Content Generation using AI Agents

2025

›Agenda

1. Definition of Agents
2. Agents vs Scripts
3. Example

"Introduction"

4. Agentic Frameworks
5. Why CrewAI
6. Basic blocks of the framework

"Tools"

7. Case-Study: Content generation
8. Pipeline overview
9. Task-specific tools
10. Demo

"Task"

11. Best-practices: Phase isolation
12. Best-practices: Prompts
13. Best-practices: Helpers
14. Additional section: Bad vs Good example

"Best-practice"

15. Results
16. Recap
17. Useful Links

"Conclusion"

Introduction

Definition of Agents

AI Agent - Autonomous program that uses LLMs to reason, plan, and execute tasks.

Key Characteristics:

- Goal-oriented (given objective, figures out how to achieve it)
- Uses tools (search, web scraping, databases, APIs)
- Multi-step reasoning (breaks down complex tasks)
- Self-correcting (retries on failure)

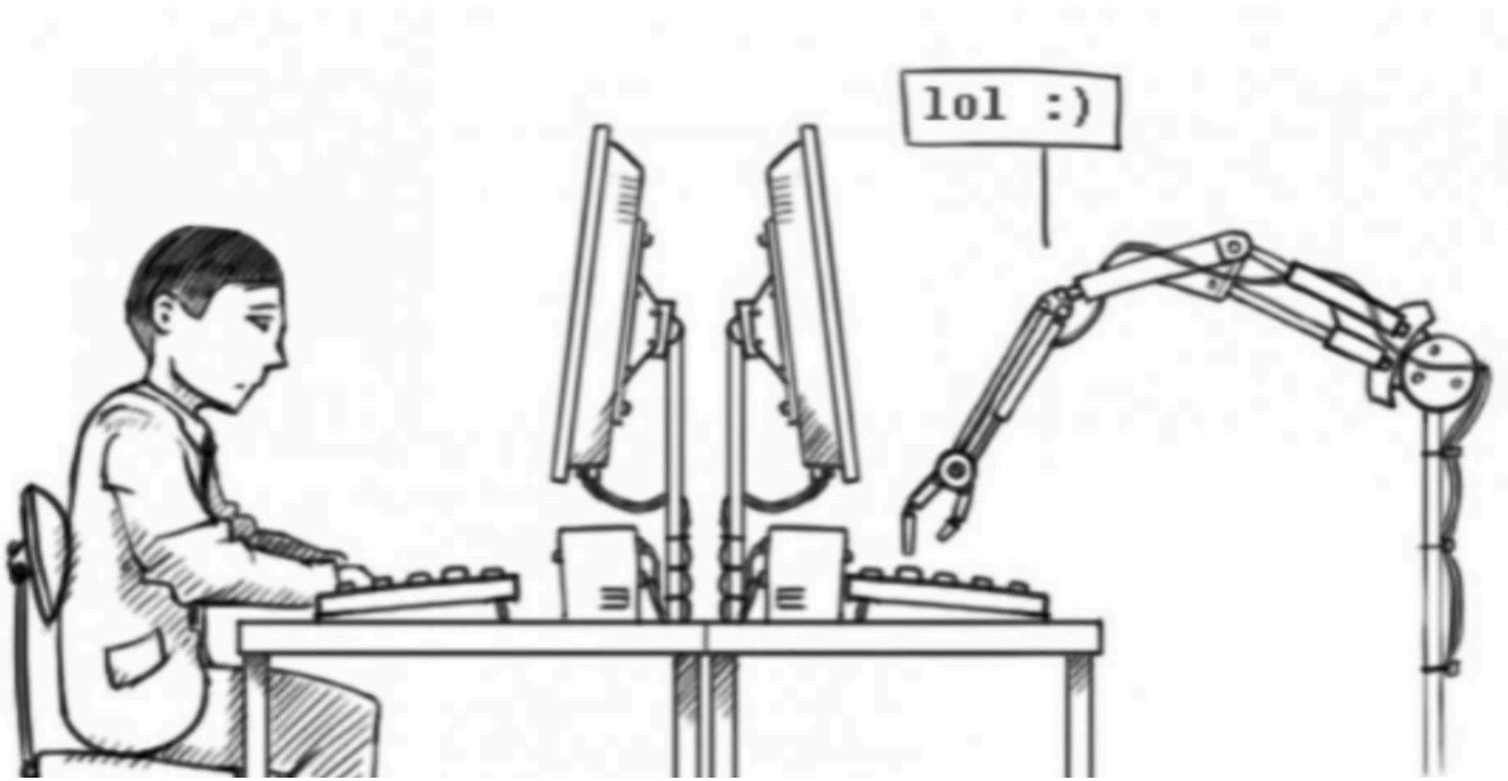


image source

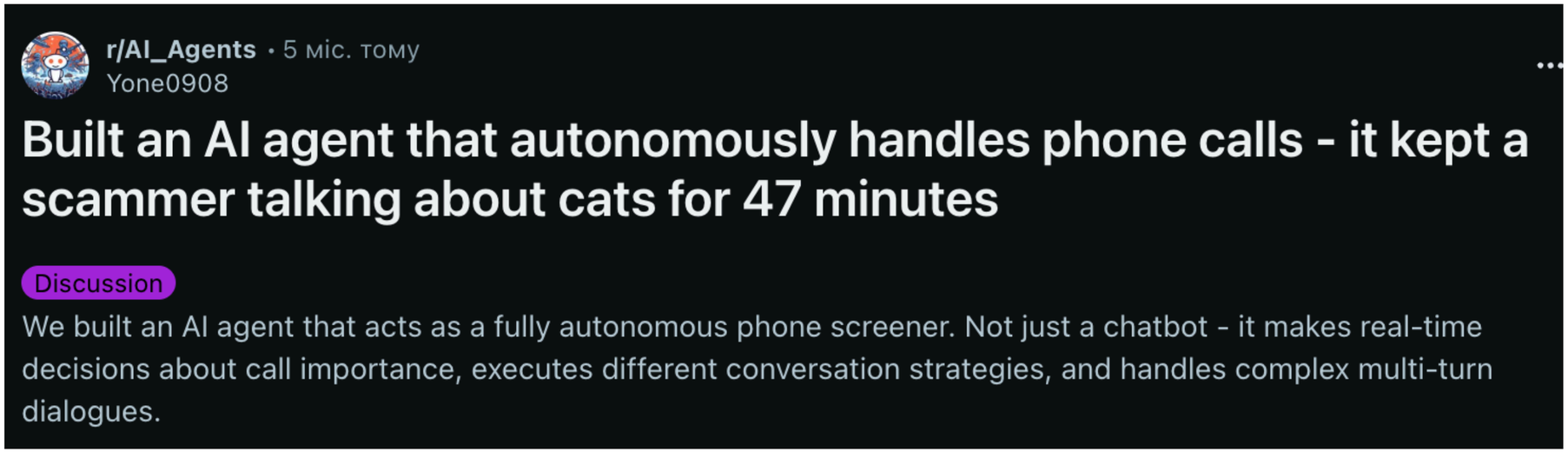
Agents vs Scripts

Scripts

Agents

Traditional Scripts	AI Agents
Fixed logic: `if condition: do_action()`	Reasoning: "What should I do to achieve X?"
Manual error handling	Self-correcting behavior
Hard-coded workflows	Dynamic planning based on context
Limited to predefined paths	Explores multiple strategies

