Multi-Person, Multi-Agent Team Coordination

Two-Layer Architecture

```
OUTER LOOP — GitHub Projects (team coordination)
│
│  Humans assign Issues, track progress on kanban board
│  Agents read/update the board via GitHub MCP Server
│  PRs are the integration point
│
└── INNER LOOP — Each person's local agent orchestration
    │
    │  Git worktrees for isolation
    │  Claude Code / Codex / Copilot — whatever each person prefers
    │  Agent Teams or Vibe Kanban for within-person parallelism
```

Step 1: Shared Foundation

Every team member needs these three things aligned:

a) A shared Git repo — the persistent memory across all agents and humans. No exceptions.

b) A well-maintained `CLAUDE.md` — institutional knowledge. Keep it small (~2.5k tokens). Every team member's agents read it. Contents:

- Project architecture overview
- Coding conventions
- What NOT to do (add rules as mistakes happen)

c) Branch discipline — each person (and each agent) works on its own branch. Never two agents on the same branch.

Step 2: GitHub Projects as the Outer Loop

GitHub Projects V2 is the team coordination layer. Free, built into GitHub, and — critically — **accessible to agents via MCP**.

Board Setup

Column	Purpose
Backlog	Ideas and future work
To Do	Ready for this sprint/iteration
In Progress	Someone (or their agent) is working on it
In Review	PR opened, awaiting review
Done	Merged

Custom Fields

Field	Type	Purpose
Priority	Single select (P0/P1/P2)	What to work on first
Assignee	Person	Who owns this (human, not agent)
Iteration	Iteration field	Sprint/week grouping
Estimate	Number	Story points or T-shirt size

Built-in Automations (Zero Config)

Trigger	Action
Issue added to project	Moves to "Backlog"
PR opened linking an issue	Moves issue to "In Progress"
PR merged	Moves issue to "Done"
Issue closed	Moves to "Done"
Issue reopened	Moves to "To Do"

Step 3: GitHub MCP Server — The Bridge Between Board and Agents

The official GitHub MCP server (`github/github-mcp-server`) gives agents direct read/write access to GitHub Projects. This is the key integration that makes the outer loop work with agents.

What Agents Can Do via MCP

Action	Description
List & retrieve projects	Agent sees the kanban board
List items in a project	Agent reads all cards, status, assignee, priority
List fields	Agent knows the custom fields
Update fields on items	Agent moves cards between columns (To Do → In Progress → Done)
Add issues/PRs to project	Agent puts new work on the board
Remove items from project	Agent cleans up completed work
Create/update issues	Agent manages work items
Create pull requests	Agent submits code for review

MCP Server Configuration

The `projects` toolset is **not enabled by default**. Enable it explicitly:

```
{
  "mcpServers": {
    "github": {
```

```
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "<your-token>",
        "GITHUB_TOOLSETS": "repos,issues,pull_requests,projects"
      }
    }
  }
}
```

Or with the binary:

```
github-mcp-server --toolsets repos,issues,pull_requests,projects
```

The Full Agent-Board Loop

```
GitHub Projects board (team kanban)
    |
    |  Issue #12: "Build auth" → assigned to Person A → status: To Do
    |
    ▼
Person A's agent (Claude Code + GitHub MCP Server)
    |
    |  1. Agent reads board via MCP → sees Issue #12 assigned to them
    |  2. Agent updates #12 status → "In Progress"
    |  3. Agent works on code in git worktree
    |  4. Agent creates PR → "Closes #12"
    |  5. Agent adds PR to the project board
    |  6. Agent updates #12 status → "In Review"
    |
    ▼
Person B reviews PR on GitHub
    |
    |  Merges → GitHub automation moves #12 → "Done"
```

Claude Code GitHub Action (Async Agent Trigger)

For fully asynchronous work, the `claude-code-action` triggers Claude when someone writes `@claude` in an Issue or PR comment:

```
on:
  issues:
    types: [opened, assigned]
  issue_comment:
    types: [created]

jobs:
  claude-pr:
    if: contains(github.event.comment.body, '@claude')
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: anthropics/claude-code-action@v1
        with:
          trigger_phrase: '@claude'
          use_vertex: 'true' # works with Vertex AI credits
```

This enables:

1. Human creates Issue on GitHub Projects board
2. Writes `@claude implement this` in a comment
3. Claude Code Action fires in CI
4. Claude reads the issue, writes code, opens a Draft PR
5. PR appears on the Projects board in "In Review"

