30-MIN SESSİON - SUI BASECAMP

SUI BASECAMP WORKSHOP

From binary to agents: a tour of how software gets built, and where AI takes it next.

BİAR

· OCTOBER 7-8, 2026

CONTENTS

BEFORE WE START

	QUESTİON	SHOW OF HANDS
01	Uses vibe-coding today	____
02	Knows a high-level language (Python, JS...)	____
03	Knows a low-level language (C, Assembly...)	____

PART 01

THE EVOLUTION OF SOFTWARE

BINARY

- Every computer, at bottom, only understands two states: on and off.
- Those states are written as 0 and 1 - bits, grouped into bytes.
- Every program that has ever run has been binary at some point.

LOW-LEVEL LANGUAGES

- Assembly and C let you write closer to what the machine actually does.
- You manage memory yourself - no safety net.
- Fast and precise, but every detail is your responsibility.

HIGH-LEVEL LANGUAGES

- Python, JavaScript, Java - languages that read closer to English than to machine code.
- The language handles memory, so you focus on logic.
- A trade: some speed and control, for a lot of speed writing it.

VIBE-CODING

- Describe what you want in plain language; the AI writes the code.
- The AI decides the syntax, the structure, even the language.
- Another layer of abstraction - the same evolution, one step further.

FOUR LAYERS, ONE EVOLUTION

1940S	1950S	1980S	TODAY
BINARY	LOW-LEVEL	HIGH-LEVEL	VIBE-CODING
TRANSISTORS, ON OR OFF	DIRECT MEMORY ACCESS	HUMAN-READABLE SYNTAX	NATURAL-LANGUAGE PROMPTS

Each layer hides the one below it. Vibe-coding doesn't replace the others - it sits on top of them, the way Python sits on top of C, and C sits on top of binary.

"Maybe in ten years, talking to an AI will feel just as outdated. Maybe we'll just think, and a product appears."

- where this evolution might go

PART 02

BUILDING AN AGENT

WHAT IS AN AGENT?

- A program that can reason, decide, and act - not just answer.
- It can call tools: search the web, run code, send a transaction.
- The loop: observe -> think -> act -> observe again.

CHOOSING YOUR STACK

OPEN-SOURCE, LOCAL

Run Qwen 2.5 (7B) or Hermes 3 (8B) through Ollama, 4-bit quantized (Q4_K_M GGUF) to fit in ~4.3–4.7 GB. Runs on Windows, Linux and macOS, with an 8 GB RAM floor.

HOSTED, VIA OPENROUTER

Call a model in the cloud through OpenRouter. No local compute needed - a good fallback when your machine isn't powerful enough.

RUNNING AN OPEN-SOURCE AGENT

Ollama runs the model locally, 4-bit quantized so it fits in ~4.3–4.7 GB of RAM - comfortable alongside Cursor, VS Code or Obsidian on an 8 GB floor. On Apple Silicon (M4), Metal acceleration gives 40–50+ tokens/sec.

```
$ ollama run qwen2.5:7b-instruct-q4_K_M
pulling manifest... done
$ ollama run hf.co/NousResearch/Hermes-3-Llama-3.1-8B-GGUF:Q4_K_M
>>> ready
```

