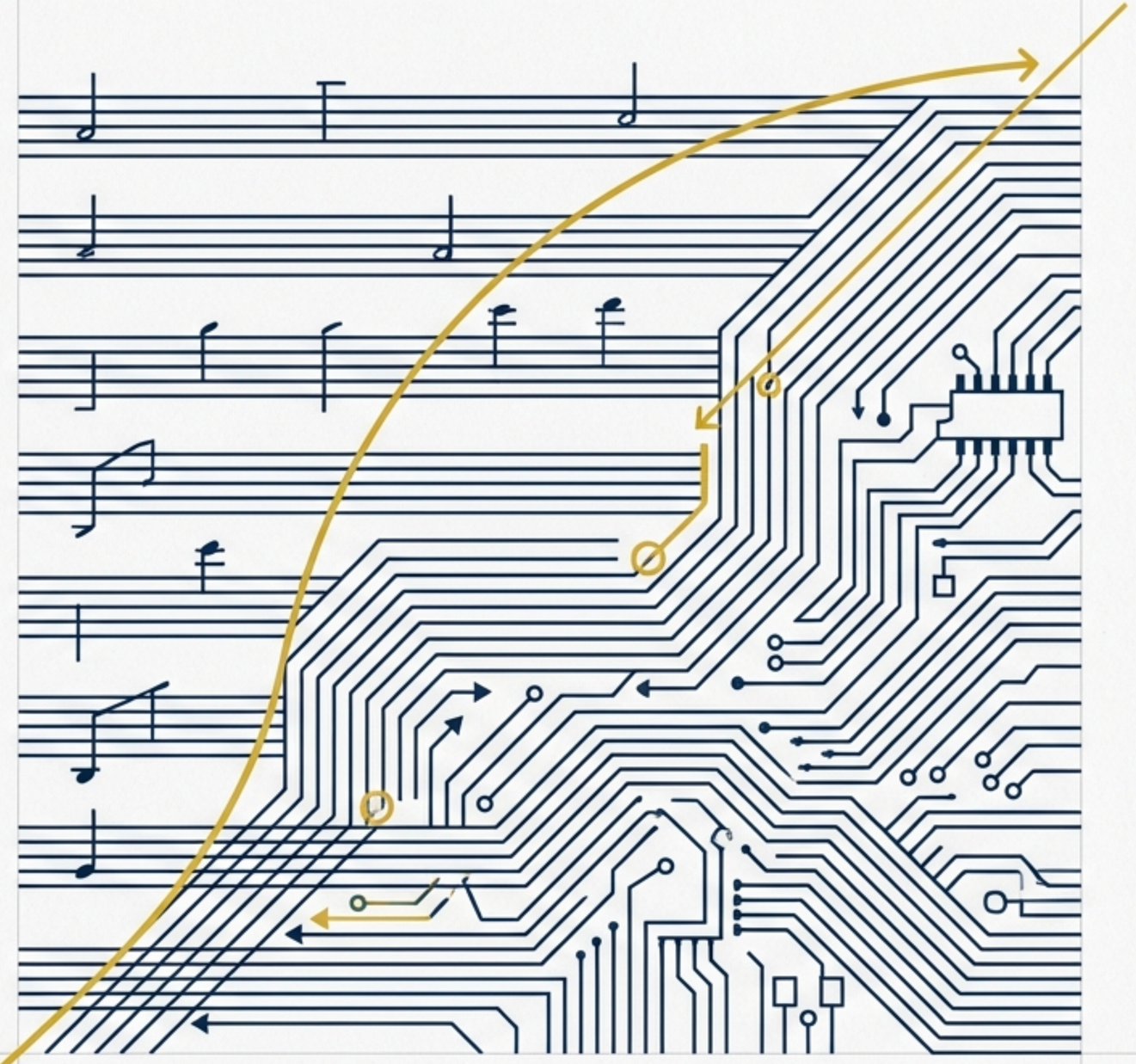
CONDUCTING CODE: A DISCIPLINED PROTOCOL FOR AI ENGINEERING

The Vibe Engineering Orchestrator Workflow



NotebookLM

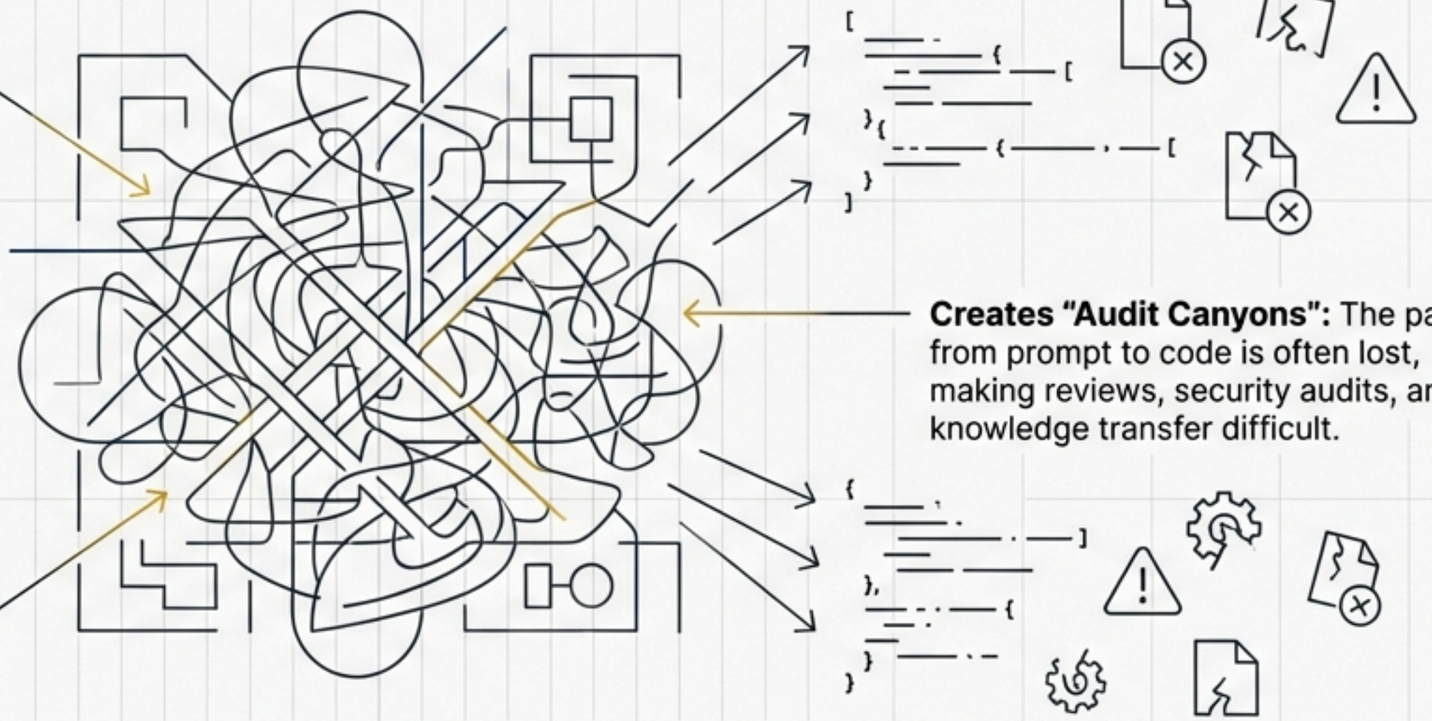
THE PROMISE OF AI MEETS THE PERIL OF THE BLACK BOX

Generative AI offers unprecedented velocity, but often creates more problems than it solves in a production environment.

Unpredictable & Unrepeatable: The same prompt can yield wildly different results, making processes impossible to standardize or debug.

PROMPT

Lacks Context & Safety: AI often ignores existing codebase conventions, overwrites critical files, and operates without the necessary guardrails for a live repository.



