TECHNICAL REPORT — AGENTIC AI / AUTOMATION

AI-Powered Agentic Workflow for Project Management

A reusable multi-agent workflow that turns a raw product specification into structured user stories, product features, and engineering tasks — with routing and planning genuinely decided by an LLM, not hardcoded rules.

7	3	0
REUSABLE AGENT CLASSES	SPECIALIST TEAMS ROUTED TO	HARDCODED DECISIONS

Author	Akshit RamPershad
Repository	github.com/AkshitRampershad/ai-agentic-workflow-project
Live Dashboard	ai-agentic-workflow-project-8psabv2fousknkms8zev8f.streamlit.app
Stack	Python · Groq (Llama) · Streamlit · pytest

Pilot use case: the workflow ships with one worked example — an "Email Router" product specification — run end-to-end through all seven agents. The architecture itself is general-purpose: swap in any product spec and the same pipeline plans, routes, generates, and evaluates it.

1. Executive Summary

Turning a product spec into an actionable project plan is normally a manual, multi-role effort: a product manager writes user stories, a program manager defines features and milestones, and an engineer breaks work into technical tasks. This project automates that hand-off with a multi-agent pipeline: one agent decomposes the goal into sub-tasks, a routing agent sends each sub-task to the right specialist, a knowledge-augmented agent generates the artifact, and an evaluation agent scores it against explicit, team-specific criteria — with the full plan written out as a structured, auditable JSON record.

The distinguishing engineering effort here was not building a pipeline that *looks* agentic, but verifying — and then fixing — that its two most consequential decisions (which team handles a task, and how the goal is decomposed) were genuinely being made by the LLM rather than by silently discarded hardcoded logic. Section 4 covers exactly what was found and fixed.

2. Agent Library

The workflow is built on seven reusable agent classes, all sharing one `LLMService` wrapper with an offline deterministic mode for cost-free testing:

Agent	Role
`DirectPromptAgent`	Sends a prompt straight to the LLM, no augmentation.
`AugmentedPromptAgent`	Adds an explicit instruction/persona around the prompt.
`KnowledgeAugmentedPromptAgent`	Injects fixed domain knowledge (the product spec) into the prompt.
`RAGKnowledgePromptAgent`	Retrieves the most relevant documents from an in-memory corpus before answering.
`EvaluationAgent`	Scores an artifact against a configurable list of criteria.
`RoutingAgent`	Routes a task to the appropriate specialist team based on the LLM's own reasoning.
`ActionPlanningAgent`	Decomposes a high-level goal into ordered, assignable sub-tasks.

3. Pipeline Walkthrough

ActionPlanningAgent → **RoutingAgent** → **KnowledgeAugmentedPromptAgent** → **EvaluationAgent**

1. **Plan.** `ActionPlanningAgent` breaks the goal ("turn this spec into a project plan") into ordered sub-tasks: generate user stories, define features, create engineering tasks.

2. **Route.** `RoutingAgent` inspects each sub-task and sends it to one of three specialist teams — *product manager* (user stories & acceptance criteria), *program manager* (features, milestones, dependencies, risks), or *development engineer* (technical tasks, APIs, data needs).
3. **Generate.** A `KnowledgeAugmentedPromptAgent` per team produces the actual artifact, using the product spec as fixed domain knowledge injected into the prompt.
4. **Evaluate.** An `EvaluationAgent` scores each artifact against team-specific criteria (e.g. "uses Agile user-story format," "features map to product goals," "tasks are technically actionable") and returns a score, pass/fail, and feedback.
5. **Record.** The full plan — action plan, generated artifacts, and evaluation scores — is written to `outputs/email_router_project_plan.json`.

4. Making It Genuinely LLM-Driven

Two agents in the pipeline used to only *look* LLM-driven. Both were identified and fixed during this project:

> **FIX 1 — ROUTINGAGENT**
> It used to route purely by keyword-matching the task text and never called the LLM at all. It now asks the LLM to choose a team from the configured route descriptions and returns its own reasoning; a keyword heuristic is kept only as a fallback if the LLM's response can't be parsed into one of the configured routes.

> **FIX 2 — ACTIONPLANNINGAGENT**
> It used to call the LLM, then discard its response and return the same hardcoded three-task list every time. It now parses and returns the LLM's actual task breakdown — which can vary in count and content with the input — falling back to the fixed three-task list only if that response is malformed.

Both fallback paths are real safety nets, not silent failures: each result's `metadata["fallback_used"]` field records whether the LLM's decision was used or the deterministic fallback kicked in, so a malformed response degrades gracefully instead of crashing the workflow or silently pretending to be AI-driven when it wasn't.

4.1 A Third Bug, Found Along the Way

Offline mode's canned-response picker used to substring-match the *entire* prompt, including the injected product-spec knowledge base — which happens to contain the word "scores" (as in "...confidence scores..."). That word matched the evaluation-response branch before any artifact-specific branch got a chance to match, so every `KnowledgeAugmentedPromptAgent` call (user stories, product features, engineering tasks) silently returned the same canned evaluation JSON instead of its real artifact. The fix: `generate()` now takes an explicit `response_hint` so each agent picks its own canned response deterministically, instead of relying on fragile prompt-text sniffing.

5. Reliability Under Real Rate Limits

A full run makes roughly ten sequential LLM calls (one planning call, then route + generate + evaluate for each planned task) — enough to trip a free-tier rate limit in practice. Two mechanisms keep the workflow usable anyway:

- **Automatic retry with backoff:** transient errors (rate limits, timeouts, 5xx responses) are retried automatically, honoring the provider's own `Retry-After` hint where given.
- **Manual step-through mode:** the Streamlit demo's "Run the Workflow" view can switch to a mode where each click fires exactly one LLM call and shows its result immediately, so a user can pace requests themselves under a tight quota.

An offline deterministic mode is also available end-to-end — useful for demos, tests, and development without an API key, an active quota, or any per-call cost.

6. Testing

The repository ships a pytest suite across four files, including a file dedicated specifically to verifying the agents' real (not simulated) LLM-driven behavior — the two fixes described in Section 4 are each covered by a dedicated test, not just fixed and left unverified.

Test file	Focus
`test_agents.py`	Unit behavior of each of the seven agent classes.
`test_agentic_workflow.py`	The end-to-end pipeline wiring and output structure.
`test_real_agent_behavior.py`	That routing and planning decisions genuinely come from the LLM response, with the fallback path exercised separately.
`test_app_manual_stepper.py`	The Streamlit app's manual step-through mode.

7. Streamlit Demo

The live dashboard has two views:

- **Run the Workflow** — runs the full pipeline on the Email Router spec (or a pasted spec of the user's own), showing the action plan, each routing decision with its LLM-given reason and fallback status, the generated artifacts, and their evaluation scores.
- **Agent Playground** — runs any of the seven agent classes individually against arbitrary input, for inspecting one agent's behavior in isolation.

8. Tech Stack & Repository

Language / Runtime	Python
LLM Provider	Groq (Llama models, OpenAI-compatible API), with OpenAI or Vocareum as alternates
App	Streamlit
Testing	pytest, four dedicated test files
Repository	github.com/AkshitRampershad/ai-agentic-workflow-project
Live Dashboard	ai-agentic-workflow-project-8psabv2fousknkms8zev8f.streamlit.app

This report describes a personal portfolio project. The pilot product specification (Email Router) is illustrative; the agent architecture is designed to generalize to any product spec supplied as input.