PyVenice

Coding assistants Open Source with custom Agents AI

Alessandra Bilardi - Data & Automation Specialist @ Corley Cloud

@PyVenice #1 #Meetup #PythonItalia #Python

agenda

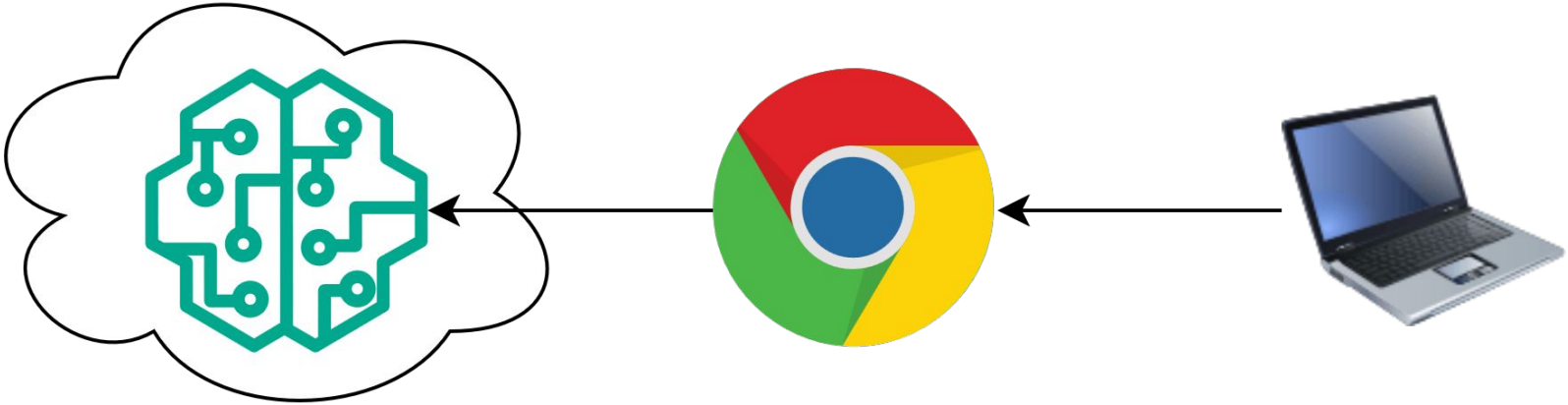
Goal

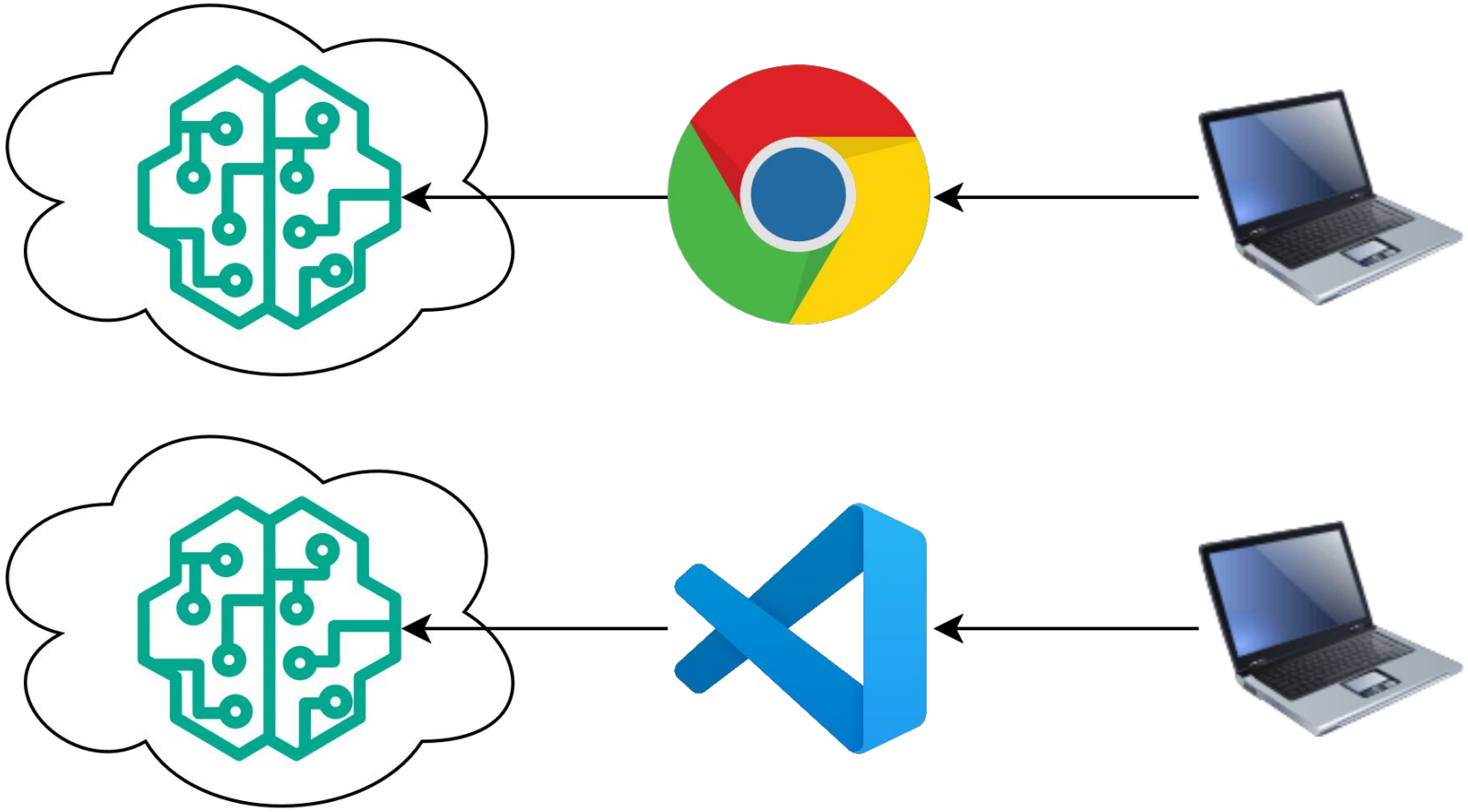
Tools

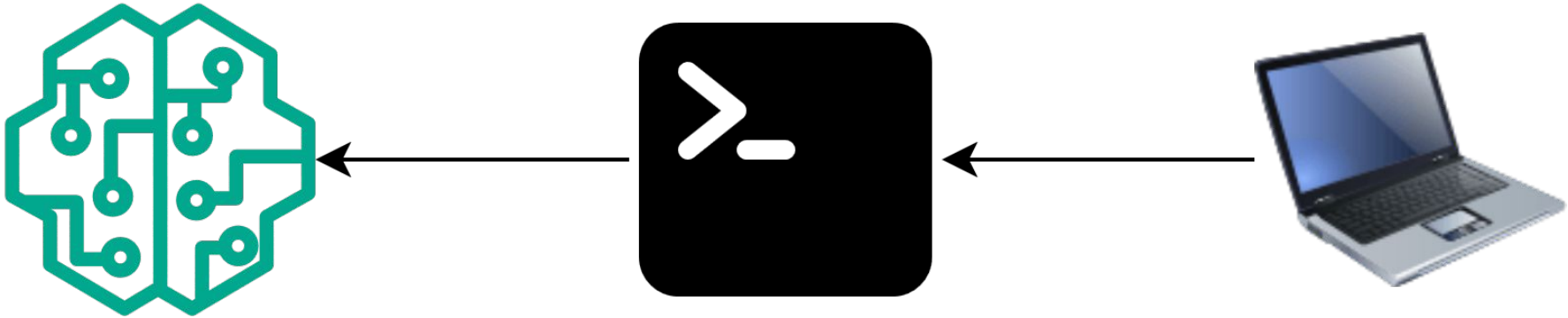
Challenge

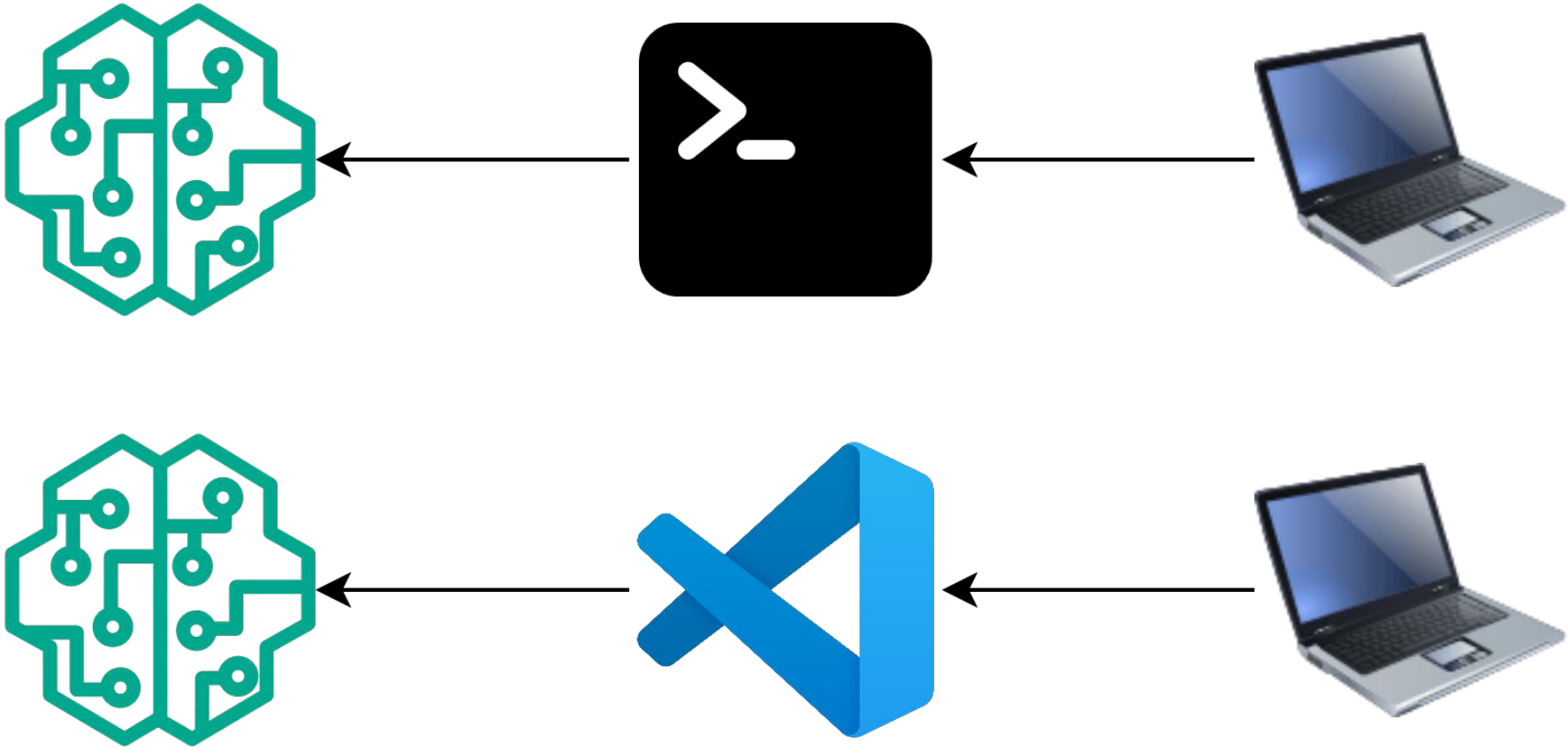
Coding

Goal



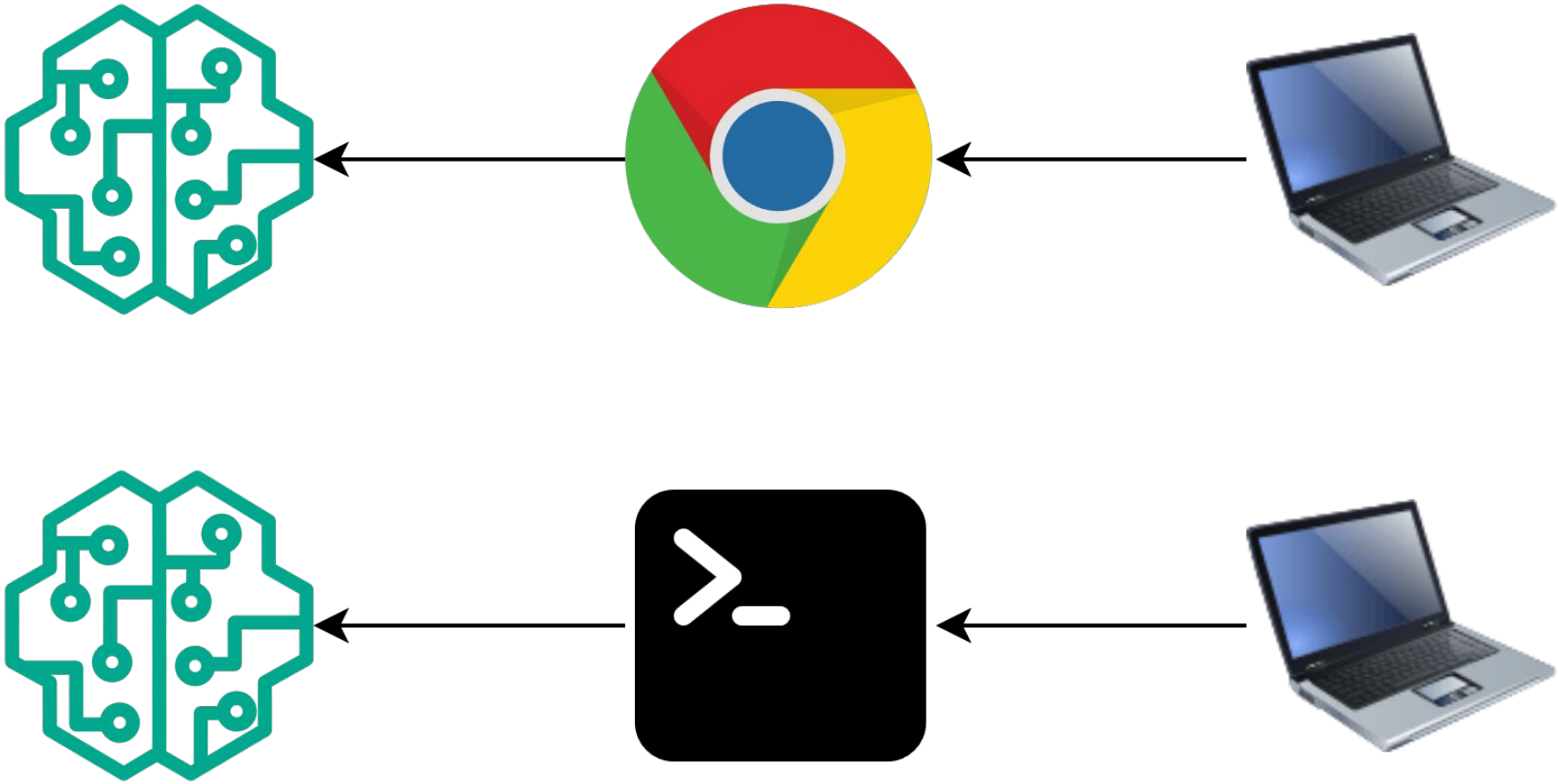






PyVenice #0

- Ollama
- LM Studio
- WASM
- any-agent



Goal

- Always add system prompt

Goal

- Always add system prompt