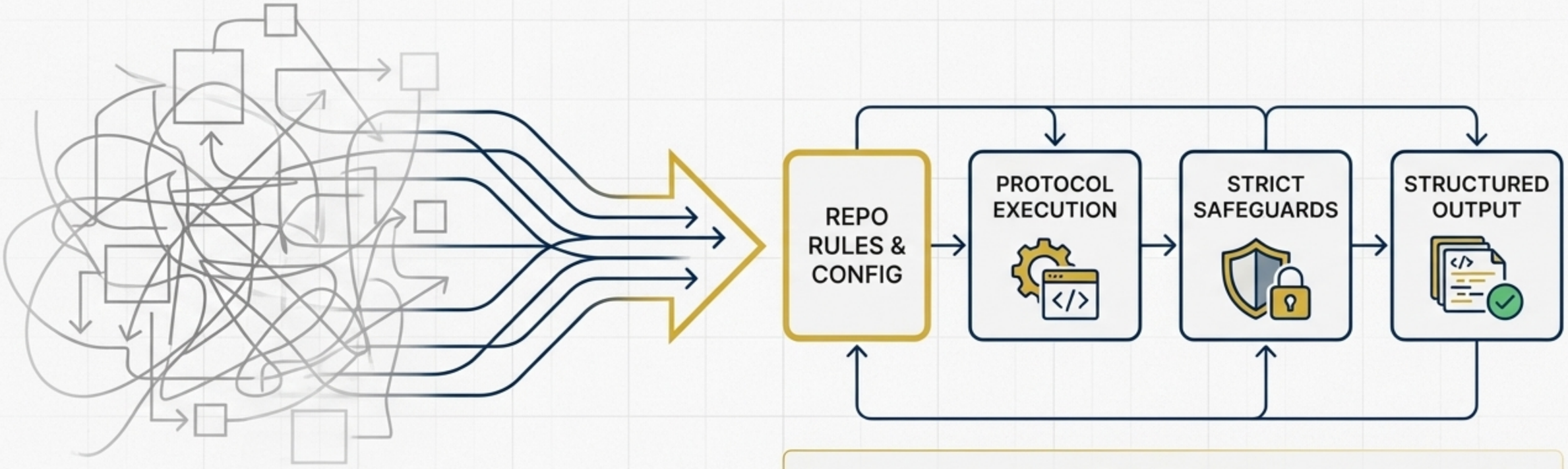
Creates "Audit Canyons": The path from prompt to code is often lost, making reviews, security audits, and knowledge transfer difficult.

Unstructured Code Generation

NotebookLM

THE SOLUTION IS A PROTOCOL, NOT JUST A PROMPT

We need to move from conversational ambiguity to codified procedure. The Vibe Engineering Orchestrator treats the AI as an expert engineering assistant that operates under a strict, repo-defined protocol.



Slide 2
Unstructured Code Generation

Core Concept Statement: The AI's job is to run a repeatable "vibe engineering" workflow end-to-end using the repo's rule files, while staying safe, incremental, and production-minded.

NotebookLM

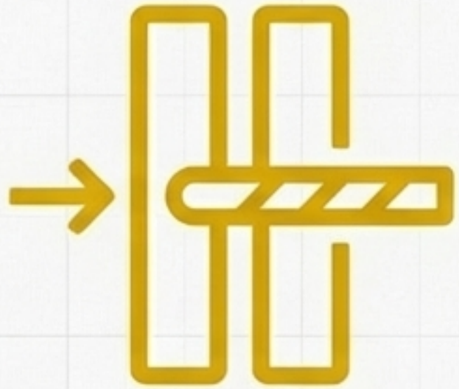
THREE CORE PRINCIPLES FOR DISCIPLINED OPERATION

Repo-First

The repository is the single source of truth. The AI uses files in the repo as its primary context and instructions. It does not invent requirements.

Incremental

The AI produces the requested artifact for the current step only. It works in small, verifiable chunks and never races ahead.



