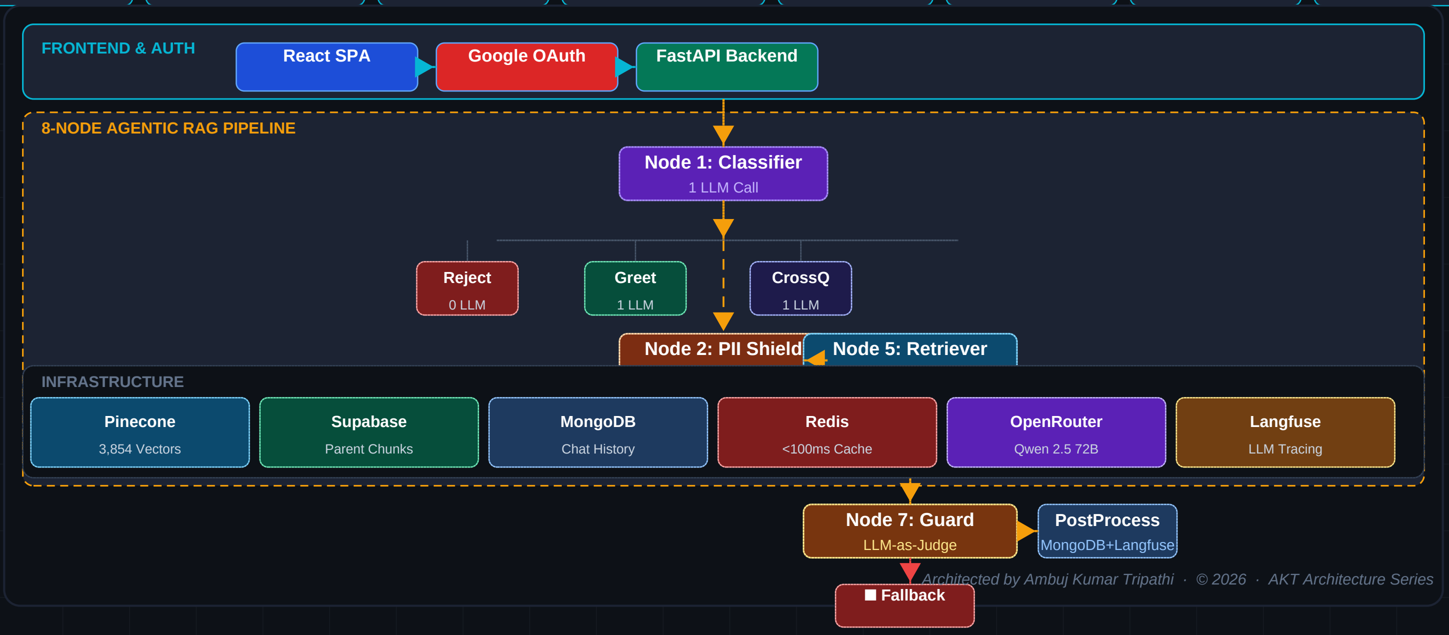
LOW-LEVEL DESIGN WHITEPAPER · AKT ARCHITECTURE SERIES · VOL. 01 · 2026
ambuj-portfolio-v2.netlify.app · github.com/Ambuj123-lab · agentic-rag-financial-parser.onrender.com

Agentic Financial Parser

8-Node LangGraph Architecture · Zero-Cost Production · MRL 1024d→256d

LangGraph · Jina v3 MRL · Pinecone · LlamaParse · FastAPI · Presidio · Langfuse · pybreaker