Sei un assistente esperto di programmazione.

Rispondi sempre in italiano con spiegazioni dettagliate.

Per tutti gli snippet di codice:

- Usa nomi di variabili e funzioni in inglese.
- Usa commenti in inglese.
- Per ogni funzione, includi una **docstring di una riga** seguendo le convenzioni di PEP257.
- Annota tutti gli argomenti delle funzioni e i tipi di ritorno usando le **type hints di Python** (PEP526)

Goal

- Always add system prompt
- Callable via API

Goal

- Always add system prompt
- Callable via API
- OpenAI compatible

Goal

- Always add system prompt
- Callable via API
- OpenAI compatible
- RAG

Goal

- Always add system prompt
- Callable via API
- OpenAI compatible
- RAG
- Online search

Tools

Tools

- Ollama

Tools

- Ollama
- Continue.dev

Tools

- Ollama
- Continue.dev
  - Kilo Code

Tools

- Ollama
- Continue.dev
  - Kilo Code
- FastAPI

Tools

- Ollama
- Continue.dev
  - Kilo Code
- FastAPI
- ChromaDB

Tools

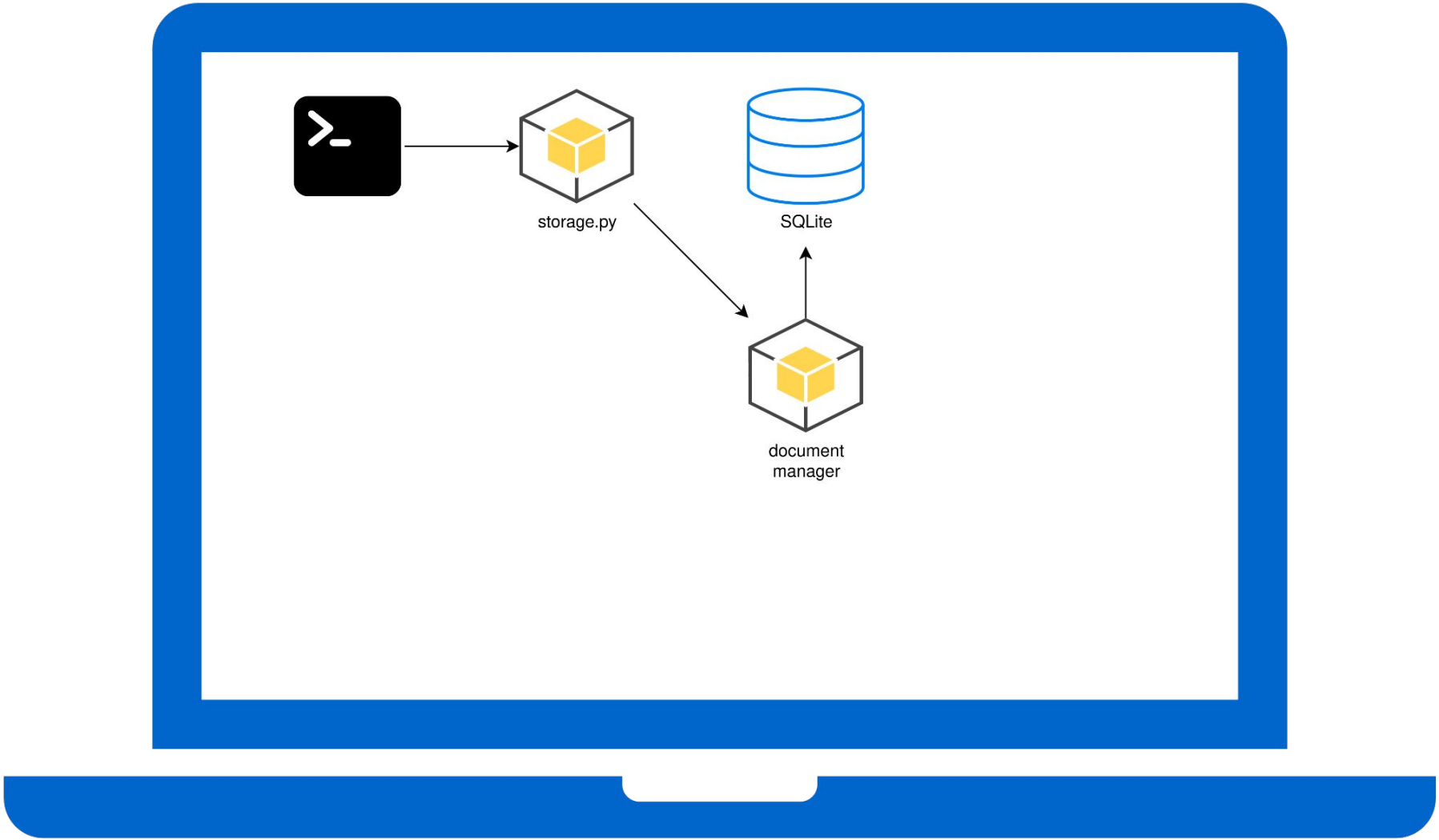
- Ollama
- Continue.dev
  - Kilo Code
- FastAPI
- ChromaDB
  - BeautifulSoup
  - pdfplumbler

