GOSIM

GOSIM AI Paris 2025

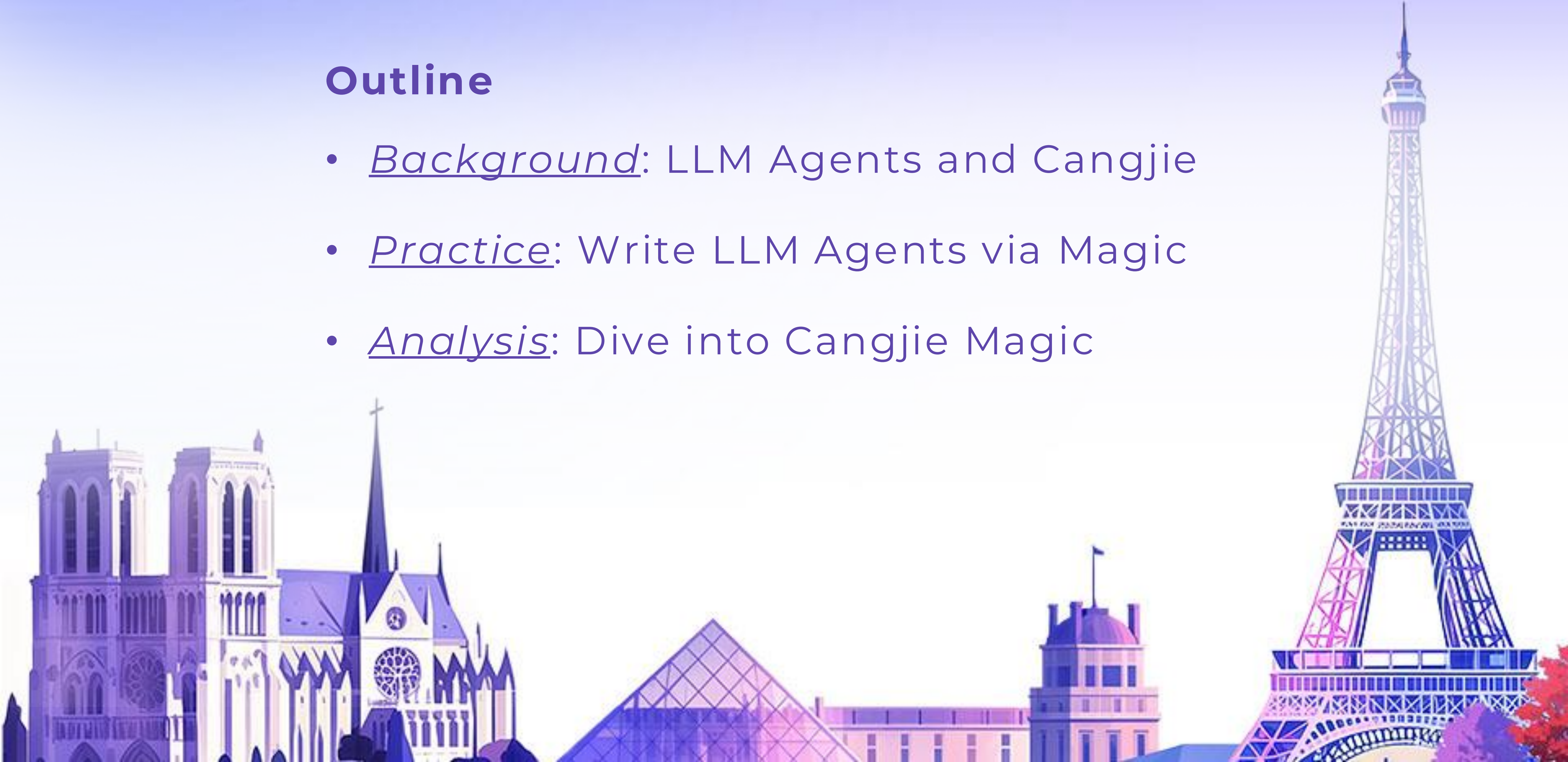
Cangjie Magic : New Choices for Developers in the Age of Large Models

Dongjie Chen

Huawei

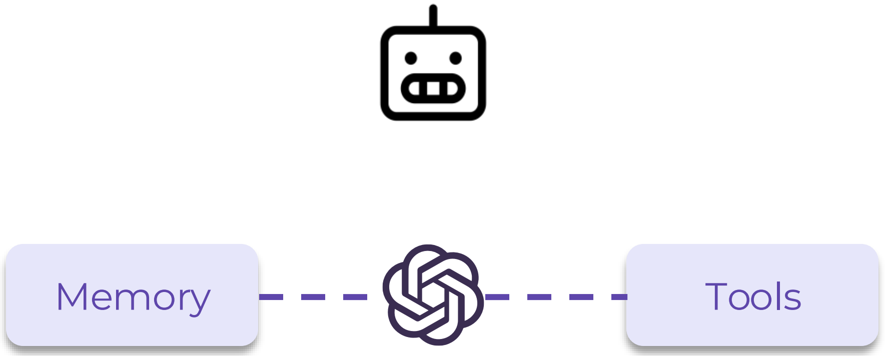
Outline

- *Background*: LLM Agents and Cangjie
- *Practice*: Write LLM Agents via Magic
- *Analysis*: Dive into Cangjie Magic



