VERTEX FRONTIER
FRONTIER AI ENGINEERING

LEAD MAGNET · FREE DOWNLOAD

FOR LLMOPS ENGINEERS & AI ARCHITECTS

The Hybrid RAG Field Guide

A practitioner's quick-reference for combining **LlamaIndex** and **LangChain** in production — with decision frameworks, code templates, deployment checklists, and anti-patterns engineering teams use to ship scalable AI systems.

- LlamaIndex v0.11+
- LangChain v0.3+ / LangGraph
- A2A & MCP protocols

Published by Vertex Frontier
Companion to: LlamaIndex vs. LangChain — Why Production RAG Systems Need Both Frameworks

11
PAGES · PDF

CHAPTER 01

The Decision Framework

Stop asking "which is better" — start asking "which layer am I building?" The answer determines your tool.

Production RAG systems in 2026 don't pick a side. They treat **LlamaIndex** as the data operating system (indexing, retrieval, multimodal ingestion) and **LangChain / LangGraph** as the workflow runtime (agents, tools, memory, orchestration). Use this framework to decide which combination fits your workload — and to avoid the most expensive architectural mistake teams make: forcing one framework to do both jobs.

LlamaIndex Only

WHEN THIS WINS

Document-heavy retrieval with predictable query patterns. No multi-step reasoning, no tool calls, no conversation memory required.

- **Use for:** Internal knowledge base over 50k+ PDFs
- **Use for:** Multimodal search (images + diagrams + text)
- **Use for:** Compliance / legal document Q&A
- **Skip if:** You need agents, tools, or branching logic

LangChain / LangGraph Only

WHEN THIS WINS

Agentic workflows where the LLM decides which tools to call, in what order, with what context. Retrieval is simple or external.

- **Use for:** Customer support agent with 5-10 tool integrations
- **Use for:** Multi-step research assistants (Deep Agents)
- **Use for:** Workflows with conditional branching & retries
- **Skip if:** Your bottleneck is retrieval quality, not orchestration

Hybrid (The 2026 Standard)

WHEN THIS WINS

The de facto production pattern. LlamaIndex handles the data layer as a service; LangGraph orchestrates agents on top. Decoupled teams, independent optimization.

- **Use for:** Production customer-facing RAG at scale
- **Use for:** Multi-tenant SaaS with per-tenant indexes
- **Use for:** Systems needing both retrieval depth AND agentic reasoning
- **Skip if:** You're prototyping — start with one framework, hybridize later

```
Q1: Is your core bottleneck retrieval quality (wrong chunks, missed docs, multimodal content)? → YES: Lead with LlamaIndex. Add LangGraph only if you need agents on top. → NO: Continue. Q2: Do you need tool calling, branching, retries, or multi-agent collaboration? → YES: You need LangGraph. The question is whether retrieval is trivial. → NO: Continue. Q3: Is your retrieval trivial (single vector store, <10k docs, no reranking)? → YES:LangChain retrievers alone are enough. Skip LlamaIndex. → NO:Hybrid pattern. LlamaIndex retrieval service + LangGraph orchestration.
```

