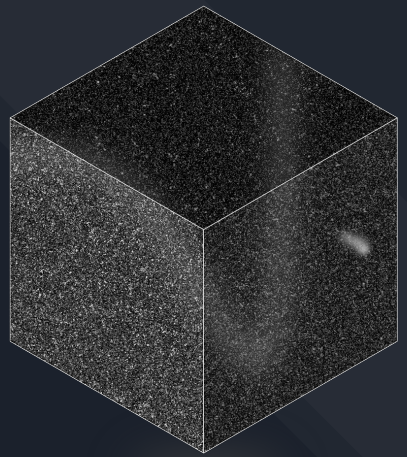
The Concrete Architecture of VOID



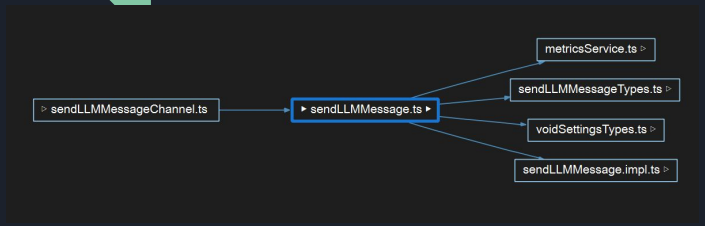
Youtube Video:
https://youtu.be/px5jdkEnupk?si=k5bJOYgua4My1QFD