Background

LLM Agent X Cangjie



LLM-powered autonomous agents is able to resolve user requirements automatically! 🔥

- LLM functions the brain
- Use tools to interact with the external environment
- Remember and learn from past actions

[LLM Powered Autonomous Agents | Lil'Log (lilianweng.github.io)](https://lilianweng.github.io)

Cangjie
仓颉

```
TypeInfo.of(Cangjie) != "Chinese PL."

enum SupportedOS {
   | HarmonyOS | Linux    // Run
   | Windows | MacOS      // Develop
   | NextOne              // (TBD)
}
```

Cangjie: A modern programming language for efficient, secure, and full-scenario development

- Support multiple platforms
- Reliable security by a powerful type system
- Superior performance
- High extensibility by its macro system
- …

Preliminaries of Cangjie

😊 Cangjie basics you'll need to know in this talk

Function

```
func add(a: Int64, b: Int64): {
  return a + b
}
```

Class

```
class Foo {
  func say(): Unit {
    println("hello")
  }
}
```

Macro

```
@macro
func add(a: Int64, b: Int64): { … }

@macro
class Foo { … }
```

Expr. & Types

```
"string"  // String
[1, 2, 3] // Array
for (num in [1, 2, 3]) { println(num) } // for-
loop
```

LLM Agents with Magic

Agent 1: AI Automation

Config a chat model

Select an executor

Write agent system prompt

```
@agent[
  model: "deepseek:deepseek-chat",
  executor: "plan-react",
  mcp: [
    stdio("node ${MARKDOWNIFY_DIR}/dist/index.js"),
    stdio("docker run mcp/filesystem ...")
  ]
]
class FileAssistant {
  @prompt[pattern: ERA] (
    expectation: "Follow instructions to complete tasks step by step",
    role: "You are a file assistant, helping users manage files",
    action: "For each request, plan step by step. You can use tools such
as..."  )
}
```

Markdownify MCP Server

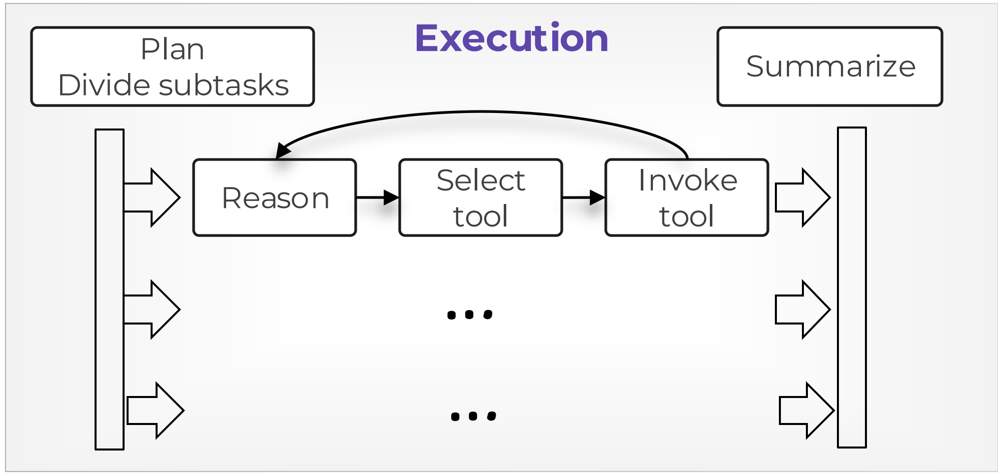
Filesystem MCP Server

Add MCP servers

Run The Agent 🚀

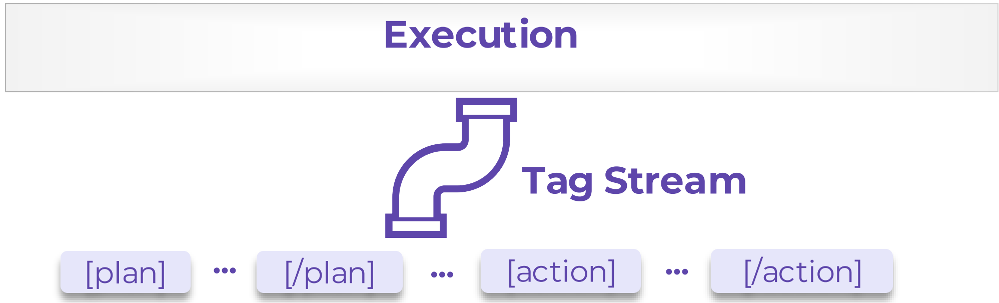
"Convert all PDF
files to Markdown"

```
let agent = FileAssistant()
for (data in agent.asyncChat(input)) {
  print(data)
}
```



“Files a.pdf, ... have
been converted to..."