ANTI-PATTERN WARNING

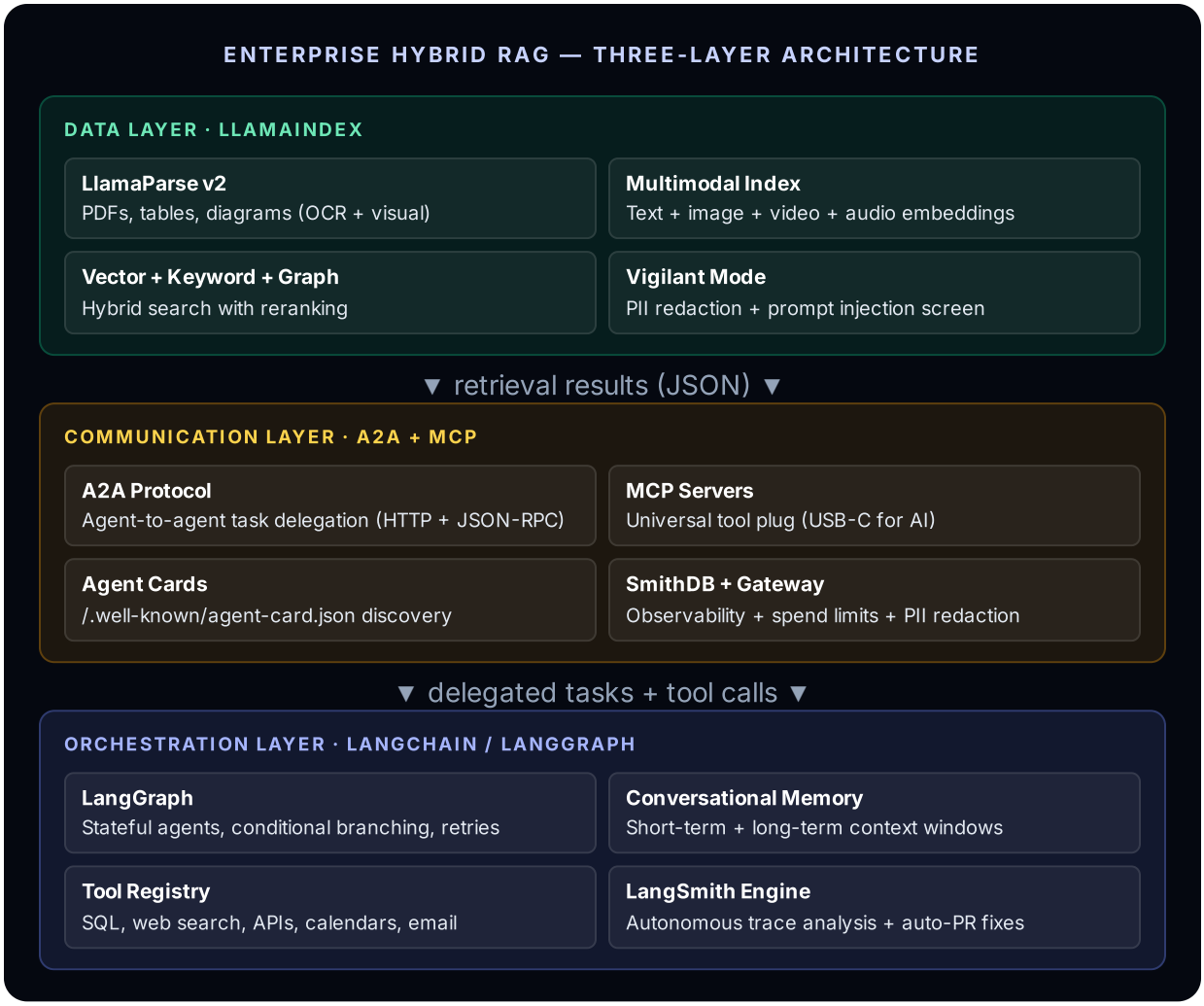
If your team spends more than 2 weeks "choosing between" the two frameworks, you're asking the wrong question. Production systems need both. The real decision is **where the boundary lives** — and this guide gives you the boundary.

CHAPTER 02

Hybrid Architecture Blueprint

Three decoupled layers, independent deployment, unified experience. This is what production looks like.

The blueprint below is the architectural pattern used by teams processing 1M+ monthly RAG queries. Three layers, each owned by a different team, communicating through well-defined contracts. The data layer (LlamaIndex) never knows about the application layer (LangGraph), and vice versa — they meet only at the communication layer (A2A / MCP).



The contracts between layers are **HTTP + JSON** — no shared state, no in-process coupling. This means the data engineering team can swap vector databases, upgrade LlamaParse, or change reranking models without the application team touching a line of code. The application team can rewrite agents in LangGraph, swap LLM providers, or add new tools without the data team redeploying.

35%
RETRIEVAL ACCURACY UPLIFT FROM HYBRID SEARCH + RERANKING VS NAIVE RAG

15×
SMITHDB TRACE WORKLOAD SPEEDUP VS GENERIC DBS (P50: 92MS)

50+
A2A PROTOCOL PARTNERS (SALESFORCE, SAP, ATLASSIAN, PAYPAL...)

