



☰ Core Concepts > Agents

Core Concepts

# Agents

Detailed guide on creating and managing agents within the CrewAI framework.

## Overview of an Agent

In the CrewAI framework, an `Agent` is an autonomous unit that can:

- Perform specific tasks
- Make decisions based on its role and goal
- Use tools to accomplish objectives
- Communicate and collaborate with other agents
- Maintain memory of interactions
- Delegate tasks when allowed

💡 Think of an agent as a specialized team member with specific skills, expertise, and responsibilities. For example, a `Researcher` agent might excel at gathering and analyzing information, while a `Writer` agent might be better at creating content.



## Core Concepts > Agents

<b>Goal</b>	<code>goal</code>	<code>str</code>	The individual objective that guides the agent's decision-making.
<b>Backstory</b>	<code>backstory</code>	<code>str</code>	Provides context and personality to the agent, enriching interactions.
<b>LLM (optional)</b>	<code>llm</code>	<code>Union[str, LLM, Any]</code>	Language model that powers the agent. Defaults to the model specified in <code>OPENAI_MODEL_NAME</code> or "gpt-4".
<b>Tools (optional)</b>	<code>tools</code>	<code>List[BaseTool]</code>	Capabilities or functions available to the agent. Defaults to an empty list.
<b>Function Calling LLM (optional)</b>	<code>function_calling_llm</code>	<code>Optional[Any]</code>	Language model for tool calling, overrides crew's LLM if specified.
<b>Max Iterations (optional)</b>	<code>max_iter</code>	<code>int</code>	Maximum iterations before the agent must provide its best answer. Default is 20.
<b>Max RPM (optional)</b>	<code>max_rpm</code>	<code>Optional[int]</code>	Maximum requests per minute to avoid rate limits.
<b>Max Execution Time (optional)</b>	<code>max_execution_time</code>	<code>Optional[int]</code>	Maximum time (in seconds) for task execution.
<b>Memory (optional)</b>	<code>memory</code>	<code>bool</code>	Whether the agent should maintain memory of interactions. Default is True.
<b>Verbose (optional)</b>	<code>verbose</code>	<code>bool</code>	Enable detailed execution logs for debugging. Default is False.



Core Concepts > Agents

<i>(optional)</i>			crew callback.
<b>Cache</b> <i>(optional)</i>	cache	bool	Enable caching for tool usage. Default is True.
<b>System Template</b> <i>(optional)</i>	system_template	Optional[str]	Custom system prompt template for the agent.
<b>Prompt Template</b> <i>(optional)</i>	prompt_template	Optional[str]	Custom prompt template for the agent.
<b>Response Template</b> <i>(optional)</i>	response_template	Optional[str]	Custom response template for the agent.
<b>Allow Code Execution</b> <i>(optional)</i>	allow_code_execution	Optional[bool]	Enable code execution for the agent. Default is False.
<b>Max Retry Limit</b> <i>(optional)</i>	max_retry_limit	int	Maximum number of retries when an error occurs. Default is 2.
<b>Respect Context Window</b> <i>(optional)</i>	respect_context_window	bool	Keep messages under context window size by summarizing. Default is True.
<b>Code Execution Mode</b> <i>(optional)</i>	code_execution_mode	Literal["safe", "unsafe"]	Mode for code execution: 'safe' (using Docker) or 'unsafe' (direct). Default is 'safe'.
<b>Embedder</b> <i>(optional)</i>	embedder	Optional[Dict[str, Any]]	Configuration for the embedder used by the agent.
<b>Knowledge Sources</b> <i>(optional)</i>	knowledge_sources	Optional[List[BaseKnowledgeSource]]	Knowledge sources available to the agent.



Core Concepts > Agents

## Creating Agents

There are two ways to create agents in CrewAI: using **YAML configuration (recommended)** or defining them **directly in code**.

### YAML Configuration (Recommended)

Using YAML configuration provides a cleaner, more maintainable way to define agents. We strongly recommend using this approach in your CrewAI projects.

