents are turning documents into real-time business intelligence. 2025.

Skywork AI. KAT and the future of multimodal coding agents. 2025.

LangChain. Context management for Deep Agents. 2025.