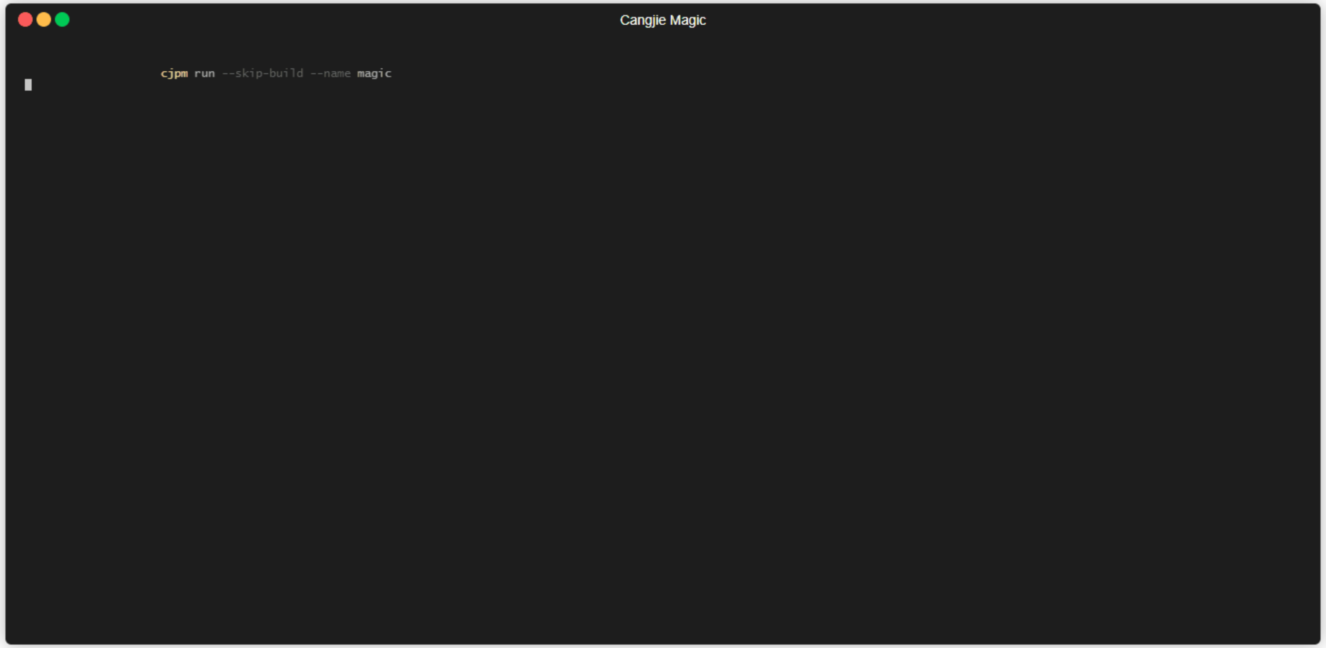
Display Execution Process



```
class ConsolePrinter <: TagStreamVisitor {

  override protected func onTag(tag: String): Unit {
    Console.stdOut.writeln(tag)
  }

  override protected func onChunk(chunk: String): Unit {
    Console.stdOut.write(chunk)
  }
  …
}
```



```
cjpm run --skip-build --name magic
```

Implement customized tag stream visitor to display the agent execution

⇨ Mobile Apps PC Apps …

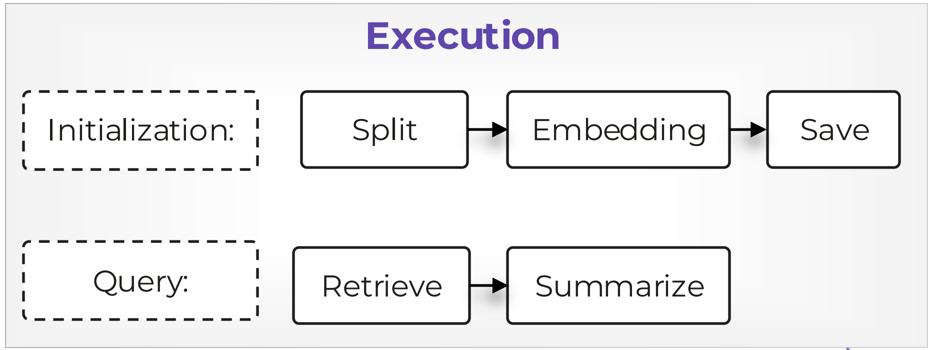
Agent 2: AI Document Q&A

Specify RAG configs

```
@agent[
  model: "deepseek:deepseek-chat",
  executor: "naive",
  rag: {
    source: "./docs/tutorial.md",
    mode: "static"
  }
]
class QABot {
  @prompt[pattern: ERA] (
    expectation: "Code blocks are wrapped
between ```cangjie and ```",
    role: "Simple Q&A assistant",
    action: "Search documents to retrieve
knowledge and answer questions"
  )
}
```

"How to write an agent using Cangjie Magic?"

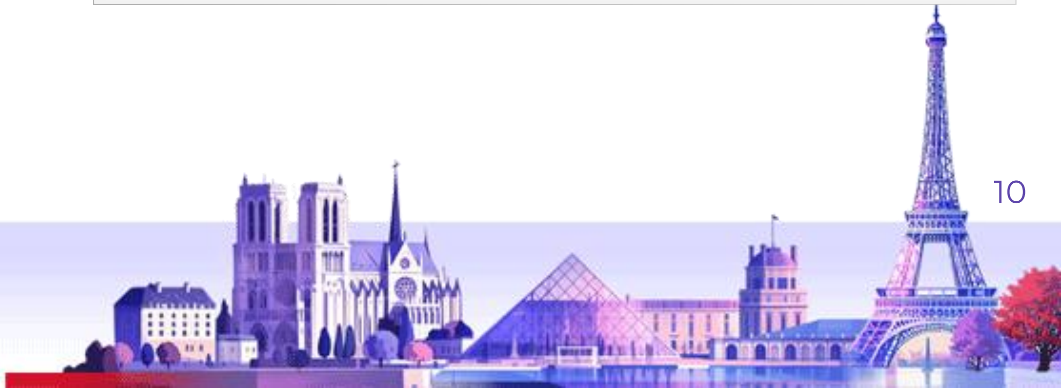
```
let agent = QABot()
for (data in agent.asyncChat(input)) {
  print(data)
}
```