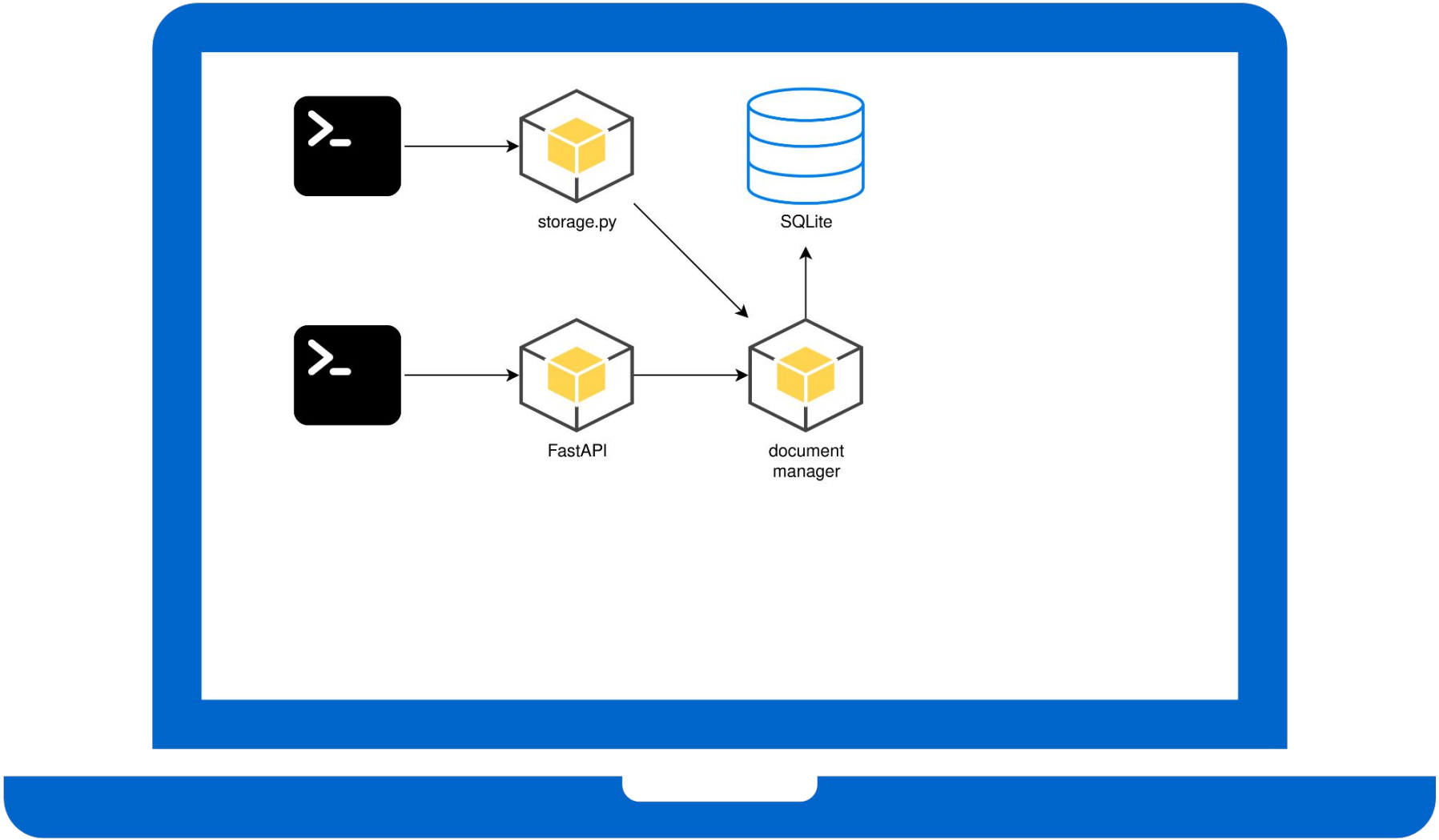
Tools

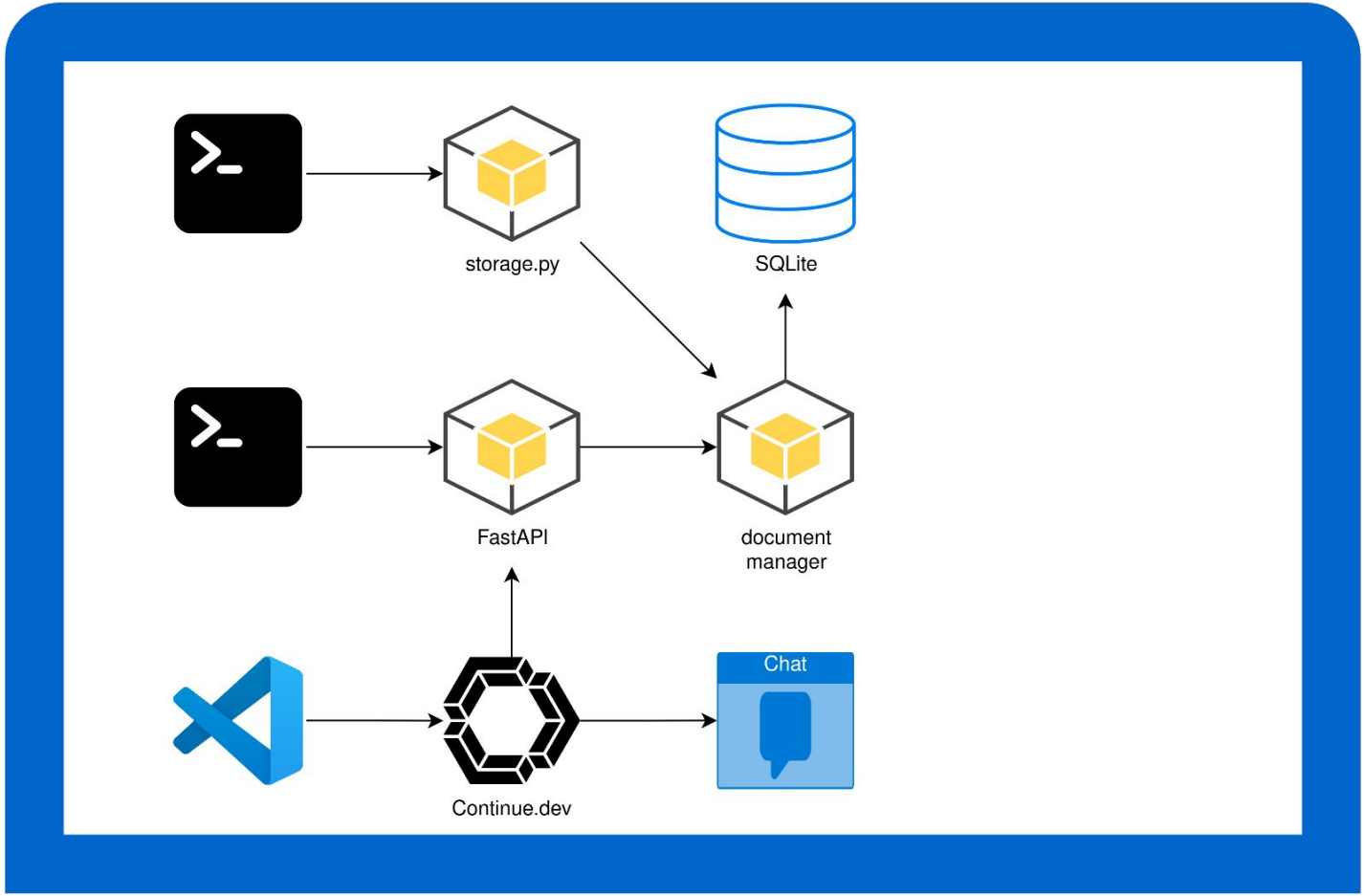
- Ollama
- Continue.dev
  - Kilo Code
- FastAPI
- ChromaDB
  - BeautifulSoup
  - pdfplumbler
- any-agent

