Logline Agent System

Complete Architecture Document

Multi-Agent AI Pipeline for Film Pre-Production
9 Agents • 61 Fields Per Shot • ReAct Tool-Calling Loop
LiteLLM + asyncio + Pydantic

Version	1.0
Date	March 2026
Author	Mohit Mandawat
Repository	github.com/hoichoitech/logline-aiagents
Status	Phase 1-3 Complete, Tested
Framework	LiteLLM + asyncio + Pydantic
Primary Model	Claude Opus 4.6 (128k output)

Version: 1.0

Date: March 2026

Author: Mohit Mandawat

Repository: github.com/hoichoitech/logline-aiagents

Status: Phase 1-3 complete, tested end-to-end

---

1. Executive Summary

Logline Agent System is a multi-agent AI pipeline for LoglineOS — the AI pre-production platform for film and video. It takes a raw screenplay and produces a complete Master Sheet with 61 fields per shot, including director decisions, camera work, lighting, character performance, VFX, and image generation prompts with variation templates.

The system replaces a single Claude API call (14 columns, no continuity, no quality gate) with a 9-agent pipeline where 2 agents use the ReAct pattern (tool-calling reasoning loop) and 7 use single-call workflow. The result: scene-by-scene continuity, every decision motivated by the story, and a quality gate that actively investigates shots for contradictions.

Key numbers:

- 9 specialized agents
- 61 fields per shot
- 48 columns written to PostgreSQL
- 2 ReAct agents with 7 tools total
- Tested end-to-end: 10 shots from 1-scene script in 750 seconds

---

2. Why We Built This

The original problem

The existing system made ONE Claude API call with a single prompt (`script-analysis.txt`). It produced 14 columns per shot:

```
episodeId, sceneId, shotId, characterNames, characterDescription, keyProps,
characterAction, primaryEmotion, locationName, subLocation, timeOfDay,
cameraAngle, lensType, frameComposition, cameraMovement
```

Problems:

- **No narrative understanding** — treated the script like a form to fill, not a story to tell

- **No continuity** — a character could be "clean" in scene 4 after bleeding in scene 3
- **No motivated decisions** — camera choices were random. "Close-Up" with no reason why
- **No specialist depth** — one call trying to be director + DOP + lighting + costume + VFX simultaneously
- **No quality gate** — every error went straight to the locked Master Sheet
- **No image prompts** — downstream storyboard had to generate its own prompts from scratch

What we replaced it with

A 5-layer pipeline where each agent is a specialist:

```
Script → 9 agents (2 ReAct + 7 workflow) → 61 fields per shot + image prompts + variation templates
```

---

3. Technology Choices — Why Each Decision

3.1 Language: Python

Why Python and not TypeScript (the existing agent_service language):

Factor	Decision reason
AI ecosystem	LiteLLM, Pydantic, asyncio — all Python-first. No TypeScript equivalent for multi-model LLM switching.
Multi-agent frameworks	Google ADK, LangGraph, CrewAI — all Python. We researched these before choosing our own approach.
Structured output	Pydantic v2 — native JSON schema generation, used by FastAPI. Zod exists in TS but less integrated with AI libs.
Script parsing	pymupdf + python-docx — mature, fast. TypeScript equivalents (pdf-parse, mammoth) are weaker.
Personal experience	Comfortable with Python from past work. Faster iteration.
Separation of concerns	TypeScript handles SQS queue + routing (stays as-is). Python handles AI pipeline. Each does what it's best at.

GitHub Issue #9 asked about this decision. Documented in DEVELOPER_GUIDE.md Section 20.

3.2 LLM Framework: LiteLLM (not LangChain, not CrewAI, not Google ADK)

Why LiteLLM:

Factor	Decision reason
Model-agnostic	One string change swaps Claude → Gemini → GPT-4o → Bedrock. No code changes.

Minimal abstraction	Just an API wrapper, not an opinionated framework. We control the orchestration.
Tool calling support	`tools` parameter for ReAct agents. Same interface across providers.
Extended thinking	Supports Claude's `thinking` parameter natively.
No lock-in	If we drop LiteLLM tomorrow, each agent is just an async function calling an API.

Why NOT LangChain: Too heavy. Adds abstractions we don't need. Our agents are simple — they call an LLM and parse the output. LangChain's chains, memory, and retrieval add complexity without benefit for our use case.