Execution

Initialization: Split → Embedding → Save

Query: Retrieve → Summarize

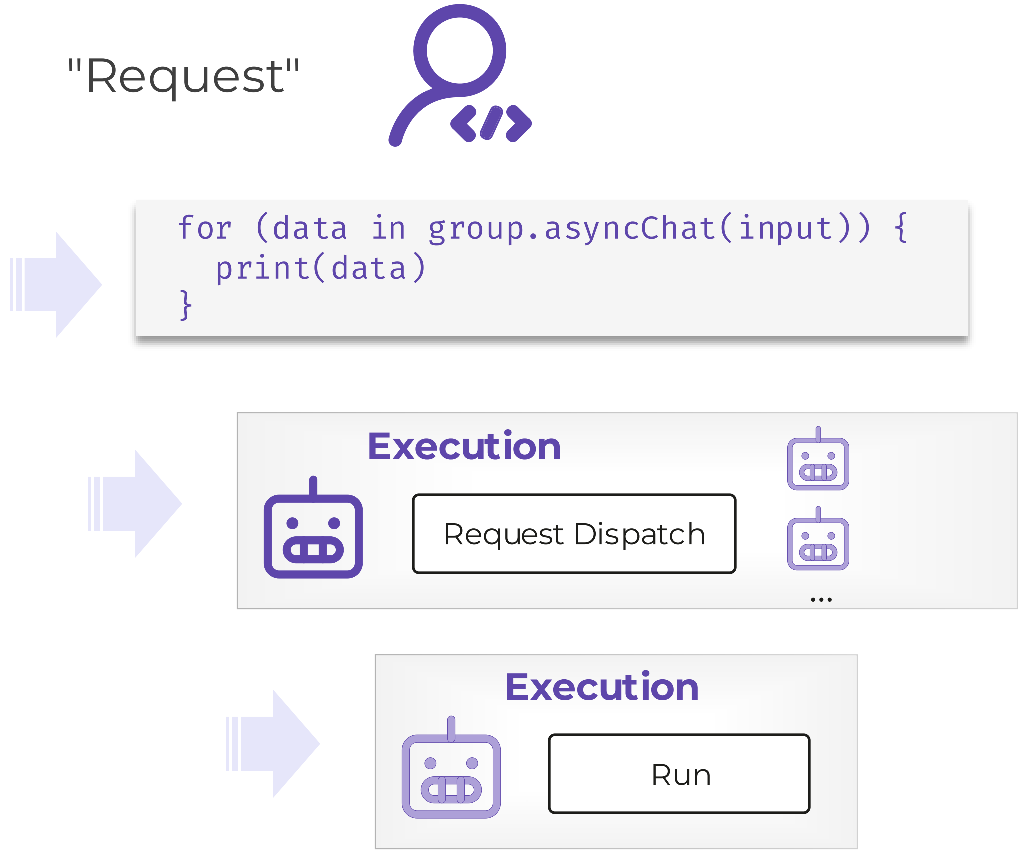


Put Agents Together

```
let group = DispatchAgent() <= [
    FileAssistant(),
    QABot(),
    ...
]
```

Use agent cooperation DSL to compose agents

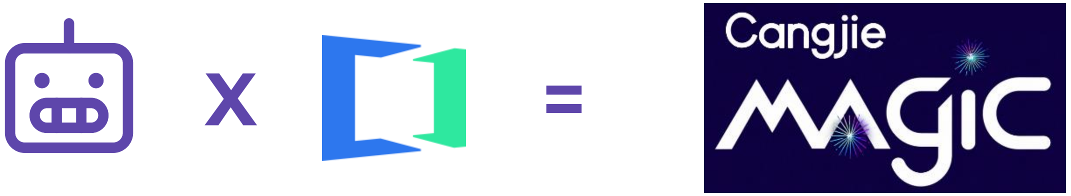
- Leader cooperation is used here
- DispatchAgent is a builtin agent



