

# MULTI-AGENT ARCHITECTURE

Di Maio, Montesi, Traversa



# HOW WE CREATED OUR CREW

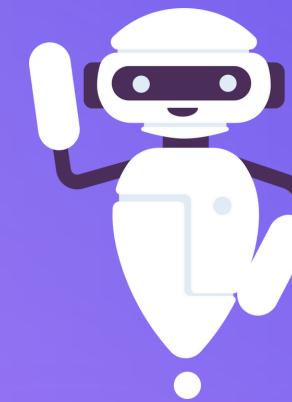


## We used Crew AI

We found Crew AI to be the most intuitive when building:

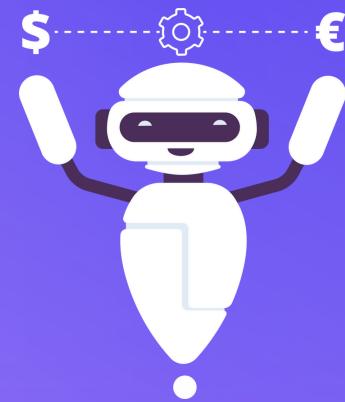
- Agent definition
- Tools
- Workflow structure
- Minimal coding, Maxiaml prompting

# OUR AGENTS



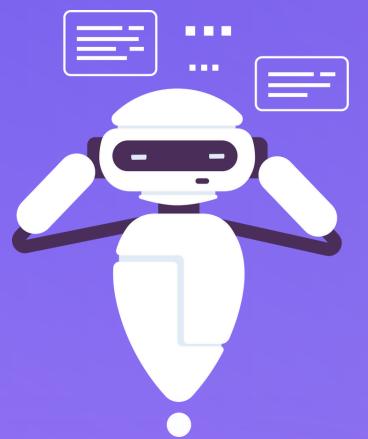
## ANALYST

- **Analyzes aggregated Italian public administration personnel data from the NoiPA portal, using an analysis tool to process queries and extract insights**
- **Produce both human-readable summaries and machine-readable CSV output specifically formatted for visualization.**



## VISUALIZER

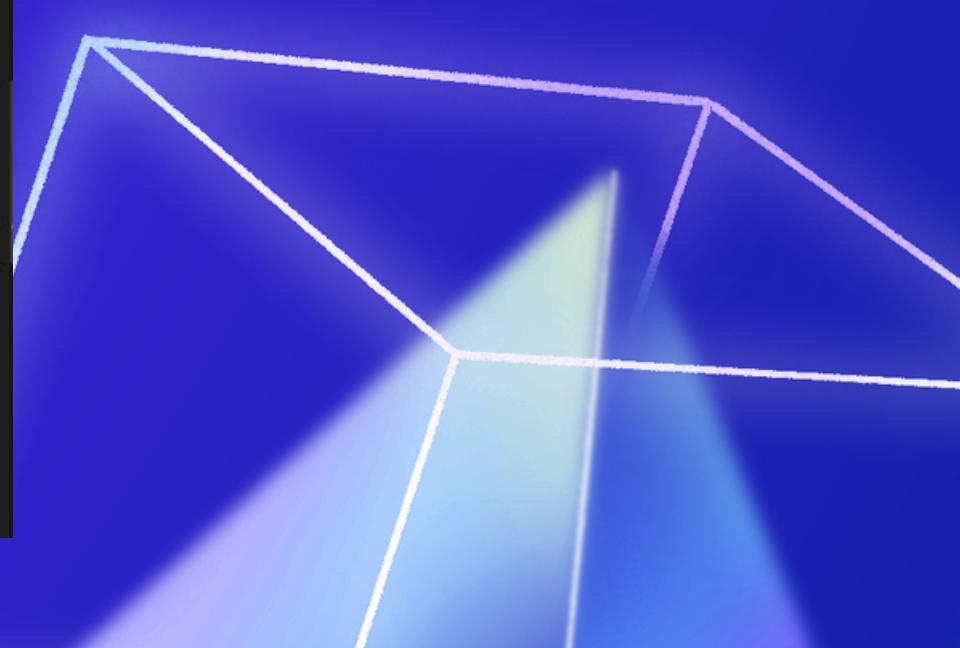
- **Transforms the Analyst's CSV data into visualizations by generating the most appropriate chart type using Python**
- **Outputs a structured JSON containing visualization parameters, plot descriptions, and the file path to the saved image, adapting visualization approaches when necessary.**



