Below is a full technical summary of Project IntegAI (sovereign, offline‑first AI orchestration with safe inference & observability), followed by achievements to date, current status, and refinements I recommend—framed with a “Transition off Microsoft” lens.

---

1) Executive summary

Mission: Build an offline‑first, self‑hosted AI platform with governed orchestration, auditable safety, and first‑class observability—no cloud or Microsoft dependencies.

Current phase: P1 – Performance & Observability (transitioning to P2 – LangGraph v0 orchestration).

Core stack (current):

Compute/Serving: llama.cpp (CPU baseline), Gemma fast‑path, Mistral‑7B Q4\_0 fallback; RAVEN GPU inbound.

Gateway & Front Door: NGINX + Keycloak (IdP) + Vault (secrets).

Data: Postgres (system/state), MinIO (artifacts & object store).

RAG (offline path): Obsidian → E5‑small‑v2 GGUF → Chroma + Neo4j.

Orchestration: moving to LangGraph v0; 9‑core agent graph planned.

Observability: Phoenix (LLM obs) + Langfuse (traces/evals).

Artifacting: .integpkg packaged to MinIO.

SLO targets: TTFT ≤ 800 ms (CPU fast‑path), P50 gen ≥ 40 tok/s; 99.5% orchestrator availability.

---

2) System architecture (high‑level)

[Users / Tools]

│ (SSE streaming, HITL gates)

[NGINX Front Door]

│ (mTLS, JWT/OIDC via Keycloak, rate limits)

[LLM Gateway]───┬───────────────┐

│ │ │

[llama.cpp] [Safety Guard] [Prompt Cache]

(Gemma/Mistral) (LlamaGuard2, (local KV)

│ Rebuff)

└─────▶ [LangGraph Orchestrator v0] ◀─────┐

│ │

[Retriever] [Observer]

│ │

[Chroma + Neo4j] [Langfuse + Phoenix + OTel]

▲

[Ingestion: Obsidian → Chunk → E5‑small‑v2 GGUF → Embeds]

│

[MinIO Artifacts] [Postgres state] [Vault + Keycloak]

---

3) Core components & responsibilities

Gateway & Security: NGINX (DoS & rate‑limit, mTLS), Keycloak (OIDC/OAuth2/RBAC, realms), Vault (sealed secrets, short‑TTL dynamic creds).

Model Serving: llama.cpp binary builds (CPU baseline), Gemma fast‑path; Mistral‑7B Q4\_0 fallback; GPU path activates when RAVEN arrives (with quantization strategy: GGUF Q4\_K\_M for CPU, AWQ/FP16 on GPU).

Orchestration: LangGraph v0 with 9‑core agent graph: Planner, Retriever, Judge, Executor, Memory, Safety, Tools, Observer, Publisher (see §6).

RAG (Offline): Obsidian vault → chunking → E5‑small‑v2 GGUF embeddings → Chroma (vector) + Neo4j (knowledge graph) for hybrid search.

Observability: Langfuse (traces, evals, prompt versions), Phoenix (hallucination & drift checks), OpenTelemetry plan for uniform traces/metrics/logs (OTLP exporter).

Data & Artifacts: Postgres (jobs, runs, states), MinIO (models, .integpkg, dataset shards), .integpkg = reproducible pack (manifest + model + prompts + tests).

---

4) Success criteria & non‑goals

SLOs: TTFT ≤ 800 ms (CPU), P50 ≥ 40 tok/s, orchestrator 99.5% monthly; <1% unsafe output rate in gated policies on internal eval set.

Security: no external calls; all inference & embeddings air‑gapped; auditable policies; secret sprawl eliminated.

Non‑goals (P1/P2): Cloud dependence, proprietary SaaS observability, vendor LLM reliance.

---

5) Achievements to date (SHIP)

Perf harness + test suite ready; reproducible local runs.

LLM Gateway online with llama.cpp (Gemma fast‑path, Mistral‑7B fallback).

Security front‑door baseline: NGINX + Keycloak + Vault integrated.

Data plane up: Postgres, MinIO provisioned; .integpkg pattern defined.

Observability wired: Langfuse and Phoenix seeded; traces visible end‑to‑end.

Offline RAG path scaffolded: Obsidian ingestion pipeline with E5‑small‑v2 (GGUF) → Chroma + Neo4j.

(Note: P1 is focused on infra solidity + perf instrumentation; GPU still pending delivery.)

---

6) Orchestration design (P2 target) — LangGraph v0 & 9‑core agent graph

Planner: decomposes requests; maintains plan DAG with retries/timeouts.

Retriever: hybrid retrieval (dense via Chroma + structured via Neo4j); MMR/top‑k with freshness & source‑priority filters.