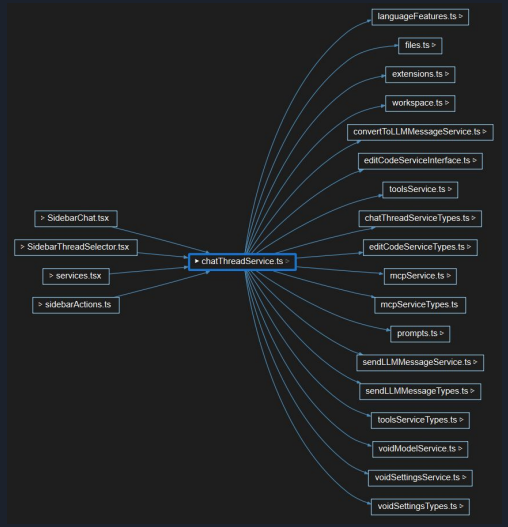
Group 23: We'll Fix It in Post

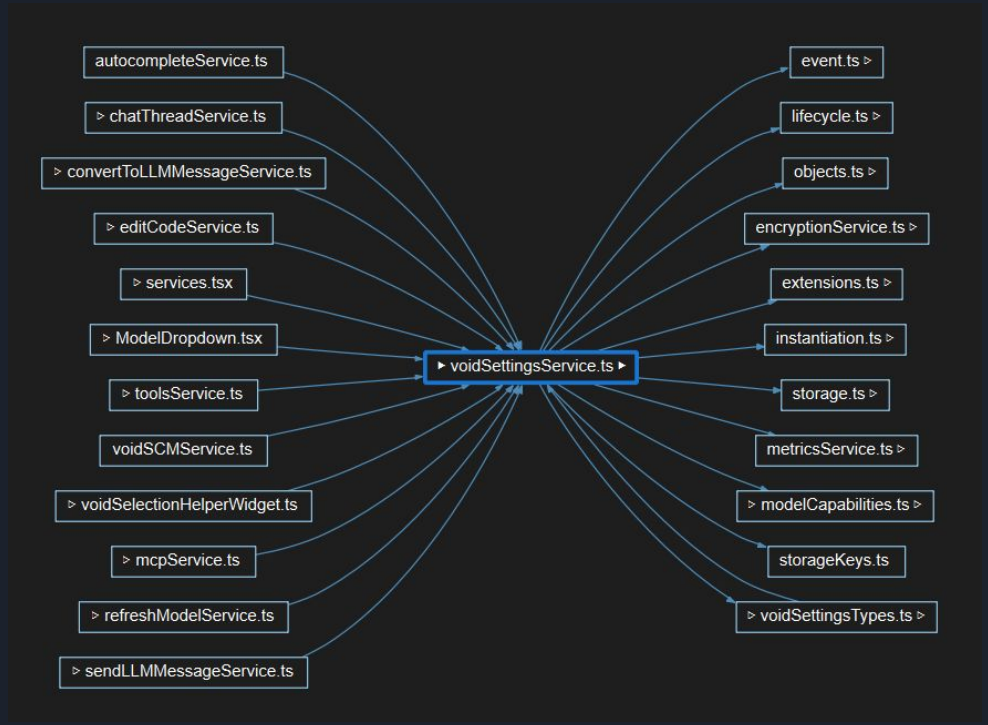
By Group 23

Jaivik Joshi
