Workshop Instructions – Part 2: Tool-Calling

Overview

In the second part of this workshop, you'll implement and experiment with **Tools**.
Tools extend what the model can do beyond text generation. Think of them as *plugins* the model can call to fetch or generate additional data.

For this workshop, we'll use tools to **fetch information from Wikipedia**:

- `ImageLookupTool` → returns image URLs of the personalities in your conversation.
- `DetailedSummaryTool` → (optional) returns a more in-depth and up-to-date description of the personalities.

These results will be returned as output from their respective tools and integrated into your generated conversation.

Your Tasks

1. Understand the Tool Protocol

A tool is a struct that conforms to `Tool` and defines:

- **Arguments** → what inputs it needs (usually a `@Generable` type).
- **Output** → what it returns (must conform to `PromptRepresentable`).
- **Metadata** → `name` and `description` to tell the model how and when to use it.

> 💡 The model decides *when* to use your tool. Your job is to provide clear contracts and Guides that make the usage obvious.

---

2. Implement `ImageLookupTool`

- Define the `Arguments` struct. Think about what input this tool requires (for example, participant names).
- Use `@Guide` annotations so the model knows exactly what values to provide.
- Implement the `call(arguments:)` function. This is where you'll write the logic to use the arguments, fetch data from Wikipedia, and return image URLs.

- The output is `[String]` — an array of URL strings pointing to images of your participants.

---

3. (Optional) Implement `DetailedSummaryTool`

- This tool is similar, but instead of returning image URLs it should return detailed text descriptions of participants.
- Add arguments to accept multiple participants.
- Implement the `call(arguments:)` function to fetch Wikipedia extracts and return them as `[String]`.
- This gives you richer, more up-to-date content for your generated personalities.

---

4. Update `WorkshopConstants`

Enable your tools by editing `WorkshopConstants.swift`:

- Add `ImageLookupTool` to the `tools` array (uncomment it).
- Optionally, add `DetailedSummaryTool` once you've implemented it.
- Adjust the **instructions** string to tell the model how it should use these tools.
- Experiment with prompts and sampling modes to see how tool usage changes.

Guiding the Model to Use Tools

- Be explicit in your **instructions**: tell the model when it should use a tool.
- Use `@Guide` annotations in your `Arguments` so the model knows exactly what inputs to provide.
- Clear, specific descriptions and consistent naming will increase the likelihood that the model calls your tools as intended.

⚠️ Notes on Tool Usage in FMF

- Tool usage in FMF can be **finicky** — the model doesn't always call tools exactly how you expect on the first try.
- You'll likely need to experiment with **instructions** and **@Guide descriptions** to steer the model toward correct tool usage.

- This is normal — treat it as trial-and-error: refine instructions, adjust wording, and rerun until the behavior stabilizes.

One Call, Many Results

- FMF can call a tool **once** and return results for **any number of participants**, as long as your `Arguments` and `Output` are designed to handle multiple values.
- For example, if your `Arguments` allow a list of names, your tool can fetch images or descriptions for all participants in a single call.
- This is the **preferred pattern**: one tool call → batched output, rather than multiple repeated calls.

Workshop Challenge

- For a larger challenge, try **combining the two tools into one** that returns both image URLs and detailed descriptions together.
- This would require designing a richer `Output` type (e.g., a struct with `images: [String]` and `summaries: [String]`).
- It's trickier, but it demonstrates how tools can be extended into more domain-specific APIs.

Notes

- Tools make your models **actionable** — they bridge AI output with external APIs.
- In this workshop, your tools bridge the model with **Wikipedia** to bring in images and detailed summaries of participants.
- Output can be simple (`[String]`) or structured (`@Generable` types).
- Experiment with tool combinations to see how they change the generated conversation.

Optional Stretch Goals

- Extend tools to return more structured data (e.g., an `ImageResult` struct).
- Add entirely new tools: e.g., a **QuoteFetcher**, **FunFactTool**, or **TranslationTool**.
- Update `Personality` to call tools automatically as part of its schema.

Next Step

Once your tools are implemented and enabled, rerun the project.
Watch how the model **decides when to call them** and how it integrates results into the conversation!