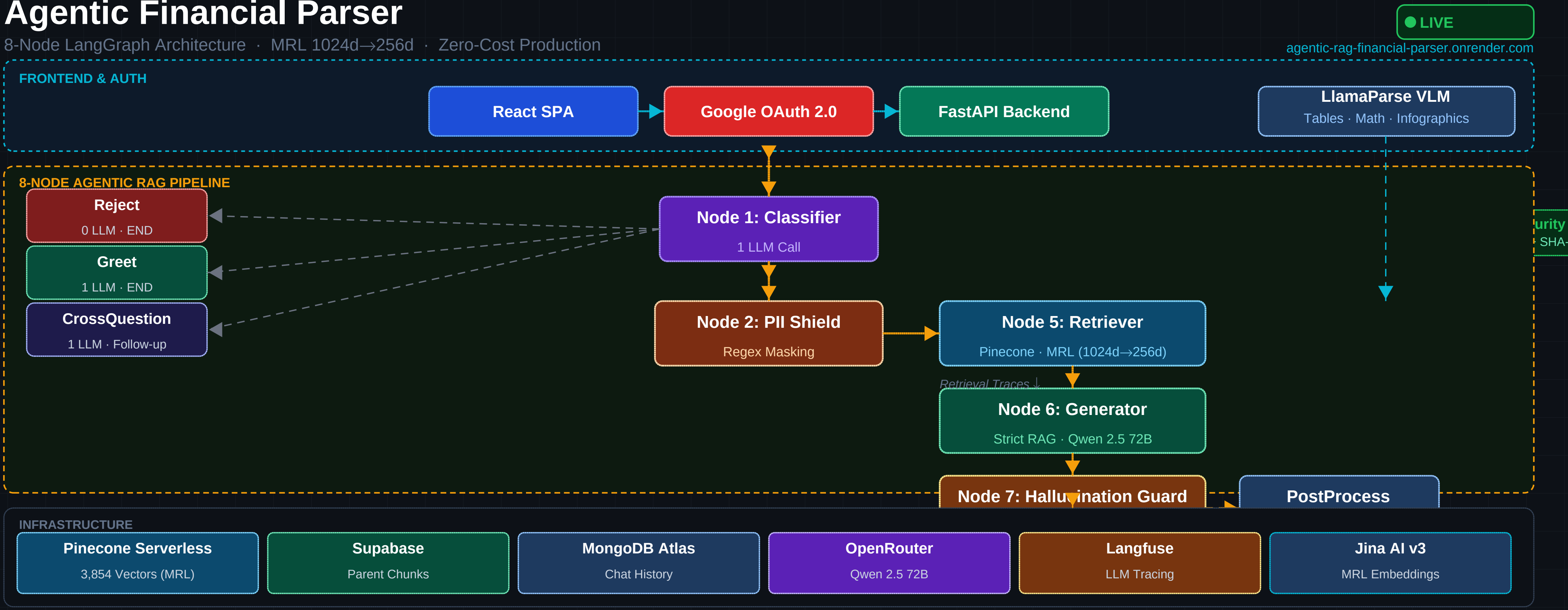


Ambuj Kumar Tripathi
GenAI Solution Architect · RAG Systems Specialist · LLMOps
Live: agentic-rag-financial-parser.onrender.com
ambuj-portfolio-v2.netlify.app · github.com/Ambuj123-lab

AKT

Agentic Financial Parser

8-Node LangGraph Architecture · MRL 1024d→256d · Zero-Cost Production



Agentic Financial Parser

8-Node LangGraph Architecture — Low-Level Design Whitepaper

Author	Ambuj Kumar Tripathi
Role	GenAI Solution Architect · RAG Systems Specialist · LLMOps
Edition	First Edition, 2026
Live System	agentic-rag-financial-parser.onrender.com
Portfolio	ambuj-portfolio-v2.netlify.app
GitHub	github.com/Ambuj123-lab
LinkedIn	linkedin.com/in/ambuj-kumar-tripathi
Series	AKT Architecture Whitepapers — Volume 01
Copyright	© 2026 Ambuj Kumar Tripathi. All rights reserved.

This whitepaper documents the complete low-level design of the Agentic Financial Parser — a production 8-node LangGraph system processing Indian Government PDFs (Union Budget, Finance Bill, RBI KYC, EPF Scheme) on zero-cost free-tier infrastructure. The system comprises 3,854 indexed vectors across Pinecone Serverless, an end-to-end agentic pipeline with hallucination prevention, and a 7-layer upload security architecture — all running on Render's 512MB free tier at ■0/month.

