Integração LLM Offline + Raycast: Guia Completo para macOS Silicon

Versão: 1.0
Plataforma: macOS Silicon (M1/M2/M3/M4)
Data: 2025-10-22

Objetivo

Integrar um **LLM rodando 100% offline** no Mac com o **Raycast** para criar workflows automatizados poderosos: coleta de dados do sistema, análise com IA local, tradução, code review, geração de commits e muito mais — tudo sem depender de APIs externas.

1. Estruturar LLM Offline no macOS Silicon

1.1. Requisitos

Item	Especificação
Hardware	Mac com Apple Silicon (M1/M2/M3/M4)
RAM mínima	16GB (recomendado 32GB para modelos maiores)
Storage	10-50GB para modelos

1.2. Opções de Aplicativos

Opção 1: LM Studio (Recomendado para Iniciantes)

O que é: Interface gráfica para baixar, gerenciar e conversar com LLMs locais.

URL: https://lmstudio.ai

Vantagens:

- Interface visual intuitiva
- Gerenciamento fácil de modelos
- Servidor local OpenAI-compatible
- Ideal para explorar modelos

Instalação:

1. Baixar de https://lmstudio.ai
2. Arrastar para pasta Aplicativos

3. Abrir e permitir nas configurações de segurança

Uso:

Baixar Modelo:

1. Aba de busca (🔍)
2. Procurar `Meta-Llama-3-8B-Instruct-GGUF`
3. Baixar versão quantizada (**Q4** ou **Q5**)
4. Modelos "Instruct" são otimizados para seguir comandos

Iniciar Servidor Local:

1. Aba **Local Server** (<->)
2. Selecionar modelo baixado
3. Clicar **Start Server**
4. API disponível em `http://localhost:1234`
5. Compatível com formato OpenAI API

Opção 2: Ollama (Recomendado para Automação/Raycast)

O que é: CLI poderosa para rodar LLMs localmente com gestão simplificada.

URL: https://ollama.com

Vantagens:

- Extremamente leve e rápido
- Ideal para automação e scripts
- API local OpenAI-compatible
- Gestão simples via terminal
- **Melhor para integração Raycast**

Instalação:

```
# Via Homebrew
brew install ollama

# Ou baixar de ollama.com
# Instala como app de barra de menus
```

Uso:

```
# Download e chat interativo em um comando
ollama run llama3:8b

# Outros modelos recomendados
ollama run mistral:7b
```

```
ollama run qwen2.5-coder:7b  # Especializado em código

# Listar modelos instalados
ollama list

# Baixar sem rodar
ollama pull llama3:8b

# Remover modelo
ollama rm llama3:8b
```

API Local:

Ollama expõe automaticamente API compatível com OpenAI em:

```
http://localhost:11434
```

Testar API:

```
curl http://localhost:11434/api/generate -d '{
  "model": "llama3:8b",
  "prompt": "Por que o céu é azul?",
  "stream": false
}'
```

Opção 3: Jan (Alternativa UI)

O que é: Open-source, alternativa offline ao ChatGPT.

URL: https://jan.ai

Instalação:

```
brew install --cask jan
```

Features:

- Interface similar ao ChatGPT
- 70+ modelos pré-instalados
- Importação de modelos Hugging Face
- 100% offline, sem telemetria

1.3. Recomendação por Caso de Uso

Caso de Uso	Ferramenta Recomendada
Iniciantes	**LM Studio** - interface gráfica amigável
Automação/Raycast	**Ollama** - CLI poderosa, ideal para scripts
Chat offline	**Jan** - experiência ChatGPT

2. Customizar Raycast com Script Commands

2.1. Criar Script Command Básico

Passos:

1. Abrir Raycast (`⌘ + Espaço`)
2. Digitar `Create Script Command`
3. Preencher campos:
   - **Title:** Nome que aparecerá no Raycast
   - **Description:** O que o script faz
   - **Package:** Categoria organizacional
   - **Language:** Bash, Python, Node.js, etc
   - **Mode:** `Full Output` (exibe resultado completo)

2.2. Estrutura de Diretório

```
~/Library/Application Support/com.raycast.macos/extensions/
└── scripts/
    ├── system/
    │   ├── generate_ai_context.sh
    │   └── system_report.py
    ├── productivity/
    │   ├── quick_note.sh
    │   └── pomodoro.sh
    └── dev/
        ├── git_status.sh
        └── docker_stats.sh
```

2.3. Metadados Obrigatórios

Todo Script Command deve ter metadados no início:

```
#!/bin/bash

# Required parameters:
# @raycast.schemaVersion 1
```

```
# @raycast.title Gerar Contexto de IA
# @raycast.mode fullOutput
# @raycast.packageName System

# Optional parameters:
# @raycast.icon 
# @raycast.description Coleta dados do sistema para configurar IA
# @raycast.author Luiz Sena

# Documentation:
# @raycast.authorURL https://github.com/luizsena
```

2.4. Script de Coleta de Dados do Sistema

Função: Coletar informações do macOS para contexto de IA.