Zack Wood
Robin Houiler
Charlie Kevill
Kai Maddocks
Jared Simon

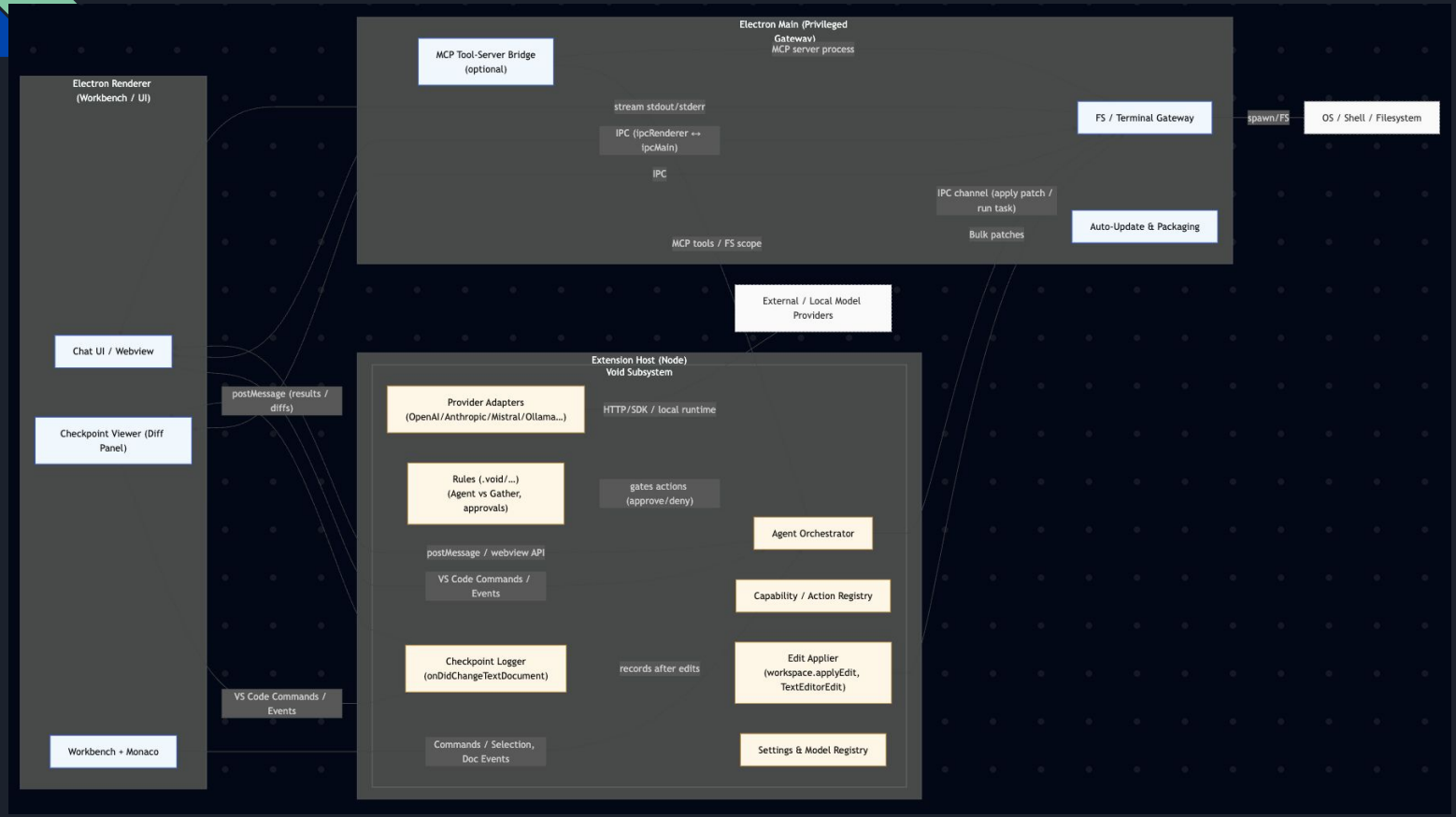
Derivation process (methodology)