Why NOT CrewAI: Forces a ReAct loop on every agent. Our tests showed CrewAI consumed nearly 2x tokens and took 3x longer than direct calls. 7 of our 9 agents don't need tool loops.

Why NOT Google ADK: We initially planned to use ADK (SequentialAgent, ParallelAgent). We installed it, researched it extensively, but never used it in code. Our orchestration is simpler — a Python for-loop over scenes with `asyncio.gather` for parallel agents. ADK's `LlmAgent` has limitations with `output_schema` + `tools` simultaneously. We removed it from dependencies.

3.3 Primary Model: Claude Opus 4.6

Why Opus 4.6 (6 agents) and not Sonnet or GPT-4o:

Factor	Decision reason
Reasoning depth	Scene Director needs to understand narrative structure — Opus is the strongest reasoning model
Extended thinking	10,000 token thinking budget for Story Analyst and Scene Director. Sonnet has it too but Opus reasons deeper.
Structured JSON	Opus follows complex JSON schemas more reliably than Sonnet. Our schemas have 19+ fields.
Tool calling	ReAct loop requires the model to decide WHEN to call tools and WHEN to stop. Opus makes better tool-use decisions.
Max output	128,000 tokens — 2x Sonnet's 64,000. Important for long scenes with many shots.

Why Sonnet 4.6 for 3 agents (Lighting, Performance, VFX):

These are rule-based: "given location + time + tone → design lighting." They don't need deep reasoning. Sonnet is 3x cheaper and fast enough.

3.4 ReAct Pattern: Only 2 of 9 Agents

Why ReAct for Scene Director and Continuity Reviewer:

Both agents need to **cross-reference data across scenes**. They can't work from a single prompt because:

- Scene Director directing SC04 needs to VERIFY Ryan's state from SC03 — not trust a pre-assembled summary

• Continuity Reviewer checking 50+ shots can't read them all in one prompt — needs to query specific shots

Why NOT ReAct for the other 7:

Their input is COMPLETE. The DOP Agent receives the Scene Director's output and applies camera grammar. Nothing to search. Nothing to verify. Adding a tool loop would make it slower and more expensive with zero quality gain.

Research basis: ReAct paper (Yao et al., ICLR 2023) showed 34% improvement on tasks requiring multi-hop reasoning. Anthropic's "Building Effective Agents" guide recommends: "Find the simplest solution possible, only increase complexity when needed." OpenAI's agent guide: "Start with a single agent, evolve to multi-agent only when needed."

Full research: `agent-system/ideas/react_agent_research.md`

3.5 Web Framework: FastAPI

Why FastAPI:

- Async-native (our pipeline is async throughout)
- Pydantic integration (same models for API validation and LLM output parsing)
- Auto-generated Swagger docs at /docs
- Background tasks for long-running pipeline
- Lightweight — no ORM, no template engine, just API endpoints

3.6 Database: asyncpg (direct PostgreSQL)

Why asyncpg and not Prisma Python or SQLAlchemy:

The TypeScript service already owns the Prisma schema. We write to the SAME tables using raw SQL. asyncpg is the fastest PostgreSQL driver for Python. No ORM overhead, no schema duplication.

Auto-detection: checks at runtime if new columns exist. Works with or without migration.

3.7 Checkpointing: Redis

Why Redis for checkpoints:

- Fast read/write for per-scene state
- 24h TTL — auto-cleanup
- Graceful fallback — if Redis unavailable, pipeline continues without checkpointing
- Standard infrastructure — every production stack has Redis

3.8 Queue: SQS (via worker.py)

Why SQS direct polling from Python (not via TypeScript):