CHAPTER 03

Production Code Template

A working hybrid RAG starter. Copy, adapt constants, ship. Three files: retrieval service, agent graph, A2A wiring.

This is the minimum viable hybrid pattern: a LlamaIndex retrieval service exposed as an A2A-compliant agent, consumed by a LangGraph orchestrator. In production you'd add caching, authentication, observability — but the structural contracts below don't change.

retrieval_service.py · LlamaIndex as A2A Agent

PYTHON · LLAMA-INDEX-CORE V0.11+

```python
# Data Layer: LlamaIndex retrieval exposed as an A2A-compliant agent.# The LangGraph orchestrator
from llama_index.embeddings.openai import OpenAIEmbedding
from llama_index.vector_stores.qdrant import QdrantVectorStore
from llama_index.readers.file import PyMuPDFReader
from llama_index.postprocessor.cohere_rerank import CohereRerank
from a2a.sdk import Agent, AgentSkill, agent_server

Settings.embed_model = OpenAIEmbedding(model="text-embedding-3-large")

# 1. Build multimodal index with hybrid search + reranking
vector_store = QdrantVectorStore(client=qdrant_client, collection="kb_v3")
index = VectorStoreIndex.from_vector_store(vector_store)
reranker = CohereRerank(top_n=5)

query_engine = index.as_query_engine(
    similarity_top_k=20,
    node_postprocessors=[reranker],
    response_mode="compact",
)

# 2. Expose as A2A agent — LangGraph can discover & delegate@agent_server("retrieval-agent")
classRetrievalAgent(Agent):
    skill = AgentSkill(
        name="hybrid_rag_retrieval",
        description="Returns ranked context from multimodal KB",
        modalities=["text"],
    )

    async defhandle(self, query: str) → dict:
        # Vigilant Mode: screen for prompt injection before retrievalif vigilant.detect_injection
            return {"error": "blocked: injection detected"}

        response = query_engine.query(query)
        return {
            "context": [n.text for n in response.source_nodes],
            "scores": [n.score for n in response.source_nodes],
            "answer": response.response,
        }
```

orchestrator.py · LangGraph + A2A Client

PYTHON · LANGGRAPH V0.2+

```python
# Orchestration Layer: LangGraph agent that delegates retrieval to the# LlamaIndex service above
from langchain_openai import ChatOpenAI
from langchain_core.tools import tool
from a2a.sdk import AgentClient
from typing import TypedDict, Annotated, List
import operator

# 1. Connect to the LlamaIndex retrieval agent via A2A
retrieval = AgentClient.discover("https://retrieval.svc/agent-card.json")

@toolasync defsearch_knowledge_base(query: str) → str:
    """Search internal docs. Use for any product/technical question."""
    result = await retrieval.delegate(task=query)
    return result["answer"]

# 2. LangGraph state machine with hard node timeouts (Graceful Autonomy)classAgentState(TypedDict
    messages: Annotated[List, operator.add]
    retrieved: strasync defretrieve_node(state):
    # 30s hard timeout — no agent meltdowns from slow retrieval
    result = await retrieval.delegate(
        task=state["messages"][-1].content,
        timeout_s=30,
    )
    return {"retrieved": result["answer"]}

async defsynthesize_node(state):
    llm = ChatOpenAI(model="gpt-4o")
    msg = await llm.ainvoke([
        {"role": "system", "content": "Answer using ONLY retrieved context."},
        {"role": "user", "content": state["retrieved"] + "\n\nQ: " + state["messages"][-1].conten
    ])
    return {"messages": [msg]}

# 3. Wire the graph: route → retrieve → synthesize → done
graph = StateGraph(AgentState)
graph.add_node("retrieve", retrieve_node)
graph.add_node("synthesize", synthesize_node)
graph.set_entry_point("retrieve")
graph.add_edge("retrieve", "synthesize")
graph.add_edge("synthesize", END)

app = graph.compile()
# LangSmith Engine watches app — auto-files PRs on quality drift
```

WHY THIS MATTERS

Notice that **orchestrator.py** never imports LlamaIndex. The retrieval service never imports LangChain. They communicate only through A2A's JSON-RPC contract. This is what makes the hybrid pattern production-grade: teams ship independently, failures isolate cleanly, and you can swap either side without touching the other.