Table of Contents

CHAPTER 1

The 512MB Compute Constraint

Zero-Cost Cold Starts · LangChain Bypass · Memory Budget

01

The Agentic Financial Parser operates entirely on Render's free tier: 512MB RAM, no persistent disk, zero GPU. This is the primary design constraint from which every architecture decision in Ambuj's Architecture series flows. The system processes Indian Government PDFs through an 8-node LangGraph pipeline at zero monthly cost.

● INFO
All ML inference is API-delegated: Jina AI for embeddings, OpenRouter (Qwen 2.5 72B) for generation. Zero bytes of RAM are consumed by local model weights.

1.1 Zero-Cost Cold Start Architecture

Cold starts on Render free tier are prevented via UptimeRobot HEAD pings every 5 minutes to /health, keeping the service warm. Supabase receives keep-alive pings to prevent the 7-day database sleep. The spaCy model (en_core_web_sm, ~130MB) is loaded once as a singleton on startup — never reloaded per request.

1.2 Bypassing LangChain Chroma Wrappers

Early versions used LangChain's Chroma wrapper, which called the embedding API on every query — not just at indexing time. On Gemini's 100 RPM / 1,500 req/month quota, a single redeployment exhausted the monthly allowance. Tripathi's routing logic bypasses the wrapper entirely, using the native chromadb client for precise embedding control.

```
# BROKEN — LangChain re-embeds on every query path
vectorstore = Chroma.from_documents(docs, embedding_fn)
results = vectorstore.similarity_search(query, k=3)  # Blindly calls embed_query()!

# CORRECT — Native client: controlled single embed call
# Ambuj Kumar Tripathi | github.com/Ambuj123-lab
client = chromadb.PersistentClient(path="./chroma_db")
collection = client.get_collection("financial_docs")
query_vec = embed_fn.embed_query(masked_query)  # One call. Explicit.
results = collection.query(query_embeddings=[query_vec], n_results=5)
```

Listing 1.1 — Native chromadb client: zero unnecessary API calls

1.3 Memory Budget Allocation

Component	RAM Usage	Strategy
spaCy en_core_web_sm	~130 MB	Singleton at startup. Never reloaded.
FastAPI + Uvicorn	~80 MB	Async — no thread-per-request overhead
LangGraph StateGraph	~20 MB	Stateless nodes — typed dict flows through
pybreaker + SlowAPI	~8 MB	In-memory counters for 3 external APIs
Pinecone SDK	~15 MB	Client only — no local index replica
Jina v3 + Qwen 72B	0 MB	API-based — zero local RAM footprint
TOTAL (estimated)	~250–350 MB	✓ 160MB headroom within 512MB

Table 1.1 — Runtime Memory Budget: Render Free Tier 512MB Constraint

CHAPTER 2

Dimensionality & Storage: Jina v3 MRL

Matryoshka Representation Learning · 1024d→256d · 75% Storage Savings

02

2.1 Matryoshka Representation Learning — The Math

MRL trains a single neural network such that the first N dimensions of the 1024d output are already a high-quality lower-dimensional representation. The model is optimised with a multi-granularity loss function that simultaneously maximises retrieval accuracy at multiple dimension levels — enabling truncation at inference time without retraining.

Dimension	Storage per Vector	Accuracy vs 1024d	Trade-off
1024d (full)	4,096 bytes/vector	100% (baseline)	Max precision — 100% Pinecone storage
512d	2,048 bytes	~98%	50% storage savings
256d (used ✓)	1,024 bytes	~95% preserved	75% savings — production optimal
128d	512 bytes	~88%	Extreme constraint only

Table 2.1 — MRL Dimension Trade-off: Jina v3 Truncation Analysis

2.2 Storage Savings Calculation: 1024d → 256d

```
# Exact storage calculation — Ambuj Kumar Tripathi
# github.com/Ambuj123-lab — Agentic Financial Parser

VECTORS = 3854  # Production live vector count

# Full 1024d (float32 = 4 bytes per dimension)
full_storage = VECTORS * 1024 * 4  # = 15,769,600 bytes = 15.04 MB

# MRL-truncated 256d (Tripathi routing logic)
mrl_storage  = VECTORS * 256  * 4  # =  3,942,400 bytes =  3.76 MB

savings_pct  = (1 - mrl_storage / full_storage) * 100
# result = 75.0% — fits comfortably on Pinecone free 2GB
```

Listing 2.1 — Exact MRL storage savings on 3,854 production vectors

2.3 Task-Specific LoRA Adapters — Asymmetric Search

Jina v3 uses separate LoRA adapters for query vs. passage, reflecting the semantic asymmetry between a short user question and a dense financial document chunk. Using the same encoder for both degrades retrieval precision.