"Request"

```
for (data in group.asyncChat(input)) {
  print(data)
}
```

A Brief Summary



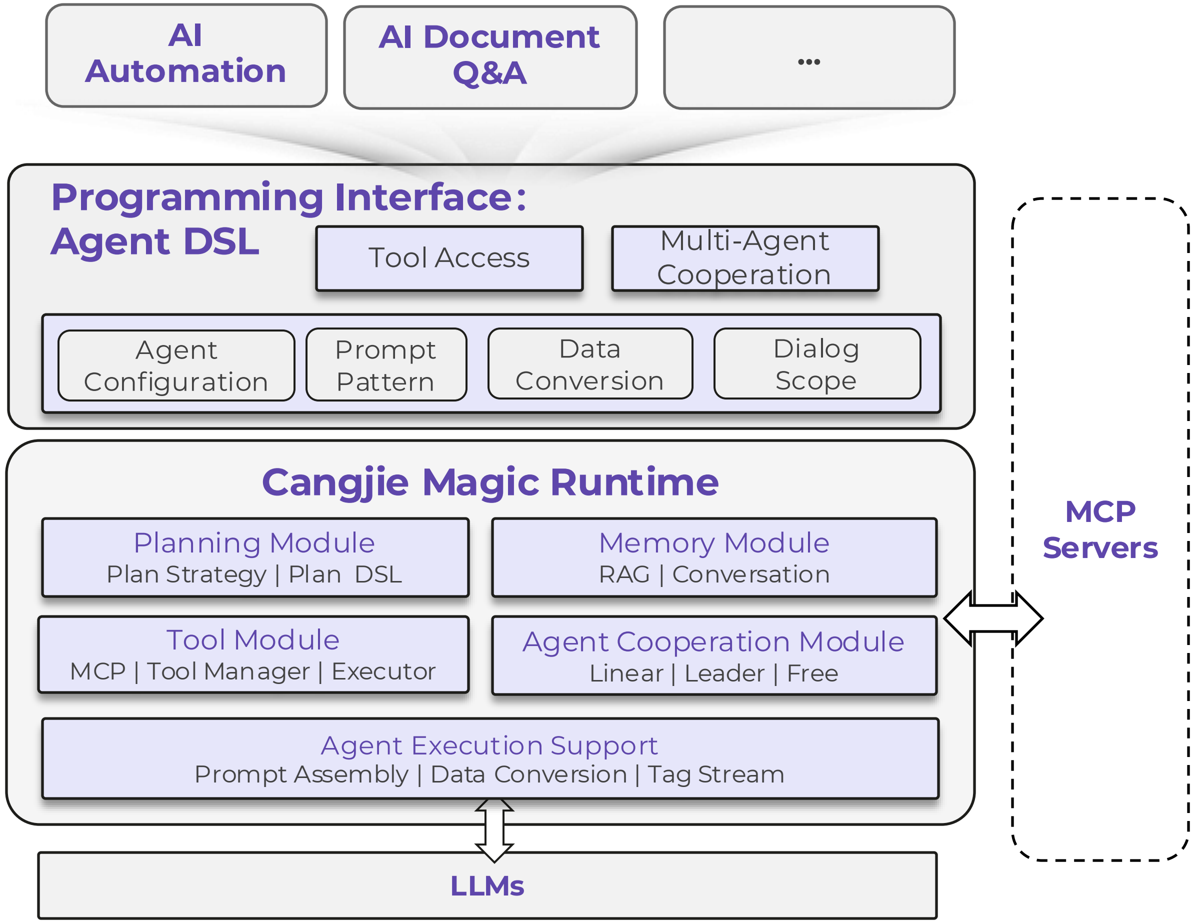
Simple Agent DSL
based on a modern programming language

Full-fledged components
to develop real agent applications

https://gitcode.com/Cangjie-TPC/CangjieMagic

Dive into Cangjie Magic

What is Cangjie Magic? 🏛️



AI Automation

AI Document Q&A

...

Programming Interface: Agent DSL

Tool Access

Multi-Agent Cooperation

Agent Configuration

Prompt Pattern

Data Conversion

Dialog Scope

Cangjie Magic Runtime

Planning Module
Plan Strategy | Plan DSL

Memory Module
RAG | Conversation

Tool Module
MCP | Tool Manager | Executor

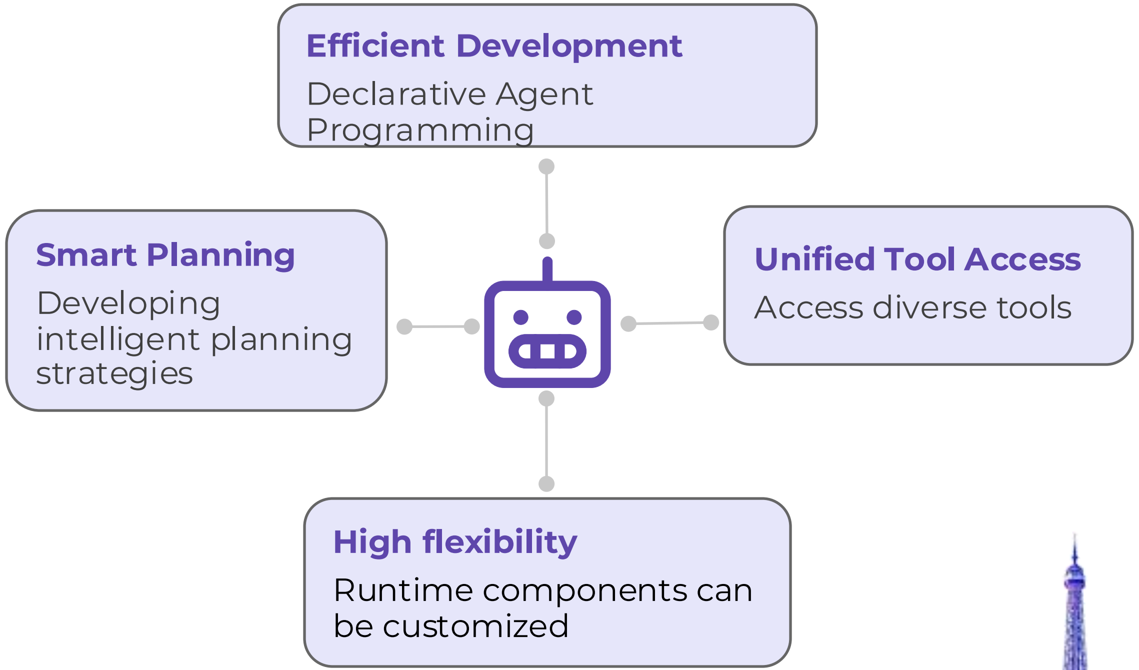
Agent Cooperation Module
Linear | Leader | Free