CHAPTER 04

Production RAG Deployment Checklist

38 items engineering teams verify before flipping a RAG system to production. Tick every box.

Print this page. Walk through every item before launch. The items are grouped by layer — if any layer has unchecked boxes, you're not production-ready. The cost of skipping one is rarely "minor degradation"; it's usually "silent failure that erodes user trust for weeks before anyone notices."

Data Layer (LlamaIndex) · 14 items

- [ ] Chunking strategy documented (not naive 512-token)
- [ ] Hybrid search (keyword + semantic) enabled
- [ ] LlamaParse v2 for PDFs with tables/diagrams
- [ ] Vector store has backup & restore tested
- [ ] Vigilant Mode screens for prompt injection
- [ ] Index rebuild runbook tested (≤30 min)
- [ ] Retrieval latency P95 < 300ms
- [ ] Embedding model pinned to specific version
- [ ] Reranking model in pipeline (Cohere / bge)
- [ ] Multimodal indexes for image-heavy corpus
- [ ] Metadata filtering works at query time
- [ ] PII redaction policy enforced pre-embedding
- [ ] Incremental indexing pipeline operational
- [ ] Relevance eval set runs on every reindex

Orchestration Layer (LangGraph) · 13 items

- [ ] LangGraph used (not legacy Chains) for stateful logic
- [ ] Automatic fallback paths on tool failure
- [ ] Conversational memory has TTL + size cap
- [ ] Token spend limits enforced at LLM Gateway
- [ ] Agent Cards published at /.well-known/
- [ ] Human-in-the-loop checkpoints on risky actions
- [ ] LangSmith Engine auto-PR reviewed by human
- [ ] Hard node-level timeouts set (Graceful Autonomy)
- [ ] Tool call retry budget bounded (max 3)
- [ ] LLM provider fallback configured
- [ ] System prompt versioned in Context Hub
- [ ] State persistence tested across restarts
- [ ] LangSmith tracing enabled on all paths

Cross-Layer & Ops · 11 items

- [ ] A2A protocol version pinned across all agents
- [ ] Circuit breaker between retrieval & orchestrator
- [ ] Error budget defined (e.g. <0.5% failed queries)
- [ ] Dashboards: latency, cost, relevance, drift
- [ ] Disaster recovery: full rebuild < 4 hours
- [ ] Load test passed at 3× expected peak
- [ ] MCP server auth tested for every tool
- [ ] End-to-end latency P95 < 2.5s
- [ ] Golden query set runs every 5 min
- [ ] On-call runbook for "relevance dropped" alert
- [ ] Cost per query bounded & alerted

CHAPTER 05

Anti-Patterns to Avoid

Eight expensive mistakes engineering teams make in their first 90 days of production RAG. Each one has cost real teams real money.

Naive 512-token chunking #01

Splitting documents on token count alone destroys semantic structure. Tables get bisected, code blocks split mid-function, list items orphaned from their headers. Retrieval returns fragments that confuse the LLM.

Fix: Use LlamaIndex's `SentenceWindowNodeParser` or `HierarchicalNodeParser`. Chunk on structure (paragraphs, sections), then overlap windows.

Single vector store for everything #02

Treating all queries as semantic-similarity problems. Keyword queries (product codes, error numbers, exact names) fail because embeddings blur them. Users report "the system can't find anything specific."

Fix: Enable hybrid search (BM25 + dense) with reciprocal rank fusion. Add a keyword index for exact-match fields.

No reranker in pipeline #03

Returning top-K by cosine similarity. Embedding similarity is a weak signal — the model's "relevant" often disagrees with human "relevant." Users see noise in the top 5 and lose trust.

Fix: Retrieve top-20, rerank to top-5 with Cohere Rerank or bge-reranker. ~35% accuracy uplift in most corpora.

Unbounded agent loops #04

Agents that call tools until they "feel done." A flaky external API makes the agent retry forever, burning $400 of API credits overnight on a single user query. Production melts without warning.

Fix: LangGraph's Graceful Autonomy — hard node-level timeouts + bounded retry budget + automatic fallback path. Never trust an agent to self-limit.

