

Apple Intelligence & IA

iOS 18 & iOS 26

Jour 3 - Atelier Swift Avancé

Agenda du Jour (3h-3h30)

- Partie 1 : Architecture Apple Intelligence
- Partie 2 : Foundation Models & On-Device AI
- Partie 3 : App Intents & Siri Integration
- Pause (15min) ☕
- Partie 4 : Writing Tools & Text APIs
- Partie 5 : Outils IA pour Développeurs
- Conclusion & Questions

Partie I



Architecture Apple Intelligence

Qu'est-ce qu'Apple Intelligence ?

Définition technique :

- Suite de modèles d'IA générative on-device (AFM iOS 26+)
- Intégration système profonde (iOS 18+, iOS 26)
- Architecture multi-niveaux pour confidentialité

Pas juste un feature :

- C'est une capability du système
- Comme UIKit, AVFoundation, Core ML
- Disponible pour toutes les apps via APIs

Changement de paradigme :

- L'app devient "proactive" et "conversationnelle"
- L'utilisateur peut parler naturellement à l'app
- L'app suggère des actions contextuelles

Concepts Fondamentaux : LLM & RLM

LLM (Large Language Model) :

- Modèle entraîné sur texte massif
- Génère du texte de manière probabiliste
- Exemples : GPT-4, Claude, Llama

RLM (Reasoning Language Model) :

- LLM optimisé pour le raisonnement
- Meilleur sur tâches logiques/complexes
- Exemples : GPT-4o, Claude Sonnet 4.5

Paramètres :

- Poids du modèle (ex: 3B, 7B, 70B)
- Plus de paramètres = plus capable
- Mais aussi plus lent et gourmand
- Apple : ~3B paramètres (2-bit quantized)

Tokens :

- Unité de traitement (≈ 0.75 mot)
- "Bonjour" = 1 token, "Hello world" = 2 tokens
- Vitesse : tokens/sec (ex: 30 t/s iPhone 15 Pro)

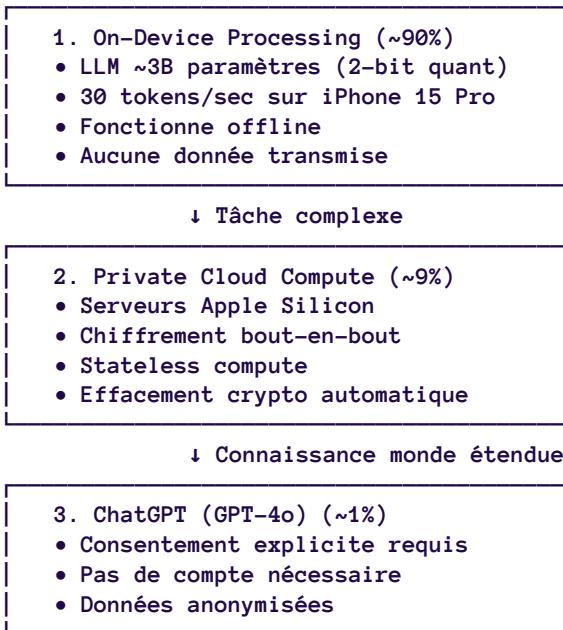
Context Window :

- Nombre max de tokens en mémoire
- Exemples : 8K, 32K, 128K, 200K
- Plus grand = plus de contexte
- Apple on-device : ~4.096 tokens

Quantization :

- Réduction précision (32-bit \rightarrow 2-bit)
- Divise taille par 16x
- Légère perte qualité, gain énorme performance

Apple Foundation Model : Architecture à 3 Niveaux¹



¹ <https://machinelearning.apple.com/research/introducing-apple-foundation-models>

On-Device : Capacités & Limites

Excellences du modèle on-device :

- Résumé de texte (jusqu'à ~4000 tokens)
- Extraction d'informations structurées
- Classification et tagging de contenu
- Génération de texte court/moyen
- Traduction entre langues courantes

Limitations connues :

- Raisonnement mathématique complexe
- Génération de code sophistiqué
- Connaissance factuelle étendue
- Contextes très longs (>8K tokens)
- Tâches multimodales complexes

Pour développeurs :

- Testez toujours on-device d'abord
- Escaladez à PCC uniquement si nécessaire

Private Cloud Compute : Garanties²

Architecture de sécurité unique :

1. Stateless Computation

- Aucune donnée stockée, jamais
- Mémoire effacée cryptographiquement après traitement

2. Enforceable Guarantees

- Garanties techniques, pas juste policies
- Vérifiables indépendamment

1. No Privileged Runtime Access

- Même les employés Apple ne peuvent pas accéder
- Pas de backdoor possible

2. Non-Targetability

- Impossible de cibler un utilisateur spécifique
- Routing aléatoire entre serveurs

1. Verifiable Transparency

- Code source inspectable par chercheurs
- Bug bounty jusqu'à \$1M

² Apple extends its privacy leadership...

ChatGPT Integration

Déclenchement automatique :

- Requêtes nécessitant connaissance monde étendue
- Système détecte automatiquement
- Popup de consentement utilisateur

Contrôle utilisateur :

Settings → Apple Intelligence → ChatGPT Extension

- Peut être désactivé globalement
- Consentement à chaque utilisation
- Pas de compte ChatGPT requis

Pour les dév :

- Pas d'API directe ChatGPT dans vos apps
- Utilisez Foundation Models framework
- Le système route automatiquement si nécessaire

Partie 2

Foundation Models : Vue d'ensemble

Disponibilité :

- iOS 26, iPadOS 26, macOS Tahoe 26
- visionOS 26

Capacités principales :

- Génération de texte libre
- Génération structurée (@Generable)
- Streaming en temps réel
- Tool calling (fonctions externes)

Requirements :

- iPhone 15 Pro / 16 / 16 Pro
- iPad M1+, Mac Apple Silicon
- 8GB RAM minimum

Gratuit : Aucun coût API, tout on-device

Génération de Texte Basique

```
import FoundationModels

func generateText(prompt: String) async throws -> String {
    let session = LanguageModelSession()
    let response = try await session.respond(to: prompt)
    return response.content
}

// Utilisation
let explanation = try await generateText(
    prompt: "Explique les optionals Swift en 3 phrases simples"
)
print(explanation)
```

Simple mais puissant :

- Génération synchrone ou async
- Contexte automatiquement géré
- Température et max_tokens configurables

