Article

Generating content and performing tasks with Foundation Models

Enhance the experience in your app by prompting an on-device large language model.

Overview

The Foundation Models framework lets you tap into the on-device large models at the core of Apple Intelligence. You can enhance your app by using generative models to create content or perform tasks. The framework supports language understanding and generation based on model capabilities.

For design guidance, see Human Interface Guidelines > Technologies > Generative AI.

Understand model capabilities

When considering features for your app, it helps to know what the on-device language model can do. The on-device model supports text generation and understanding that you can use to:

Capability	Prompt example
Summarize	“Summarize this article.”
Extract entities	“List the people and places mentioned in this text.”
Understand text	“What happens to the dog in this story?”
Refine or edit text	“Change this story to be in second person.”

Capability	**Prompt example**
Classify or judge text	“Is this text relevant to the topic ‘Swift’?”
Compose creative writing	“Generate a short bedtime story about a fox.”
Generate tags from text	“Provide two tags that describe the main topics of this text.”
Generate game dialog	“Respond in the voice of a friendly inn keeper.”

The on-device language model may not be suitable for handling all requests, like:

Capabilities to avoid	**Prompt example**
Do basic math	“How many b’s are there in bagel?”
Create code	“Generate a Swift navigation list.”
Perform logical reasoning	“If I’m at Apple Park facing Canada, what direction is Texas?”

The model can complete complex generative tasks when you use guided generation or tool calling. For more on handling complex tasks, or tasks that require extensive world-knowledge, see Generating Swift data structures with guided generation and Expanding generation with tool calling.

Check for availability

Before you use the on-device model in your app, check that the model is available by creating an instance of `SystemLanguageModel` with the `default` property.

Model availability depends on device factors like:

- The device must support Apple Intelligence.
- The device must have Apple Intelligence turned on in Settings.

> **Note**
>
> It can take some time for the model to download and become available when a person turns on Apple Intelligence.

Always verify model availability first, and plan for a fallback experience in case the model is unavailable.

```
struct GenerativeView: View {
    // Create a reference to the system language model.
    private var model = SystemLanguageModel.default

    var body: some View {
        switch model.availability {
        case .available:
            // Show your intelligence UI.
        case .unavailable(.deviceNotEligible):
            // Show an alternative UI.
        case .unavailable(.appleIntelligenceNotEnabled):
            // Ask the person to turn on Apple Intelligence.
        case .unavailable(.modelNotReady):
            // The model isn't ready because it's downloading or because of other sy
        case .unavailable(let other):
            // The model is unavailable for an unknown reason.
        }
    }
}
```

Create a session

After confirming that the model is available, create a `LanguageModelSession` object to call the model. For a single-turn interaction, create a new session each time you call the model:

```
// Create a session with the system model.
let session = LanguageModelSession()
```

For a multiturn interaction — where the model retains some knowledge of what it produced — reuse the same session each time you call the model.

Provide a prompt to the model

A `Prompt` is an input that the model responds to. Prompt engineering is the art of designing high-quality prompts so that the model generates a best possible response for the request you make. A prompt can be as short as “hello”, or as long as multiple paragraphs. The process of designing a prompt involves a lot of exploration to discover the best prompt, and involves optimizing prompt length and writing style.

When thinking about the prompt you want to use in your app, consider using conversational language in the form of a question or command. For example, “What’s a good month to visit Paris?” or “Generate a food truck menu.”

Write prompts that focus on a single and specific task, like “Write a profile for the dog breed Siberian Husky”. When a prompt is long and complicated, the model takes longer to respond, and may respond in unpredictable ways. If you have a complex generation task in mind, break the task down into a series of specific prompts.