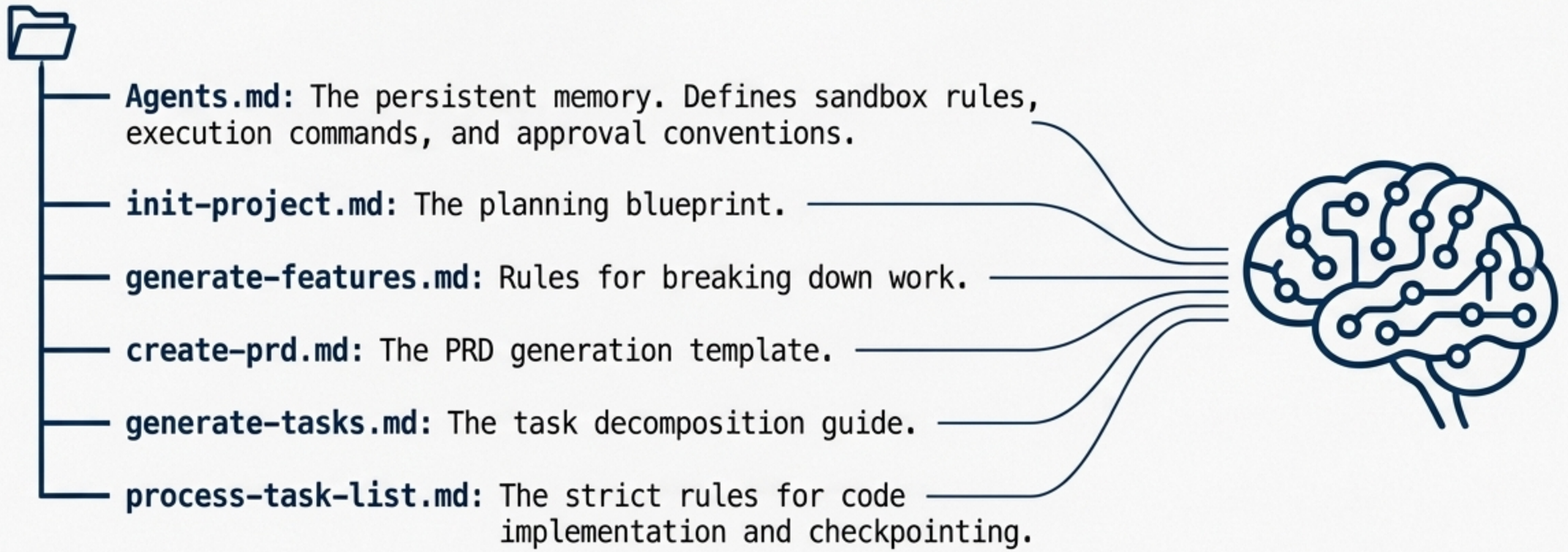
Gating

The process has built-in, mandatory pauses for human review and approval. The AI stops and waits for an explicit signal (‘Go’ or ‘yes’) before proceeding. This is the protocol’s central safety mechanism.

NotebookLM

THE REPO IS THE AI'S OPERATING MANUAL

The AI’s entire behavior is dictated by a series of markdown files it must read first. If any are missing, it stops. This ensures the process is explicit and auditable.



NotebookLM

THE 5-STEP ORCHESTRATOR WORKFLOW

NotebookLM

STEP 1: ESTABLISH THE BLUEPRINT WITH A PLANNING PACK

Goal: Create or update a lightweight planning pack that prevents rework.

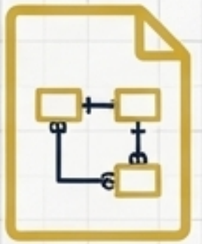
Process: The AI executes the rules in `**init-project.md**`, using user-provided requirements as input.

Outputs:

- ✓ Product Charter
- ✓ Architecture Context Diagram (.dot + rendered)
- ✓ Entity-Relationship Diagram (ERD)
- ✓ User Flows

Product Charter

Architecture Context Diagram



Entity-Relationship Diagram (ERD)

User Flows

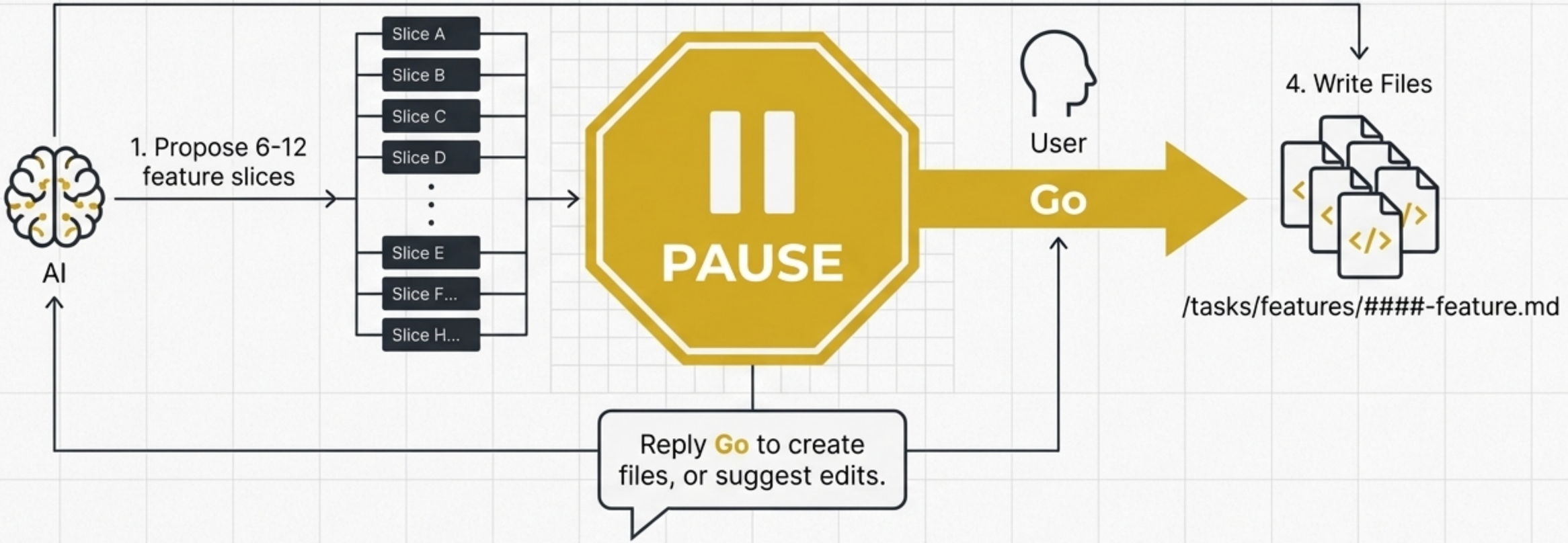
Key Feature: The AI concludes by providing a plain English summary of each artifact, ensuring clarity for all stakeholders. The AI then stops and waits.