Example

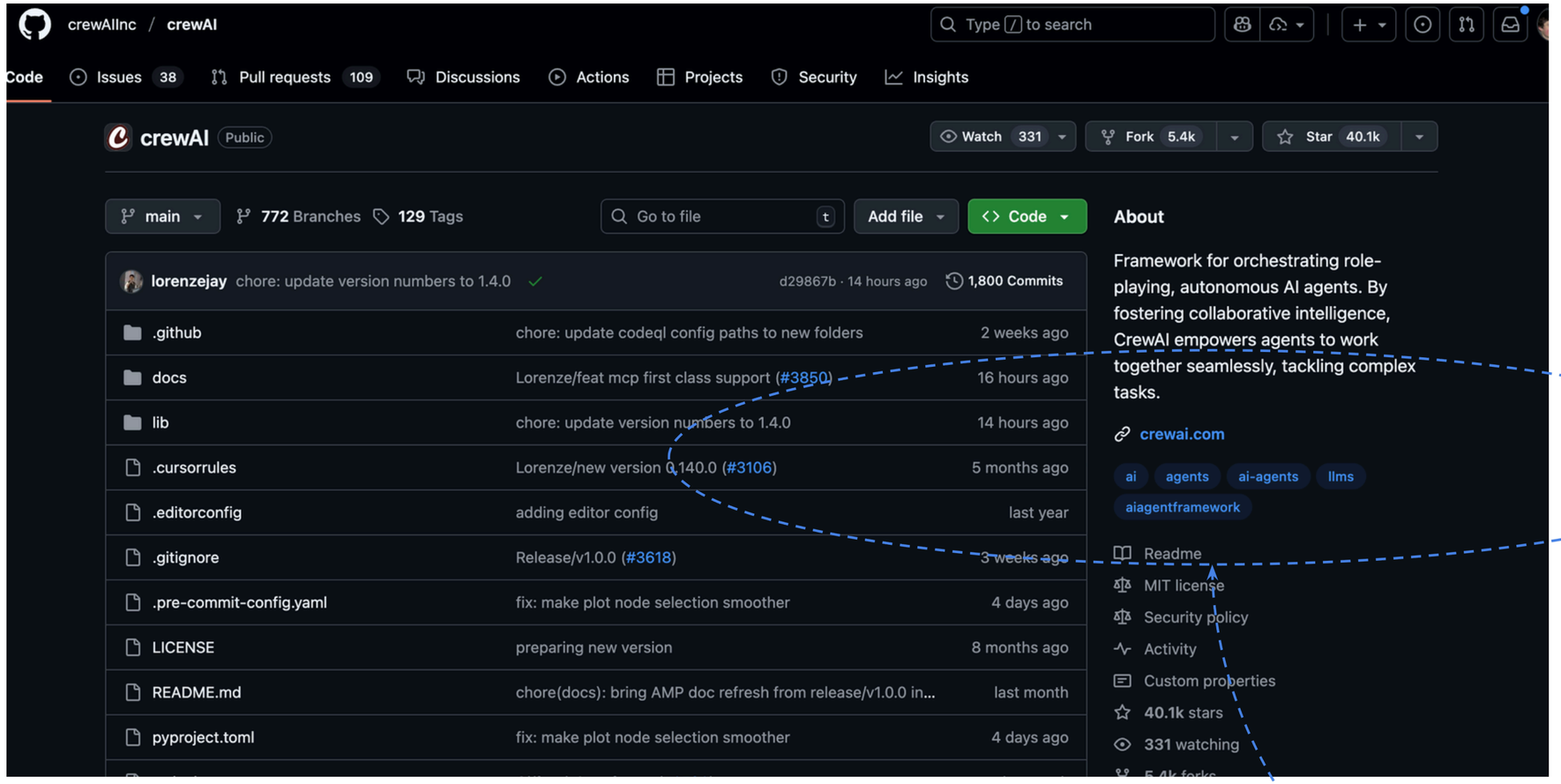

r/AI_Agents • 5 міс. тому
Yone0908
Built an AI agent that autonomously handles phone calls - it kept a scammer talking about cats for 47 minutes
Discussion
We built an AI agent that acts as a fully autonomous phone screener. Not just a chatbot - it makes real-time decisions about call importance, executes different conversation strategies, and handles complex multi-turn dialogues.

Tools (Frameworks)

Agentic Frameworks

Framework	Stars	Description
LangChain	☆90k	Large ecosystem with strong memory handling, great for research and production, **but** usually single agent system
LangGraph	☆20k	Graph-based orchestration with strong error recovery and stateful workflows, **but** quite low-level.
LlamaIndex	☆45k	Excellent for data-centric workflows, **but** less suited for general-purpose agent tasks.
CrewAI	☆40k	Pythonic style framework that supports multi-agent teamwork and role-based collaboration.
AutoGen	☆51k	Advanced multi-agent orchestration with agent-to-agent communication, **but** complex and requires strong coding expertise.
Smolagents	☆23k	Minimal, efficient, and lightweight, **but** lacks advanced orchestration features for complex workflows.
PydanticAI	☆13k	Typed, schema-safe agents with strong validation, **but** narrowly focused on schema safety over full orchestration.

Why CrewAI?