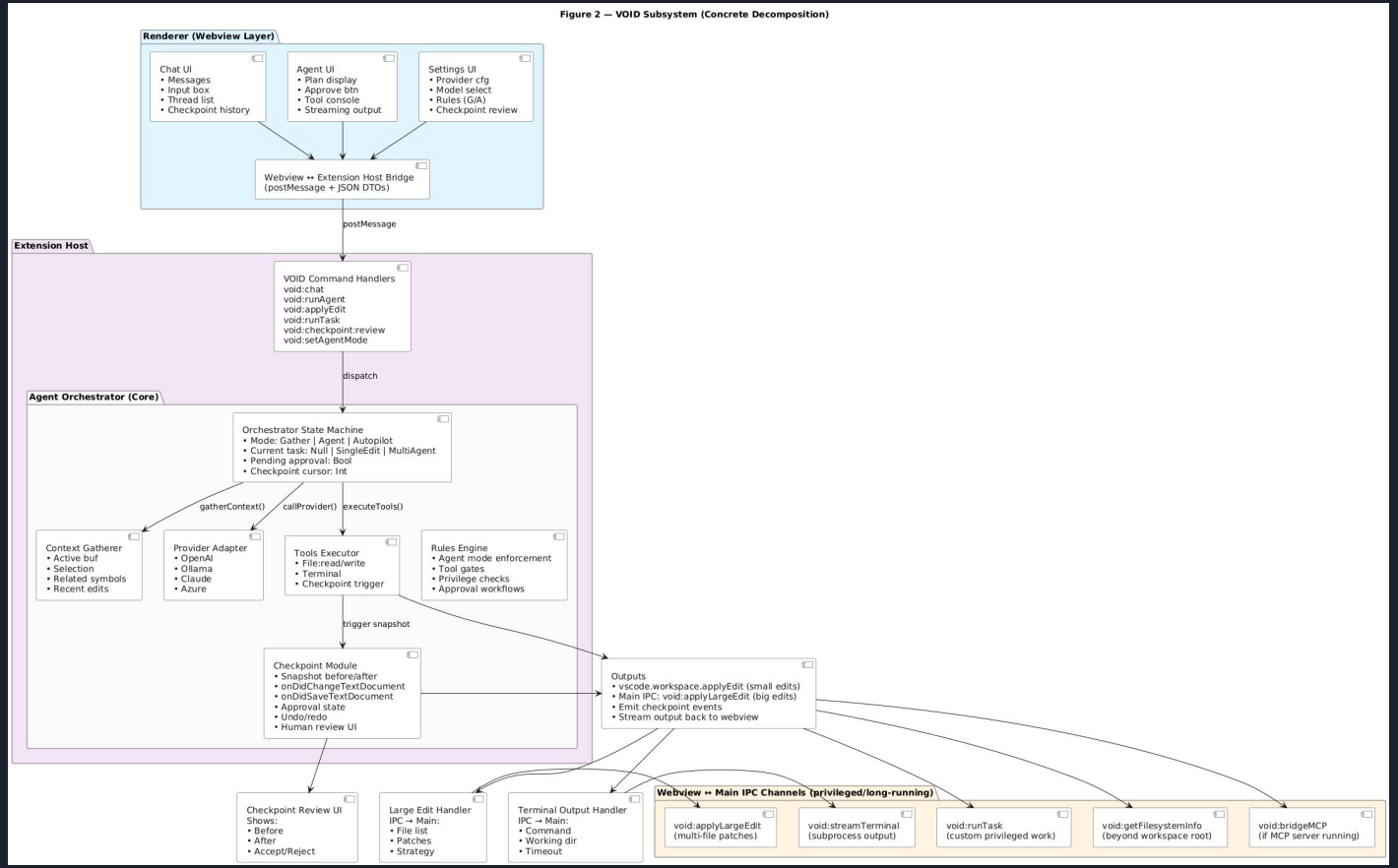




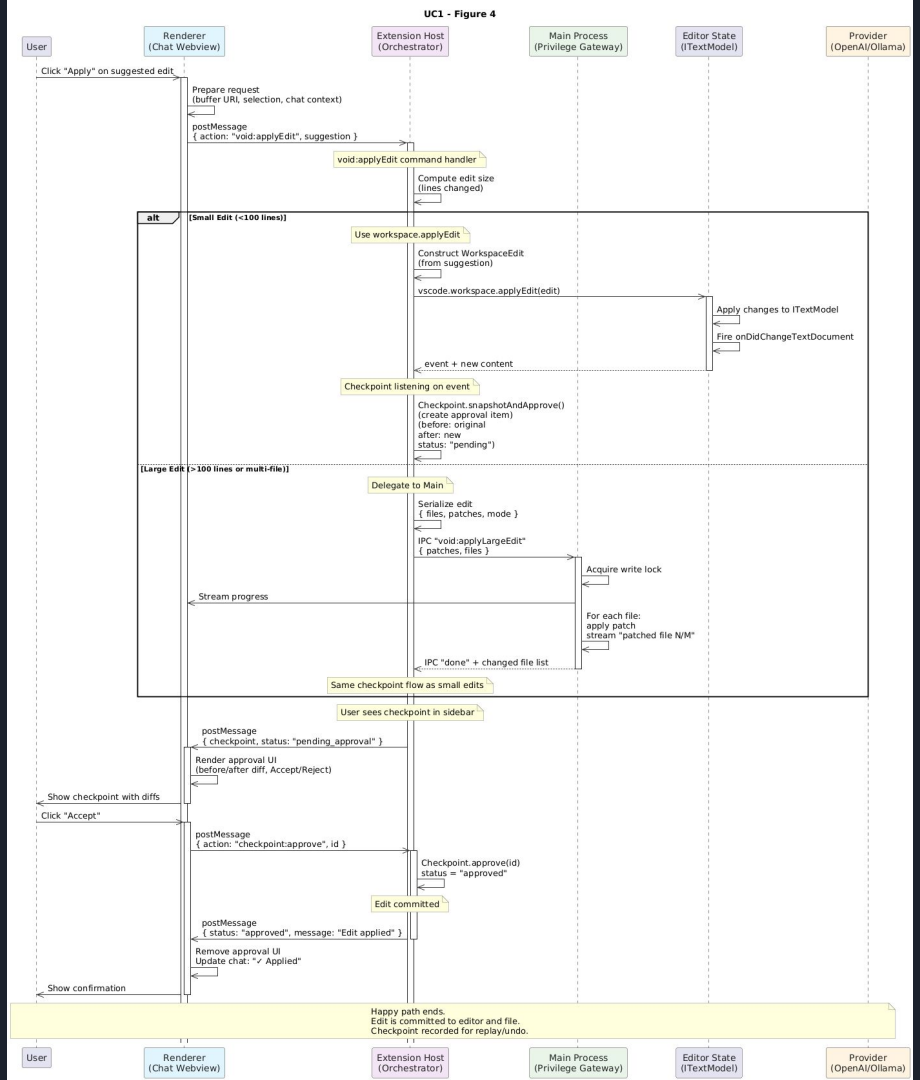
High-level concrete architecture



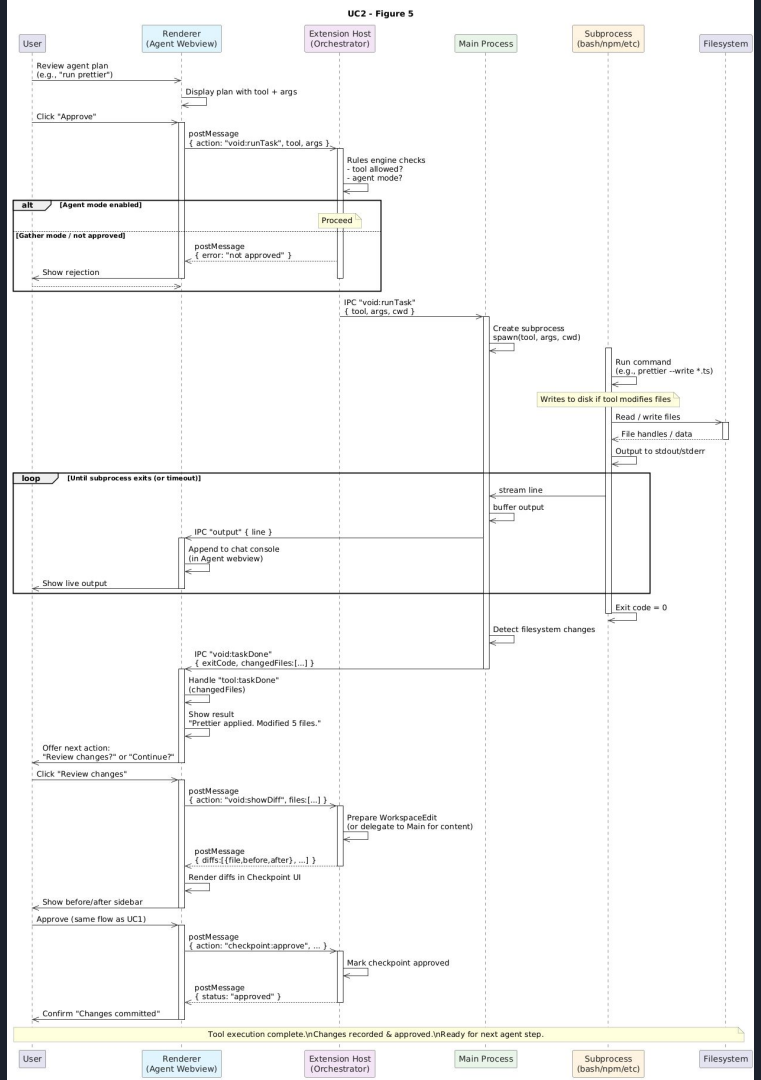
Second-level subsystem: Agent Orchestrator



Sequence diagram A — Apply AI edit with checkpoint



Sequence diagram B — Run terminal tool with approval



Concurrency & team/process effects

Divergences (Reflection)

• Renderer → Main for long tasks: we route heavy or privileged work to Main for stability; it keeps the UI smooth.

• Provider layer split: we separate Provider Router from Adapters so we can swap models without touching orchestration logic.

• EditCodeService split into Patch Engine + Diff/Review + Apply: this enables previewability and fast rollback.

• Policy/approvals around tools: we added a Rules Engine so shell commands and file operations require explicit user approval.

• Telemetry as a cross-cutting concern: we record timing and failure data across components to guide improvements.

Alternatives considered

All in Extension Host: simpler mental model, but weak for rich webviews and limited for terminal/file gateways.

All in Main: maximum control, but we'd lose VS Code ecosystem isolation and risk freezing the UI.

Chosen hybrid: rich webview UI + orchestrator in the Extension Host + gateways in Main. This balances usability, performance, and safety."

Limitations

Lessons learned

Interaction with AI teammate