Génération Structurée avec @Generable³

```
import FoundationModels

@Generable
struct MeetingSummary: Equatable {
    @Guide(description: "Sujet principal de la réunion")
    let subject: String

    @Guide(description: "Participants présents")
    let participants: [String]

    @Guide(description: "Points principaux discutés")
    let keyPoints: [String]

    @Guide(description: "Décisions prises")
    let decisions: [String]

    @Guide(description: "Actions à suivre avec responsables")
    let actionItems: [String]

    @Guide(description: "Date et sujet de la prochaine réunion si mentionnée")
    let nextMeeting: String?
}
```

³ Deep dive into the Foundation Models framework: WWDC 2025 Session 301

Génération Structurée avec @Generable

```
func analyzeMeetingNotes(notes: String) async throws -> MeetingSummary {  
    let session = LanguageModelSession()  
    let response = try await session.respond(  
        to: "Analyse ces notes de réunion: \(notes)",  
        generating: MeetingSummary.self  
    )  
    return response.content  
}
```

@Generable : Types Supportés

Value types simples :

```
@Generable
struct SimpleData {
    let name: String
    let age: Int
    let score: Double
    let isActive: Bool
}
```

Collections :

```
@Generable
struct ComplexData {
    let items: [String]
    let scores: [Int]
    let metadata: [String: String]
}
```

Nested structures :

```
@Generable
struct Author {
    let name: String
    let bio: String
}

@Generable
struct Book {
    let title: String
    let author: Author // Nested!
    let chapters: [String]
}
```

Streaming pour UX Réactive

```
import FoundationModels
import SwiftUI

struct StreamingView: View {
    @State private var partialSummary: ArticleSummary?
    @State private var isGenerating = false

    func generateSummary(text: String) async {
        isGenerating = true
        defer { isGenerating = false }

        let session = LanguageModelSession()
        let stream = session.streamResponse(
            to: "Résume cet article: \(text)",
            generating: ArticleSummary.self
        )

        for try await partial in stream {
            // partial contient des propriétés progressivement remplies
            partialSummary = partial
        }
    }
}
```

```
var body: some View {
    VStack {
        if let summary = partialSummary {
            Text(summary.title ?? "Génération...")
            Text(summary.mainPoints?.joined(separator: "\n") ?? "")
        }
    }
}
```

Tool Calling : Étendre les Capacités

```
import FoundationModels

struct WeatherTool: Tool {
    let name = "get_weather"
    let description = "Obtient la météo actuelle pour une ville"

    @Generable
    struct Arguments {
        @Guide(description: "Nom de la ville")
        let city: String
    }

    func call(arguments: Arguments) async throws -> ToolOutput {
        // Appel API météo réelle
        let weather = try await WeatherAPI.fetch(city: arguments.city)
        return ToolOutput("Météo à \(arguments.city): \(weather.temp)°C, \(weather.condition)")
    }
}

func chatWithTools(message: String) async throws -> String {
    let session = LanguageModelSession(tools: [WeatherTool()])
    let response = try await session.respond(to: message)
    return response.content
}

// Utilisation
let answer = try await chatWithTools(message: "Quel temps fait-il à Paris ?")
// Le modèle détecte qu'il doit appeler WeatherTool, le fait automatiquement
// Puis génère une réponse avec les données réelles
```

Vérifiez la disponibilité d'AFM

```
import FoundationModels

func checkAvailability() {
    switch SystemLanguageModel.default.availability {
        case .available:
            print("✅ Foundation Models disponible")

        case .unavailable(let reason):
            switch reason {
                case .appleIntelligenceNotEnabled:
                    print("⚠️ Apple Intelligence désactivé dans Settings")
                    // Montrer UI pour guider l'utilisateur

                case .deviceNotEligible:
                    print("❌ Appareil non compatible (iPhone 15 Pro+ requis)")
                    // Fallback vers alternative

                case .modelNotReady:
                    print("⏳ Modèle pas encore téléchargé")
                    // Afficher message "En cours de téléchargement"

                @unknown default:
                    print("❓ Raison inconnue")
            }
    }
}
```

Partie 3

App Intents & Siri Integration

App Intents : Pourquoi ?

Avant App Intents :

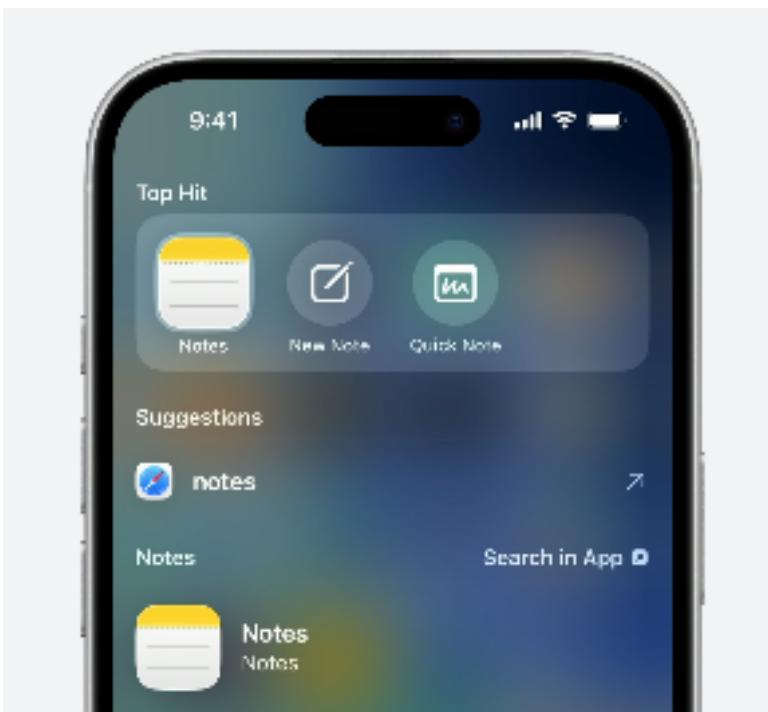
Utilisateur → Ouvre l'app → Navigation → Action

Avec App Intents :

Utilisateur → "Crée une note dans MyApp" → Fait !