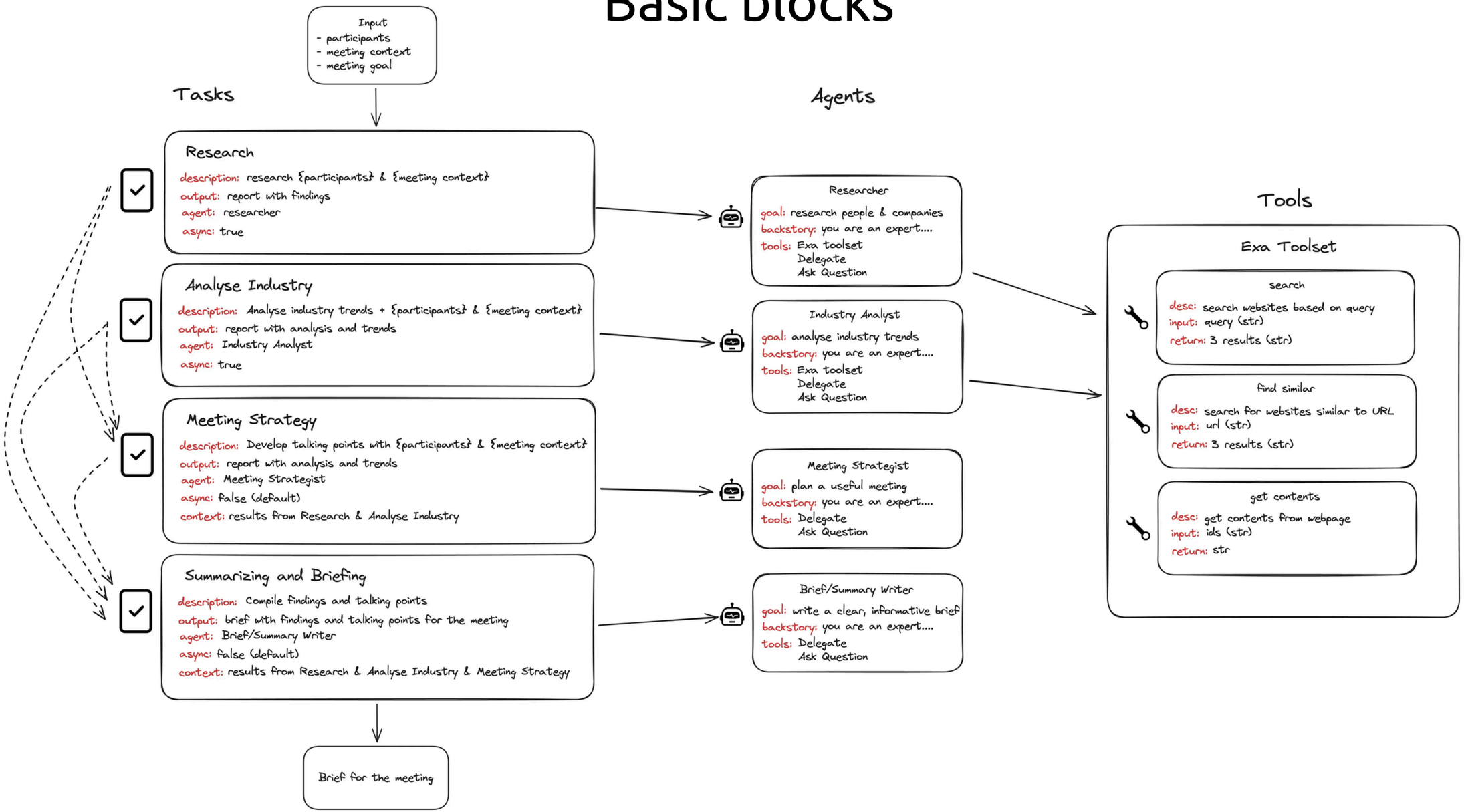


The frameworks are quite similar

The best one for production is to use none and “roll your own”