Step 4: Individual Setup (Each Team Member)

Each person picks their AI tool and sets up their workspace:

Tool	Setup
Claude Code	Install CLI, set `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1`
Codex (OpenAI)	VS Code extension or CLI
Both	VS Code 1.109+ runs them side-by-side
GitHub Agent HQ	Copilot Pro+ ($39/mo) or Enterprise — runs Claude/Codex in GitHub's cloud

Git Worktrees (Essential)

Each agent session gets its own worktree for complete file isolation:

```
# Person A working on auth
git worktree add ../worktree-auth origin/main

# Person B working on API
git worktree add ../worktree-api origin/main
```

Within-Person Parallelism

Option A — Worktree round-robin (simplest)

```
# Terminal 1: Claude on feature-X
cd ../worktree-feature-x && claude

# Terminal 2: Claude on tests
cd ../worktree-tests && claude

# Terminal 3: Codex on docs
cd ../worktree-docs && codex
```

Option B — Agent Teams (for complex single tasks)

```
"Create a team. One teammate reviews security,
 one writes tests, one refactors the API layer."
```

Step 5: CI as the Quality Gate

Once multiple agents are producing PRs in parallel, you need automated checks before a human even looks at the code. This is what makes the parallelism safe.

Start with tests

Agents are good at writing unit tests — ask them to. Once you have tests in the repo, set up GitHub Actions to run them on every PR:

```
# .github/workflows/ci.yml
on:
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4  # or setup-python, etc.
      - run: npm ci && npm test
```

Then add AI code review

The Claude Code GitHub Action (from Step 3) doubles as a CI reviewer. By the time you open a PR, both tests and an AI review have already run. You're reviewing a PR that's already been checked twice.

The progression

1. Add unit tests to the repo (agents write them as part of each task)
2. Set up GitHub Actions to run tests on every PR
3. Add Claude Code Action for AI review on CI
4. Now when you review a PR, you're the third pair of eyes — not the first

---

Step 6: Getting Started (Concrete Exercise)

Pick a real task your team needs done. Split it:

Person	Task	Agent
Person A	Implement feature X	Claude Code in worktree-A
Person B	Write tests for module Y	Claude/Codex in worktree-B
Person C	Review & refactor module Z	Claude Code in worktree-C

Each person works independently. PRs merge the work. Start small — one task per person, one agent each.

Key Rules

1. **Each agent session = one focused task.** Don't reuse long conversations.

2. **Commit after every completed sub-task.** Git is memory, not the chat.
3. **Start fresh sessions aggressively.** Context rot is real.
4. `CLAUDE.md` **is a living doc.** When an agent makes a mistake, add a rule.
5. **Human reviews every PR.** Agents handle 80% of work, humans handle judgment.
6. **GitHub Projects is the single source of truth** for what needs to be done and who's doing it.
7. **Learn CI/CD along the way.** This whole setup naturally pushes you to learn GitHub Actions, automated testing, and deployment pipelines. If you're adding unit tests (and you should — agents are good at writing them), you'll want CI running those tests on every PR. It's one of those skills that pays for itself quickly and fits right into the workflow.

Tool Stack Summary

Layer	Tool	Cost
Team coordination	GitHub Projects V2	Free
Agent ↔ Board bridge	GitHub MCP Server (`projects` toolset)	Free (open source)
Async agent triggers	Claude Code GitHub Action	Free (your API costs)
Human communication	Slack.	Free tier
Agent orchestration (local)	Claude Code CLI / Agent Teams	VertexAI Credits
Code review (AI)	Claude Agent SDK on CI / Agent HQ	VertexAI Credits

---

Practice Project: MedGemma Impact Challenge (Agentic Workflow Prize)

Use the multi-agent team workflow to compete in a real Kaggle hackathon with $100K in prizes. The **Agentic Workflow Prize ($10,000)** maps directly to the skills this project is about.