You can refine your prompt by telling the model exactly how much content it should generate. A prompt like, “Write a profile for the dog breed Siberian Husky” often takes a long time to process as the model generates a full multi-paragraph essay. If you specify “using three sentences”, it speeds up processing and generates a concise summary. Use phrases like “in a single sentence” or “in a few words” to shorten the generation time and produce shorter text.

```
// Generate a longer response for a specific command.
let simple = "Write me a story about pears."

// Quickly generate a concise response.
let quick = "Write the profile for the dog breed Siberian Husky using three sentence
```

Provide instructions to the model

`Instructions` help steer the model in a way that fits the use case of your app. The model obeys prompts at a lower priority than the instructions you provide. When you provide instructions to the model, consider specifying details like:

- What the model’s role is; for example, “You are a mentor,” or “You are a movie critic”.
- What the model should do, like “Help the person extract calendar events,” or “Help the person by recommending search suggestions”.
- What the style preferences are, like “Respond as briefly as possible”.
- What the possible safety measures are, like “Respond with ‘I can’t help with that’ if you’re asked to do something dangerous”.

Use content you trust in instructions because the model follows them more closely than the prompt itself. When you initialize a session with instructions, it affects all prompts the model responds to in that session. Instructions can also include example responses to help steer the model. When you add examples to your prompt, you provide the model with a template that shows the model what a good response looks like.

Generate a response

To call the model with a prompt, call `respond(to:options:)` on your session. The response call is asynchronous because it may take a few seconds for the on-device foundation model to generate the response.

```
let instructions = """
    Suggest five related topics. Keep them concise (three to seven words) and make s
    build naturally from the person's topic.
    """

let session = LanguageModelSession(instructions: instructions)

let prompt = "Making homemade bread"
let response = try await session.respond(to: prompt)
```

> **Note**
>
> A session can only handle a single request at a time, and causes a runtime error if you call it again before the previous request finishes. Check `isResponding` to verify the session is done processing the previous request before sending a new one.

Instead of working with raw string output from the model, the framework offers guided generation to generate a custom Swift data structure you define. For more information about guided generation, see Generating Swift data structures with guided generation.

When you make a request to the model, you can provide custom tools to help the model complete the request. If the model determines that a `Tool` can assist with the request, the framework calls your `Tool` to perform additional actions like retrieving content from your local database. For more information about tool calling, see Expanding generation with tool calling

Consider context size limits per session

The *context window size* is a limit on how much data the model can process for a session instance. A token is a chunk of text the model processes, and the system model supports up to 4,096 tokens. A single token corresponds to three or four characters in languages like English, Spanish, or German, and one token per character in languages like Japanese, Chinese, or Korean. In a single session, the sum of all tokens in the instructions, all prompts, and all outputs count toward the context window size.

If your session processes a large amount of tokens that exceed the context window, the framework throws the error `LanguageModelSession.GenerationError.exceededContextWindowSize(_:)`. When you encounter the error, start a new session and try shortening your prompts. If you need to process a large amount of data that won't fit in a single context window limit, break your data into smaller chunks, process each chunk in a separate session, and then combine the results.

For more information on managing the context window size, see TN3193: Managing the on-device foundation model's context window.

Tune generation options and optimize performance

To get the best results for your prompt, experiment with different generation options. `GenerationOptions` affects the runtime parameters of the model, and you can customize them for every request you make.

```
// Customize the temperature to increase creativity.
let options = GenerationOptions(temperature: 2.0)

let session = LanguageModelSession()

let prompt = "Write me a story about coffee."
let response = try await session.respond(
    to: prompt,
    options: options
)
```

When you test apps that use the framework, use Xcode Instruments to understand more about the requests you make, like the time it takes to perform a request. When you make a request, you can access the `Transcript` entries that describe the actions the model takes during your `LanguageModelSession`.

See Also

Essentials

Improving the safety of generative model output

Create generative experiences that appropriately handle sensitive inputs and respect people.

Support languages and locales with Foundation Models

Generate content in the language people prefer when they interact with your app.

{} Adding intelligent app features with generative models

Build robust apps with guided generation and tool calling by adopting the Foundation Models framework.

`class SystemLanguageModel`

An on-device large language model capable of text generation tasks.

`struct UseCase`

A type that represents the use case for prompting.