Outputs:

- Hardware (CPU, RAM, GPU, Storage)
- Software (OS, apps instalados, configurações)
- Network (hostname, IPs, configuração)
- Desenvolvimento (Git, Docker, IDEs)
- Credenciais (via 1Password CLI)

3. Integração Completa: Raycast → Script → LLM

3.1. Fluxo de Workflow

1. **Usuário aciona** comando no Raycast (`⌘ + Espaço` → "Analisar Meu Sistema")
2. **Script coleta** dados do sistema (JSON)
3. **Script envia** JSON para LLM local (Ollama/LM Studio)
4. **LLM processa** e retorna resumo/análise
5. **Raycast exibe** resultado formatado

3.2. Script Integrado Completo

Arquivo: `analyze_system_with_llm.sh`

```
#!/bin/bash

# Required parameters:
# @raycast.schemaVersion 1
# @raycast.title Analisar Sistema com LLM
# @raycast.mode fullOutput
# @raycast.packageName AI System
# @raycast.icon 
```

```
set -e

# --- Funções de Coleta de Dados ---

collect_macos_data() {
    cat &lt;&lt;EOF
{
  "hostname": "$(hostname)",
  "os": {
    "name": "$(sw_vers -productName)",
    "version": "$(sw_vers -productVersion)",
    "build": "$(sw_vers -buildVersion)"
  },
  "hardware": {
    "model": "$(sysctl -n hw.model)",
    "cpu": "$(sysctl -n machdep.cpu.brand_string)",
    "cores": $(sysctl -n hw.ncpu),
    "memory_gb": $(( $(sysctl -n hw.memsize) / 1073741824 ))
  },
  "storage": {
    "total_gb": $(df -h / | awk 'NR==2 {print $2}'),
    "used_gb": $(df -h / | awk 'NR==2 {print $3}'),
    "available_gb": $(df -h / | awk 'NR==2 {print $4}')
  },
  "network": {
    "local_ip": "$(ipconfig getifaddr en0 2&gt;/dev/null || echo 'N/A')"
  },
  "development": {
    "git_installed": $(command -v git &amp;&gt; /dev/null &amp;&amp; echo "true" || echo
    "docker_installed": $(command -v docker &amp;&gt; /dev/null &amp;&amp; echo "true" ||
    "python_version": "$(python3 --version 2&gt;&amp;1 | awk '{print $2}')"
  }
}
EOF
}

# --- Função Principal ---

main() {
    echo "🔍 Coletando dados do sistema..."
    echo ""

    # Coleta dados
    JSON_OUTPUT=$(collect_macos_data)

    # Escapa JSON para curl
    ESCAPED_JSON=$(echo "$JSON_OUTPUT" | sed 's/\\/\\\\/g' | sed 's/"/\\"/g' | tr -d '\n'

    echo "🤖 Enviando para LLM local (Ollama)..."
    echo ""

    # Monta payload
    PAYLOAD=$(cat &lt;&lt;EOF
{
  "model": "llama3:8b",
  "prompt": "Com base nos seguintes dados do sistema em formato JSON, escreva um resumo t
```

```
  "stream": false
}
EOF
)

    # Chama Ollama API
    RESPONSE=$(curl -s http://localhost:11434/api/generate -d "$PAYLOAD")

    # Extrai resposta
    LLM_OUTPUT=$(echo "$RESPONSE" | grep -o '"response":"[^"]*"' | cut -d'"' -f4 | sed 's

    echo "🤖 === RESUMO GERADO PELO LLM LOCAL ==="
    echo ""
    echo "$LLM_OUTPUT"
    echo ""
    echo "📊 === DADOS COMPLETOS (JSON) ==="
    echo ""
    echo "$JSON_OUTPUT" | jq '.'
}

# Executa
main
```

3.3. Como Usar

1. Salvar script acima como `analyze_system_with_llm.sh`
2. Colocar em `~/Library/Application Support/com.raycast.macos/extensions/scripts/`
3. Tornar executável: `chmod +x analyze_system_with_llm.sh`
4. Recarregar extensões no Raycast
5. Executar: `⌘ + Espaço` → `Analisar Sistema com LLM`

4. Casos de Uso Avançados

4.1. Tradução Automática

Traduzir textos copiados usando LLM local:

```
#!/bin/bash

# @raycast.title Traduzir para Português
# @raycast.mode fullOutput

TEXTO_ORIGINAL=$(pbpaste)

PAYLOAD=$(cat &lt;&lt;EOF
{
  "model": "llama3:8b",
  "prompt": "Traduza o seguinte texto para português brasileiro, mantendo o contexto técn
  "stream": false
}
```

```
EOF
)

curl -s http://localhost:11434/api/generate -d "$PAYLOAD" | \
  grep -o '"response":"[^"]*"' | cut -d'"' -f4 | sed 's/\\n/\n/g'
```