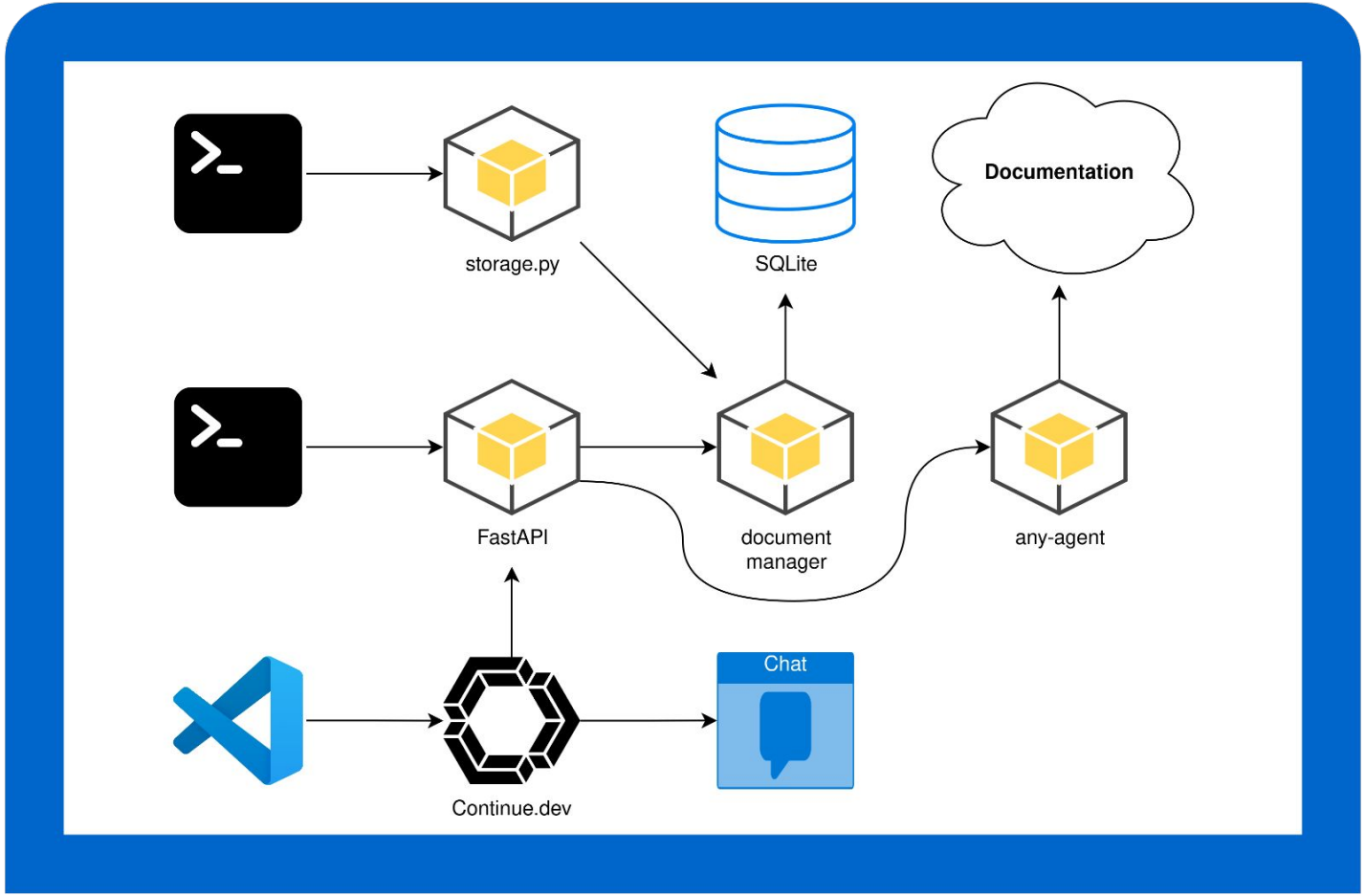
Tools

- Ollama
- Continue.dev
  - Kilo Code
- FastAPI
- ChromaDB
  - BeautifulSoup
  - pdfplumbler
- any-agent
  - any-agent-wrapper


storage.py
SQLite
document
manager


storage.py
SQLite
FastAPI
document
manager


storage.py
SQLite
FastAPI
document
manager
Chat
Continue.dev


storage.py
SQLite
Documentation
FastAPI
document
manager
any-agent
Chat
Continue.dev

Challenge

Challenge

2025 > 10 > ollama > Modelfile

```
FROM qwen3
PARAMETER num_ctx 40000
SYSTEM """
Sei un assistente esperto di programmazione.
Rispondi sempre in italiano con spiegazioni dettagliate.
Per tutti gli snippet di codice:
  - Usa nomi di variabili e funzioni in inglese.
  - Usa commenti in inglese.
  - Per ogni funzione, includi una **docstring di una riga** seguendo le convenzioni di PEP257.
  - Annota tutti gli argomenti delle funzioni e i tipi di ritorno usando le **type hints di Python** (PEP526).
"""
```

Challenge

2025 > 10 > ollama > Modelfile

```
1  FROM qwen3
2  PARAMETER num_ctx 40000
3  SYSTEM """
4  Sei un assistente esperto di programmazione.
5  Rispondi sempre in italiano con spiegazioni dett
6  Per tutti gli snippet di codice:
7    - Usa nomi di variabili e funzioni in inglese.
8    - Usa commenti in inglese.
9    - Per ogni funzione, includi una **docstring d
10   - Annota tutti gli argomenti delle funzioni e
11 """
```

ollama

- to download a model

```
ollama pull qwen3
```

- to create a model with your `system_prompt`

```
ollama create coding -f ollama/Modelfile
```

- to test the model

```
ollama run qwen3
>>> Who wrote The Betrothed?
Alessandro Manzoni
```

Challenge

2025 > 10 > ollama > Modelfile

```
FROM qwen3
PARAMETER num_ctx 40000
SYSTEM """
Sei un assistente esperto di programmazione.
Rispondi sempre in italiano con spiegazioni dett
Per tutti gli snippet di codice:
  - Usa nomi di variabili e funzioni in inglese.
  - Usa commenti in inglese.
  - Per ogni funzione, includi una **docstring d
  - Annota tutti gli argomenti delle funzioni e
"""
```

ollama

- to download a model

```
ollama pull qwen3
```

- to create a model with your `system_prompt`

```
ollama create coding -f ollama/Modelfile
```

- to test the model

```
ollama run qwen3
>>> Who wrote The Betrothed?
Alessandro Manzoni
```