## REPORTER

- **Orchestrates the presentation layer by taking the Analyst's findings and Visualizer's JSON output and using a specialized Streamlit tool to render both text and visualization in the application.**

# REPOSITORY



A screenshot of a code editor showing a Python script named `app.py`. The code defines several components: `analyst.py`, `visualizer.py`, and `reporter.py` under the `agents` directory; `data`, `plots`, `tasks`, `tools`, and `utils` modules; and environment variables `.env` and `.gitignore`. The `app.py` script orchestrates these components to perform data processing, visualization, and reporting.

```
File Edit Selection View Go Run ... ⏪ ⏩ agents 0: 0: - x  
EXPLORER app.py analyst.py visualizer.py  
AGENTS  
_pycache_ .venv  
agents  
_pycache_ _init_.py analyst.py reporter.py visualizer.py  
data plots tasks tools utils  
.env .gitignore app.py crew_tasks.log crew.py README.md requirements.txt  
OUTLINE  
TIMELINE  
122     visualization_code_generation_task.context = [analyst_data_processing_task]  
123  
124  
125     # Task for Final Reporter (to use Streamlit tool)  
126     final_report_rendering_task = create_final_reporting_task(  
127         reporter_agent=reporter_agent,  
128         original_user_query=query,  
129         analyst_findings_context_name=analyst_output_context_placeholder,  
130         visualizer_json_context_name=visualizer_output_context_placeholder  
131     )  
132     # Set context: Reporter task needs output from Analyst AND Visualizer tasks  
133     final_report_rendering_task.context = [analyst_data_processing_task, visualization_code_generation_task]  
134  
135  
136     # --- 5. Orchestrate the Crew ---  
137  
138     crew = Crew()  
139     agents=[analyst_agent, visualizer_agent, reporter_agent],  
140     tasks=[  
141         analyst_data_processing_task,  
142         visualization_code_generation_task,  
143         final_report_rendering_task  
144     ],  
145     process=Process.sequential,  
146     verbose=1,  
147     #memory = True,  
148     #short_term_memory = short_term_memory  
149     )  
150
```

- Agents (analyst, visualizer, reporter)
- Datasets
- Tasks
- Tools
- App.py

# METRICS

We used a scale from 1–5

---

- **AVERAGE ACCURACY SCORE: 2,90**
- **AVERAGE SCORE ITALIAN QUERIES : 2,96**
- **AVERAGE SCORE ENGLISH QUERIES: 2,85**
- **AVERAGE TIME: 38 secs**