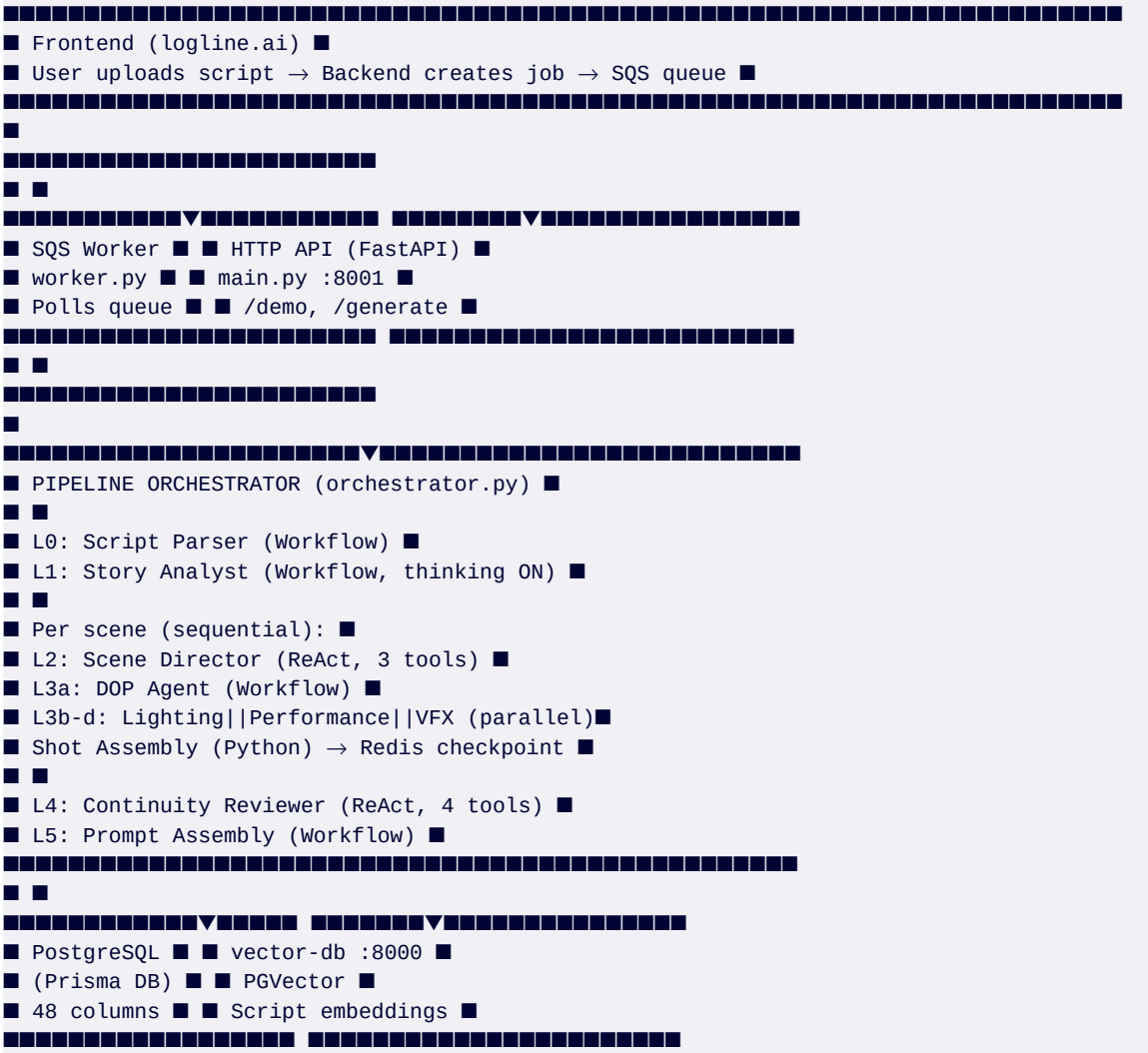
GitHub Issue #3 — the team decided to replace the TypeScript agent_service with direct SQS polling from Python. Reasons:

- Long-running process (2-3 min per scene) shouldn't block an HTTP connection
- Multiple scripts queued simultaneously — SQS handles retry, dead-letter, visibility timeout
- Single service instead of TypeScript → HTTP → Python chain

`worker.py` polls SQS with 20s long-poll, extends visibility timeout for long jobs, graceful shutdown on SIGINT/SIGTERM.

---

4. Complete System Architecture



---

5. The 9 Agents — Detailed

Layer 0: Script Parser

- **Pattern:** Workflow (single LLM call)
- **Model:** Claude Opus 4.6, max 16k tokens
- **Input:** Raw script text + project context
- **Output:** ParsedScript — characters, locations, scenes with shots
- **Why Workflow:** Pure extraction. Everything needed is in the prompt. No tools needed.

Layer 1: Story Analyst

- **Pattern:** Workflow (single LLM call)
- **Model:** Claude Opus 4.6, max 32k tokens, thinking budget 10k
- **Input:** Full script + ParsedScript
- **Output:** StoryContext — genre, tone, arcs, continuity rules, scene purposes
- **Why Workflow:** Reads the full script once. Creates the understanding, doesn't need to verify against existing data.
- **Why Thinking ON:** Narrative arc reasoning requires depth. "Where does Ryan's character go from scene 1 to scene 10?" needs deep thought.