Challenge

2025 > 10 > ollama > Modelfile

```
1  FROM qwen3
2  PARAMETER num_ctx 40000
3  SYSTEM """
4  Sei un assistente esperto di programmazione.
5  Rispondi sempre in italiano con spiegazioni dett
6  Per tutti gli snippet di codice:
7    - Usa nomi di variabili e funzioni in inglese.
8    - Usa commenti in inglese.
9    - Per ogni funzione, includi una **docstring d
10   - Annota tutti gli argomenti delle funzioni e
11 """
```

ollama

- to download a model

```
ollama pull qwen3
```

- to create a model with your `system_prompt`

```
ollama create coding -f ollama/Modelfile
```

- to test the model

```
ollama run qwen3
>>> Who wrote The Betrothed?
Alessandro Manzoni
```

Challenge

2025 > 10 > ollama > Modelfile

```
1  FROM qwen3
2  PARAMETER num_ctx 40000
3  SYSTEM """
4  Sei un assistente esperto di programmazione.
5  Rispondi sempre in italiano con spiegazioni dett
6  Per tutti gli snippet di codice:
7    - Usa nomi di variabili e funzioni in inglese.
8    - Usa commenti in inglese.
9    - Per ogni funzione, includi una **docstring d
10   - Annota tutti gli argomenti delle funzioni e
11 """
```

ollama

- to download a model

```
ollama pull qwen3
```

- to create a model with your `system_prompt`

```
ollama create coding -f ollama/Modelfile
```

- to test the model

```
ollama run qwen3
>>> Who wrote The Betrothed?
Alessandro Manzoni
```

Challenge

2025 > 10 > ollama > Modelfile

```
1  FROM qwen3
2  PARAMETER num_ctx 40000
3  SYSTEM """
4  Sei un assistente esperto di programmazione.
5  Rispondi sempre in italiano con spiegazioni dett
6  Per tutti gli snippet di codice:
7    - Usa nomi di variabili e funzioni in inglese.
8    - Usa commenti in inglese.
9    - Per ogni funzione, includi una **docstring d
10   - Annota tutti gli argomenti delle funzioni e
11 """
```

ollama

- to download a model

```
ollama pull qwen3
```

- to create a model with your `system_prompt`

```
ollama create coding -f ollama/Modelfile
```

- to test the model

```
ollama run qwen3
>>> Who wrote The Betrothed?
Alessandro Manzoni
```

Challenge

2025 > 10 > ollama > Modelfile

```
1  FROM qwen3
2  PARAMETER num_ctx 40000
3  SYSTEM """
4  Sei un assistente esperto di programmazione.
5  Rispondi sempre in italiano con spiegazioni dett
6  Per tutti gli snippet di codice:
7    - Usa nomi di variabili e funzioni in inglese.
8    - Usa commenti in inglese.
9    - Per ogni funzione, includi una **docstring d
10   - Annota tutti gli argomenti delle funzioni e
11 """
```

ollama

- to download a model

```
ollama pull qwen3
```

- to create a model with your `system_prompt`

```
ollama create coding -f ollama/Modelfile
```

- to test the model

```
ollama run ~~qwen3~~ coding
>>> Who wrote The Betrothed?
Alessandro Manzoni
```

Challenge

2025 > 10 > ollama > Modelfile

```
1  FROM qwen3
2  PARAMETER num_ctx 40000
3  SYSTEM """
4  Sei un assistente esperto di programmazione.
5  Rispondi sempre in italiano con spiegazioni dett
6  Per tutti gli snippet di codice:
7    - Usa nomi di variabili e funzioni in inglese.
8    - Usa commenti in inglese.
9    - Per ogni funzione, includi una **docstring d
10   - Annota tutti gli argomenti delle funzioni e
11 """
```

ollama

- to download a model

```
ollama pull qwen3
```

- to create a model with your `system_prompt`

```
ollama create coding -f ollama/Modelfile
```

- to test the model

```
ollama run ~~qwen3~~ coding
>>> Who wrote The Betrothed?
Alessandro Manzoni
```

Coding

```python
import requests
import os

MODEL = os.environ.get("MODEL", "qwen3")
OLLAMA_URL = os.environ.get("OLLAMA_URL", "http://localhost:11434")

url = f"{OLLAMA_URL}/api/chat"
print(url)
payload = {
  "model": MODEL,
  "messages": [
    {"role": "system", "content": """
Sei un assistente esperto di programmazione.
Rispondi sempre in italiano con spiegazioni dettagliate.
Per tutti gli snippet di codice:
  - Usa nomi di variabili e funzioni in inglese.
  - Usa commenti in inglese.
  - Per ogni funzione, includi una **docstring di una riga** seguendo le convenzioni di PEP257.
  - Annota tutti gli argomenti delle funzioni e i tipi di ritorno usando le **type hints di Python** (PEP526).
    """},
    {"role": "user", "content": "/dask forniscimi uno snippet per parallelizzare un groupby"}
  ],
  "stream": False
}

res = requests.post(url, json=payload, stream=True)
for line in res.iter_lines():
    if line:
        print(line.decode("utf-8"))

```

```python
import requests
import os

MODEL = os.environ.get("MODEL", "qwen3")
OLLAMA_URL = os.environ.get("OLLAMA_URL", "http://localhost:11434")

url = f"{OLLAMA_URL}/api/chat"
print(url)
payload = {
  "model": MODEL,
  "messages": [
    {"role": "system", "content": """
Sei un assistente esperto di programmazione.
Rispondi sempre in italiano con spiegazioni dettagliate.
Per tutti gli snippet di codice:
  - Usa nomi di variabili e funzioni in inglese.
  - Usa commenti in inglese.
  - Per ogni funzione, includi una **docstring di una riga** seguendo le convenzioni di PEP257.
  - Annota tutti gli argomenti delle funzioni e i tipi di ritorno usando le **type hints di Python** (PEP526).
    """},
    {"role": "user", "content": "/dask forniscimi uno snippet per parallelizzare un groupby"}
  ],
  "stream": False
}

res = requests.post(url, json=payload, stream=True)
for line in res.iter_lines():
    if line:
        print(line.decode("utf-8"))

```

```python
import requests
import os

MODEL = os.environ.get("MODEL", "qwen3")
OLLAMA_URL = os.environ.get("OLLAMA_URL", "http://localhost:11434")

url = f"{OLLAMA_URL}/api/chat"
print(url)
payload = {
  "model": MODEL,
  "messages": [
    {"role": "system", "content": """
Sei un assistente esperto di programmazione.
Rispondi sempre in italiano con spiegazioni dettagliate.
Per tutti gli snippet di codice:
  - Usa nomi di variabili e funzioni in inglese.
  - Usa commenti in inglese.
  - Per ogni funzione, includi una **docstring di una riga** seguendo le convenzioni di PEP257.
  - Annota tutti gli argomenti delle funzioni e i tipi di ritorno usando le **type hints di Python** (PEP526).
    """},
    {"role": "user", "content": "/dask forniscimi uno snippet per parallelizzare un groupby"}
  ],
  "stream": False
}

res = requests.post(url, json=payload, stream=True)
for line in res.iter_lines():
    if line:
        print(line.decode("utf-8"))

