Team

Laal Tamatar

Overall Round Progress

1	Basic RAG (late entry)
2	RAG with optimized retrieval ← local Milvus DB, t-e-3 large
3	Sanitization, optimized chunking ← MD introduced
4	added single tool calling, agentic
5	agentic interaction, using a cURL tool
6	more tools introduced <---- cURL v/s headless browser debate
7A	Agentic workflow with DOM snapshots v/s looped cURL approach
7B	Fixated to cURL + tools agentic workflow

Rounds 1-4 (prelims) overview

Extract Media

into MARKDOWN, sanitized

we tested 10+ extraction libraries!

Sort Text by size

character size >= 200K?
~50K tokens

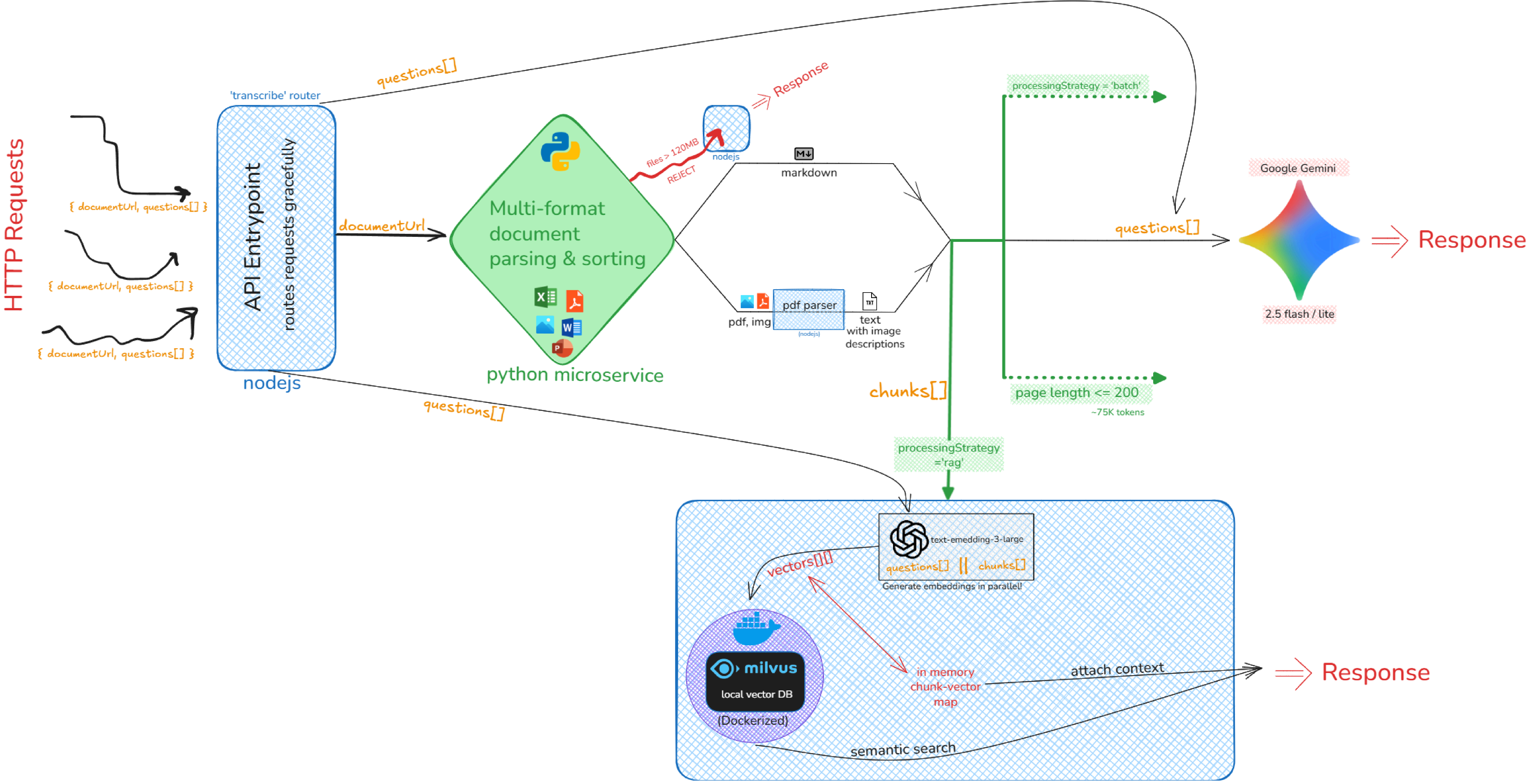
Smaller Media

Directly Passed to LLM

Larger Media

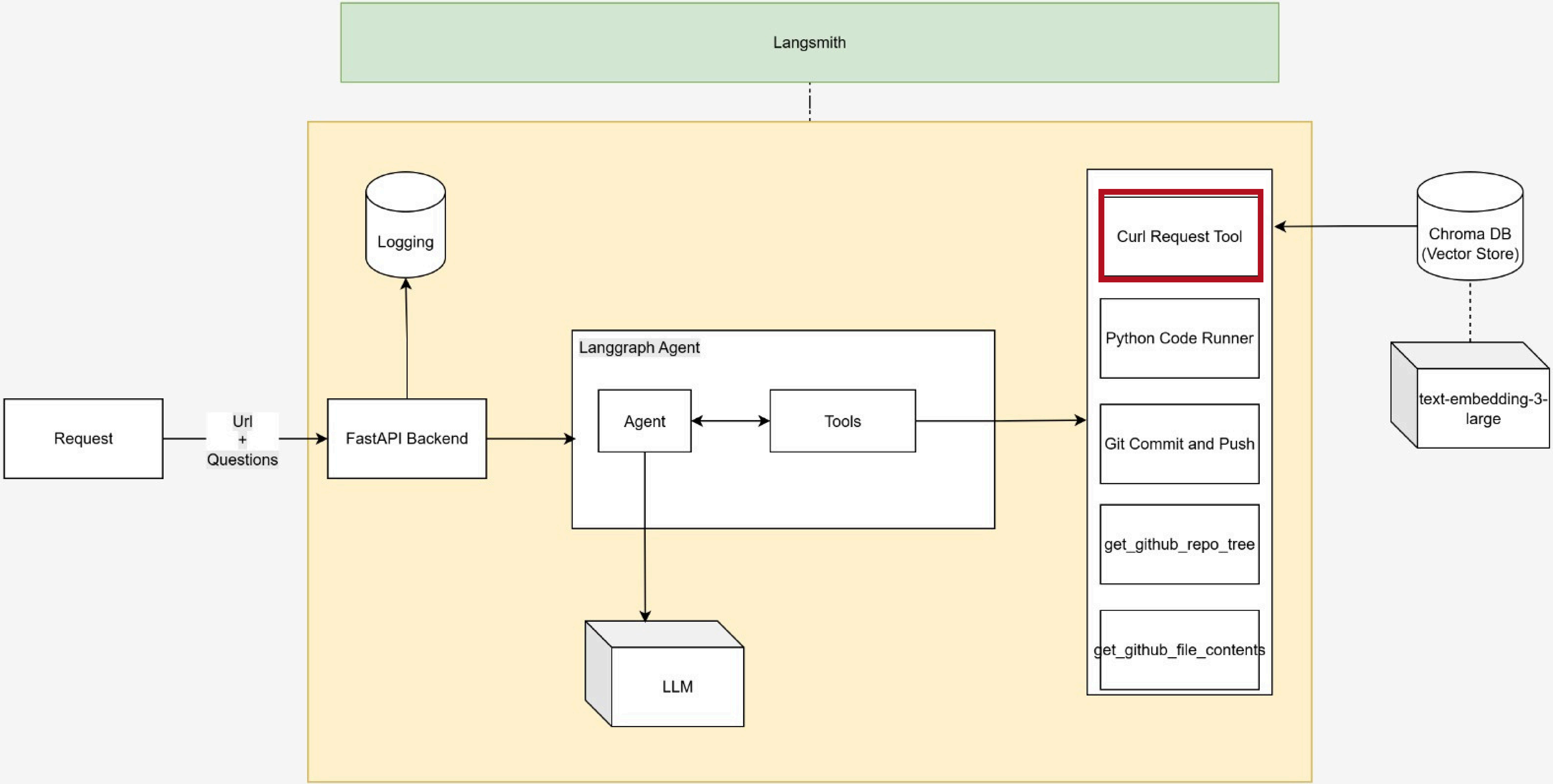