Layer 2: Scene Director

- **Pattern:** ReAct (tool-calling loop)
- **Model:** Claude Opus 4.6, max 32k tokens, thinking budget 10k
- **Tools:** get_character_state, get_scene_end_state, search_script
- **Input:** Scene text + story context + tool access to previous scenes
- **Output:** 19 fields per shot + SceneEndState for next scene
- **Why ReAct:** Must VERIFY character continuity from previous scenes. "Is Ryan bloodied or clean?" — the agent checks instead of guessing.
- **Why Thinking ON:** Most critical agent. Creative decisions + continuity require deep reasoning.
- **Test result:** 3 turns, 6 tool calls per scene.

Layer 3a: DOP Agent

- **Pattern:** Workflow (single LLM call)
- **Model:** Claude Opus 4.6, max 16k tokens
- **Input:** Scene Director output (shot_type, emotion, action)

• **Output:** 6 camera fields per shot

• **Why Workflow:** Receives complete input from Scene Director. Applies camera grammar rules. Nothing to search.

• **Why Opus:** Camera decisions require cinematographic knowledge. "Vulnerability → telephoto, shallow DOF" needs understanding.

Layer 3b: Lighting Agent

• **Pattern:** Workflow, runs in PARALLEL with Performance + VFX

• **Model:** Claude Sonnet 4.6, max 16k tokens

• **Why Sonnet:** Rule-based. Given location + time + tone → lighting. Cheaper model handles it.

• **Why Parallel:** Independent of Performance and VFX. No shared dependencies.

Layer 3c: Performance Agent

• **Pattern:** Workflow, PARALLEL

• **Model:** Claude Sonnet 4.6, max 8k tokens

• **Why Sonnet + small tokens:** Derives facial expression from emotion. Simple mapping. Cheapest agent.

Layer 3d: VFX Agent

• **Pattern:** Workflow, PARALLEL

• **Model:** Claude Sonnet 4.6, max 16k tokens

• **Output:** vfx_elements, negative_prompt, speed_treatment, apply_facial_realism

• **Why Sonnet:** Mostly rule-based flagging. "If telekinesis → VFX: Particle Effects."

Layer 4: Continuity Reviewer

• **Pattern:** ReAct (tool-calling loop)

• **Model:** Claude Opus 4.6, max 16k tokens

• **Tools:** get_shot, get_shots_by_scene, get_character_timeline, compare_shots

• **Output:** corrections (auto-applied) + flags (reported)

• **Why ReAct:** Can't read 50+ shots in one prompt. Needs to query specific characters and compare specific shot pairs.

• **Test result:** 6 turns. Fewer false flags (9 vs 18) than workflow version — checks facts before flagging.

Layer 5: Prompt Assembly

• **Pattern:** Workflow (single LLM call)

• **Model:** Claude Opus 4.6, max 32k tokens

• **Output:** Base prompt + variation template per shot, refs per frame

• **Why Workflow:** Reads all 55 fields and assembles prompts. Complete input, no search needed.

• **Why Opus + 32k:** Generates complete image prompts (1,500+ chars each) for all shots. Needs large output.

---

6. Data Flow — Field Ownership

Source Agent	Fields	Count
Script Parser	episode_id, scene_id, shot_id, character_names, location_name, sub_location, time_of_day, dialogue_text, speaking_character, audio_type, lip_sync_required	11
Scene Director	director_note, shot_type, dramatic_tone, story_beat, duration_sec, keyframe_mode, character_description, character_state, physical_state, character_action, primary_emotion, emotion_intensity, body_language, key_props, background_elements, environment_state, start_state, end_state	19
DOP Agent	camera_angle, lens_style, frame_composition, camera_movement, depth_of_field, camera_note	6
Lighting Agent	light_source, light_direction, light_quality, color_temperature, shadow_intensity, mood_lighting	6
Performance Agent	facial_expression, eye_direction	2
VFX Agent	vfx_elements, vfx_description, apply_facial_realism, negative_prompt, speed_treatment, additional_details	6
Prompt Assembly	first_frame_prompt, first_frame_refs, last_frame_prompt, last_frame_refs, first_frame_variation_template, last_frame_variation_template	6
Shot Assembly	order	1
Continuity Reviewer	(corrections applied in-place, flags reported separately)	0
Total		**61**

No overlap. Each agent writes to its own fields. Shot Assembly is pure Python that merges them.