Surfaces d'exposition :

-  Siri (vocal et texte)
-  Spotlight Search
-  Shortcuts app
-  Control Center
-  Action Button
-  Proactive
-  Apple Intelligence



Structure d'un App Intent

```
import AppIntents

struct CreateTaskIntent: AppIntent {
    // 1. Métdonnées
    static var title: LocalizedStringResource = "Créer une tâche"
    static var description = IntentDescription("Crée une nouvelle tâche")
    static var openAppWhenRun: Bool = false // Ne lance pas l'app

    // 2. Paramètres
    @Parameter(title: "Titre de la tâche")
    var taskTitle: String

    @Parameter(title: "Liste", default: TaskList.inbox)
    var list: TaskList

    @Parameter(title: "Date limite")
    var dueDate: Date?

    // 3. Phrase suggérée pour Siri
    static var parameterSummary: some ParameterSummary {
        Summary("Créer \$(taskTitle) dans \$(list)")
    }

    // 4. Exécution
    func perform() async throws -> some IntentResult {
        try await TaskManager.shared.createTask(
            title: taskTitle,
            list: list,
            dueDate: dueDate
        )
        return .result(dialog: "Tâche créée !")
    }
}
```

App Entity : Objets Métier

```
struct TaskListQuery: EntityQuery {
    func entities(for identifiers: [UUID]) async throws -> [TaskList] {
        TaskDataManager.shared.lists
            .filter { identifiers.contains($0.id) }
            .map { TaskList(id: $0.id, name: $0.name, color: $0.color) }
    }

    func suggestedEntities() async throws -> [TaskList] {
        TaskDataManager.shared.lists
            .map { TaskList(id: $0.id, name: $0.name, color: $0.color) }
    }
}
```

App Shortcuts Provider

```
import AppIntents

struct MyAppShortcutsProvider: AppShortcutsProvider {
    @AppShortcutsBuilder
    static var appShortcuts: [AppShortcut] {
        AppShortcut(
            intent: CreateTaskIntent(),
            phrases: [
                "Créer une tâche dans \(.applicationName)",
                "Ajouter à faire dans \(.applicationName)",
                "Create task in \(.applicationName)",
                "Add todo in \(.applicationName)"
            ],
            shortTitle: "Nouvelle tâche",
            systemImageName: "plus.circle"
        )
        AppShortcut(
            intent: ShowTodayTasksIntent(),
            phrases: [
                "Mes tâches du jour dans \(.applicationName)",
                "What's on my list in \(.applicationName)"
            ],
            shortTitle: "Tâches aujourd'hui",
            systemImageName: "calendar"
        )
    }
}
```

Activation :

```
swift
// Dans votre app delegate ou scene delegate
MyAppShortcutsProvider.updateAppShortcutParameters()
```

Assistant Schemas (iOS 18+)

```
import AppIntents

@AssistantIntent(schema: .system.search)
struct SearchPhotosIntent: AppIntent {
    static var title: LocalizedStringResource = "Rechercher photos"

    @Parameter(title: "Critères de recherche")
    var criteria: String

    func perform() async throws -> some IntentResult {
        let results = await PhotoManager.shared.search(query: criteria)

        // Retourner résultats pour Apple Intelligence
        return .result(
            value: results,
            dialog: "J'ai trouvé \(results.count) photos"
        )
    }
}
```

Assistant Schemas disponibles :

- **.system.search** - Recherche de contenu
- **.system.open** - Ouvrir un élément
- **.system.settings** - Paramètres app
- **.photos.search** - Recherche photos spécifique
- **.messages.send** - Envoyer messages

Partie 4

Writing Tools & Text APIs

Writing Tools : Intégration Automatique

```
import SwiftUI

struct EditorView: View {
    @State private var text = ""

    var body: some View {
        VStack {
            // ✅ Writing Tools activé par défaut
            TextEditor(text: $text)
                .frame(height: 200)

            // ⚠️ Désactiver si nécessaire
            TextField("Sans outils", text: $text)
                .writingToolsBehavior(.disabled)

            // 📁 Mode limité (copier/partager uniquement)
            TextEditor(text: $text)
                .writingToolsBehavior(.limited)

            // 🖊️ Comportement complet (default)
            TextEditor(text: $text)
                .writingToolsBehavior(.complete)
        }
    }
}
```

Actions disponibles pour l'utilisateur :

- Rewrite (Professional / Concise / Friendly)
- Proofread (Grammaire & orthographe)
- Summarize (Key Points / Paragraphe)

Writing Tools : Delegates UIKit

```
import UIKit

class MyTextViewController: UIViewController, UITextViewDelegate {
    let textView = UITextView()

    override func viewDidLoad() {
        super.viewDidLoad()
        textView.delegate = self
    }

    // Appelé quand Writing Tools commence
    func textViewWritingToolsWillBegin(_ textView: UITextView) {
        print("⚠️ Writing Tools commence")

        // Désactiver sync cloud temporairement
        CloudSyncManager.shared.pauseSync()

        // Désactiver auto-save
        autoSaveTimer?.invalidate()
    }

    // Appelé quand Writing Tools termine
    func textViewWritingToolsDidEnd(_ textView: UITextView) {
        print("✅ Writing Tools terminé")

        // Réactiver sync
        CloudSyncManager.shared.resumeSync()

        // Réactiver auto-save
        scheduleAutoSave()
    }

    // Optionnel : Ignorer certains ranges
    func textView(
        _ textView: UITextView,
        writingToolsIgnoredRangesIn range: NSRange
    ) -> [NSValue] {
        // Ignorer les code blocks, markdown syntax, etc.
        return findCodeBlockRanges(in: textView.text)
    }
}
```

Natural Language Framework

```
import NaturalLanguage

// 1. Extraction d'entités nommées
func extractEntities(from text: String) -> [(text: String, type: NLTag)] {
    var entities: [(String, NLTag)] = []
    let tagger = NLTagger(tagSchemes: [.nameType])
    tagger.string = text

    let tags: [NLTag] = [.personalName, .placeName, .organizationName]

    tagger.enumerateTags(
        in: text.startIndex..
```

```
// Exemple
let text = "Clément SAUVAGE remplace Tim COOK à la tête d'Apple à Cupertino pour Apple"
let entities = extractEntities(from: text)
// [("Clément SAUVAGE", .personalName), ("Tim COOK", .personalName), ("Cupertino", .placeName), ("Apple", .organizationName)]
```

Résumé Automatique

```
import NaturalLanguage

func summarize(text: String, sentenceCount: Int = 3) -> String {
    let tagger = NLTagger(tagSchemes: [.tokenType])
    tagger.string = text

    var sentences: [String] = []
    tagger.enumerateTags(
        in: text.startIndex..
```

Partie 5



Outils IA pour Développeurs

Xcode Predictive Code Completion

Status :

- Disponible : Xcode 16.0+ (Predictive Completion)
- Swift Assist : Annoncé WWDC24, disponible (réellement) dans Xcode 26

Activation :

Xcode → Settings → Components
→ Predictive Code Completion Model
→ Download & Enable

Caractéristiques :

- Modèle ML ~2GB on-device
- Apple Silicon Mac avec 16GB RAM minimum
- Completion contextuelle basée sur projet
- Génération depuis commentaires

```
// Create a function to fetch user data from API
// Le model génère automatiquement :
func fetchData() async throws -> User {
    let url = URL(string: "https://api.example.com/user")!
    let (data, _) = try await URLSession.shared.data(from: url)
    return try JSONDecoder().decode(User.self, from: data)
}
```

ChatGPT pour Développement Swift

Cas d'usage excellents :

1. Génération de code boilerplate

Prompt: "Create a SwiftData model for a Book with title, author, publicationDate, and rating. Include sample data."