Agent Execution Support
Prompt Assembly | Data Conversion | Tag Stream

LLMs

MCP Servers

Features



Efficient Development
Declarative Agent Programming

Smart Planning
Developing intelligent planning strategies

Unified Tool Access
Access diverse tools

High flexibility
Runtime components can be customized

Efficient Development With DSL

DSL-powered agent programming

```
@agent[
  model: "deepseek:deepseek-chat",
  executor: "react"
]
class Foo {
   @prompt[pattern: ERA](
     expectation: "Responses should be concise and
precise",
     role: "You serve as an inquiry assistant",
     action: "…"
    )
}
```

One-liner config for models & planning

Prompt patterns guide writing better prompts

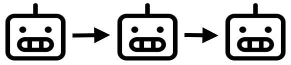
Maintain the conversation automatically

```
@dialog[agent: foo](
  "Birthdate of Einstein" -> d1: MyDate
  "Birthdate of Newton" -> d2: MyDate
  "Who is the elder" -> name: String
)
```

Streaming like symbols simplify multi-agent cooperation

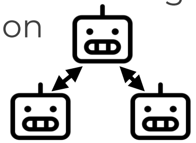
```
let linearGroup: LinearGroup = ag1 ▷ ag2 ▷ ag3
```

Linear cooperation: Sequential execution with message passing in order



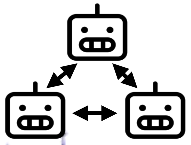
```
let leaderGroup: LeaderGroup = ag1 <= [ag2, ag3]
```

Leader cooperation: The master agent selects member Agents for communication



```
let freeGroup: FreeGroup = ag1 | ag2 | ag3
```

Free cooperation: Agents communicate with message broadcasting



Unified Tool Access

GOSIM

The DSL provides declarative tool configuration with unified agent access syntax

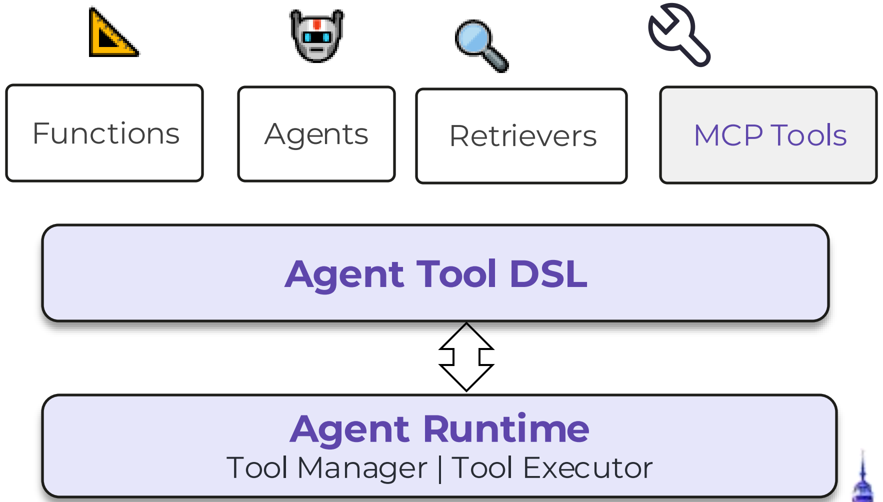
```
@tool[description: "Search relevant content
based on the question"]
func search(q: String):String { … }

@agent[
  mcp: [
    stdio("docker mcp/filesystem"),
    tools(search,
          BarAgent().asTool(),
          VectorDB().asRetriever())
    ]
]
class Foo { ... }
```

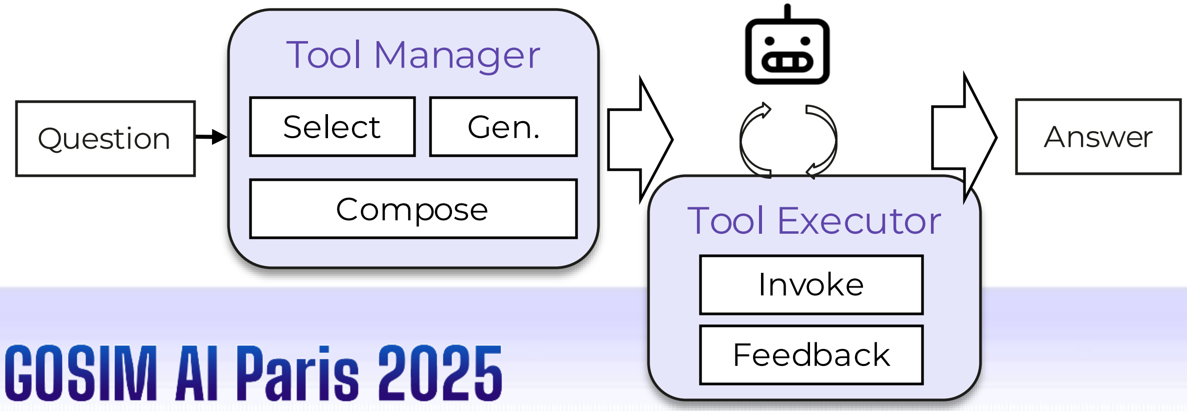
Enhance Agent execution capabilities through tools