After creating your CrewAI project as outlined in the [Installation](#) section, navigate to the `src/latest_ai_development/config/agents.yaml` file and modify the template to match your requirements.

❗ Variables in your YAML files (like `{topic}` ) will be replaced with values from your inputs when running the crew:

Code

```
crew.kickoff(inputs={'topic': 'AI Agents'})
```

Here's an example of how to configure agents using YAML:



## Core Concepts > Agents

**role:** >

{topic} Senior Data Researcher

**goal:** >

Uncover cutting-edge developments in {topic}

**backstory:** >

You're a seasoned researcher with a knack for uncovering the latest developments in {topic}. Known for your ability to find the most relevant information and present it in a clear and concise manner.

**reporting\_analyst:**

**role:** >

{topic} Reporting Analyst

**goal:** >

Create detailed reports based on {topic} data analysis and research findings

**backstory:** >

You're a meticulous analyst with a keen eye for detail. You're known for your ability to turn complex data into clear and concise reports, making it easy for others to understand and act on the information you provide.

To use this YAML configuration in your code, create a crew class that inherits from `CrewBase` :

Code



## Core Concepts > Agents

```
@CrewBase
class LatestAiDevelopmentCrew():
    """LatestAiDevelopment crew"""

    agents_config = "config/agents.yaml"

    @agent
    def researcher(self) -> Agent:
        return Agent(
            config=self.agents_config['researcher'],
            verbose=True,
            tools=[SerperDevTool()]
        )

    @agent
    def reporting_analyst(self) -> Agent:
        return Agent(
            config=self.agents_config['reporting_analyst'],
            verbose=True
        )
```

❗ The names you use in your YAML files ( `agents.yaml` ) should match the method names in your Python code.



## Core Concepts &gt; Agents

## Code

```
from crewai import Agent
from crewai_tools import SerperDevTool

# Create an agent with all available parameters
agent = Agent(
    role="Senior Data Scientist",
    goal="Analyze and interpret complex datasets to provide actionable insights",
    backstory="With over 10 years of experience in data science and machine learning, "
            "you excel at finding patterns in complex datasets.",
    llm="gpt-4", # Default: OPENAI_MODEL_NAME or "gpt-4"
    function_calling_llm=None, # Optional: Separate LLM for tool calling
    memory=True, # Default: True
    verbose=False, # Default: False
    allow_delegation=False, # Default: False
    max_iter=20, # Default: 20 iterations
    max_rpm=None, # Optional: Rate limit for API calls
    max_execution_time=None, # Optional: Maximum execution time in seconds
    max_retry_limit=2, # Default: 2 retries on error
    allow_code_execution=False, # Default: False
    code_execution_mode="safe", # Default: "safe" (options: "safe", "unsafe")
    respect_context_window=True, # Default: True
    use_system_prompt=True, # Default: True
    tools=[SerperDevTool()], # Optional: List of tools
```



Core Concepts > Agents

```
    step_callback=None, # Optional: Callback function for monitoring
)
```

Let's break down some key parameter combinations for common use cases:

## Basic Research Agent

Code

```
research_agent = Agent(
    role="Research Analyst",
    goal="Find and summarize information about specific topics",
    backstory="You are an experienced researcher with attention to detail",
    tools=[SerperDevTool()],
    verbose=True # Enable logging for debugging
)
```

## Code Development Agent

Code





## Core Concepts > Agents

```
allow_code_execution=True,  
code_execution_mode="safe", # Uses Docker for safety  
max_execution_time=300, # 5-minute timeout  
max_retry_limit=3 # More retries for complex code tasks  
)
```

## Long-Running Analysis Agent

### Code

```
analysis_agent = Agent(  
    role="Data Analyst",  
    goal="Perform deep analysis of large datasets",  
    backstory="Specialized in big data analysis and pattern recognition",  
    memory=True,  
    respect_context_window=True,  
    max_rpm=10, # Limit API calls  
    function_calling_llm="gpt-4o-mini" # Cheaper model for tool calls  
)
```