2. Décodage JSON vers Codable

Prompt: "Convert this JSON to Swift Codable struct:
{ "user": { "name": "John", "age": 30, "email": "..." } }

1. Génération de tests unitaires

Prompt: "Generate XCTest unit tests for this ViewModel:
[paste code]"

2. Debugging et explication

Prompt: "Explain why this code crashes: [paste crash log]"
[.column]

3. Optimisation performance

Prompt: "Optimize this SwiftUI View for better performance: [paste code]"

Prompt Engineering pour Swift

Template efficace :

[PERSONA]

Act as a senior iOS developer with 10+ years Swift experience.

[CONTEXT]

I'm building a SwiftUI app using MVVM architecture, SwiftData for persistence, and async/await for networking.

[TASK]

Create a ViewModel for fetching and displaying a list of products from this API endpoint: <https://api.example.com/products>

[CONSTRAINTS]

- Use `@Observable` (iOS 17+)
- Include error handling with custom errors
- Add loading states
- Use Swift 6 strict concurrency

[FORMAT]

Provide complete code with documentation comments.

Claude Code : Coding Agentique

Qu'est-ce que c'est ? :

- Outil de coding agentique terminal-native
- Lancé par Anthropic en mai 2025
- Fenêtre contexte 200K tokens

Capabilities :

- Construire features complètes from scratch
- Debugger automatiquement avec execution
- Lire codebases existantes (context aware)
- Intégration MCP (Google Drive, Jira, etc.)
- Refactoring intelligent multi-fichiers

Installation :

```
brew install claude  
claude login
```

Usage :

```
cd MyiOSProject  
claude "Add SwiftData persistence to this app"
```

LM Studio : LLMs Locaux GUI

Installation :

```
bash  
brew install lm-studio
```

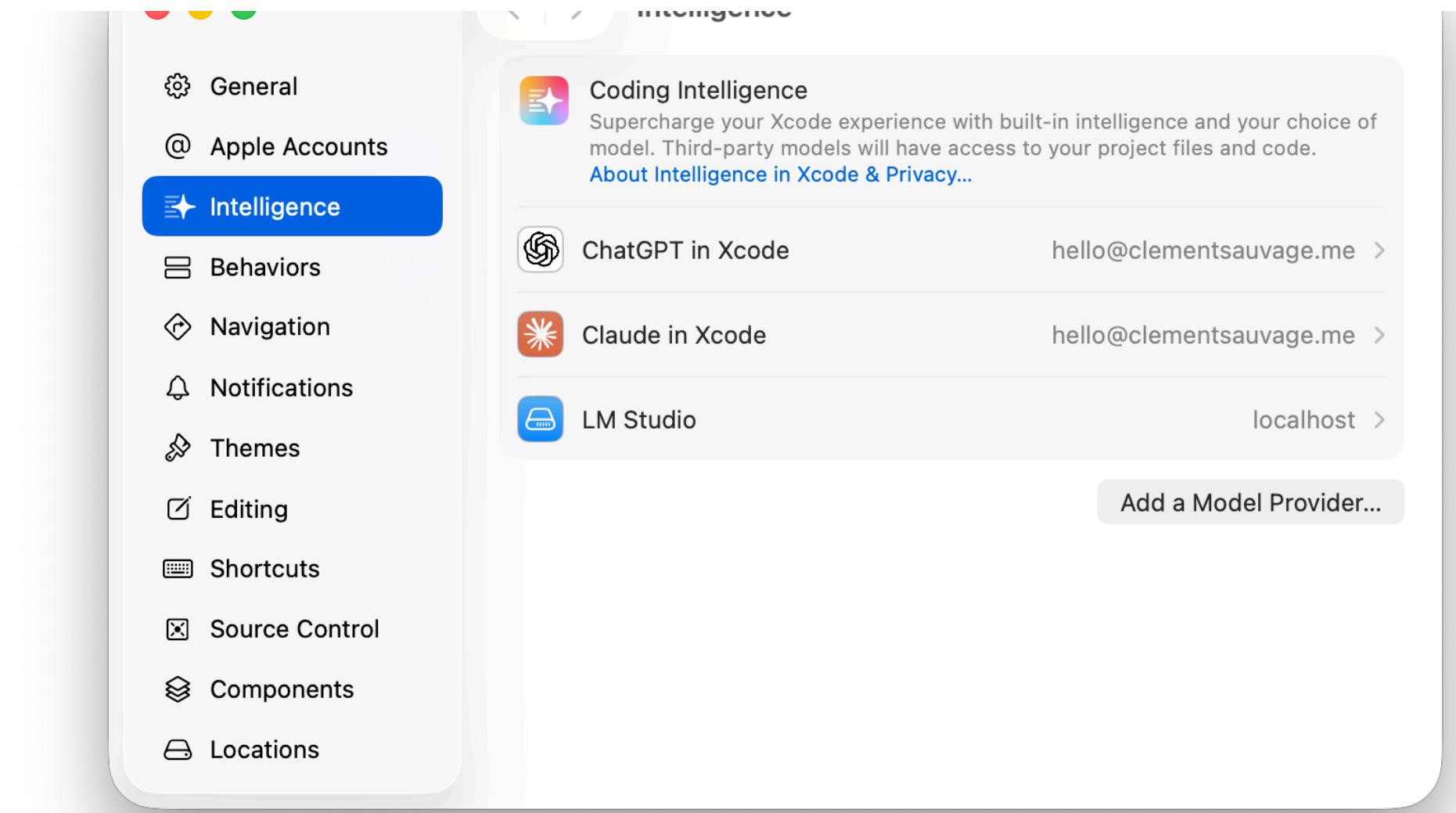
Features :

- Interface graphique intuitive
- Découverte de modèles (HuggingFace)
- Serveur API local (compatible OpenAI)
- Inference GPU-accelerated (Metal)
- Gratuit