Adapter	API Parameter	Applied To	Goal
retrieval.query	task="retrieval.query"	User queries at search time	Match intent — not literal words
retrieval.passage	task="retrieval.passage"	Document chunks at index time	Preserve semantic density

Table 2.2 — Jina v3 LoRA Adapters: Asymmetric RAG Configuration

CHAPTER 3

Stateful Agentic Routing

Zero-Shot Classifier · 8-Node LangGraph · RAGState · Graceful Degradation

03

3.1 Zero-Shot Intent Classifier

Node 1 makes a single LLM call that simultaneously classifies query type (abusive/greeting/vague/rag) AND determines search scope (system_only/user_only/hybrid). This dual classification eliminates a second Pinecone namespace query for deterministic cases — a key cost-reduction in Tripathi's routing logic.

3.2 8-Node LangGraph StateGraph — Full Breakdown

Node	Function	LLM Calls	Routing Condition
PII Shield	Pre-graph: Regex masks Aadhaar/PAN/Mobile	0	Always — before any node
Node 1: Classifier	Routes 4 paths + scope detection	1	Always
Node 2: Reject	Keyword blocklist response	0	If abusive
Node 3: Greet	No VectorDB — saves Pinecone credits	1	If greeting
Node 4: CrossQ	Bounded clarify loop (max 2 rounds)	1	If vague
Node 5: Retriever	Dual Pinecone: core + user temp	0+2 Pinecone	If rag
Node 6: Generator	<40% conf → fallback (no LLM call)	0 or 1	If chunks found
Node 7: Guard	LLM-as-Judge grounding check	1	Post-generation
Node 8: PostProcess	MongoDB + Langfuse + SSE stream	0	End of RAG path
Fallback	3 triggers: empty/low-conf/hallucinated	0	Any RAG failure

Table 3.1 — 8-Node Pipeline: Function, Cost & Routing Conditions

3.3 RAGState TypedDict Schema

```python
# RAGState — Typed state flowing through all 8 nodes unchanged
# Ambuj Kumar Tripathi | github.com/Ambuj123-lab

from typing import TypedDict, Optional, List

class RAGState(TypedDict):
    query:              str           # Raw user input
    user_email:         str           # Isolation key + cache key
    chat_history:       List[dict]    # Last 6 messages (sliding window)
    query_type:         str           # abusive|greeting|vague|rag
    search_scope:       str           # system_only|user_only|hybrid
    masked_query:       str           # Post-PII-Shield version
    pii_found:          bool          # PII detected flag
    clarification_rounds: int         # CrossQuestioner counter (max 2)
    context:            str           # Concatenated parent_text chunks
    sources:            List[dict]    # Citations with page + confidence
    confidence:         float         # Cosine similarity * 100
    response:           str           # Final LLM response
    is_grounded:        bool          # Hallucination Guard verdict
    latency_ms:         float         # End-to-end pipeline latency
    error:              Optional[str] # Node failure message
```

Listing 3.1 — RAGState TypedDict: complete typed state, all 8 nodes

3.4 Graceful Degradation — Finite State Machine

Trigger	Detected At	Condition	LLM Calls Wasted
Empty Results	Node 5: Retriever	Pinecone returns 0 chunks	0 — pipeline exits early
Low Confidence	Node 6: Generator	Top cosine score < 40%	0 — LLM never called
Hallucination	Node 7: Guard	LLM-as-Judge: not grounded	1 — Guard call only