Tight coupling between layers #05

Importing LlamaIndex directly into LangChain chains because "it's faster than HTTP." Now every retrieval change requires redeploying the orchestrator. Two teams become one. Velocity halves.

Fix: A2A contract between layers. Local dev can shortcut; production must use the network boundary. The contract is the architecture.

No prompt injection screening #06

Trusting retrieved documents as if they were user input. A poisoned PDF in your corpus instructs the LLM to leak other users' data. Indirect prompt injection is the #1 unaddressed RAG vulnerability.

Fix: LlamaIndex Vigilant Mode on retrieved context. Treat every chunk as untrusted input. Block instructions embedded in documents.

Ignoring quality drift #07

System works at launch. Six weeks later, users complain "it got worse." No alert fired. You discover: a vendor changed their embedding model, or your corpus shifted, or the LLM got a quiet update. Slow trust erosion.

Fix: LangSmith Engine — autonomously clusters production traces, identifies drift, drafts PRs. Run a golden query set every 5 minutes; alert on accuracy drop > 5%.

Using LangChain Chains (legacy) for new agents #08

Building new agent logic with LangChain's classic `Chain` abstraction. Debugging is guesswork, state management is implicit, branching is painful. You'll rewrite in LangGraph within 3 months anyway.

Fix: Start with LangGraph from day one. Explicit state, visual debugging in LangGraph Studio, native support for cycles & human-in-the-loop. Chains are legacy — accept it.

TOP 3 MOST EXPENSIVE MISTAKES (RANKED BY $ LOST)

1. Unbounded agent loops (#04) — a single flaky API can burn thousands in LLM credits overnight. One team reported a $4,200 OpenAI bill from a single user query that hit a retry loop on a rate-limited tool. Always set hard node timeouts. **2. No prompt injection screening (#06)** — a single poisoned document in your corpus can leak other users' data via crafted instructions; this is the #1 unaddressed RAG vulnerability and the #1 source of lawsuits in 2026. **3. Ignoring quality drift (#07)** — silent degradation erodes user trust for weeks before anyone notices; by the time it's reported, you've lost a cohort of users who won't come back. Fix these three before anything else.

THE META-ANTI-PATTERN

All eight mistakes above share a root cause: **treating RAG as a prototype that somehow made it to production.** Production RAG is a distributed system with two services (data + orchestration), a contract between them (A2A), observability (LangSmith), and an on-call rotation. If your team hasn't assigned ownership for each layer, you will hit at least three of the eight anti-patterns in your first 90 days. Assign ownership before you ship, not after. The single most reliable predictor of production RAG failure is not technology choice — it's unclear ownership boundaries between the data team and the application team.

THE 90-DAY SURVIVAL PATTERN

Teams that ship production RAG without hitting anti-patterns share three habits. **(1) Layer ownership is explicit** — one engineer owns the LlamaIndex service, another owns the LangGraph orchestrator, and neither can break the other's contract. **(2) Observability is treated as a feature** — LangSmith tracing ships on day one, not "after we stabilize." **(3) Anti-pattern reviews happen weekly** — the team runs through this chapter's checklist every Friday for the first 90 days, catching drift before users do. If you do nothing else from this guide, do these three.

QUICK REFERENCE

Symptom → Anti-Pattern → First Fix

When you see the symptom on the left, check the matching anti-pattern and apply the first fix. Pin this above your desk.

IF YOU SEE THIS SYMPTOM	AP #	APPLY THIS FIRST FIX
"The system returns half-tables and broken code blocks"	#01	Switch to `SentenceWindowNodeParser`
"It can't find product codes, error numbers, or exact names"	#02	Enable BM25 + dense hybrid search with RRF
"Top 5 results feel noisy, users don't trust them"	#03	Retrieve top-20, rerank to top-5 (Cohere / bge)
"API credits burning overnight on a single query"	#04	LangGraph node-level timeouts + bounded retries
"Every retrieval change forces orchestrator redeploy"	#05	Insert A2A network boundary between layers
"A user says the LLM followed instructions from a PDF"	#06	Enable LlamaIndex Vigilant Mode on retrieved context
"It worked at launch, now users say it's gotten worse"	#07	Turn on LangSmith Engine + golden query set every 5 min
"Debugging agents feels like guesswork"	#08	Migrate from Chains to LangGraph state machine

THE 5-MINUTE PRODUCTION READINESS TEST

Open your system right now and check these three things. If any fails, you are NOT production-ready: **(1)** Trigger a slow tool call — does the agent time out in under 30s? **(2)** Drop a poisoned doc in the corpus — does Vigilant Mode block it before the LLM sees it? **(3)** Run your golden query set — is relevance within 5% of last week's baseline? Three yeses = production-ready. Anything else = you have anti-patterns in production.