Quite good docs
High abstraction
Is natively multi-agentic

Basic blocks



Case-study: Content generation

The Problem

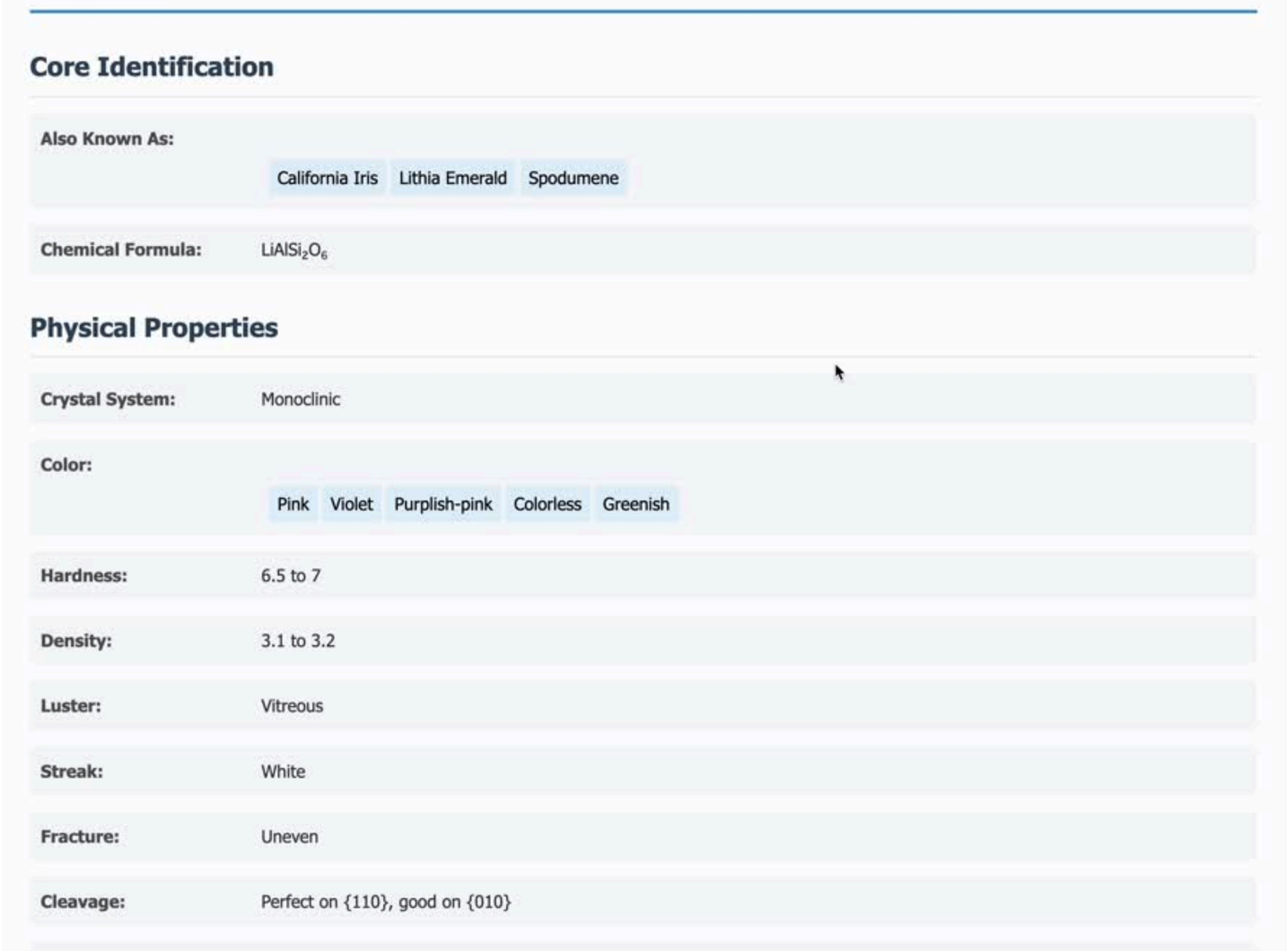
Goal: Generate structured research for 800+ rocks

Requirements:

- 40+ fields per rock (chemical formula, description, ...)
- Scientific accuracy from authoritative sources

Why agents? Traditional scraping fails due to:

- Inconsistent website structures
- Missing data fields
- Need for synthesis across sources
- Quality validation required



Core Identification

Also Known As: California Iris · Lithia Emerald · Spodumene

Chemical Formula: $LiAlSi_2O_6$

Physical Properties

Crystal System: Monoclinic

Color: Pink · Violet · Purplish-pink · Colorless · Greenish

Hardness: 6.5 to 7

Density: 3.1 to 3.2

Luster: Vitreous

Streak: White

Fracture: Uneven

Cleavage: Perfect on {110}, good on {010}

Pipeline Overview

3-Phase Architecture (independent, resumable):