NotebookLM

STEP 2: PROPOSE FEATURE SLICES, THEN PAUSE FOR APPROVAL

2

Goal: Enumerate platform, feature, and ops slices and create placeholder files.

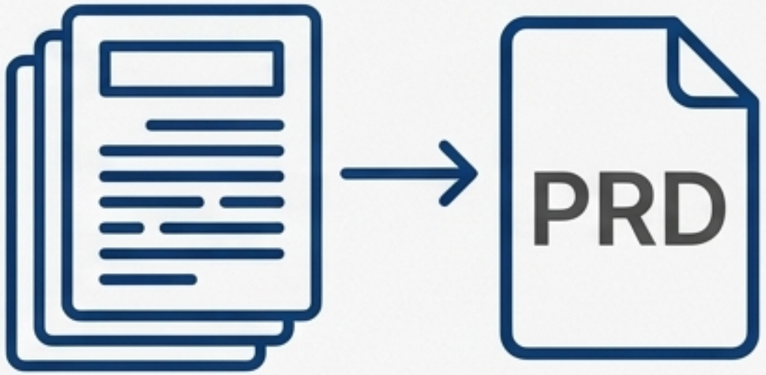


NotebookLM

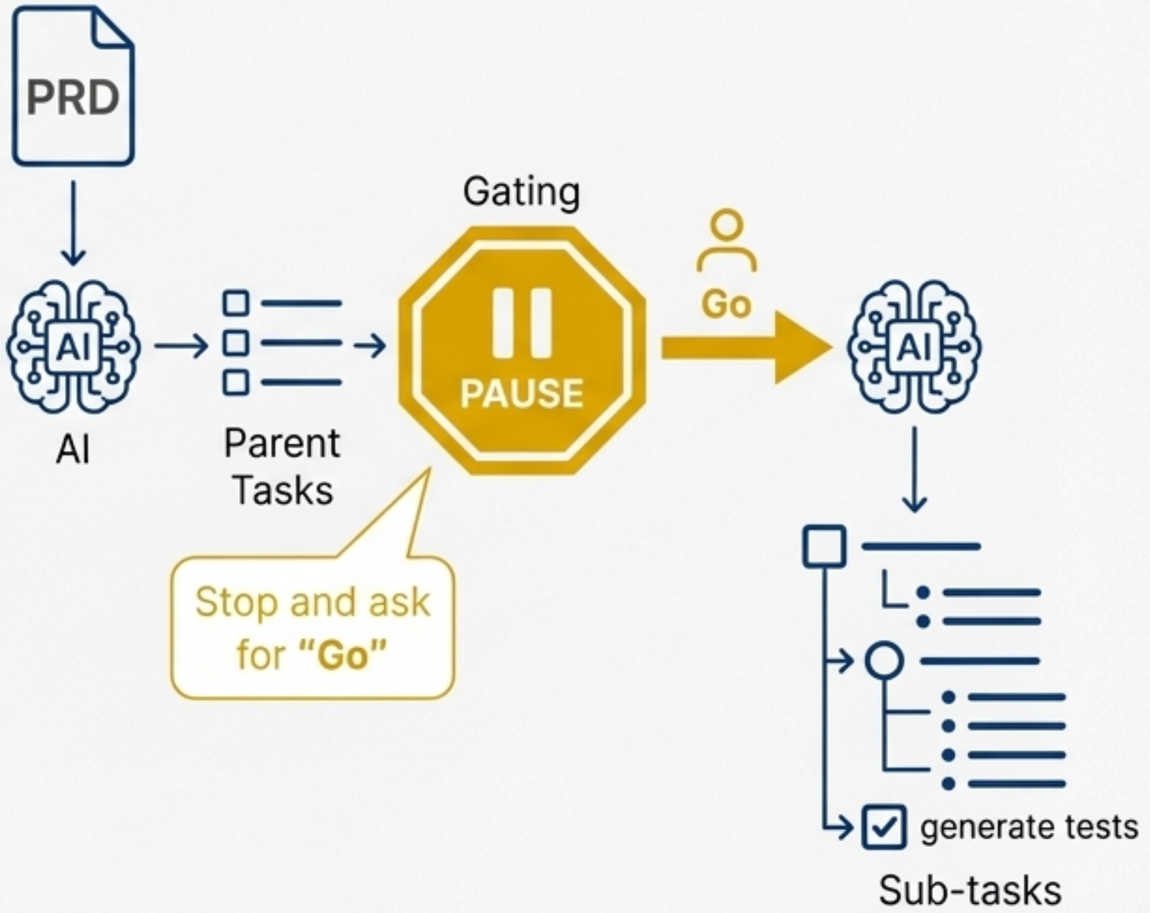
STEPS 3 & 4: FROM A SELECTED FEATURE TO A GATED TASK LIST