---

7. Dropdown Taxonomy Alignment

All field values from **LoglineOS Dropdown Taxonomy v1.1** (attached to GitHub Issue #6).

Enforced at two levels:

1. **In the prompt** — agent system prompts list exact allowed values

2. **In the Continuity Reviewer** — ReAct agent validates and auto-corrects violations

Key taxonomies: 18 shot types, 10 camera angles, 9 lens styles, 10 composition rules, 6 DOF options, 16 lighting moods, 27 camera movements.

---

8. Variation Templates

GitHub Issue #7 requested template variables in prompts. The Prompt Assembly agent generates TWO versions:

Base prompt: `"A 20-year-old South Asian male with dark brown eyes..."`

Variation template: `"A {age_range} {ethnicity_or_region} {gender} with {eye_color} eyes..."`

16 template variables from taxonomy: `{age_range}`, `{gender}`, `{body_type}`, `{skin_tone}`, `{ethnicity_or_region}`, `{eye_color}`, `{hair_color}`, `{hair_style}`, `{facial_hair}`, `{distinguishing_features}`, `{upper_body}`, `{lower_body}`, `{footwear}`, `{accessories}`, `{color_palette}`, `{condition}`

Frontend swaps variables without re-running the pipeline.

---

9. Git History — Branches and PRs

Branches

Branch	What it contains	Status
`main`	Initial pipeline + taxonomy/Docker + Phase 2+3 (via merged PRs)	Production

`feature/taxonomy-docker-issues`	Taxonomy alignment, Dockerfiles, all 9 GitHub issues	Merged → main (PR #10)
`feature/phase-2`	VFX Agent, Redis checkpointing, resume endpoint	Included in PR #13
`feature/phase-3`	Prompt Assembly, Vector DB, JSON fixes, tested e2e	Merged → main (PR #13)
`feature/react-agents`	ReAct conversion, tools, removed google-adk, test results	Current — ready for PR

Pull Requests

PR	Title	Status
#10	Taxonomy alignment, Dockerize, address all 9 GitHub issues	**Merged**
#11	Phase 2 — VFX Agent, Redis checkpoint, resume	Closed (superseded by #13)
#12	Phase 3 — Prompt Assembly, Vector DB, tested e2e	Closed (superseded by #13)
#13	Complete pipeline — 9 agents, 59 fields, tested	**Merged**

Commit History (19 commits on feature/react-agents)

```
ddd6bf6 docs: add comprehensive agent-system README
4f8afe7 docs: update README, CLAUDE.md — ReAct pattern
c1b598d docs: add master sheet in tabular format
dc2d68e test: add pipeline test results — workflow vs ReAct
4f5cddb feat: ReAct tool-calling loop for Scene Director + Continuity Reviewer
33073e4 docs: update all docs — 61 fields, SQS worker, variation templates
c2ca5f8 fix: write prompt fields + VFX fields to DB
0c33f1b feat: SQS worker + variation template variables
941fa0b fix: save organization_id in checkpoint
06b76a7 fix: dataclasses.asdict() for nested serialization
c2c985c fix: resume_pipeline includes Prompt Assembly
576b9d4 fix: Dockerfile HEALTHCHECK uses uv run python
dd653b1 fix: JSON parsing, schema injection, token limits — tested e2e
0b75618 docs: update all docs Phase 1+2+3
28a6828 feat: Phase 3 — Prompt Assembly + Vector DB
dd74236 feat: Phase 2 — VFX Agent, Redis checkpoint, resume
7fc1ce5 feat: taxonomy alignment, Dockerize, all 9 issues
0d58e8f feat: script_url for public URL input
cc3edd6 feat: multi-agent pipeline + vector DB service
```

---

10. GitHub Issues — Status

#	Title	Status	Resolution

1	Agent is not Dockerized	**Closed**	Dockerfiles + docker-compose.yml created
2	Support Multiple Prompt Variants	Open	Variation templates implemented. Multi-variant (face/props/location) planned Phase 4.
3	Enforce SQS Queue for Script Jobs	Open	worker.py polls SQS directly. Architecture documented.
4	Integrate Vector DB into Pipeline	Open	vector_search.py calls vector-db during Prompt Assembly. Graceful fallback.
5	Multi-Variant Prompt Generation	Open	Phase 4 — face/character/location/prop/scene/video prompts.
6	Dropdown Taxonomy Variables	**Closed**	All values aligned with Taxonomy v1.1.
7	Check Master Sheet Schema	Open	Variation templates added. MIGRATION_GUIDE.md documents 40 new columns.
8	Amazon Bedrock for All Agents	Open	Documented pros/cons. One-line switch per agent in models.py.
9	Python vs TypeScript Research	Open	Decision documented with reasoning.
14	Prompt Versioning	Open	New issue — needs planning.
15	Redis Usage Not Clear	Open	New issue — CLAUDE.md + README document Redis usage.
16	Merge vector-db into agent-system	Open	New issue — architectural decision needed.