Competition Overview

Detail	Value
Host	Google Research on Kaggle
Prize pool	$100,000 total
Agentic Workflow Prize	$10,000
Deadline	February 24, 2026
Results	March 17–24, 2026
Teams	~1,700 active
URL	kaggle.com/competitions/med-gemma-impact-challenge

What You Submit

1. **A working demo application** using at least one HAI-DEF model
2. **A writeup** following their template (problem, solution, technical details)
3. **Reproducible code** (public repo)
4. **A video** (3 minutes or less)

Evaluation Criteria

1. Effective use of HAI-DEF models
2. Importance of the problem
3. Potential real-world impact
4. Technical feasibility
5. Execution and communication quality

Prize Tracks

Track	Prize	Focus
Main Track	$75,000	Best overall projects
Agentic Workflow Prize	$10,000	Reimagine complex workflows using HAI-DEF models
Novel Task Prize	$10,000	Best fine-tuned model for a useful new task
Edge AI Prize	$5,000	Solutions running on local/edge devices

Available HAI-DEF Models

Model	Size	What it does	Hugging Face
MedGemma 1.5 4B (multimodal)	4B params	Medical images + text — CXR, CT, MRI, derm, pathology, EHR	`google/medgemma-1.5-4b-it`
MedGemma 27B (multimodal)	27B params	Same but higher accuracy, needs more compute	`google/medgemma-27b-it`
MedGemma 27B Text	27B params	Text-only clinical reasoning, no images	`google/medgemma-27b-text-it`