Enhance Agent capabilities through tools

- Access extra-model knowledge
- Sense contextual information
- Execute concrete operations



Agents intelligently select and assemble tools at runtime



Smart Planning Strategies 🧠

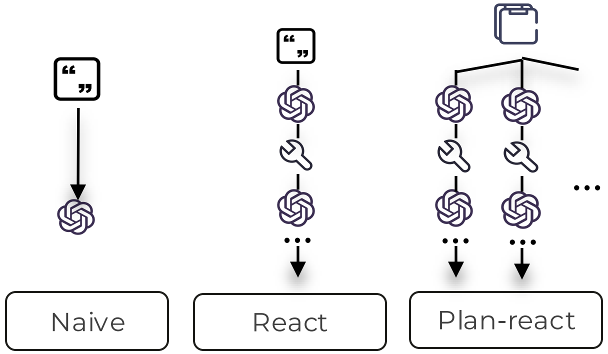
Agent DSL offers two approaches to planning

- One-liner configuration for built-in strategies
- Custom DSL development to create new strategies*

```
@agent[
  executor: "react"
]
class Foo { … }
```

```
@agent
class Foo {
  @loop(
    think |> act
  )
}
```

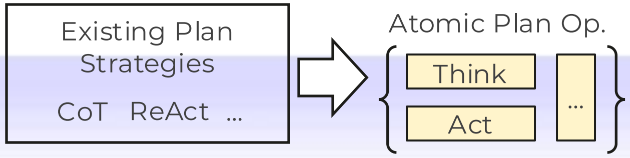
Built-in planning strategies



Customized planning strategies

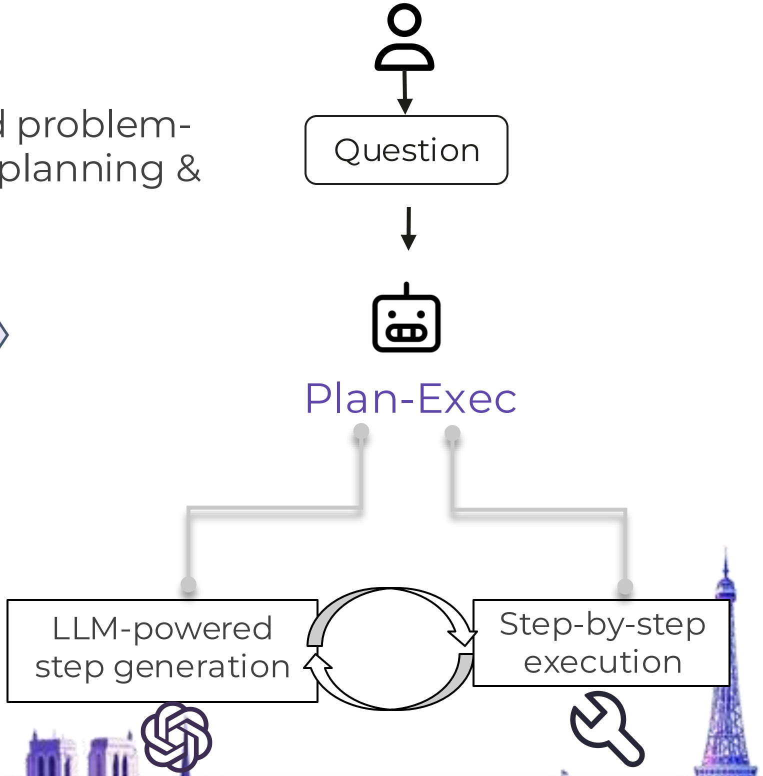
Planning DSL

- Extract atomic planning operations from common planning strategies
- Compose them into new strategies via DSL



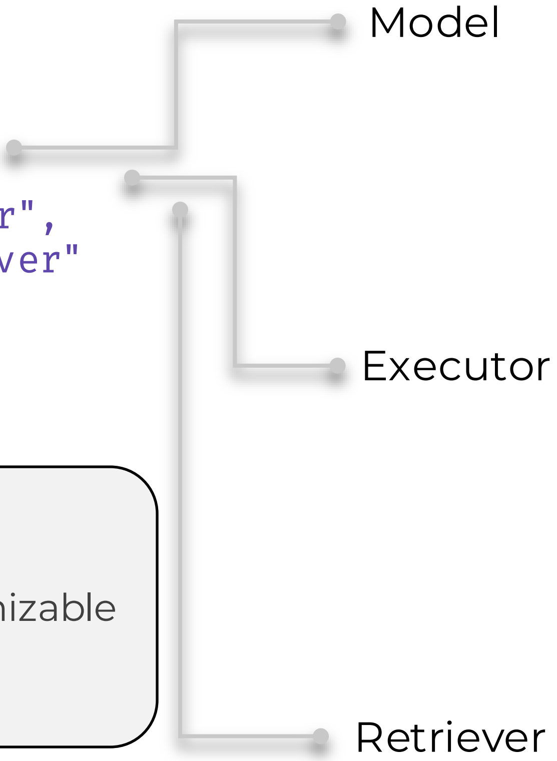
Planning strategies guide the Agent through problem-solving processes

