O'REILLY®

Agent Tools & Features

Bert Gollnick

Learning Objectives

By the end of this module, you will know:

- Which tools can be applied by an agent
- which features are available

Which tools are available?

Searching the Internet

Scraping Websites

Reading Files

Tool	Description
CodeDocsSearchTool	A RAG tool optimized for searching through code documentation and related technical documents.
CSVSearchTool	A RAG tool designed for searching within CSV files, tailored to handle structured data.
DirectorySearchTool	A RAG tool for searching within directories, useful for navigating through file systems.
DOCXSearchTool	A RAG tool aimed at searching within DOCX documents, ideal for processing Word files.
DirectoryReadTool	Facilitates reading and processing of directory structures and their contents.
FileReadTool	Enables reading and extracting data from files, supporting various file formats.
GithubSearchTool	A RAG tool for searching within GitHub repositories, useful for code and documentation search.
SerperDevTool	A specialized tool for development purposes, with specific functionalities under development.
TXTSearchTool	A RAG tool focused on searching within text (.txt) files, suitable for unstructured data.

...

Source: https://docs.crewai.com/core-concepts/Tools/#available-crewai-tools

Memory

Short-Term Memory

- temporary storage of interactions
- enables agents to recall information to current context

Long-Term Memory

- preserves valuable insights and outcomes
- allows agents to build up knowledge over time

Entity Memory

- captures and organizes information on entities, e.g. people, places

Contextual Memory

- keeps context of interactions
- increases relevance of agent responses

Memory

- implementation is pretty simple
- by default
  - memory is disabled
  - uses OpenAI embeddings

```
from crewai import Crew, Agent, Task, Process

my_crew = Crew(
    agents = [...],
    tasks = [...],
    process = Process.sequential,
    memory = True,
    verbose = True
)
```

Memory

Adaptive Learning

- Crews adapt to new information and refine their approach to tasks

Enhanced Personalisation

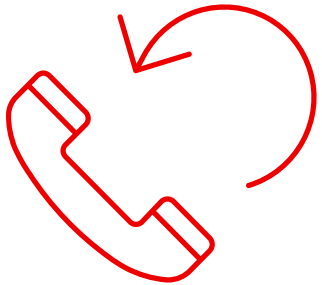
- Agents remember user preferences and historical interactions

Improved Performance

- More informed decisions
- Use past learnings and contextual insights

Callbacks

- Task callback and step callback
- Executed after task or step-completion
- Can be used for
  - Notifications
  - actions
- Parameter passed inside task



Expected Task Outcome

- Output formats can be defined in detail

```
class OutputFormat(BaseModel):
    chapter_title: str
    bullet_points: list[str]
```

```
Task(
    description=("..."),
    expected_output="A well-written slideset …“,
    agent=editor,
    output_format="markdown",
    output_format_model=OutputFormat,
    output_format_description=(
        "The output format is a markdown file …"
    ),
    output_file = "slideset.md"
)
```

Use of other LLMs

- Set up an llm-object
- Pass it as a parameter

```
from langchain_groq import ChatGroq

llm=ChatGroq(temperature=0,
             model_name=MODEL,
             api_key=os.environ["GROQ_API_KEY"])

planner = Agent(
    role="…",
    goal="…",
    backstory="…",
    allow_delegation=False,
    llm=llm,
    verbose=True
)
```


O'REILLY®