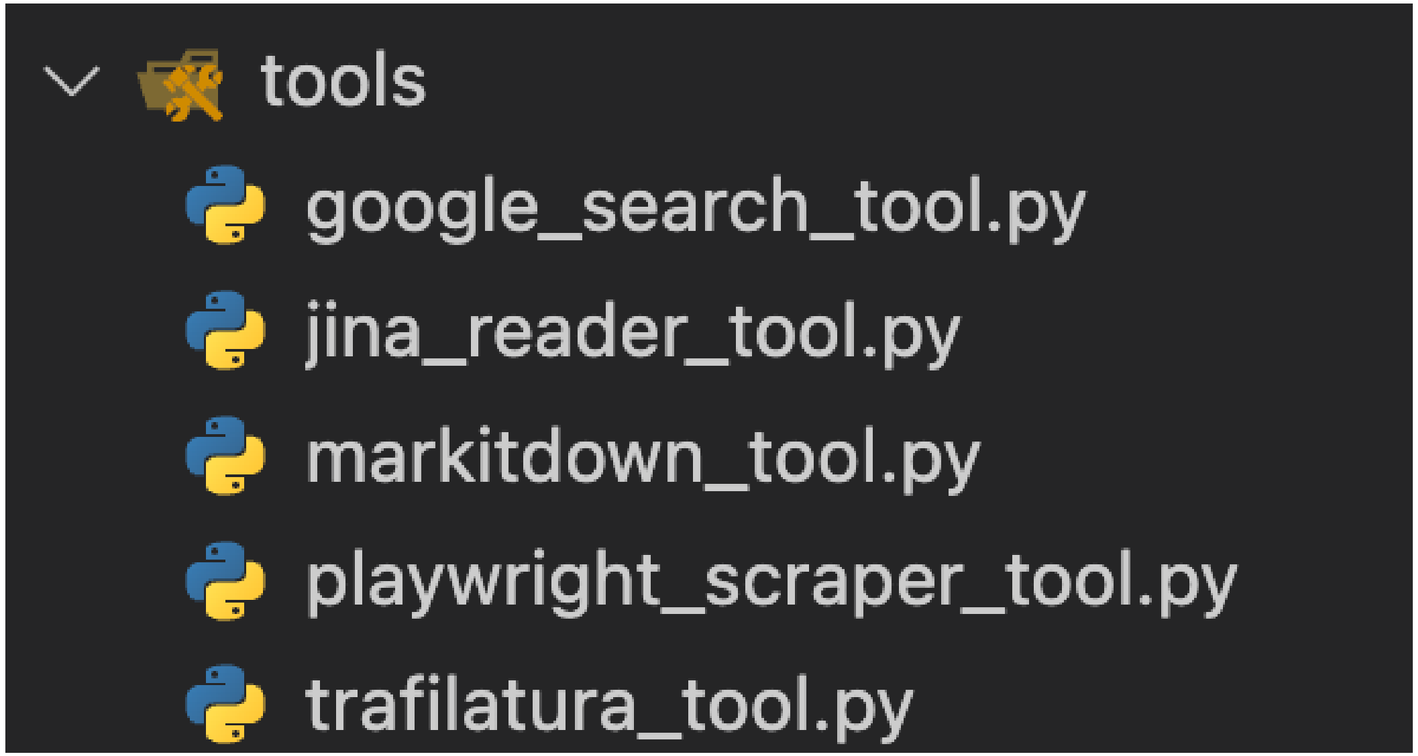
```
Phase 1: Link Collection
  Input:  Rock name
  Agent:  Planner + Link Collector
  Output: 8–10 URLs
  Status: NOT_STARTED → LINKS_COLLECTED

Phase 2: Content Scraping
  Input:  URLs from Phase 1
  Tool:   Playwright + Trafilatura
  Output: Clean markdown content
  Status: LINKS_COLLECTED → CONTENT_SCRAPED

Phase 3: Research Generation
  Input:  Scraped content
  Agent:  Researcher
  Output: 40+ structured fields
  Status: CONTENT_SCRAPED → RESEARCH_COMPLETED
```

4 Specialized Agents (like a research team):

```
Planner → Link Collector → Web Scraper → Researcher
   ↓             ↓                ↓             ↓
Strategy     Find URLs     Get Content   Synthesize
```

1. **Planner**: Generates targeted search queries
2. **Link Collector**: Finds authoritative sources (Wikipedia, Mindat, USGS)
3. **Web Scraper**: Extracts clean content from varied site structures
4. **Researcher**: Synthesizes 40+ fields from multiple sources