Automated problem-solving via planning & tools



High Extensibility 📦

```
@agent[
  model: "myModel",
  executor: "myExecutor",
  retriever: "myRetriever"
]
class Foo { … }
```



```
@chatModel[
  name: "myModel",
]
class MyModel {
  func chat(request: ChatRequest) { … }
}
```

```
@executor[
  name: "myExecutor",
]
class MyExecutor {
  func execute(agent: Agent, request: AgentRequest { … }
}
```

```
@retriever[
  name: "myRetriever",
]
class MyRetriever {
  func search(query: String): Array<Document> {… }
}
```

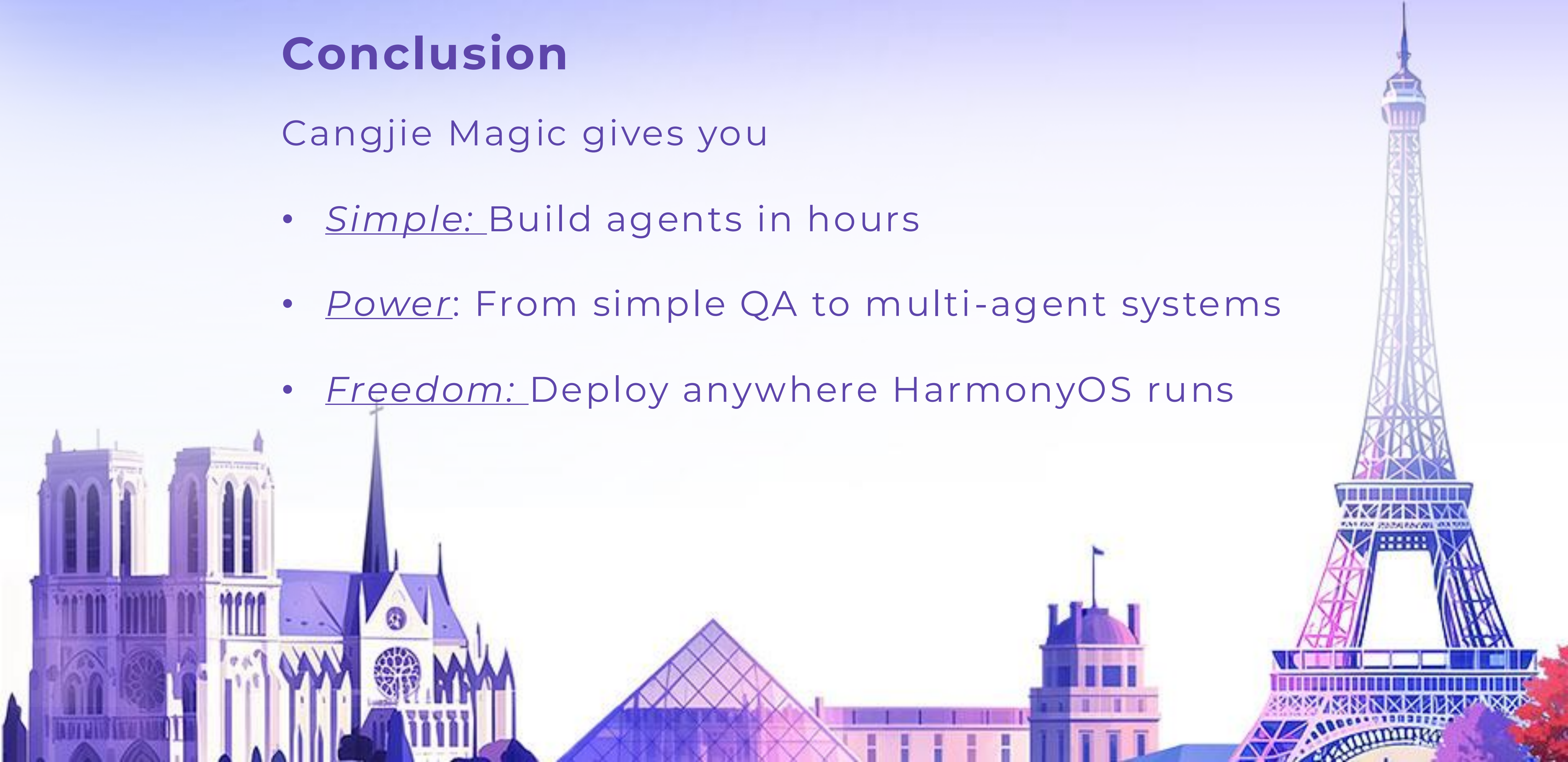
Modular Implementation

- High extensibility: agent components are fully customizable
- Low coupling: modules can operate independently

Conclusion

Cangjie Magic gives you

- *Simple:* Build agents in hours
- *Power*: From simple QA to multi-agent systems
- *Freedom:* Deploy anywhere HarmonyOS runs



Q&A

THANK YOU