4.2. Code Review

Revisar código copiado:

```
#!/bin/bash

# @raycast.title Code Review com LLM
# @raycast.mode fullOutput

CODE=$(pbpaste)

PAYLOAD=$(cat &lt;&lt;EOF
{
  "model": "qwen2.5-coder:7b",
  "prompt": "Revise este código e forneça: 1) Potenciais bugs, 2) Sugestões de melhoria,
  "stream": false
}
EOF
)

curl -s http://localhost:11434/api/generate -d "$PAYLOAD" | \
  grep -o '"response":"[^"]*"' | cut -d'"' -f4 | sed 's/\\n/\n/g'
```

4.3. Resumir Documento

Resumir documento markdown ou texto longo:

```
#!/bin/bash

# @raycast.title Resumir Documento
# @raycast.mode fullOutput
# @raycast.argument1 { "type": "text", "placeholder": "Caminho do arquivo" }

ARQUIVO="$1"
CONTEUDO=$(cat "$ARQUIVO")

PAYLOAD=$(cat &lt;&lt;EOF
{
  "model": "llama3:8b",
  "prompt": "Resuma o seguinte documento em bullet points (máximo 10 pontos), destacando
  "stream": false
}
EOF
)
```

```
curl -s http://localhost:11434/api/generate -d "$PAYLOAD" | \
  grep -o '"response":"[^"]*"' | cut -d'"' -f4 | sed 's/\\n/\n/g'
```

4.4. Gerar Commit Message

Gerar mensagem de commit baseada em git diff:

```
#!/bin/bash

# @raycast.title Gerar Commit Message
# @raycast.mode fullOutput

cd $(git rev-parse --show-toplevel)
DIFF=$(git diff --staged)

if [ -z "$DIFF" ]; then
    echo "❌ Nenhuma mudança staged. Use 'git add' primeiro."
    exit 1
fi

PAYLOAD=$(cat &lt;&lt;EOF
{
  "model": "qwen2.5-coder:7b",
  "prompt": "Com base no seguinte git diff, gere uma mensagem de commit seguindo o padrão
  "stream": false
}
EOF
)

COMMIT_MSG=$(curl -s http://localhost:11434/api/generate -d "$PAYLOAD" | \
  grep -o '"response":"[^"]*"' | cut -d'"' -f4)

echo "🤖 Sugestão de commit message:"
echo ""
echo "$COMMIT_MSG"
echo ""
echo "Para usar, execute:"
echo "git commit -m \"$COMMIT_MSG\""
```

5. Otimização de Performance

5.1. Modelos Recomendados para macOS

Uso	Modelo	RAM	Velocidade	Qualidade
Uso Geral	`llama3:8b` (Q4)	~5GB	Rápido	Excelente para maioria das tarefas
Código	`qwen2.5-coder:7b`	~4GB	Muito rápido	Especializado em código, melhor que Llama3
Tarefas Simples	`mistral:7b`	~4GB	Ultra-rápido	Bom para summaries, tradução, FAQ

Uso	Modelo	RAM	Velocidade	Qualidade
Avançado	`llama3:70b`	~40GB	Lento	Próximo de GPT-4 (requer 64GB+ RAM)

5.2. Dicas de Performance

- ✓ Manter Ollama rodando em background (app de barra)
- ✓ Usar modelos quantizados (Q4, Q5) para velocidade
- ✓ Primeiro load é lento, subsequentes são instantâneos (cache)
- ✓ Fechar apps pesados antes de rodar modelos grandes
- ✓ Usar `"stream": false` em API calls para Raycast
- ✓ Limitar tamanho do prompt (máx 2000 tokens)

6. Troubleshooting

6.1. Ollama Não Responde

```
# Verificar se Ollama está rodando
ps aux | grep ollama

# Reiniciar Ollama
killall ollama
ollama serve &amp;

# Testar conectividade
curl http://localhost:11434/api/tags
```

6.2. Modelo Muito Lento

```
# Usar modelo menor/quantizado
ollama pull llama3:8b  # Em vez de 70b

# Verificar uso de RAM
top -o MEM

# Limitar contexto (reduzir tamanho do prompt)
```

6.3. Raycast Não Executa Script

```
# Verificar permissões
chmod +x ~/path/to/script.sh

# Verificar metadados Raycast
# Deve ter @raycast.schemaVersion no topo
```

```
# Recarregar extensões
# Raycast → Extensions → Reload All
```

Conclusão

Com este setup, você tem um **workflow completamente offline** no macOS:

1. **LLM local** (Ollama) rodando em background
2. **Scripts Raycast** para automação
3. **Integração perfeita** via API local
4. **Zero dependências** de serviços externos
5. **Performance excelente** em Apple Silicon

Casos de uso práticos: análise de sistema, tradução, code review, resumos, geração de commits — tudo instantâneo e privado.

⁂