```

```python
import requests
import os

MODEL = os.environ.get("MODEL", "qwen3")
OLLAMA_URL = os.environ.get("OLLAMA_URL", "http://localhost:11434")

url = f"{OLLAMA_URL}/api/chat"
print(url)
payload = {
  "model": MODEL,
  "messages": [
    {"role": "system", "content": """
Sei un assistente esperto di programmazione.
Rispondi sempre in italiano con spiegazioni dettagliate.
Per tutti gli snippet di codice:
  - Usa nomi di variabili e funzioni in inglese.
  - Usa commenti in inglese.
  - Per ogni funzione, includi una **docstring di una riga** seguendo le convenzioni di PEP257.
  - Annota tutti gli argomenti delle funzioni e i tipi di ritorno usando le **type hints di Python** (PEP526).
    """},
    {"role": "user", "content": "/dask forniscimi uno snippet per parallelizzare un groupby"}
  ],
  "stream": False
}

res = requests.post(url, json=payload, stream=True)
for line in res.iter_lines():
    if line:
        print(line.decode("utf-8"))

```

```python
import requests
import os

MODEL = os.environ.get("MODEL", "qwen3")
OLLAMA_URL = os.environ.get("OLLAMA_URL", "http://localhost:11434")

url = f"{OLLAMA_URL}/api/chat"
print(url)
payload = {
  "model": MODEL,
  "messages": [
    {"role": "system", "content": """
Sei un assistente esperto di programmazione.
Rispondi sempre in italiano con spiegazioni dettagliate.
Per tutti gli snippet di codice:
  - Usa nomi di variabili e funzioni in inglese.
  - Usa commenti in inglese.
  - Per ogni funzione, includi una **docstring di una riga** seguendo le convenzioni di PEP257.
  - Annota tutti gli argomenti delle funzioni e i tipi di ritorno usando le **type hints di Python** (PEP526).
    """},
    {"role": "user", "content": "/dask forniscimi uno snippet per parallelizzare un groupby"}
  ],
  "stream": False
}

res = requests.post(url, json=payload, stream=True)
for line in res.iter_lines():
    if line:
        print(line.decode("utf-8"))

```

```python
import requests
import os

MODEL = os.environ.get("MODEL", "qwen3")
OLLAMA_URL = os.environ.get("OLLAMA_URL", "http://localhost:11434")

url = f"{OLLAMA_URL}/api/chat"
print(url)
payload = {
  "model": MODEL,
  "messages": [
    {"role": "system", "content": """
Sei un assistente esperto di programmazione.
Rispondi sempre in italiano con spiegazioni dettagliate.
Per tutti gli snippet di codice:
  - Usa nomi di variabili e funzioni in inglese.
  - Usa commenti in inglese.
  - Per ogni funzione, includi una **docstring di una riga** seguendo le convenzioni di PEP257.
  - Annota tutti gli argomenti delle funzioni e i tipi di ritorno usando le **type hints di Python** (PEP526).
    """},
    {"role": "user", "content": "/dask forniscimi uno snippet per parallelizzare un groupby"}
  ],
  "stream": False
}

res = requests.post(url, json=payload, stream=True)
for line in res.iter_lines():
    if line:
        print(line.decode("utf-8"))

```

```python
@app.post("/api/chat")
async def chat(request: Request) -> Response:
    """Simulate namesake Ollama API"""
    body = await request.json()
    messages = body.get("messages", [])
    last_message = messages[-1]["content"] if messages else ""

    # Routing for calling the Python agents directly
    content = call_agents_by_router(last_message)
    if content is not None:
        response = {"message":{"role":"assistant","content":content},"done": False}
        response = json.dumps(response) + '''
        {"message":{"role":"assistant","content":""},"done":true}
        '''
        return Response(
            content=response.encode('utf-8'),
            status_code=200
        )

    response = requests.post(f"{OLLAMA_URL}/api/chat", json=body)
    return Response(
            content=response.content,
            status_code=response.status_code,
            headers={"Content-Type": "application/json"}
    )

```

```python
@app.post("/api/chat")
async def chat(request: Request) -> Response:
    """Simulate namesake Ollama API"""
    body = await request.json()
    messages = body.get("messages", [])
    last_message = messages[-1]["content"] if messages else ""

    # Routing for calling the Python agents directly
    content = call_agents_by_router(last_message)
    if content is not None:
        response = {"message":{"role":"assistant","content":content},"done": False}
        response = json.dumps(response) + '''
        {"message":{"role":"assistant","content":""},"done":true}
        '''
        return Response(
            content=response.encode('utf-8'),
            status_code=200
        )

    response = requests.post(f"{OLLAMA_URL}/api/chat", json=body)
    return Response(
            content=response.content,
            status_code=response.status_code,
            headers={"Content-Type": "application/json"}
    )

```

```python
@app.post("/api/chat")
async def chat(request: Request) -> Response:
    """Simulate namesake Ollama API"""
    body = await request.json()
    messages = body.get("messages", [])
    last_message = messages[-1]["content"] if messages else ""

    # Routing for calling the Python agents directly
    content = call_agents_by_router(last_message)
    if content is not None:
        response = {"message":{"role":"assistant","content":content},"done": False}
        response = json.dumps(response) + '''
        {"message":{"role":"assistant","content":""},"done":true}
        '''
        return Response(
            content=response.encode('utf-8'),
            status_code=200
        )

    response = requests.post(f"{OLLAMA_URL}/api/chat", json=body)
    return Response(
            content=response.content,
            status_code=response.status_code,
            headers={"Content-Type": "application/json"}
    )

```

```python
@app.post("/api/chat")
async def chat(request: Request) -> Response:
    """Simulate namesake Ollama API"""
    body = await request.json()
    messages = body.get("messages", [])
    last_message = messages[-1]["content"] if messages else ""

    # Routing for calling the Python agents directly
    content = call_agents_by_router(last_message)
    if content is not None:
        response = {"message":{"role":"assistant","content":content},"done": False}
        response = json.dumps(response) + '''
        {"message":{"role":"assistant","content":""},"done":true}
        '''
        return Response(
            content=response.encode('utf-8'),
            status_code=200
        )

    response = requests.post(f"{OLLAMA_URL}/api/chat", json=body)
    return Response(
            content=response.content,
            status_code=response.status_code,
            headers={"Content-Type": "application/json"}
    )

```

```python

@app.post("/api/chat")
async def chat(request: Request) -> Response:
    """Simulate namesake Ollama API"""
    body = await request.json()
    messages = body.get("messages", [])
    last_message = messages[-1]["content"] if messages else ""

    # Routing for calling the Python agents directly
    content = call_agents_by_router(last_message)
    if content is not None:
        response = {"message":{"role":"assistant","content":content},"done": False}
        response = json.dumps(response) + '''
        {"message":{"role":"assistant","content":""},"done":true}
        '''
        return Response(
            content=response.encode('utf-8'),
            status_code=200
        )

    response = requests.post(f"{OLLAMA_URL}/api/chat", json=body)
    return Response(
            content=response.content,
            status_code=response.status_code,
            headers={"Content-Type": "application/json"}
    )