MedASR	—	Medical speech-to-text (dictation, notes)	Part of HAI-DEF
CXR Foundation	EfficientNet-L2	Chest X-ray embeddings — classify findings with very few labels	`google/cxr-foundation`
Derm Foundation	—	Skin image embeddings for dermatology	`google/derm-foundation`
Path Foundation	—	Histopathology image embeddings	`google/path-foundation`
HEAR	—	Health audio embeddings (coughs, breaths)	`google/hear`

MedGemma Key Capabilities

- **Medical image interpretation** — CXR, CT, MRI, dermatology, pathology, ophthalmology
- **Medical text comprehension** — EHR summaries, discharge notes, clinical Q&A
- **Clinical reasoning** — patient triaging, decision support, differential diagnosis
- **EHR understanding** — FHIR-based records, discharge summaries, lab reports
- **Agentic orchestration** — MedGemma as a tool within an agent system, paired with web search, FHIR generators, Gemini for function calling, or local privacy-preserving parsing before sending anonymized requests to cloud models

Project Idea: Agentic Clinical Intake & Triage Assistant

Target: Agentic Workflow Prize ($10,000)

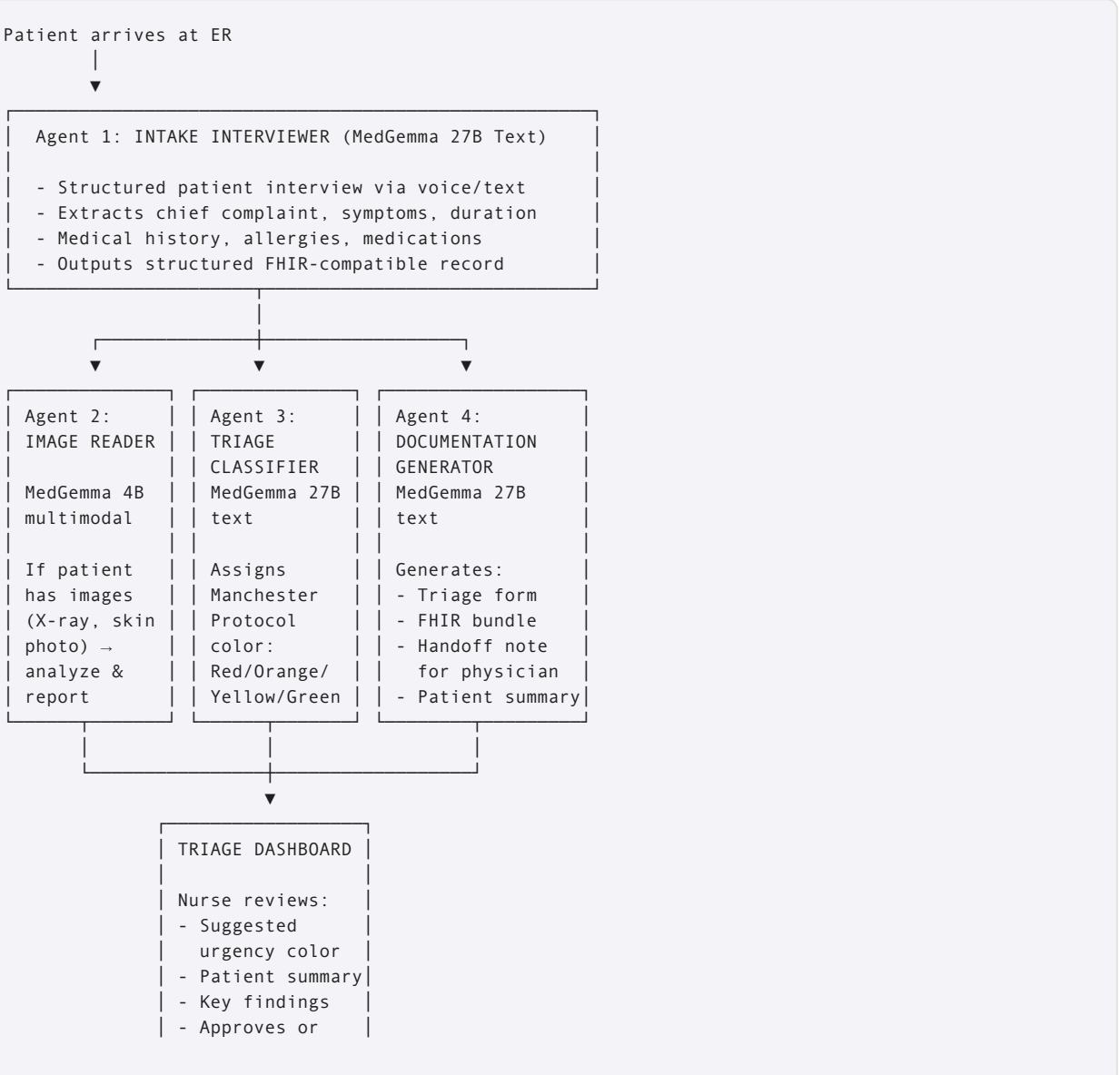
Who this is for: The **triage nurse** — the first clinical contact when a patient arrives at the ER. The tool does not replace the nurse. It runs alongside them as an AI-powered assistant, while the nurse retains full authority to approve, override, or adjust every decision.