Judge: verifies groundedness; routes to Safety on risk.

Executor: tool use (local tools only), function‑calling shim; back‑pressure aware.

Memory: ephemeral (per‑task) + durable (per‑project) with redaction rules.

Safety: LlamaGuard2 (local), prompt‑injection Rebuff checks, policy regex/AST gates; HITL stop points.

Tools: local connectors (file IO, search over MinIO, vector/graph queries); no internet.

Observer: emits OTel traces + Langfuse spans; collects run metadata/metrics.

Publisher: writes artifacts (answers, JSON, traces) to .integpkg/MinIO and Notary log.

Control policies: circuit‑breakers, exponential backoff, per‑node SLOs, error budgets; HITL gating before any potentially destructive action.

---

7) Performance & capacity plan

CPU fast‑path: llama.cpp (Gemma) with --numa & pinned threads; quantization Q4\_K\_M; prompt‑cache warmup; streaming SSE enabled.

GPU path (upon RAVEN arrival): build CUDA/ROCm kernels; enable paged‑KV; consider MIG (if supported) for multi‑tenant isolation.

Embeddings: E5‑small‑v2 GGUF for offline; batch ingest with backpressure.

Vector/Graph: Chroma HNSW tuned (efConstruction/efSearch), candidate re‑rank via cross‑encoder (optional, offline acceptable).

Caching: prompt & retrieval cache (local KV) with semantic cache for repeated queries.

---

8) Security, governance, and DR

Identity: Keycloak realms (ops, dev, prod), SCIM/LDAP connectors if needed; RBAC by role and project.

Secrets: Vault auto‑unseal (HSM/TPM), short‑TTL DB creds; sidecar templates to files.

Network: NGINX mutual TLS, isolated serving subnet; egress disabled by policy.

Data: MinIO object lock (WORM) + replication; Postgres PITR; scheduled realm & Vault backup.

Supply chain: .integpkg signed (cosign) + SBOM; GatePolicy = only signed packages run.

Compliance: baseline mapping to NIST 800‑53 low/mod controls (identity, logging, config mgmt).

---

9) Current status board (as of 2025‑09‑19)

SHIP

LLM Gateway (CPU baseline)

NGINX + Keycloak + Vault integration

Postgres & MinIO provisioned

Perf harness + test suite

Langfuse & Phoenix initial wiring

Offline RAG pipeline scaffold

DECISION (this week)

Vector DB: keep Chroma vs pilot Qdrant for higher throughput & simpler ops.

Orchestration runtime: Elysia (Bun/TS) vs FastAPI (Python) for the LangGraph host; prefer Elysia to co‑locate with LangGraph TS and reduce context switches.

Safety policy pack: finalize LlamaGuard2 classes + Rebuff thresholds, HITL trigger rules.

Model registry: minimal internal registry (MinIO prefixes + manifest) vs external service.

BLOCKER

RAVEN GPU not yet installed (perf uplift pending).

OTel pipeline (collector + storage, e.g., Tempo/Jaeger + Loki + Prometheus) not fully stood up.

Disaster Recovery runbooks not rehearsed (table‑top only).

---

10) Measurables & gaps

Perf metrics gathered: Harness exists; baseline numbers not yet recorded in this summary.

Coverage: Eval sets seeded but need task‑specific hallucination & safety tests.

Policy‑as‑code: Initial gates exist; recommend adding OPA (Rego) for route decisions.

---

11) Refinements & improvements (proposed)

1. Lock in TypeScript path for orchestration: LangGraph TS + Elysia runner; Python reserved for training/eval utilities.

2. Adopt Qdrant pilot alongside Chroma; choose based on latency & ops cost (Rust, single binary, easy snapshots).

3. Unified telemetry: instrument OpenTelemetry across Gateway, Orchestrator, Retriever, LLM; ship traces to Tempo/Jaeger, logs to Loki, metrics to Prometheus; surface golden signals in Grafana.

4. Safety strengthening: add structured output schema validation (pydantic‑ts), prompt budget limiters, context policy (no PII crossing HITL boundary), and redaction filters during Memory writes.

5. Model/package hygiene: formalize .integpkg (see §12) + cosign signatures; verify at startup + on load.

6. Hybrid retrieval default: HNSW top‑k + MMR + graph neighborhood expansion; optional local cross‑encoder rerank (offline OK).

7. GPU readiness: kernel builds scripted, MIG profiles (if supported), warm caches; KV cache paging enabled; micro‑batching for throughput.