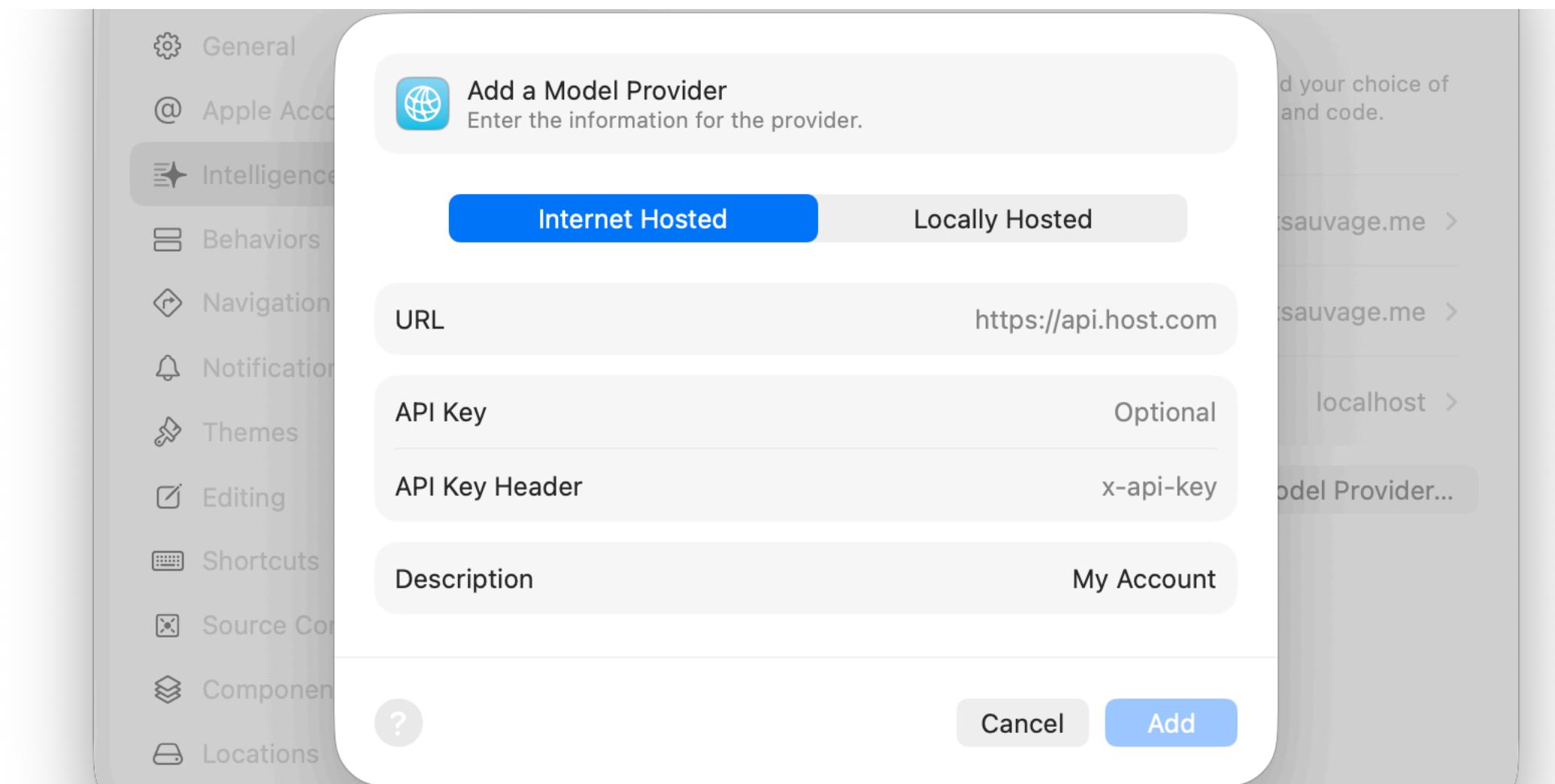
API Usage :

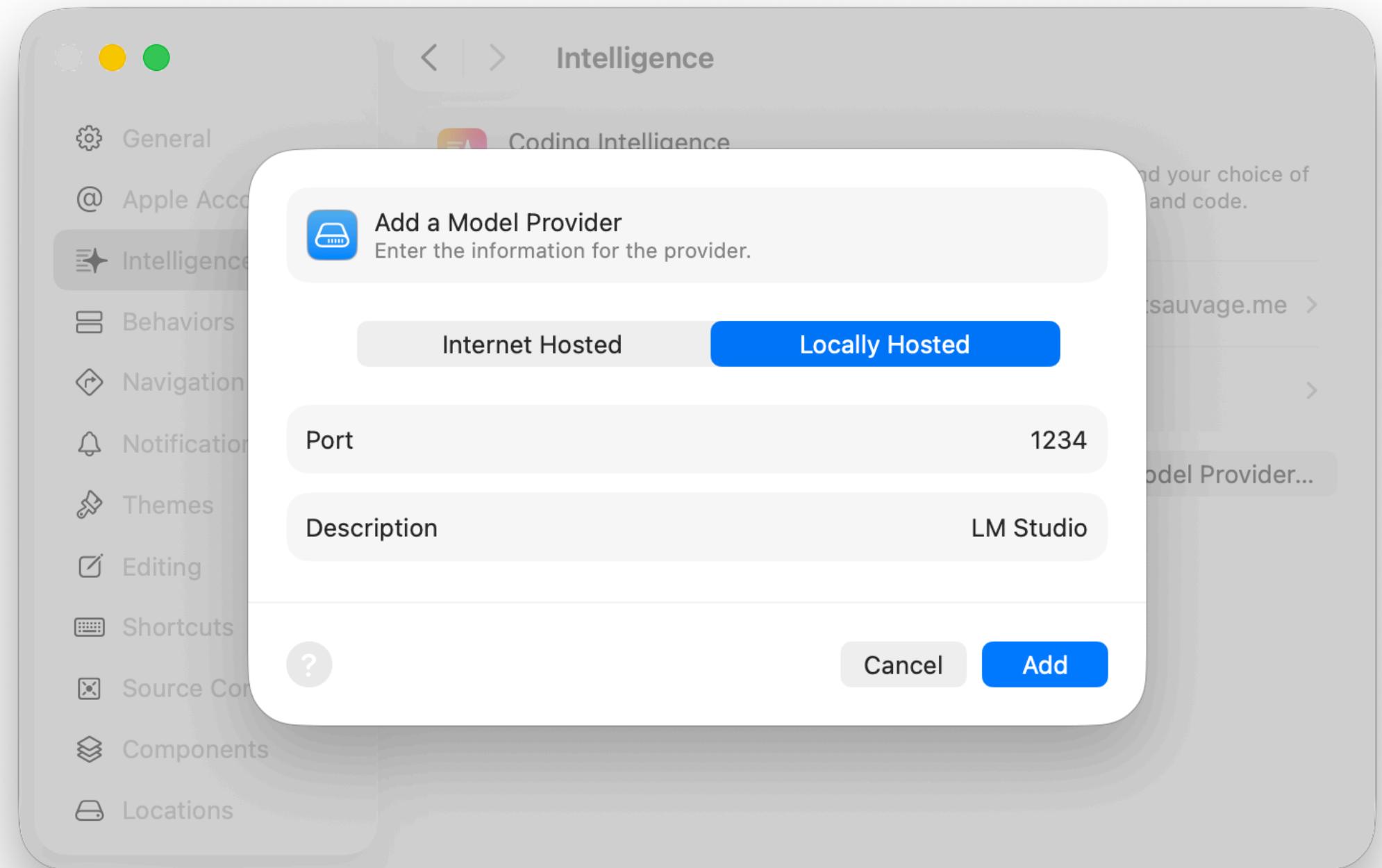
```
bash  
curl http://localhost:1234/v1/chat/completions  
\  
-H "Content-Type: application/json" \  
-d '{"messages": [{"role": "user",  
"content": "Explain Swift actors"}]}'
```