```

```python
@app.post("/api/chat")
async def chat(request: Request) -> Response:
    """Simulate namesake Ollama API"""
    body = await request.json()
    messages = body.get("messages", [])
    last_message = messages[-1]["content"] if messages else ""

    # Routing for calling the Python agents directly
    content = call_agents_by_router(last_message)
    if content is not None:
        response = {"message":{"role":"assistant","content":content},"done": False}
        response = json.dumps(response) + '''
        {"message":{"role":"assistant","content":""},"done":true}
        '''
        return Response(
            content=response.encode('utf-8'),
            status_code=200
        )

    response = requests.post(f"{OLLAMA_URL}/api/chat", json=body)
    return Response(
            content=response.content,
            status_code=response.status_code,
            headers={"Content-Type": "application/json"}
    )

```

```python
@app.post("/api/chat")
async def chat(request: Request) -> Response:
    """Simulate namesake Ollama API"""
    body = await request.json()
    messages = body.get("messages", [])
    last_message = messages[-1]["content"] if messages else ""

    # Routing for calling the Python agents directly
    content = call_agents_by_router(last_message)
    if content is not None:
        response = {"message":{"role":"assistant","content":content},"done": False}
        response = json.dumps(response) + '''
        {"message":{"role":"assistant","content":""},"done":true}
        '''
        return Response(
            content=response.encode('utf-8'),
            status_code=200
        )

    response = requests.post(f"{OLLAMA_URL}/api/chat", json=body)
    return Response(
            content=response.content,
            status_code=response.status_code,
            headers={"Content-Type": "application/json"}
    )

```

```python
@app.post("/api/chat")
async def chat(request: Request) -> Response:
    """Simulate namesake Ollama API"""
    body = await request.json()
    messages = body.get("messages", [])
    last_message = messages[-1]["content"] if messages else ""

    # Routing for calling the Python agents directly
    content = call_agents_by_router(last_message)
    if content is not None:
        response = {"message":{"role":"assistant","content":content},"done": False}
        response = json.dumps(response) + '''
        {"message":{"role":"assistant","content":""},"done":true}
        '''
        return Response(
            content=response.encode('utf-8'),
            status_code=200
        )

    response = requests.post(f"{OLLAMA_URL}/api/chat", json=body)
    return Response(
            content=response.content,
            status_code=response.status_code,
            headers={"Content-Type": "application/json"}
    )

```

```python
@app.post("/api/chat")
async def chat(request: Request) -> Response:
    """Simulate namesake Ollama API"""
    body = await request.json()
    messages = body.get("messages", [])
    last_message = messages[-1]["content"] if messages else ""

    # Routing for calling the Python agents directly
    content = call_agents_by_router(last_message)
    if content is not None:
        response = {"message":{"role":"assistant","content":content},"done": False}
        response = json.dumps(response) + '''
        {"message":{"role":"assistant","content":""},"done":true}
        '''
        return Response(
            content=response.encode('utf-8'),
            status_code=200
        )

    response = requests.post(f"{OLLAMA_URL}/api/chat", json=body)
    return Response(
            content=response.content,
            status_code=response.status_code,
            headers={"Content-Type": "application/json"}
    )

```

FastAPI

Ollama API

- @app.post("/api/chat")
- @app.post("/api/embeddings")
- @app.post("/api/generate")
- @app.post("/api/show")
- @app.get("/api/tags")

FastAPI

Ollama API

- @app.post("/api/chat")
- @app.post("/api/embeddings")
- @app.post("/api/generate")
- @app.post("/api/show")
- @app.get("/api/tags")

OpenAI Compatible

- @app.post("/v1/chat/completions")
- @app.post("/v1/completions")
- @app.post("/v1/embeddings")
- @app.get("/v1/models")

Document Manager

File conversion

- extract_text_from_html()
- extract_text_from_pdf()

Document Manager

File conversion

- extract_text_from_html()
- extract_text_from_pdf()

Text preparation

- chunk_text()
- get_embedding()

Document Manager

File conversion

- extract_text_from_html()
- extract_text_from_pdf()

Text preparation

- chunk_text()
- get_embedding()

ChromaDB

- add_to_collection()
- query_collection()

Document Manager - storage.py

File conversion

- extract_text_from_html()
- extract_text_from_pdf()

Text preparation

- chunk_text()
- get_embedding()

ChromaDB

- add_to_collection()
- query_collection()

Document Manager - FastAPI

File conversion

- extract_text_from_html()
- extract_text_from_pdf()

Text preparation

- chunk_text()
- get_embedding()

ChromaDB

- add_to_collection()
- query_collection()

```python

async def use_any_agent(tool_name: str, message: str) -> Optional[str]:
    """Retrieve information from the web"""
    tools = [search_web, visit_webpage]
    if tool_name == "search_web":
        tools = [search_web]
    elif tool_name == "visit_webpage":
        tools = [visit_webpage]
    agent = await AnyAgent.create_async(
        "smolagents",
        AgentConfig(
            model_id="ollama/" + MODEL,
            api_base=OLLAMA_URL,
            instructions="""You must use the available tools to find an answer.""",
            tools=tools,
        ),
    )
    agent_trance = await agent.run_async(message)
    return str(agent_trance.final_output)

```

```python

async def use_any_agent(tool_name: str, message: str) -> Optional[str]:
    """Retrieve information from the web"""
    tools = [search_web, visit_webpage]
    if tool_name == "search_web":
        tools = [search_web]
    elif tool_name == "visit_webpage":
        tools = [visit_webpage]
    agent = await AnyAgent.create_async(
        "smolagents",
        AgentConfig(
            model_id="ollama/" + MODEL,
            api_base=OLLAMA_URL,
            instructions="""You must use the available tools to find an answer.""",
            tools=tools,
        ),
    )
    agent_trance = await agent.run_async(message)
    return str(agent_trance.final_output)

```

```python

async def use_any_agent(tool_name: str, message: str) -> Optional[str]:
    """Retrieve information from the web"""
    tools = [search_web, visit_webpage]
    if tool_name == "search_web":
        tools = [search_web]
    elif tool_name == "visit_webpage":
        tools = [visit_webpage]
    agent = await AnyAgent.create_async(
        "smolagents",
        AgentConfig(
            model_id="ollama/" + MODEL,
            api_base=OLLAMA_URL,
            instructions="""You must use the available tools to find an answer.""",
            tools=tools,
        ),
    )
    agent_trance = await agent.run_async(message)
    return str(agent_trance.final_output)

```

```python

async def use_any_agent(tool_name: str, message: str) -> Optional[str]:
    """Retrieve information from the web"""
    tools = [search_web, visit_webpage]
    if tool_name == "search_web":
        tools = [search_web]
    elif tool_name == "visit_webpage":
        tools = [visit_webpage]
    agent = await AnyAgent.create_async(
        "smolagents",
        AgentConfig(
            model_id="ollama/" + MODEL,
            api_base=OLLAMA_URL,
            instructions="""You must use the available tools to find an answer.""",
            tools=tools,
        ),
    )
    agent_trance = await agent.run_async(message)
    return str(agent_trance.final_output)

```

PyVenice #1

- Ollama
- FastAPI
- ChromaDB
- any-agent

Thank you for listening !

@PyVenice #1 #Meetup #PythonItalia #Python