Problem: In Brazilian public health (SUS), patients wait hours in emergency rooms. Triage nurses are overwhelmed — they must simultaneously:

1. Interview the patient (complaint, symptoms, history)
2. Check vitals
3. Review any available images (X-ray, skin lesion)
4. Classify urgency using the Manchester Protocol (Red/Orange/Yellow/Green/Blue)
5. Document everything manually
6. Hand off to the physician

All of this is done manually, which is slow and error-prone. The nurse is the bottleneck: every minute spent on paperwork is a minute not spent on the next patient.

Solution: An agentic workflow where MedGemma orchestrates the clinical intake process, giving the triage nurse a pre-filled assessment to review instead of building one from scratch. MedGemma handles the structured interview and documentation, analyzes any images, and suggests a triage color. The nurse reviews, approves or overrides, and moves on to the next patient — faster and with a second opinion on classification.



```
                │   overrides      │
                └──────────────────┘
```

Why this wins the Agentic Workflow Prize:

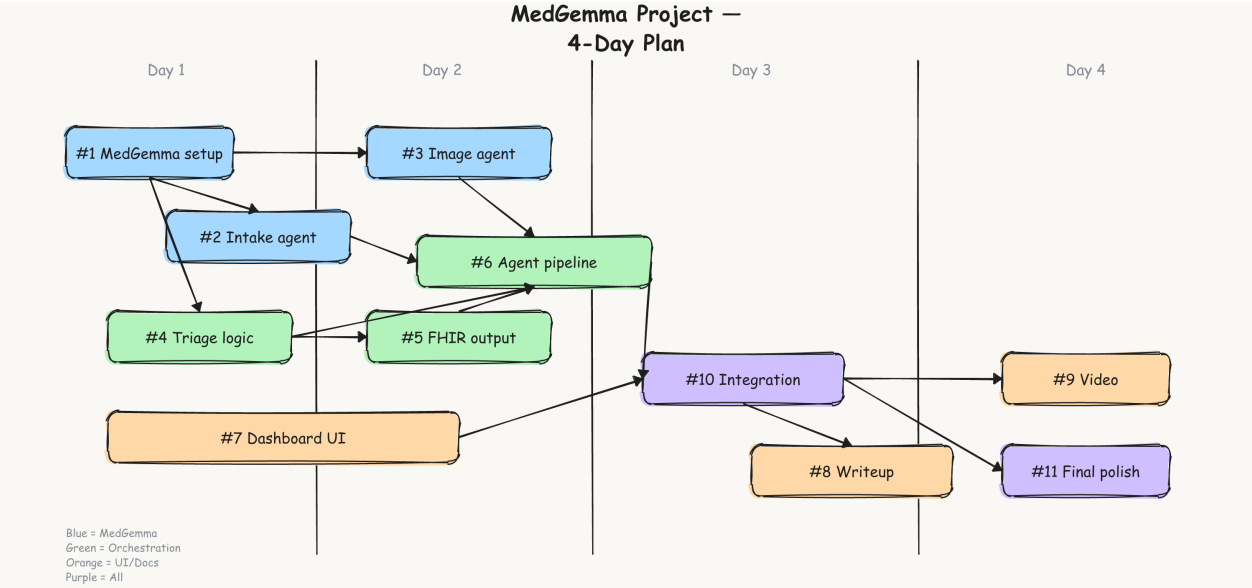
- Uses multiple HAI-DEF models in a coordinated pipeline (not just one model)
- Reimagines a real, broken clinical workflow (ER triage)
- Human-in-the-loop — nurse always has final say
- Privacy-first — MedGemma runs locally, patient data never leaves the facility
- FHIR output — integrates with existing hospital systems
- Addresses underserved populations (SUS serves 150M+ Brazilians)

Task Split for 3 People

Person	Domain	What they build
Person A	MedGemma integration	Model loading, inference pipeline, prompt engineering for each agent role. Test with sample medical cases.
Person B	Agentic orchestration	The multi-agent workflow — intake → image analysis → triage → documentation. FHIR output generation. Agent coordination logic.
Person C	UI + writeup + video	Triage dashboard (web UI), patient intake form, nurse review screen. Kaggle writeup. Record 3-min demo video.

GitHub Projects Board (4-day plan)

With each person running multiple agents in parallel, this compresses to 4 days — that's the whole point of this workflow.




```
#1 MedGemma setup     (day 1)      ← foundation, no deps
#2 Intake agent       (day 1-2)    ← depends on #1
#3 Image agent        (day 2-3)    ← depends on #1
#4 Triage logic       (day 1-2)    ← depends on #1 (uses MedGemma 27B)
#5 FHIR output        (day 2-3)    ← depends on #4
#6 Agent pipeline     (day 2-3)    ← depends on #2, #3, #4, #5 (start with stubs day 2)
#7 Dashboard UI       (day 1-3)    ← no deps (mock data)
#10 Integration       (day 3)      ← depends on #6, #7
```

```
#8 Writeup            (day 3-4)     ← depends on #10
#9 Video              (day 4)       ← depends on #10
#11 Final polish      (day 4)       ← depends on #10
```

Tech Stack

Component	Tool
Models	MedGemma 4B (images) + MedGemma 27B Text (reasoning) via Vertex AI
Orchestration	Python — LangGraph or simple agent loop
FHIR output	`fhir.resources` Python library
UI	Streamlit or Gradio (fast to build)
Hosting	Vertex AI endpoints for models, local for demo
Code	Public GitHub repo

References

- MedGemma docs
- MedGemma model card
- HAI-DEF on Hugging Face
- MedGemma agentic orchestration — official docs mention pairing with FHIR, Gemini, web search
- How to Build Agentic Workflows with MedGemma 1.5 (YouTube)
- MedGemma Kaggle submission example (YouTube)

References

Why Multi-Agent Coordination

- SemiAnalysis — Claude Code is the Inflection Point — 4% of GitHub public commits from Claude Code, projected 20%+ by end of 2026
- TLDR Dev — Building a C Compiler with Parallel Claudes — 16 agents, 100K lines of Rust, $20K, compiles the Linux kernel
- 10 Claude Code Secrets from the Team — Boris Cherny's workflow: 5-10 parallel sessions, 50-100 PRs/week
- Anthropic — 2026 Agentic Coding Trends Report (PDF)

Tooling

- GitHub MCP Server — includes `projects` toolset for agent ↔ board integration
- Claude Code Agent Teams — team lead + teammates, shared task list, peer messaging
- Claude Code GitHub Action — `@claude` trigger in Issues/PRs, works with Vertex AI
- Claude Code Multi-Agent Orchestration (IndyDevDan)