Utilisation dans Xcode



Utilisation dans Xcode





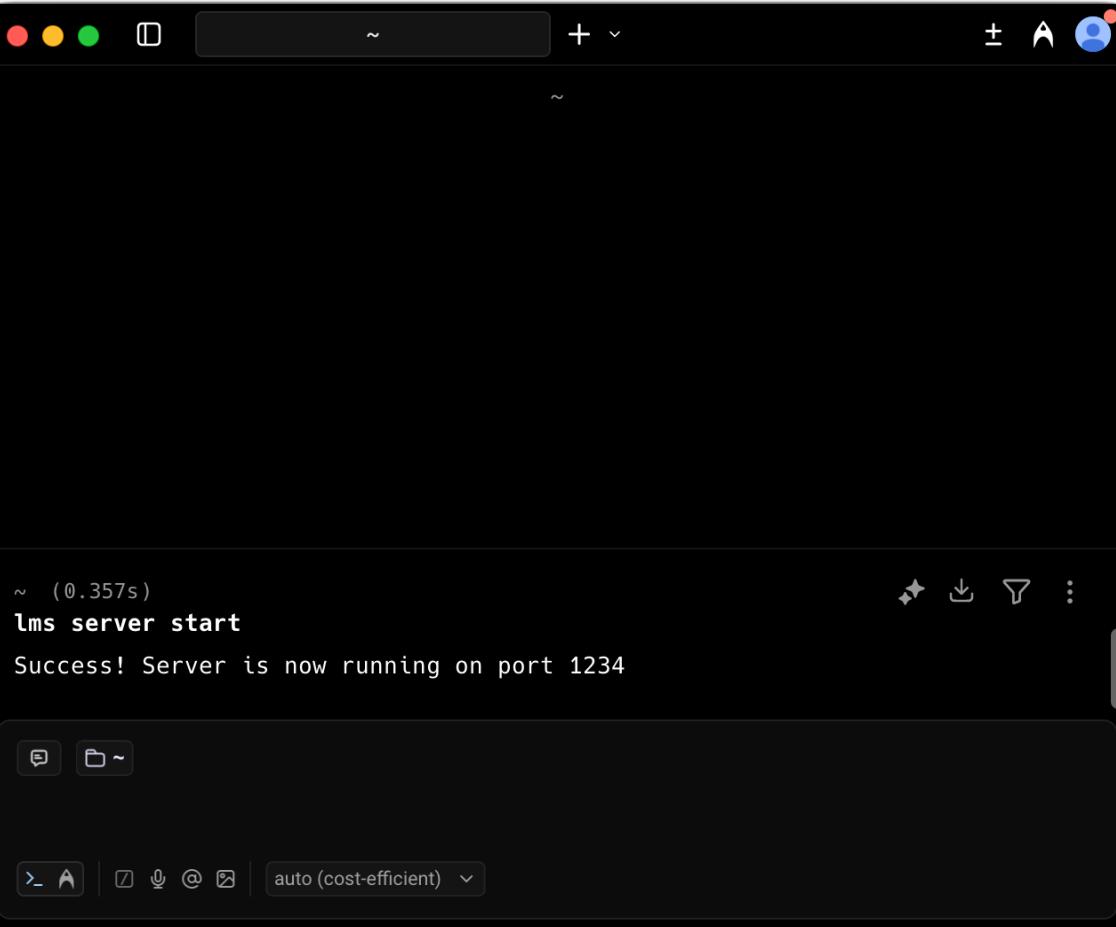


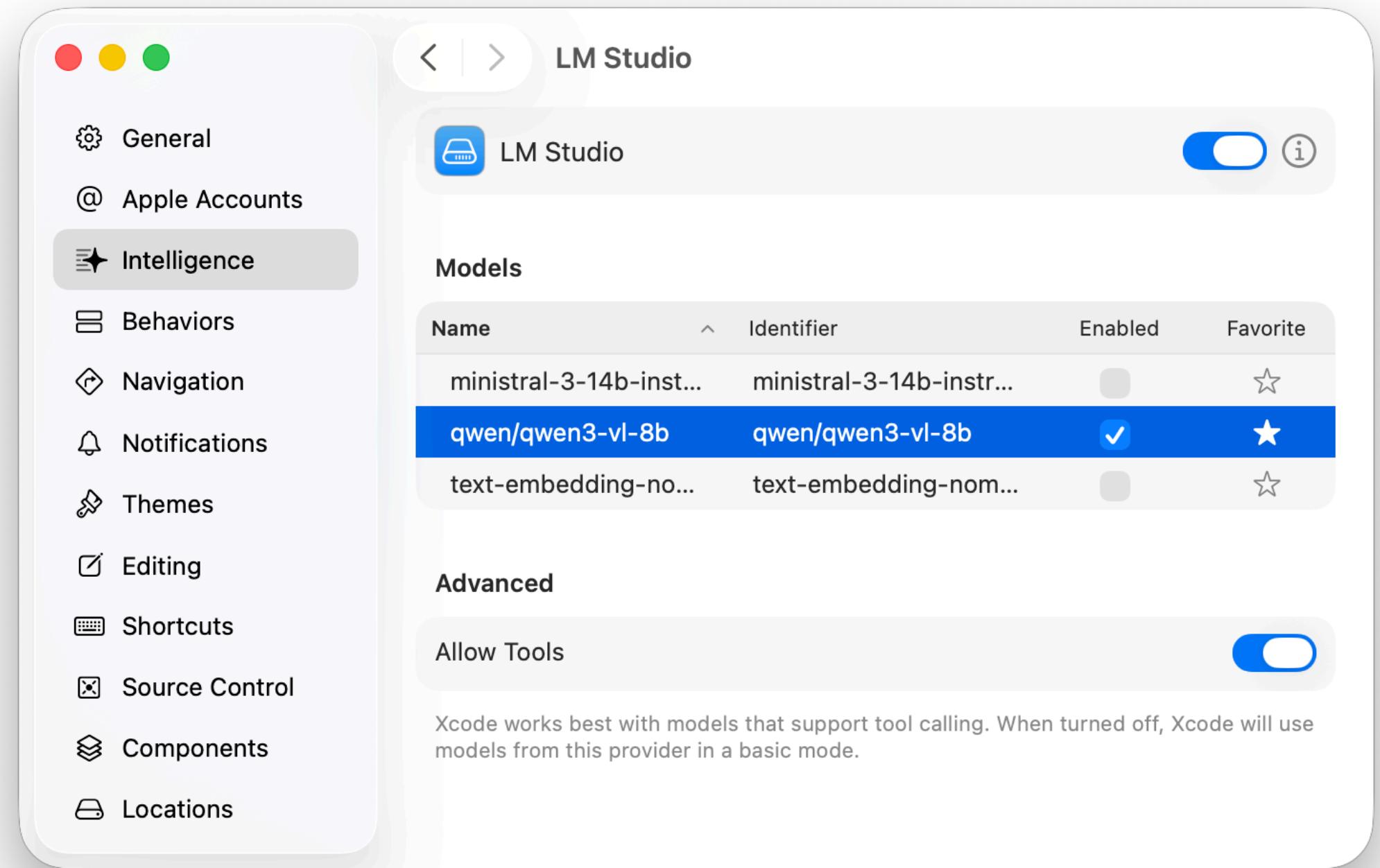