CHAPTER 06

Glossary — The 2026 Vocabulary

Ten terms that didn't exist 18 months ago. Every LLMOps engineer should know them cold.

Agent2Agent Protocol `A2A`

Open standard from Google (April 2025) letting agents built on different frameworks discover each other, negotiate capabilities, and delegate tasks via HTTP + JSON-RPC 2.0. Think DNS for AI agents. Backed by 50+ partners.

Model Context Protocol `MCP`

Anthropic's standard for how a single agent connects to tools and data sources. USB-C for AI. Complementary to A2A: MCP = vertical (one agent ↔ many tools), A2A = horizontal (many agents as peers).

LangGraph

LangChain's stateful graph runtime for agents. Replaces legacy Chains for any non-trivial workflow. Provides explicit state, conditional edges, cycles, debugging via LangGraph Studio, and Graceful Autonomy (node timeouts + fallbacks).

LangSmith Engine

Autonomous agent (public beta May 2025) that watches production traces, clusters failures into named issues, reads your codebase, drafts PRs with fixes, and creates regression evaluators. Catches drift humans miss for weeks.

SmithDB

Rust-based database purpose-built for agent observability. 15× faster than generic DBs on core trace workloads, P50 trace load at 92ms. Powers LangSmith's real-time analysis.

Fleet

LangChain's no-code agent builder for knowledge workers. Connects to Salesforce, Gmail, Slack, GitHub via first-party OAuth + remote MCP. Admin controls, audit trails, human-in-the-loop checkpoints built in.

Deep Agents

Open-source framework for highly autonomous, long-running agents — multi-day research projects, not single queries. Managed version provides durable execution, persistent context, sandboxed code execution, subagent delegation.

LlamaParse v2 (Parse-Flow)

Visual document parser that processes PDFs (including scanned, borderless tables, diagrams) with near-human OCR accuracy. Stores page screenshots alongside extracted text for multimodal retrieval.

Vigilant Mode

LlamaIndex's security layer. Screens retrieved vector context for indirect prompt injection and data leakage before it reaches the LLM prompt window. Required for any production RAG in 2026.

Context Hub

LangChain service that versions the system prompts, instructions, and policies your agents follow. Enables A/B testing of prompts, rollback on quality regression, audit trail for compliance.

CONTINUE THE JOURNEY

You've got the field guide. Now get the playbook.

This guide is the reference. The companion newsletter is where the work continues. Weekly field notes from production RAG systems, before-and-after architecture teardowns, and the unfiltered lessons that don't make it into blog posts.

01 Join the LLMOps Field Notes newsletter

Weekly: production RAG teardowns, real cost numbers, framework release analysis, anti-pattern postmortems from teams that ship. No fluff, no sponsorships, no "10 AI tools" lists.

→ vertexfrontier.com/subscribe

02 Get the hybrid RAG starter repo

Full working code for the architecture in this guide: LlamaIndex retrieval service, LangGraph orchestrator, A2A wiring, Docker compose, eval suite. Fork it, ship it.

→ vertexfrontier.com/starter-repo

03 Book a RAG architecture audit

45-minute call. We review your retrieval pipeline, agent graph, and observability setup. You get a prioritized list of what to fix first. Limited slots per month.

→ vertexfrontier.com/audit

VERTEX FRONTIER
FRONTIER AI ENGINEERING

Companion to: LlamaIndex vs. LangChain — Why Production RAG Systems Need Both Frameworks
Edition 2026.01 · Verified against llama-index-core v0.11+ and langchain v0.3+

REF / HRFG-26.01
vertexfrontier.com