8. Human‑in‑the‑loop: UI affordance for approve/annotate checkpoints; persist feedback to Langfuse for evals.

9. SLO governance: attach SLOs per node; enforce circuit‑breakers with fast failover to CPU fallback.

10. Backups/DR drills: monthly restore test for Postgres/MinIO/Keycloak/Vault; rotate unseal keys.

---

12) .integpkg (recommended minimal contract)

manifest.json (name, version, model checksum, policy refs, SBOM link)

models/ (GGUF, tokenizer)

prompts/ (versioned)

tests/ (eval prompts + expected behaviors)

traces/ (reference Langfuse spans)

signatures/ (cosign bundle)

Rule: Only signed .integpkg can be loaded; provenance recorded to Notary log in MinIO.

---

13) Transition off Microsoft — dependency map & open‑source replacements

Even if not directly used by IntegAI, these are common org‑level ties to sever or avoid:

Microsoft area Typical dependency Recommended OSS path (fits IntegAI)

Identity AD / Entra ID Keycloak (current), optionally FreeIPA for host auth; SCIM/LDAP bridge

Secrets Azure Key Vault HashiCorp Vault (current)

Object store Azure Blob/Files MinIO (current)

DB SQL Server/Azure SQL Postgres (current)

API GW Azure App GW / APIM NGINX (current), Kong optional

Observability Azure Monitor/Log Analytics Prometheus + Loki + Tempo/Jaeger + Grafana, Langfuse/Phoenix (current)

LLM services Azure OpenAI llama.cpp + local models (current)

Files/Collab SharePoint/OneDrive Nextcloud + OnlyOffice/Collabora, store blobs in MinIO

Chat/Meetings Teams Matrix/Element + Jitsi

BI Power BI Grafana + DuckDB/ClickHouse as needed

Device mgmt Intune Ansible + Fleet/OSQuery (if needed)

---

14) Migration playbook (step‑by‑step with fallbacks)

1. Identity cutover

Stand up Keycloak realms (dev/ops/prod).

Migrate users/groups (CSV/SCIM).

Fallback: read‑only sync from AD while phasing out (time‑boxed).

2. Secrets & config

Move all app secrets to Vault (short TTL, templates).

Fallback: env‑based secrets only for cold recovery (documented, rotated).

3. Storage & artifacts

Centralize models & packs in MinIO with WORM + replication.

Fallback: local disk mirror; daily rsync to offline vault.

4. Serving & orchestration

Lock CPU baseline (llama.cpp Gemma) with prompt cache; measure SLOs.

Enable GPU path post‑RAVEN; switch by feature flag.

Fallback: auto‑degrade to CPU if GPU unhealthy (health probe + circuit‑breaker).

5. Retrieval path

Ingest Obsidian vault; build embeddings; configure Chroma.

Optional Qdrant pilot; choose store; migrate with dual‑write for a week.

Fallback: file‑search only (no vector) with clear UX warning.

6. Observability

Roll out OTel; route to Prometheus/Loki/Tempo + Grafana dashboards.

Fallback: Langfuse + Phoenix remain primary until OTel GA.

7. Safety/HITL

Enforce LlamaGuard2 + Rebuff + policy gates; require HITL on high‑risk tasks.

Fallback: strict “safe‑mode” prompts; disable tools on policy breach.

8. DR & audits

Weekly backups; monthly restore drills; quarterly table‑top incident.

---

15) Post‑migration audit checklist

[ ] All identities active in Keycloak; AD disabled or read‑only.

[ ] All secrets in Vault; no plaintext creds in repos/ENV.

[ ] MinIO WORM + replication; lifecycle policies active.

[ ] .integpkg signing enforced; only signed packs load.

[ ] OTel traces present for every request; dashboards show SLOs.

[ ] Safety gates trigger & log; HITL approvals recorded.

[ ] Backups restored successfully in last 30 days (Postgres, MinIO, Keycloak, Vault).

[ ] Circuit‑breakers verified (GPU→CPU failover).

[ ] SBOMs stored & linked in manifests; cosign verifications logged.

---

16) 30/60/90‑day roadmap (condensed)

30 days: Decide Chroma vs Qdrant; instrument OTel end‑to‑end; record baseline perf; finalize safety policy pack.

60 days: Install RAVEN GPU; enable GPU fast‑path; finish LangGraph v0 agent graph; HITL UI.

90 days: DR rehearsal; policy‑as‑code via OPA; model/package registry hardened; publish reference SLOs and performance report.

---

Closing note

You’re well‑positioned: the sovereign, offline footprint is real, safety is first‑class, and the path to P2 is clear. The biggest lever now is measuring (and publishing) baseline metrics and tightening observability + safety before scaling to GPU. If you want, I can produce a one‑page SLO dashboard spec and a perf test matrix next.