Step 3 - Create PRD

- The user selects a feature placeholder file.
- The AI asks clarifying questions to resolve ambiguities.
- It generates a structured PRD in `/tasks/`.
- The AI stops and waits.



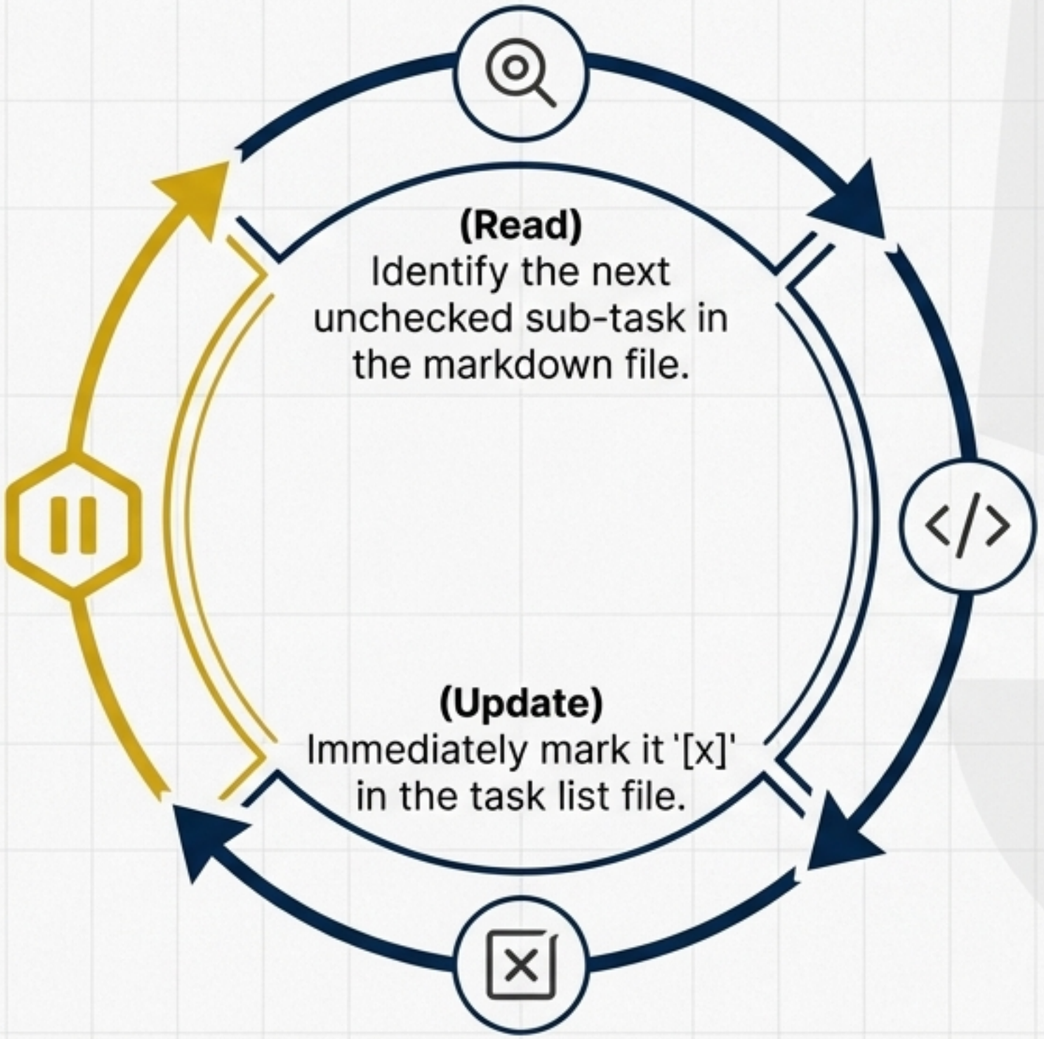
Step 4 - Generate Task List



NotebookLM

STEP 5: EXECUTE WITH PRECISION, ONE SUB-TASK AT A TIME

Goal: Implement the feature safely and track progress directly in the task list file.