Task-specific tools



Demo

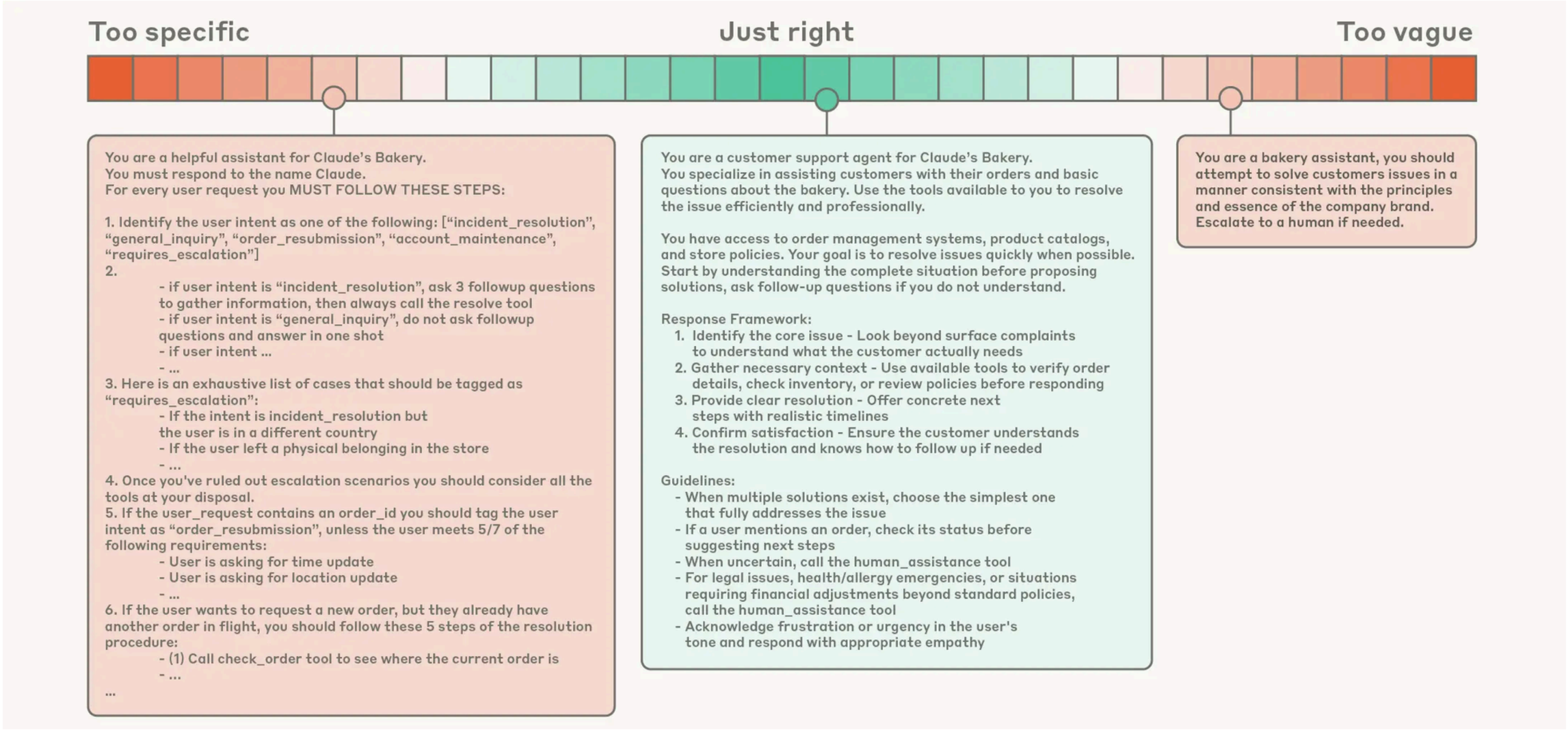
Best practices

Best practices: Phase isolation

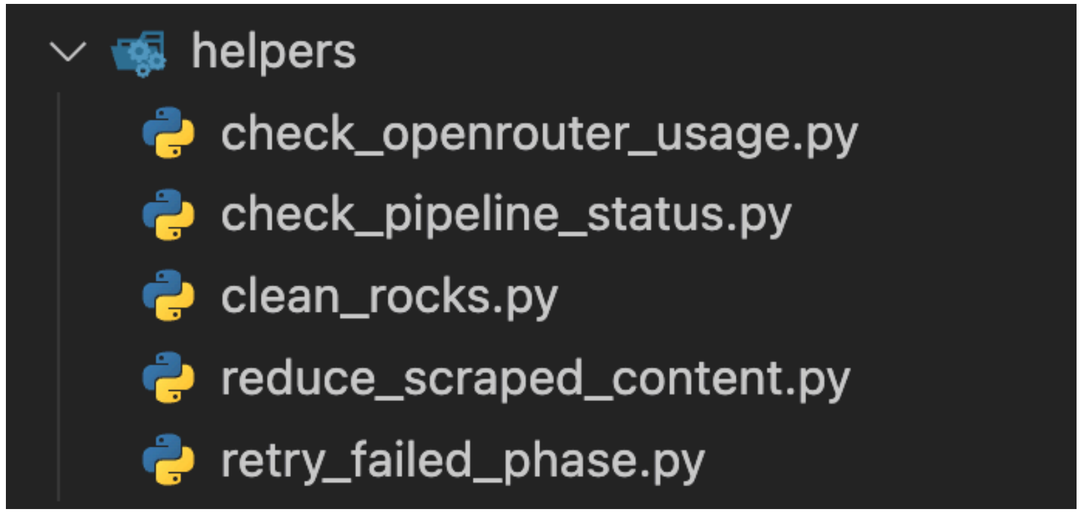
```
# Phase 2 fails, but Phase 1 data preserved
{
    "pipeline_status": "SCRAPING_FAILED",
    "collected_links": [...],  # Still here
    "scraping_error": "Timeout",
    "scraped_content": []
}
```

```python
class PipelineStatus(str, Enum):
    NOT_STARTED = "NOT_STARTED"
    LINKS_COLLECTED = "LINKS_COLLECTED"
    CONTENT_SCRAPED = "CONTENT_SCRAPED"
    RESEARCH_COMPLETED = "RESEARCH_COMPLETED"
    LINKS_COLLECTION_FAILED = "LINKS_COLLECTION_FAILED"
    SCRAPING_FAILED = "SCRAPING_FAILED"
    RESEARCH_FAILED = "RESEARCH_FAILED"
```

Best practices: Prompts



Best practices: Helpers



1. Status Checker (`check_pipeline_status.py`):

```
$ uv run helpers/check_pipeline_status.py --phase=scraping --execute
Total: 832 rocks
  ✓ Completed: 280 (33%)
  ✗ Failed: 17 (2%)

Failed: ID 45 (Timeout), ID 78 (403 Forbidden)...
```

2. Retry Script (`retry_failed_phase.py`):

```
$ uv run helpers/retry_failed_phase.py --phase=scraping --execute
# Resets failed rocks to retry specific phase
```

3. Cost Monitor (`check_openrouter_usage.py`):

```
$ uv run helpers/check_openrouter_usage.py
Usage: $8.47 / $50.00 limit
```

Additional section: Bad vs Good

❌ Wrong way

1. **Turning a few knobs at once** → Multiple changes simultaneously → Can't isolate root cause
2. **Do not debugging immediately** → Guessing blindly → Potential wasted time
3. **No hypothesis testing** → Making random fixes → Having no clear result