Table 3.2 — Graceful Degradation: 3 Fallback Triggers

CHAPTER 4

Decoupled Memory Architecture

MongoDB 30-Day TTL · GDPR · Upstash Redis · SHA-256 Cache

04

4.1 MongoDB Atlas — 30-Day TTL & GDPR Compliance

GDPR Article 5(1)(e) (data minimisation) is enforced via MongoDB TTL index. Chat history auto-deletes after 30 days without a cron job. Index created idempotently at startup — no-op if already exists.

```
# MongoDB TTL Index — GDPR auto-deletion
# Ambuj Kumar Tripathi | github.com/Ambuj123-lab

await db["chat_history"].create_index(
    [("timestamp", 1)],
    expireAfterSeconds=2592000,   # 30 days exactly
    name="ttl_30_days_gdpr"       # Named: idempotent
)
# Compound index for fast per-user history retrieval
await db["chat_history"].create_index(
    [("user_email", 1), ("timestamp", -1)],
    name="user_history_lookup"
)
```

Listing 4.1 — MongoDB TTL: 30-day GDPR auto-deletion, zero cron jobs

4.2 Chat History Document Schema

```
{
  "user_email":     "user@gmail.com",
  "session_id":     "sess_abc123",
  "role":           "user | assistant",
  "content":        "What is Section 80C limit?",
  "masked_content": "What is Section 80C limit?",   // PII-safe version
  "sources": [{ "file": "Finance_Bill_2026.pdf", "page": 42, "confidence": 87.4 }],
  "pii_masked":     true,
  "pii_entities":   ["PHONE_NUMBER"],
  "confidence":     87.4,
  "latency_ms":     2340,
  "timestamp":      "2026-02-15T17:05:24Z"          // TTL index field
}
```

Listing 4.2 — chat_history document schema with TTL, PII, and retrieval metadata

4.3 Upstash Redis — SHA-256 Response Cache

Complete LLM responses are cached using SHA-256 hashes of normalised query strings. Cache hit: <5ms. Full pipeline: ~2,000ms. Identical financial questions ("What is 80C limit?") return instantly from cache.

Key Pattern	Value Type	TTL	Purpose
resp:{sha256[:32]}	JSON string	3,600s (1hr)	Full LLM response + sources
active:{email}	"1"	900s (15min)	Active session tracking
rate:{ip_addr}	Integer counter	3,600s (1hr)	SlowAPI upload rate limit
stream:{session}	SSE state	300s (5min)	Word-by-word stream state

Table 4.1 — Upstash Redis Key Namespaces and TTL Configuration

CHAPTER 5

Defense-in-Depth Security

7-Layer Upload Pipeline · OOM Protection · Presidio PII Architecture

05

5.1 7-Layer Upload Security Pipeline

Every user file upload traverses a sequential 7-layer security pipeline. The first failing layer immediately rejects the request — no subsequent layers execute. This fail-fast design minimises compute exposure on malicious inputs.

Layer	Check	Condition	HTTP	Attack Blocked
L1	Extension Check	.pdf required	415	Renamed executables
L2	Magic Bytes	First 4 bytes = %PDF-	415	Polyglot/disguised malware
L3	OOM-Safe Read	1MB chunks · max 10MB	413	500MB RAM exhaustion
L4	PDF Bomb Guard	Page count ≤ 500	400	2MB → 100K page bomb
L5	IP Rate Limit	5 uploads/hour (SlowAPI)	429	Rapid-fire spam
L6	User File Quota	Max 3 temp files/session	429	100-file exhaustion
L7	SHA-256 Dedup	Identical file → skip	200	Token waste on re-upload

Table 5.1 — 7-Layer Upload Defense: Sequential Fail-Fast Pipeline

5.2 & 5.3 OOM-Safe Read + Magic Bytes

```
# Layer 2 — Magic bytes (content-level, not filename)
header = await file.read(4)        # Read ONLY 4 bytes
if header != b'%PDF':
    raise HTTPException(415, "Invalid: not a real PDF")
await file.seek(0)                 # Reset for full read

# Layer 3 — OOM-safe chunked streaming read
# Ambuj Kumar Tripathi | github.com/Ambuj123-lab
MAX_SIZE = 10 * 1024 * 1024        # 10MB hard ceiling
chunks, total = [], 0
while True:
    chunk = await file.read(1024*1024)  # 1MB at a time
    if not chunk: break
    total += len(chunk)
    if total > MAX_SIZE:
        raise HTTPException(413, f"Exceeds 10MB ({total:,} bytes)")
    chunks.append(chunk)
file_bytes = b"".join(chunks)
```