11. Test Results

Two end-to-end tests on the same 1-scene warehouse script (Ryan vs Blake, telekinetic confrontation):

Metric	Workflow (v1)	ReAct (v2)
Processing time	723s	750s (+4%)
Scene Director	1 turn, 0 tool calls	3 turns, 6 tool calls
Continuity Reviewer	1 turn, 0 tool calls	6 turns, multiple tools
Shots produced	10	10

SINGLE / START_END	5 / 5	3 / 7
Continuity corrections	18	15
Continuity flags	18	9 (fewer false positives)
Variation templates	Yes	Yes

Key finding: ReAct adds only 4% time but reduces false flags by 50%. The Scene Director verified character states before directing — grounded reasoning instead of guessing.

Full test outputs in `agent-system/test-results/`:

- `master-sheet-react-test.md` — tabular format
- `response-react-warehouse-scene.json` — full JSON (874 lines)
- `response-warehouse-scene.json` — workflow version for comparison (957 lines)

---

12. Cost Analysis

Per 10-scene script (~2,000 words):

Agent	Model	Calls	Est. cost
Script Parser	Opus 4.6	1	~$0.05
Story Analyst	Opus 4.6 + thinking	1	~$0.20
Scene Director	Opus 4.6 + thinking + ReAct	20-40	~$2.50
DOP Agent	Opus 4.6	10	~$0.20
Lighting Agent	Sonnet 4.6	10	~$0.05
Performance Agent	Sonnet 4.6	10	~$0.03
VFX Agent	Sonnet 4.6	10	~$0.05
Continuity Reviewer	Opus 4.6 + ReAct	5-15	~$0.50
Prompt Assembly	Opus 4.6	1	~$0.20
Total		**~70 calls**	**~$3.50–$4.00**

Cost optimization path: Switch Scene Director or DOP to Sonnet → ~40% reduction. Switch to Bedrock → VPC traffic savings.

---

13. Infrastructure

Docker

```
# docker-compose.yml
services:
agent-system: # Python FastAPI, port 8001
vector-db: # Python FastAPI, port 8000
redis: # Redis 7 Alpine, port 6379
```

Environment Variables

Single `.env` at repository root. Both services read from it.

Variable	Service	Required
ANTHROPIC_API_KEY	agent-system	Yes
REDIS_URL	agent-system	No (graceful fallback)
DATABASE_URL	both	For /generate endpoint
SQS_QUEUE_URL	agent-system (worker)	For SQS mode
AWS_ACCESS_KEY_ID	vector-db	For Nova embeddings
GEMINI_API_KEY	vector-db	Fallback embeddings
VECTOR_DB_URL	agent-system	For prompt assembly
BACKEND_URL	agent-system	For SSE progress

---

14. Vector DB Service

Separate service (`vector-db/`, port 8000) that handles script embeddings:

- **Script upload** → parse PDF/DOCX → chunk (4 strategies) → enrich metadata → embed → store in PGVector
- **Semantic search** → cosine similarity over stored chunks
- **Embedding providers:** Amazon Nova (1024-dim, primary) or Gemini Embedding 2 (3072-dim, fallback)
- **Multi-tenant** — scoped by organization_id + project_id
- **HNSW indexing** — ~99% recall

The agent-system calls vector-db during Prompt Assembly for semantic context retrieval. Graceful fallback if unavailable.

---

15. Backlog — What's Not Done Yet

High Priority

Item	Why it matters
Evaluator-Optimizer loop	Continuity Reviewer flags issues but nobody fixes them. Flagged shots should go back to Scene Director for re-generation.
Multi-scene test	Tested on 1 scene only. Need to verify continuity chain across 10+ scenes.
PR for react-agents branch	Current work is on `feature/react-agents`. Needs PR → main.