Unable to load models for LM Studio

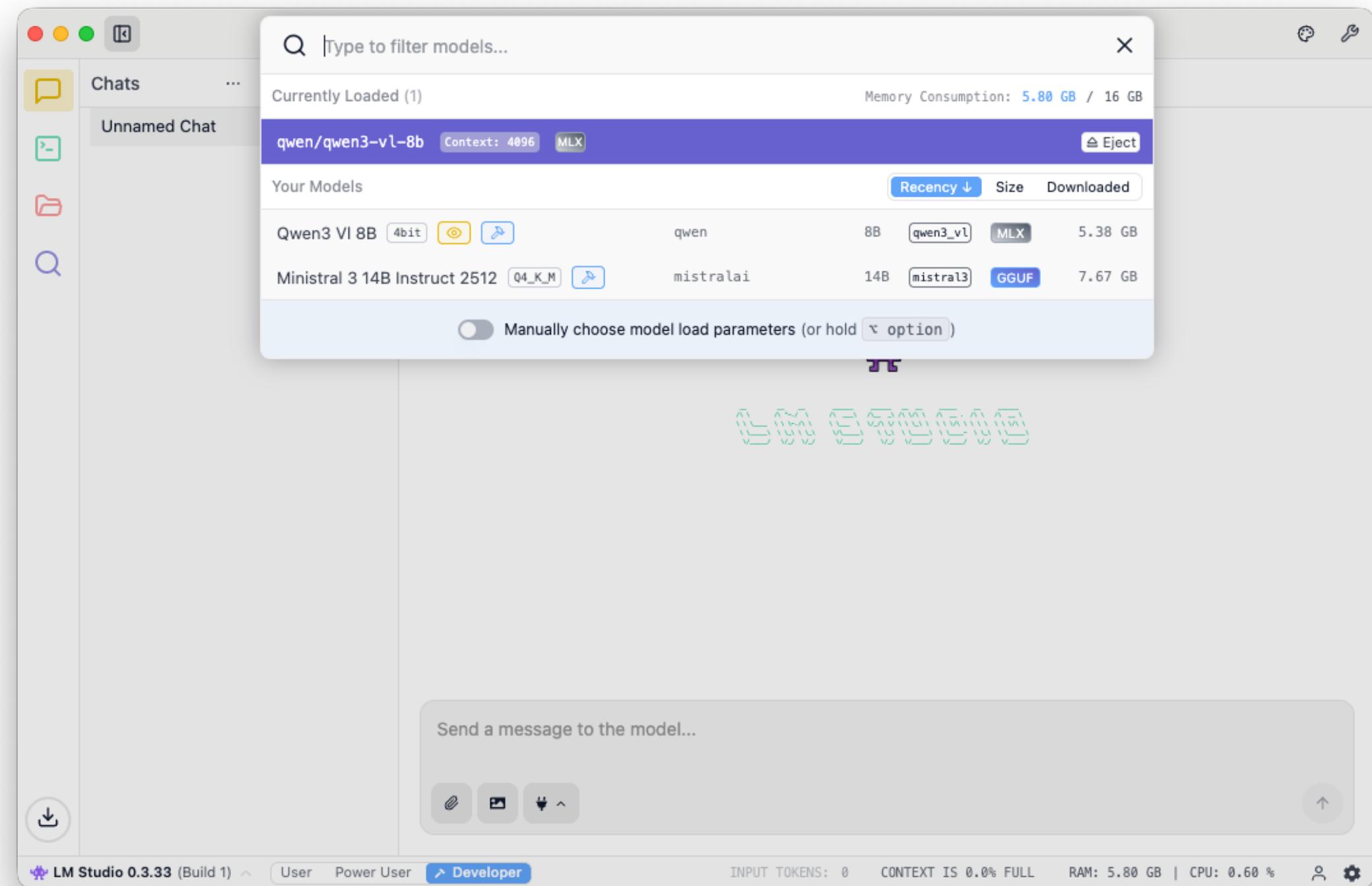
Check your internet connection and try again.