Through a comprehensive RAG model

→ parallel chunking
→ tables, media processed into markdown
→ images to description
-> DB collection caching based on first chunk's hash


HTTP Requests
{ documentUrl, questions[] }
{ documentUrl, questions[] }
{ documentUrl, questions[] }
'transcribe' router
API Entrypoint
routes requests gracefully
nodejs
questions[]
documentUrl
Multi-format document parsing & sorting
python microservice
files > 120MB
REJECT
nodejs
Response
markdown
pdf, img
pdf parser
(nodejs)
text with image descriptions
chunks[]
processingStrategy = 'batch'
questions[]
page length <= 200
~75K tokens
Google Gemini
2.5 flash / lite
Response
questions[]
processingStrategy ='rag'
text-emedding-3-large
questions[] || chunks[]
Generate embeddings in parallel!
vectors[][]
milvus
local vector DB
(Dockerized)
in memory chunk-vector map
attach context
semantic search
Response

Finale Approach

we had two!


Langsmith
Logging
Curl Request Tool
Chroma DB
(Vector Store)
Python Code Runner
Langgraph Agent
text-embedding-3-large
Request
Url
+
Questions
FastAPI Backend
Agent
Tools
Git Commit and Push
get_github_repo_tree
get_github_file_contents
LLM

The dilemma

the ‘simple’ approach

master cURL tool + util tools

- Works for both, server and client side rendering
- after the context limit approaches 50K tokens → switch to RAG
- **(chunks are created, stored in a vector DB, semantically searched and returned)**
- Faster
- pretty generic