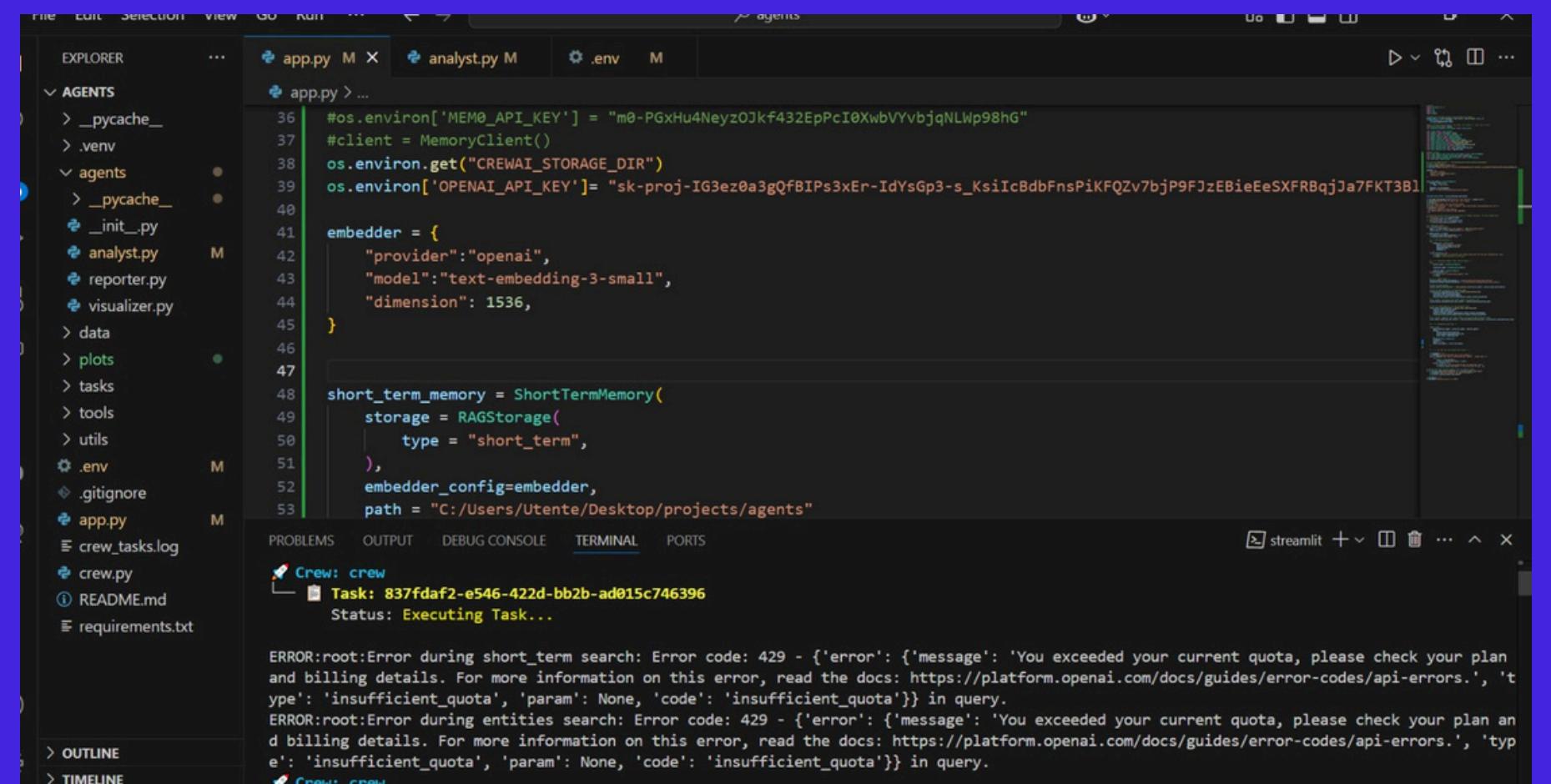


# MEMORY

## WHAT WE DISCOVERED:

- **No memory buffer supported**
- **Default configuration for storing memory in embeddings**
- **Memory intuitive to construct but only works with a paid text embedder model**

## THE CODE



The screenshot shows a code editor interface with several files open in tabs: app.py, analyst.py, .env, and requirements.txt. The app.py file contains the following code:

```
#os.environ['MEMO_API_KEY'] = "m0-PGxHu4Neyz0Jkf432EpPcI0XwbVYvbjqNLWp98hG"
client = MemoryClient()
os.environ.get("CREWAI_STORAGE_DIR")
os.environ['OPENAI_API_KEY']= "sk-proj-IG3ez0a3gQfBIPs3xEr-IdYsGp3-s_KsiIcBdbFnsPiKFQZv7bjP9FJzEBieEeSXFRBqjJa7FKT3B1"

embedder = {
    "provider": "openai",
    "model": "text-embedding-3-small",
    "dimension": 1536,
}

short_term_memory = ShortTermMemory(
    storage = RAGStorage(
        type = "short_term",
    ),
    embedder_config=embedder,
    path = "C:/Users/Utente/Desktop/projects/agents"
)
```

The terminal tab shows a task named 'crew' executing, with the status 'Executing Task...'. The output of the task shows two errors related to OpenAI API quota exceedance.

```
ERROR:root>Error during short_term search: Error code: 429 - {'error': {'message': 'You exceeded your current quota, please check your plan and billing details. For more information on this error, read the docs: https://platform.openai.com/docs/guides/error-codes/api-errors.', 'type': 'insufficient_quota', 'param': None, 'code': 'insufficient_quota'}} in query.
ERROR:root>Error during entities search: Error code: 429 - {'error': {'message': 'You exceeded your current quota, please check your plan and billing details. For more information on this error, read the docs: https://platform.openai.com/docs/guides/error-codes/api-errors.', 'type': 'insufficient_quota', 'param': None, 'code': 'insufficient_quota'}} in query.
```

# MAIN TAKE AWAYS

01

- A sequential architecture worked better than a hierarchical architecture for this task

02

- Queries in italian performed slightly better, this may be cause by the nature of the data

THANK YOU!

