GUL'S NOTEBOOK · AI RESEARCH · OPEN ACCESS

Agents that do real work: a pragmatic multi-agent design

Muhammad Gulzaib
https://orcid.org/0009-0000-8709-6989
SynergyFlow Labs · Islamabad, Pakistan

Published	2026-11-16
URL	https://gulzaib.vercel.app/blog/agents-that-do-real-work-a-pragmatic-multi-agent-design
Licence	Creative Commons Attribution 4.0 International (CC BY 4.0), https://creativecommons.org/licenses/by/4.0/

ABSTRACT

Start with a workflow and add an agent only where the path can't be known. The design I use at SynergyFlow Labs: narrow tools, state, budgets and approvals.

Keywords: agents, llm, architecture, automation

> **TL;DR**
>
> Start with a workflow, not an agent. Add autonomy only where the path genuinely can't be known in advance. When you do use agents, give them narrow tools, explicit state, budgets and a human approval step for anything irreversible.

At SynergyFlow Labs we build software that does work rather than just showing it: following up on leads, drafting WhatsApp replies in Urdu, reconciling invoices. "Agentic AI" is the label, but the hard part isn't the model. It's everything around it.

Workflows first

Most business processes are predictable. If you can draw the steps on a whiteboard, write them as code and use the LLM *inside* specific steps: classify this message, extract these fields, draft this reply. That's a workflow, and it's easier to test, cheaper to run and simpler to debug than an autonomous agent [3].

Fixed workflow vs. autonomous agent

Pros

- Workflow: predictable cost and latency
- Workflow: easy to test step by step
- Agent: handles open-ended tasks

- Agent: adapts when a step fails

Cons

- Workflow: breaks on cases you didn't plan for
- Agent: harder to test and audit
- Agent: cost and latency vary a lot
- Agent: can loop or take wrong actions

I reach for an agent loop only when the number and order of steps genuinely depend on what the model finds along the way, like researching a lead across several sources.

The agent loop

The core loop is the ReAct pattern [1]: the model reasons about what to do, calls a tool, observes the result and repeats until it's done.

1. **Goal + state**: what we know so far
2. **Think**: pick the next action
3. **Act**: call one tool
4. **Observe**: append the result
5. **Done?**: or loop, within budget

`agent.ts`

```
async function runAgent(goal: string, tools: Tool[], budget = { steps: 12, usd: 0.5 }) {
  const state: Step[] = []
  for (let i = 0; i < budget.steps; i++) {
    const action = await llm.decide({ goal, state, tools: tools.map((t) => t.spec) })
    if (action.type === 'finish') return action.result

    const tool = tools.find((t) => t.name === action.tool)!
    // irreversible actions wait for a human
    if (tool.irreversible && !(await approve(action))) {
      state.push({ action, result: 'rejected by human' })
      continue
    }
    state.push({ action, result: await tool.run(action.input) })
    if (spent() > budget.usd) break
  }
  return escalate(goal, state) // hand over to a person with full context
}
```

Design rules

1. **Narrow tools with typed inputs**

   `send_whatsapp(to, template_id, vars)` is safer than `send_message(anything)`. Models are good at learning to use well-described tools [2], and narrow tools limit how much damage a mistake can do.

2. **Explicit state, outside the prompt**

   Store each step's action and result in the database. You get audit trails, resumability after crashes, and the ability to replay a run when debugging.

3. **Budgets on everything**

   Max steps, max cost, max wall-clock time. An agent without a budget will eventually find a loop.

4. **Humans approve the irreversible**

   Sending money, emailing a customer or deleting data needs a person to click approve, at least until you have weeks of logs showing the agent gets it right.

5. **Escalate gracefully**

   When the agent gives up, hand the task to a person *with the full trace*, so they don't start from zero.

> **Prompt injection is real**
>
> If an agent reads emails or web pages, those pages can contain instructions. Treat all retrieved content as untrusted data, never as instructions, and never give a browsing agent access to destructive tools.

Multi-agent, carefully

On my portfolio I show six agents: research, planning, coding, sales, support and operations. In practice, "multiple agents" mostly means **multiple prompts and tool sets with clear handoffs**, coordinated by plain code. A planner that delegates to specialists can work well, but every extra agent adds latency, cost and a new place for things to go wrong.

My rule of thumb: split into separate agents when the tools or context are genuinely different (the sales agent should never see deployment tools), not just because it looks good in a diagram.

How I evaluate agents

- **Task success rate** on a fixed set of realistic tasks, rerun on every prompt or model change.

- **Steps and cost per successful task**, because a correct agent that costs too much is still a failure.
- **Human override rate:** how often people reject or edit the agent's proposed actions.
- **Trace review:** reading 20 full runs a week catches problems no metric shows.

> **Key takeaways**
>
> - Use workflows for predictable processes. Use agents only where the path is unknown.
> - Narrow, typed tools; explicit, stored state; budgets on everything.
> - Humans approve anything irreversible.
> - Split into multiple agents for different tools or context, not for the diagram.

Questions readers ask

When should I use an AI agent instead of a workflow?

Use a fixed workflow whenever you can draw the steps in advance, and call the LLM inside specific steps. Reach for an agent loop only when the number and order of steps depend on what the model finds along the way.

How do you keep AI agents safe in production?

Give them narrow, typed tools, explicit stored state and budgets on steps, time and cost, and require a human approval step for anything irreversible.

When does a multi-agent design make sense?

Split work into multiple agents when parts need different tools or different context, not because the architecture diagram looks better.

References

1. Yao, S. et al. ReAct: Synergizing Reasoning and Acting in Language Models. ICLR, 2023. https://doi.org/10.48550/arXiv.2210.03629
2. Schick, T. et al. Toolformer: Language Models Can Teach Themselves to Use Tools. NeurIPS, 2023. https://doi.org/10.48550/arXiv.2302.04761
3. Anthropic. Building effective agents, 2024.

CITE AS

Gulzaib, M. (2026, November 16). Agents that do real work: a pragmatic multi-agent design. Gul's Notebook. https://gulzaib.vercel.app/blog/agents-that-do-real-work-a-pragmatic-multi-agent-design

You may share and adapt this text for any purpose, including commercially, as long as you credit the author and link to the original. Licence: https://creativecommons.org/licenses/by/4.0/