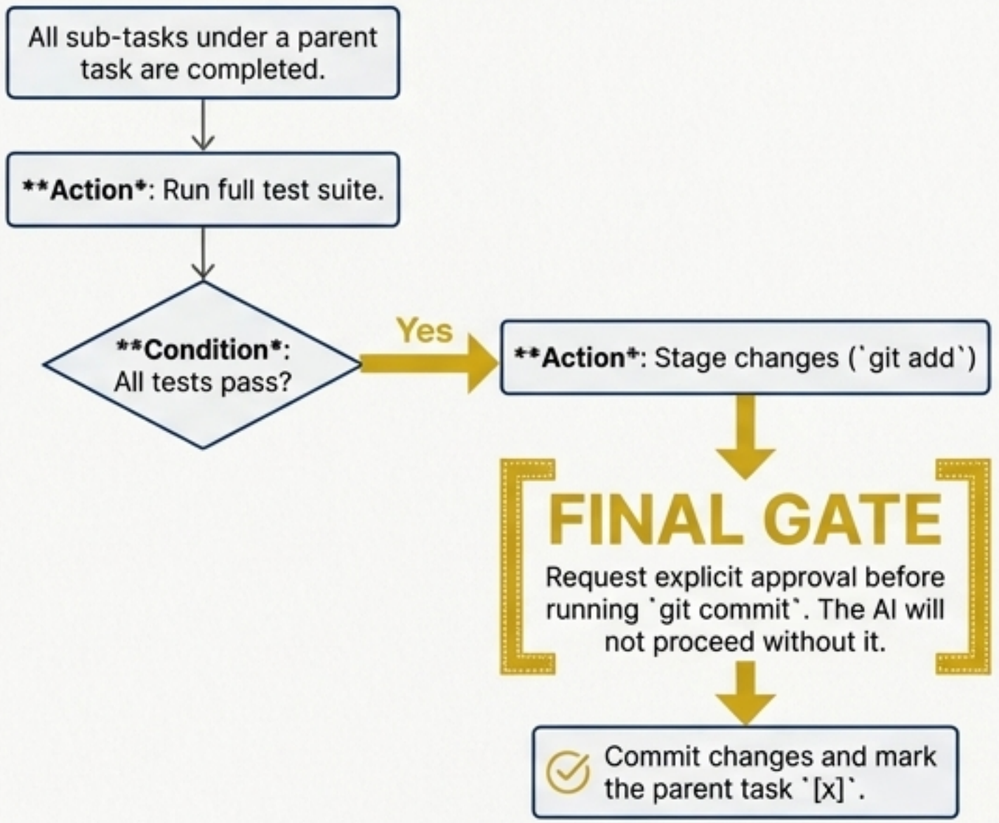


NotebookLM

SAFEGUARDING THE REPOSITORY: THE COMMIT APPROVAL GATE

The protocol enforces strict procedures before any code is committed, mirroring best practices for production engineering.

The Parent Task Completion Workflow



Quote from `Agents.md`: "only request explicit approval when performing publishing/escalated actions (e.g., `git commit`, `git push`, tags, PR writes)."