## Custom Template Agent



## Core Concepts > Agents

```
goal="Assist customers with their inquiries",
backstory="Experienced in customer support with a focus on satisfaction",
system_template="""<|start_header_id|>system<|end_header_id|>
                {{ .System }}<|eot_id|>""",
prompt_template="""<|start_header_id|>user<|end_header_id|>
                {{ .Prompt }}<|eot_id|>""",
response_template="""<|start_header_id|>assistant<|end_header_id|>
                {{ .Response }}<|eot_id|>""",
)
```

## Parameter Details

### Critical Parameters

`role` , `goal` , and `backstory` are required and shape the agent's behavior

`llm` determines the language model used (default: OpenAI's GPT-4)

### Memory and Context

`memory` : Enable to maintain conversation history

`respect_context_window` : Prevents token limit issues

`knowledge_sources` : Add domain-specific knowledge bases



## Core Concepts > Agents

`max_rpm` : Rate limiting for API calls

`max_retry_limit` : Retries on error

## Code Execution

`allow_code_execution` : Must be True to run code

`code_execution_mode` :

"safe" : Uses Docker (recommended for production)

"unsafe" : Direct execution (use only in trusted environments)

## Templates

`system_template` : Defines agent's core behavior

`prompt_template` : Structures input format

`response_template` : Formats agent responses

ⓘ When using custom templates, you can use variables like `{role}` , `{goal}` , and `{input}` in your templates. These will be automatically populated during execution.

## Agent Tools



Core Concepts > Agents

Here's how to add tools to an agent:

#### Code

```
from crewai import Agent
from crewai_tools import SerperDevTool, WikipediaTools

# Create tools
search_tool = SerperDevTool()
wiki_tool = WikipediaTools()

# Add tools to agent
researcher = Agent(
    role="AI Technology Researcher",
    goal="Research the latest AI developments",
    tools=[search_tool, wiki_tool],
    verbose=True
)
```

## Agent Memory and Context



## Core Concepts > Agents

```
from crewai import Agent

analyst = Agent(
    role="Data Analyst",
    goal="Analyze and remember complex data patterns",
    memory=True, # Enable memory
    verbose=True
)
```

ⓘ When `memory` is enabled, the agent will maintain context across multiple interactions, improving its ability to handle complex, multi-step tasks.

## Important Considerations and Best Practices

### Security and Code Execution

When using `allow_code_execution`, be cautious with user input and always validate it

Use `code_execution_mode: "safe"` (Docker) in production environments

Consider setting appropriate `max_execution_time` limits to prevent infinite loops



Core Concepts > Agents

Enable `cache: true` to improve performance for repetitive tasks

Adjust `max_iter` and `max_retry_limit` based on task complexity

## Memory and Context Management

Use `memory: true` for tasks requiring historical context

Leverage `knowledge_sources` for domain-specific information

Configure `embedder_config` when using custom embedding models

Use custom templates ( `system_template` , `prompt_template` , `response_template` ) for fine-grained control over agent behavior

## Agent Collaboration

Enable `allow_delegation: true` when agents need to work together

Use `step_callback` to monitor and log agent interactions

Consider using different LLMs for different purposes:

    Main `llm` for complex reasoning

`function_calling_llm` for efficient tool usage



Core Concepts > Agents

## Troubleshooting Common Issues

### 1. **Rate Limiting:** If you're hitting API rate limits:

Implement appropriate `max_rpm`

Use caching for repetitive operations

Consider batching requests

### 2. **Context Window Errors:** If you're exceeding context limits:

Enable `respect_context_window`

Use more efficient prompts

Clear agent memory periodically

### 3. **Code Execution Issues:** If code execution fails:

Verify Docker is installed for safe mode

Check execution permissions

Review code sandbox settings


### 4. **Memory Issues:** If agent responses seem inconsistent:



---

Core Concepts > **Agents**

Remember that agents are most effective when configured according to their specific use case. Take time to understand your requirements and adjust these parameters accordingly.

Was this page helpful?  Yes  No

< **Quickstart**

**Tasks** >

---

Powered by Mintlify