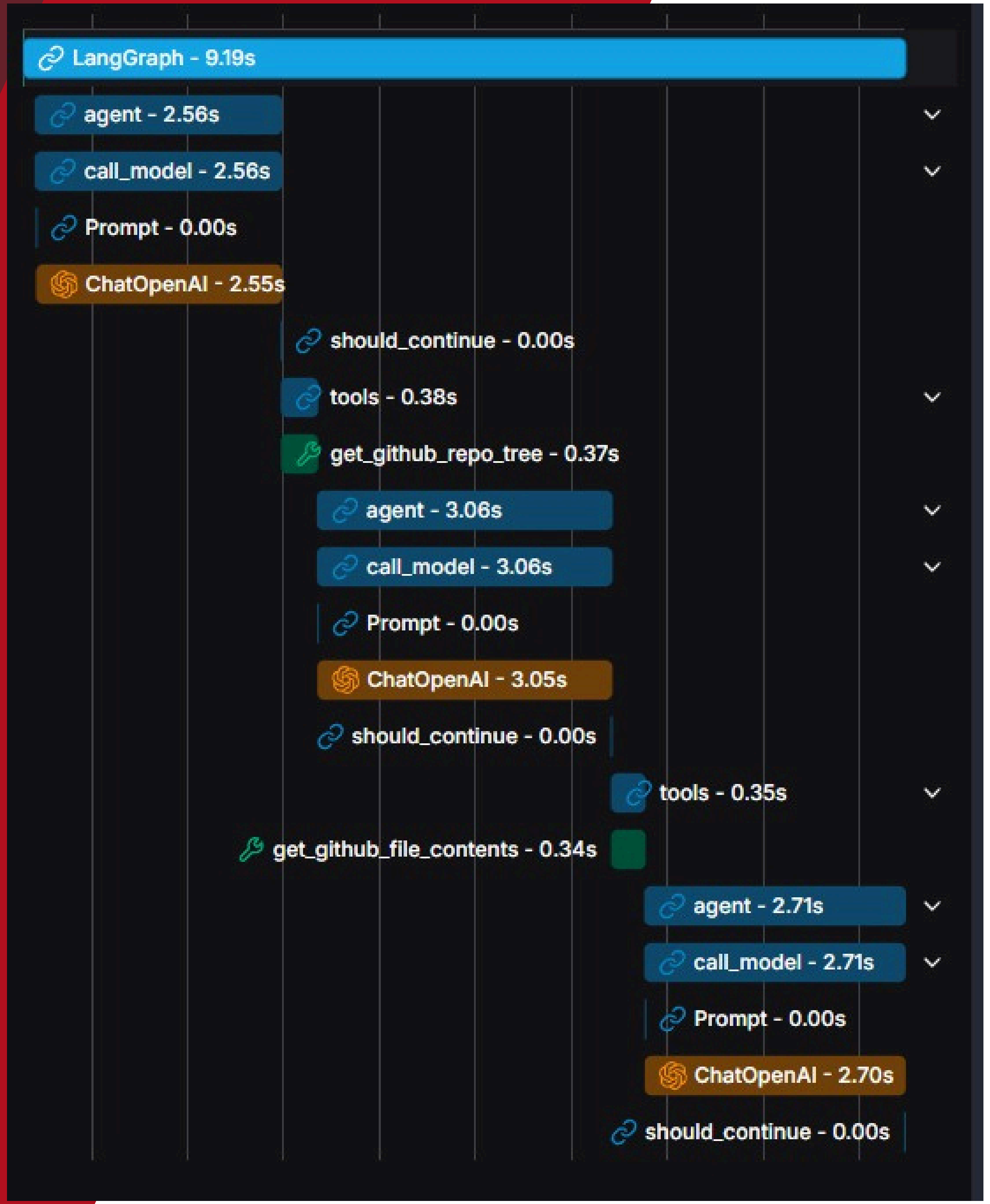
the ‘optimal’ approach

headless browser + DOM snapshots on change

- Headless browser + DOM snapshots on every change
- A tool for the agent to read sanitized DOM snapshots, and inject JS into the latest page.
- Memory overhead, less scalable
- cleaner