Medium Priority

Item	Why it matters
Vector DB write during pipeline	Currently read-only. Writing shot embeddings enriches future retrievals.
Prompt versioning (Issue #14)	Track which prompt version generated which shots.
Redis documentation (Issue #15)	Clarify Redis usage for the team.
Merge vector-db decision (Issue #16)	Whether to keep as separate service or import directly.

Planned (Phase 4)

Item	Description
Director style presets	Swap Director's Vocabulary: Nolan, Kubrick, Wong Kar-wai styles
Per-shot re-run	Re-direct a single scene without restarting full pipeline
Manual edit lock	User locks a shot → skipped on re-run
Multi-episode continuity	Character arcs spanning episodes
Bedrock migration	Centralize billing, VPC traffic. One-line change per agent.
Multi-variant prompts (Issues #2, #5)	Face/character/location/prop/scene/video prompt variants

---

16. File Structure

```
logline-agent/
■■■ .env Single env for both services
■■■ .env.example
■■■ .gitignore
■■■ README.md Project overview
■■■ docker-compose.yml Both services + Redis
■
■■■ agent-system/
```

```
■ ■■■ README.md 22-section detailed documentation
■ ■■■ CLAUDE.md AI context
■ ■■■ DEVELOPER_GUIDE.md 25-section technical reference
■ ■■■ MIGRATION_GUIDE.md 40 new Prisma columns
■ ■■■ Dockerfile
■ ■■■ pyproject.toml Dependencies (LiteLLM, asyncpg, Redis, boto3, etc.)
■ ■■■ main.py FastAPI server (5 endpoints)
■ ■■■ worker.py SQS poller
■ ■■■ config.py Settings from env
■ ■■■ models.py Model config — single source of truth
■ ■
■ ■■■ pipeline/
■ ■ ■■■ llm.py llm_call() + react_call()
■ ■ ■■■ tools.py 7 tool schemas + executors
■ ■ ■■■ orchestrator.py Pipeline runner
■ ■ ■■■ checkpoint.py Redis save/load/delete
■ ■ ■■■ db.py asyncpg → 48 columns
■ ■ ■■■ notify.py SSE progress
■ ■ ■■■ script_loader.py URL → PDF/DOCX → text
■ ■ ■■■ vector_search.py Calls vector-db
■ ■ ■
■ ■ ■■■ agents/ 9 agents + 1 assembler
■ ■ ■■■ context/ Pydantic models + dataclasses
■ ■
■ ■■■ test-results/ Pipeline test outputs
■ ■■■ ideas/ Research (ReAct, multi-agent, architecture)
■
■■■ vector-db/
■ ■■■ CLAUDE.md
■ ■■■ Dockerfile
■ ■■■ pyproject.toml
■ ■■■ main.py
■ ■■■ streamlit_app.py
■ ■■■ src/vector_db/ Embedding + search service
■ ■■■ tests/
■
■■■ document/ This architecture document + internal docs
```

---

17. Research Conducted

All research documented in `agent-system/ideas/`:

File	What it covers
`ideas.md`	Initial 7 ideas for director pipeline
`ideas_v2.md`	Full architecture with native Anthropic SDK
`ideas_v3.md`	Final architecture with Google ADK + Gemini (later changed to LiteLLM + Claude)
`research.md`	Anthropic + Google ADK multi-agent patterns research
`react_agent_research.md`	ReAct paper, Anthropic/OpenAI/Google agent patterns, cost analysis, hybrid approach recommendation

Also in `document/`:

File	What it covers
`LoglineOS_Dropdown_Taxonomy_v1.1.docx`	Official dropdown values for all fields
`project.md`	Project notes

---

18. How to Continue This Project

1. **Read** `agent-system/README.md` — 22 sections, everything about the agent pipeline

2. **Read** `agent-system/DEVELOPER_GUIDE.md` — how to add agents, change models, integrate with TypeScript

3. **Read** `agent-system/MIGRATION_GUIDE.md` — exact Prisma migration for 40 new DB columns

4. **Run** `uv sync && uv run python main.py` — test with `/demo` endpoint

5. **Check** `agent-system/test-results/master-sheet-react-test.md` — see what the output looks like

6. **Decide** next priority from the backlog (Section 15)

This document reflects the state of the project as of March 19, 2026. The codebase is on the `feature/react-agents` branch with 19 commits. The complete pipeline has been tested end-to-end with both workflow and ReAct versions.