NotebookLM

THE PROTOCOL CHEAT SHEET: GATING AND APPROVALS

TRIGGERING STEP / FILE	GATING MECHANISM
`generate-features.md`	• **Action:** Generate proposed slice list. ⏸ • **Gate:** Stop and wait for user to reply **‘Go”**.
`generate-tasks.md`	• **Action:** Generate parent tasks. ⏸ • **Gate:** Stop and wait for user to reply **‘Go”**.
`process-task-list.md` (Sub-task)	• **Action:** Implement one sub-task and mark `[x]`. ⏸ • **Gate:** Stop and wait for user to reply **“yes”** or **“y”**.
`process-task-list.md` (Parent task)	• **Action:** Run tests and stage changes. ⏸ • **Gate:** Request explicit user approval before `git commit`.

NotebookLM

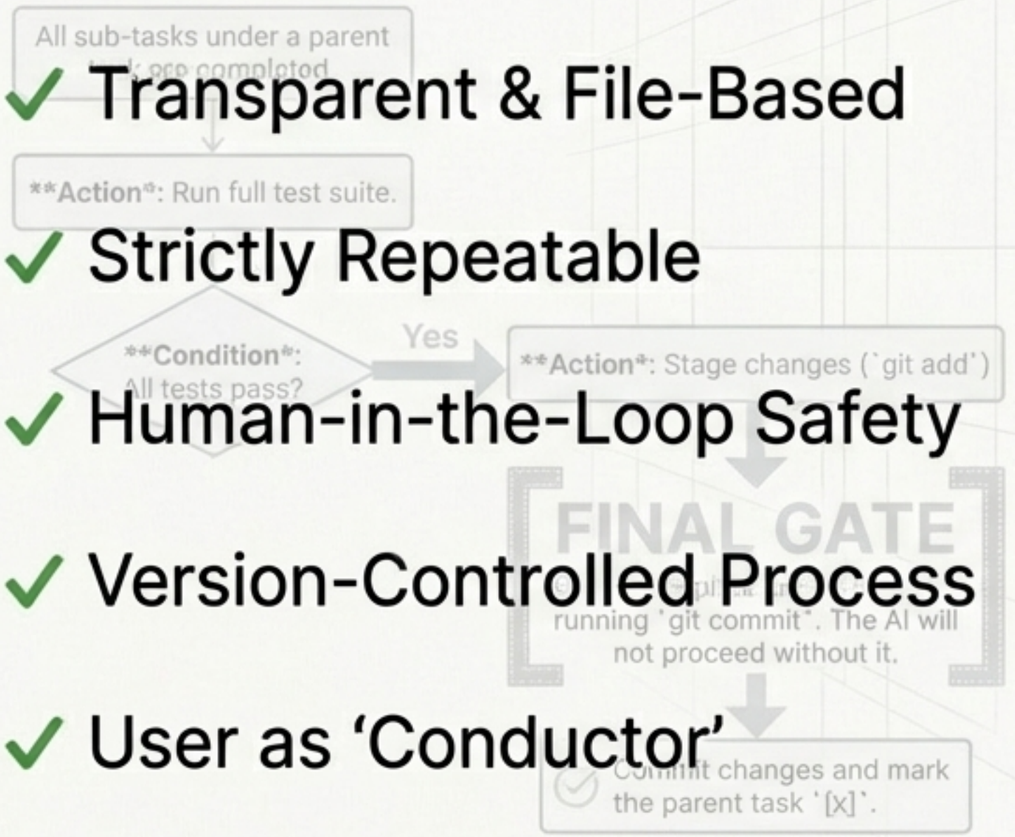
FROM UNPREDICTABLE CODER TO DISCIPLINED ORCHESTRATOR

CHAOS

- ✗ Black Box Operation
- ✗ Unrepeatable Results
- ✗ High Risk of Errors
- ✗ No Audit Trail
- ✗ User as ‘Prompter’

ORCHESTRA

- ✓ Transparent & File-Based
- ✓ Strictly Repeatable
- ✓ Human-in-the-Loop Safety
- ✓ Version-Controlled Process
- ✓ User as ‘Conductor’



NotebookLM

THIS IS ENGINEERING, NOT MAGIC

The Vibe Engineering Orchestrator integrates AI into a rigorous software development lifecycle. It's a system built for professionals who value precision, safety, and repeatability over speed at any cost.

It treats the repository as the ultimate source of truth and human oversight not as a bottleneck, but as the most critical feature.

NotebookLM