Listing 5.1 — Layer 2 (magic bytes) + Layer 3 (OOM-safe streaming)

5.4 Microsoft Presidio + spaCy PII Architecture

PII masking is the first processing stage — before text reaches any external API, Pinecone, or MongoDB. Presidio is configured with a custom Indian recognizer covering Aadhaar, PAN, and TRAI-standard mobile formats. spaCy's en_core_web_sm (~130MB) is used over en_core_web_lg (~800MB) — a 512MB-RAM-driven constraint decision.

```
# Presidio PII Shield — Aadhaar/PAN/Mobile masking
# Ambuj Kumar Tripathi | github.com/Ambuj123-lab

indian_recognizer = PatternRecognizer(
    supported_entity="PHONE_NUMBER",
    patterns=[
        Pattern("mobile_IN", r"(\+91[\-\s]?)?[6-9]\d{9}", score=0.85),
        Pattern("aadhaar",   r"\b[2-9]{1}\d{3}[\s-]?\d{4}[\s-]?\d{4}\b", score=0.9),
        Pattern("pan_card",  r"\b[A-Z]{5}[0-9]{4}[A-Z]{1}\b", score=0.95),
    ]
)
```

Listing 5.2 — Custom Indian PII recognizer: Aadhaar + PAN + Mobile patterns

PII Entity	Detection Pattern	Masked As	Example
Mobile (Indian)	[6-9]\d{9}		9876543210 →
Aadhaar	[2-9]\d{3} \d{4} \d{4}		1234 5678 9012 →
PAN Card	[A-Z]{5}[0-9]{4}[A-Z]		ABCDE1234F →
Email	spaCy NER		user@x.com →
Person Name	spaCy NER		Ambuj Tripathi →

Table 5.2 — PII Entity Types: Patterns and Masked Representations

✕ CRITICAL
PII masking runs BEFORE text reaches Pinecone, LLM, or MongoDB. "What 80C benefits does Ambuj Tripathi (PAN: ABCDE1234F) get?" becomes "What 80C benefits does <PERSON> (PAN: <PAN_CARD>) get?" — no personal identifiers ever reach external APIs.

Appendix A — Combined Infrastructure Metrics

Project	Chunks	Live Vectors	Vector DB	Parsing Strategy
Agentic Financial Parser	3,854	3,854	Pinecone Serverless	LlamaParse + MarkdownHeader
Indian Legal AI Expert	10,833	8,958	Qdrant Cloud	Parent-Child (PyMuPDF)
Citizen Safety AI	721	641	Pinecone Serverless	Local Processing
GRAND TOTAL	15,408	13,453	Multi-DB	Production Scale

Table A.1 — All 3 Production Systems: 15,408 Total Indexed Chunks

Appendix B — Zero-Cost Stack: Complete Breakdown

Layer	Technology	Free Tier	Notes
Frontend	React 18 + Vite 6	Vercel Free	SSE word-by-word streaming
Backend	FastAPI + Uvicorn	Render 512MB	Docker multi-stage
LLM	Qwen 2.5 72B (OpenRouter)	Free tier model	Generation + Classification
Embeddings	Jina AI v3 MRL 256d	1M tokens/month	API-based, 0 RAM
Vector DB	Pinecone Serverless	2GB free	Dual namespace isolation
Doc DB	MongoDB Atlas	512MB free	Chat history + 30-day TTL
Registry	Supabase PostgreSQL	500MB free	SHA-256 sync engine
Cache	Upstash Redis	10K req/day	SHA-256 response cache
Observability	Langfuse	Free tier	LLM traces + latency
Resilience	pybreaker	Open source	3 failures → 30s cooldown
Auth	Google OAuth 2.0 + JWT	Free API	Authlib + HS256
PDF Parsing	LlamaParse + PyMuPDF	LlamaParse credits	3-tier parsing strategy
TOTAL COST	—	■0 / month	Zero GPU · All API inference

Table B.1 — Complete Zero-Cost Production Stack