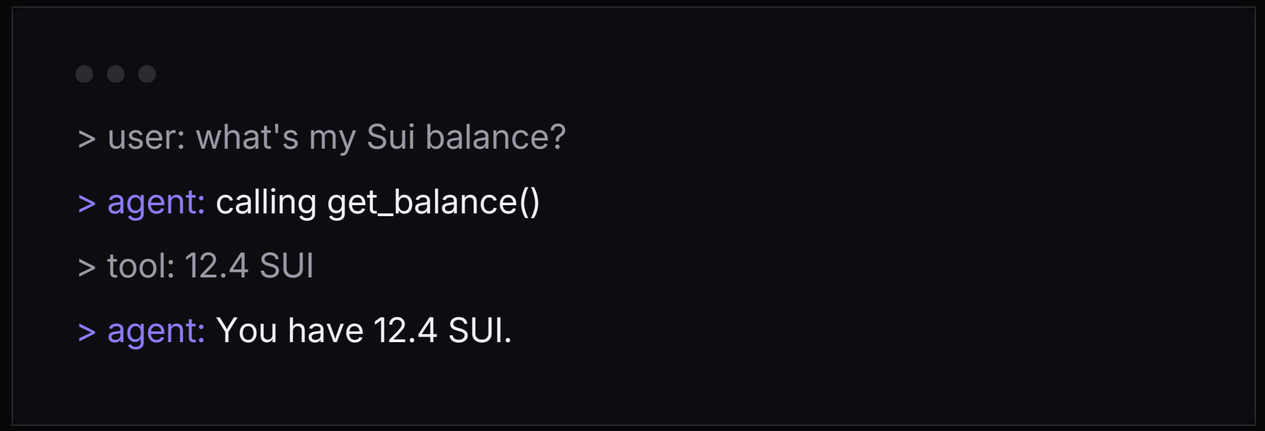
CALLING A MODEL IN THE CLOUD

No GPU, no problem. OpenRouter gives you API access to dozens of models, including free tiers - a fallback for lower-power machines.

```
$ export OPENROUTER_API_KEY=sk-...
$ agent.model = "openrouter/qwen-2.5-7b-instruct"
>>> connected
```

WATCHING THE AGENT THİNK

Every step is visible: what the agent decided, which tool it called, and what came back.



PART 03

AGENTIC WALLETS ON SUI

FROM REQUEST TO ON-CHAIN ACTION

YOU

A PLAIN-LANGUAGE REQUEST

AGENT

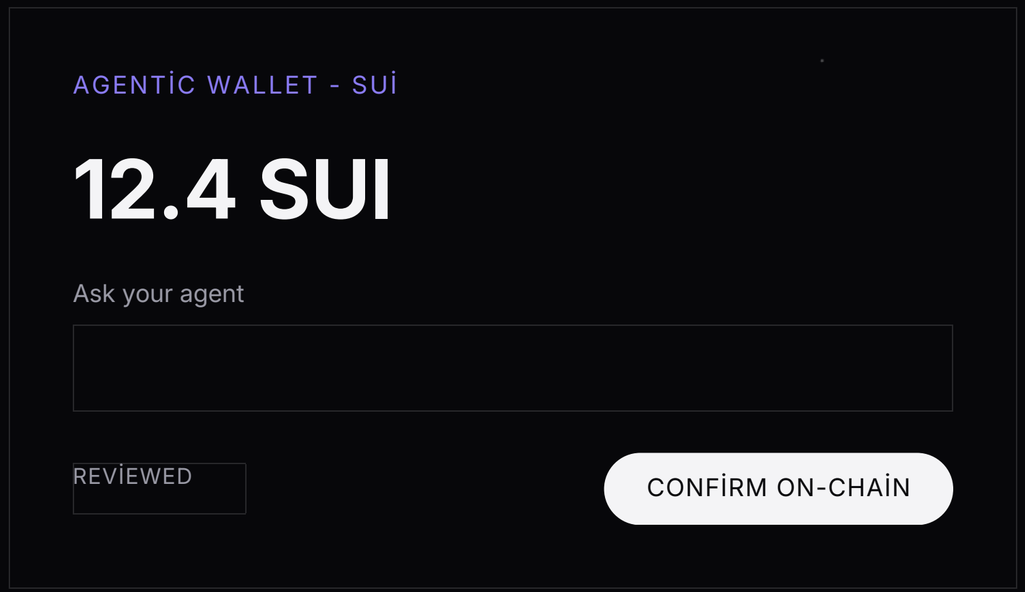
REASONS, DECIDES, CALLS TOOLS

WALLET ON SUI

SIGNS AND EXECUTES ON-CHAIN

The agent reasons in natural language, but every action it takes is a real, signed transaction on Sui.

WHAT WE'RE BUILDING



The agent proposes the transaction in plain language; you stay the one who signs.

THANK YOU

Questions? Let's build.