Let’s talk numbers

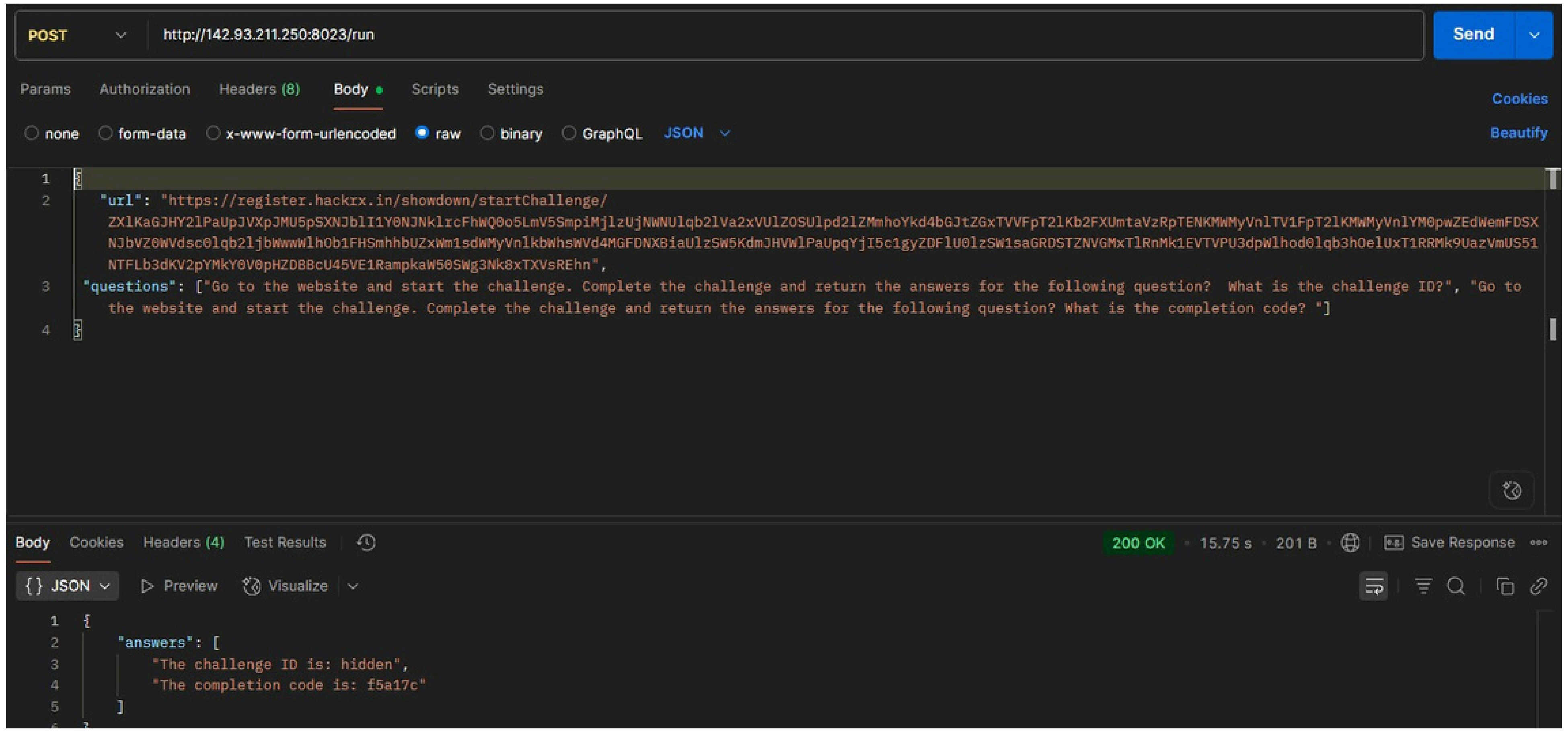
what worked out for us!



LangGraph - 9.19s
agent - 2.56s
call_model - 2.56s
Prompt - 0.00s
ChatOpenAI - 2.55s
should_continue - 0.00s
tools - 0.38s
get_github_repo_tree - 0.37s
agent - 3.06s
call_model - 3.06s
Prompt - 0.00s
ChatOpenAI - 3.05s
should_continue - 0.00s
tools - 0.35s
get_github_file_contents - 0.34s
agent - 2.71s
call_model - 2.71s
Prompt - 0.00s
ChatOpenAI - 2.70s
should_continue - 0.00s

9s

V/S



http://142.93.211.250:8023/run

```
{
  "url": "https://register.hackrx.in/showdown/startChallenge/
   ZXlKaGJHY2lPaUpJVXpJMU5pSXNJblI1Y0NJNklrcFhWQ0o5LmV5SmpiMjlzUjNWNUlqb2lVa2xVUlZOSUlpd2lZMmhoYkd4bGJtZGxTVVFpT2lKb2FXUmtaVzRpTENKMWMyVnlTV1FpT2lKMWMyVnlYM0pwZEdWemFDSX
   NJbVZ0WVdsc0lqb2ljbWwwWlhOb1FHSmhhbUZxWm1sdWMyVnlkbWhsWVd4MGFDNXBiaUlzSW5KdmJHVWlPaUpqYjI5c1gyZDFlU0lzSW1saGRDSTZNVGMxTlRnMk1EVTVPU3dpWlhod0lqb3hOelUxT1RRMk9UazVmUS51
   NTFLb3dKV2pYMkY0V0pHZDBBcU45VE1RampkaW50SWg3Nk8xTXVsREhn",
 "questions": ["Go to the website and start the challenge. Complete the challenge and return the answers for the following question?  What is the challenge ID?", "Go to
   the website and start the challenge. Complete the challenge and return the answers for the following question? What is the completion code? "]
}
```

200 OK • 15.75 s • 201 B

```
{
    "answers": [
        "The challenge ID is: hidden",
        "The completion code is: f5a17c"
    ]
```

15s

Thank you!