4. **Rushing** → Thinking of wasting time → Making more mistakes
5. **Manual fixes** → Fixing issues by hand → Unintentionally resetting db

Result: Spent entire day stuck, little progress

✅ Right way

1. **Changing ONE thing at a time** → Clear cause-effect → Quick iterations
2. **Using tools to debug (MLFlow)** → Inspecting actual inputs/outputs → Finding the bug
3. **Testing hypotheses systematically** → Is it scraper? Crawler? Blocked IP? → Identifying exact culprit
4. **Staying calm & confident** → Trusting own problem-solving skills → Clear thinking
5. **Automation with scripts** → Status checker, retry tool → Saving a lot of manual work

Result: Problem solved in morning after a couple of minutes

Conclusion

Results

Investments

- **40 hours** of development
- **$50** for OpenRouter API

Returns

- **830+** mineral records
- **40+ fields** per record
- **1 minute** generation time for new entries
- Content team freed from manual, repetitive tasks

Recap

Agents vs Scripts

Agentic Frameworks

Basic blocks

Case-study: Content generation

Isolation, Prompts, Helpers

Context/memory management

Optimizations

Links

1. https://docs.crewai.com/how-to/create-efficient-teams (CrewAI best practices)
2. https://tarekeesa7.medium.com/crew-ai-crash-course-step-by-step-c801f37220a5 (Crewai 5-min crash-course)
3. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents (Effective context engineering)
4. https://agenticai-learning.org/f25 (Agentic AI course, star lecturers, recently started)