Refresh

```
lms server start  
# Puis quand on a fini  
lms server stop
```







GGUF vs MLX : Comprendre la Différence

GGUF :

- Format universel basé sur llama.cpp
- Compatible macOS / Windows / Linux
- Très léger grâce aux quantizations (Q2-Q8)
- Excellentes perfs CPU, bonnes perfs GPU
- Large écosystème, très stable dans LM Studio

Limites GGUF :

- Perte de qualité possible selon la quantization
- Pas optimisé à 100% pour Apple Silicon
- Conçu uniquement pour l'inférence

MLX :

- Le format Apple Silicon
- Framework créé par Apple
- Exploite GPU + Neural Engine
- Performances supérieures sur Mac
- Idéal pour les modèles 7B-13B

Limites MLX :

- Fonctionne uniquement sur Mac
- Moins de quantizations disponibles
- Demande plus de VRAM

Quand utiliser GGUF ? :

- Multi-plateforme (Windows/Linux)
- Modèles très quantifiés (Q2-Q4)
- Écosystème large et stable
- Besoin de compatibilité maximale

Quand utiliser MLX ? :

- Développement exclusif Mac
- Modèles 7B-13B non quantifiés
- Besoin de performances maximales
- Exploitation Neural Engine

Ollama : LLMs en CLI

Installation :

```
brew install ollama
```

```
brew services start ollama
```

Commandes essentielles :

```
# Lister modèles disponibles  
ollama list
```

```
# Télécharger un modèle  
ollama pull qwen/qwen3-v1-8b
```

```
# Exécuter interactivement  
ollama run qwen/qwen3-v1-8b
```

```
# One-shot  
ollama run qwen/qwen3-v1-8b "Explain Swift Sendable protocol"
```

```
# API serveur (localhost:11434)  
curl http://localhost:11434/api/generate -d '{  
    "model": "qwen3-v1-8b",  
    "prompt": "Write a SwiftUI login view"  
}'
```

Cursor IDE pour iOS Development

Qu'est-ce que Cursor ? :

- IDE basé sur VS Code
- IA native intégrée (GPT-5.1, Claude Sonnet 4.5)
- Autocomplete AI-powered
- Chat contextuel
- Edit via prompts

Raccourcis essentiels :

- Tab : Accepter autocomplete
- Cmd+K : Éditer sélection via prompt
- Cmd+L : Ouvrir chat

The screenshot shows the homepage of the XcodeBuildMCP website. At the top, there's a navigation bar with links for Features, Installation, Usage Examples, Contributing, and GitHub. Below the navigation is a banner with the text "AI-Powered Xcode Automation". The banner features a logo with a hammer and gear icon, followed by the text "XcodeBuild MCP". Below the banner are buttons for "Get Started" and "View on GitHub". Underneath, there's a section titled "Powerful Xcode Integration" with the subtext "Everything you need to integrate Xcode workflows with AI assistants and automation tools." At the bottom, there are three large buttons with icons: a lightning bolt, a greater than sign, and a shield.

The screenshot shows the GitHub repository page for XcodeBuildMCP. The repository has 2938 stars, 135 forks, and is version v1.14.1. The main tab is selected, showing 19 branches and 55 tags. The commit history lists 676 commits from the user cameroncooke, starting with "Ignore derived data" (commit 015c14d) and ending with "Create LICENSE" (commit 4ce793). The commits are organized into several folders like .claude/agents, .cursor, .github, .vscode, build-plugins, docs, example_projects, scripts, src, .axe-version, .cursrrules, .gitignore, .prettierignore, .prettierrc.js, AGENTS.md, CHANGELOG.md, CLAUDE.md, CODE_OF_CONDUCT.md, Dockerfile, and LICENSE. On the right side, there's an "About" section with details about the project, including its purpose (integrating AI assistants with Xcode), links to its website and GitHub pages, and various GitHub statistics like 2.9k stars, 13 watching, and 135 forks. There are also sections for Releases (with a latest release at v1.14.1), Sponsor this project, and Contributors.

Best Practices : IA au Service du Dev

Règle d'or :

"L'IA accélère un senior, elle ne remplace pas un senior."

DO :

- Review tout code généré par IA
- Comprendre avant d'utiliser
- Tester extensively les suggestions
- Utiliser IA pour boilerplate et tests
- Apprendre des explications IA

DON'T :

- Copier-coller sans comprendre
- Faire confiance aveuglément
- Ignorer les warnings du compilateur
- Utiliser IA pour crypto/sécurité critique
- Remplacer code review humain

Conclusion

Apple Intelligence & IA pour Développeurs

Messages Clés

Apple Intelligence :

- Architecture à 3 niveaux (on-device / PCC / ChatGPT)
- Foundation Models framework = accès direct au LLM
- Privacy-first avec garanties vérifiables

App Intents :

- "Anything your app does should be an App Intent"
- Exposition dans Siri, Spotlight, Shortcuts, etc.
- Proactive suggestions by AI

Writing Tools & Text APIs :

- Intégration automatique dans text fields
- Natural Language framework pour analyse
- Résumé et extraction structurée

Outils IA Dev :

- ChatGPT / Claude Code pour génération
- LM Studio / Ollama pour LLMs locaux
- Cursor IDE pour workflow augmenté

L'IA n'est pas Magique

"L'IA est un super assistant, pas un remplaçant."

Réalités :

- L'IA fait des erreurs (hallucinations)
- Elle nécessite supervision humaine
- Elle excelle sur patterns connus
- Elle struggle sur nouveautés

Rôle du développeur :

- Architecte et reviewer
- Validator et testeur
- Curator de qualité
- Gardien de la sécurité

Ressources pour Aller Plus Loin

Documentation Apple :

- developer.apple.com/apple-intelligence/
- developer.apple.com/documentation/FoundationModels
- developer.apple.com/documentation/appintents
- security.apple.com/documentation/private-cloud-compute

WWDC Sessions :

- "Meet the Foundation Models framework" (WWDC25)
- "Bring your app to Siri" (WWDC24)
- "Get started with Writing Tools" (WWDC24)
- "Get to know App Intents" (WWDC25)

Outils & Communities :

- lmstudio.ai (LLMs locaux GUI)
- ollama.com (LLMs CLI)
- cursor.com (IDE AI-powered)
- anthropic.com/cl Claude-code

Merci ! Questions ?

Prochaines étapes :

- Intégrez App Intents dans vos apps
- Testez Foundation Models
- Adoptez les outils IA dans votre workflow
- Partagez vos apprentissages !

Restons en contact :

- Twitter/X : [@clementsauvage](https://twitter.com/clementsauvage)
- GitHub : github.com/csauvage

Au travail !