**Pesquisa 1:**

# 📘 \*\*ENCICLOPÉDIA COMPLETA DO AGENTE LOCAL DE IA OPEN-SOURCE\*\*

## 🧠 O Guia Absoluto para Criar Seu Próprio Agente Inteligente 100% Offline e Gratuito

> Um TCC vivo, uma bíblia técnica e um curso completo para dominar a construção de agentes locais com IA open-source.

---

## 📖 \*\*SEÇÃO 1 – INTRODUÇÃO GERA\*\*

### 1.1 O que é um Agente de IA?

Um \*\*agente de IA\*\* é um sistema autônomo que percebe o ambiente, toma decisões e executa ações para atingir objetivos. Em vez de simplesmente responder perguntas (como um chatbot), ele pode:

- Ler arquivos, navegar na web, rodar código, instalar programas, gerenciar APIs, criar vídeos, enviar e-mails, controlar IoT, etc.

- Memorizar contexto, aprender com feedback, corrigir erros automaticamente.

- Funcionar como um “assistente digital completo” que age sem supervisão.

### 1.2 Agente Local vs. Agente em Nuvem

| \*\*Critério\*\* | \*\*Agente Local\*\* | \*\*Agente em Nuvem\*\* |

|----------------------|-------------------------------------------|-------------------------------------------|

| \*\*Privacidade\*\* | Tudo fica no seu PC. Zero vazamento. | Dados viajam por servidores externos. |

| \*\*Custo\*\* | 100% gratuito após instalação. | Pago por tokens, execuções ou assinatura. |

| \*\*Velocidade\*\* | Sem latência de rede. | Depende da internet e da API. |

| \*\*Personalização\*\* | Você controla tudo: modelos, prompts, UI. | Limitado ao que a plataforma permite. |

| \*\*Requisitos\*\* | Precisa de um PC com 8 GB+ de RAM. | Funciona até em celulares. |

### 1.3 Por que Construir um Agente Local Gratuito?

- \*\*Autonomia total:\*\* sem dependência de OpenAI, Google ou Microsoft.

- \*\*Infinitas customizações:\*\* desde um bot de Discord até um assistente de produção de vídeos.

- \*\*Zero custo operacional:\*\* ideal para freelancers, pequenas empresas, educadores, devs.

- \*\*100% privado:\*\* dados sensíveis nunca saem do seu computador.

---

## 🛠️ \*\*SEÇÃO 2 – FERRAMENTAS E TECNOLOGIAS\*\*

### 2.1 Mapa Completo do Ecossistema

| \*\*Categoria\*\* | \*\*Ferramentas Open-Source\*\* | \*\*Função Principal\*\* |

|-----------------------|------------------------------------------------------|---------------------------------------------|

| \*\*LLM Local\*\* | LLaMA 3.2, Mistral 7B, Code Llama, GPT4All, Ollama | Modelos de linguagem rodando offline. |

| \*\*Framework\*\* | LangChain, LangGraph, LlamaIndex | Orquestração de prompts, memória, ferramentas.|

| \*\*Vetores/Memória\*\* | Qdrant, ChromaDB, FAISS, Milvus | Armazenar embeddings e recuperar contexto. |

| \*\*Automação\*\* | n8n, Flowise, CrewAI, AutoGen | Criar pipelines visuais ou código. |

| \*\*UI/UX\*\* | Gradio, Streamlit, React, Electron, Tauri | Interfaces de chat desktop ou web. |

| \*\*Estrutura\*\* | Docker, Docker-Compose, FastAPI | Containerizar e servir o agente. |

### 2.2 Instalação Passo a Passo de Cada Ferramenta

#### 🔧 \*\*1. Instalar Docker (Base para Tudo)\*\*

```bash

# Windows / Mac / Linux

# 1. Baixe em: https://docs.docker.com/get-docker/

# 2. Após instalar, rode no terminal:

docker --version

```

#### 🔧 \*\*2. Instalar Ollama (LLMs Locais)\*\*

```bash

# Windows (PowerShell Admin)

winget install ollama

# Mac / Linux

curl -fsSL https://ollama.ai/install.sh | sh

# Baixar modelos

ollama pull llama3.2

ollama pull mistral

ollama pull codellama

```

#### 🔧 \*\*3. Instalar n8n (Automação Visual)\*\*

```bash

# Via Docker (recomendado)

docker run -it --name n8n -p 5678:5678 \

-v ~/.n8n:/home/node/.n8n \

n8nio/n8n

# Acesse: http://localhost:5678

```

#### 🔧 \*\*4. Instalar Qdrant (Vetores Locais)\*\*

```bash

docker run -p 6333:6333 \

-v $(pwd)/qdrant\_storage:/qdrant/storage \

qdrant/qdrant

```

#### 🔧 \*\*5. Instalar Flowise (Pipeline RAG Visual)\*\*

```bash

docker run -d --name flowise -p 3000:3000 \

-e FLOWISE\_USERNAME=admin \

-e FLOWISE\_PASSWORD=admin \

flowiseai/flowise

```

---

## 🧱 \*\*SEÇÃO 3 – ARQUITETURA COMPLETA DO AGENTE LOCAL\*\*

### 3.1 Diagrama de Arquitetura

```

[Usuário]

↓ (Texto/Voz/Imagem)

[Interface Gradio/Streamlit]

↓

[n8n Workflow]

↓

[LangChain Chain]

↓

[LLM Local (Ollama)]

↓

[Memória Vetorial (Qdrant)]

↓

[Ferramentas: Terminal, Navegador, Python, APIs]

↓

[Output: Código, Vídeo, Resposta, Ação]

```

### 3.2 Fluxo de Dados Passo a Passo

| \*\*Etapa\*\* | \*\*O que Acontece\*\* |

|-------------------|-------------------------------------------------------------------------------------|

| \*\*Input\*\* | Usuário digita ou fala uma tarefa. |

| \*\*Parsing\*\* | n8n recebe e roteia para o agente. |

| \*\*Contexto\*\* | LangChain busca no Qdrant histórico e documentos. |

| \*\*LLM\*\* | Modelo local (ex: Mistral) gera plano de ação. |

| \*\*Execução\*\* | Ferramentas são disparadas: terminal, script, navegador, etc. |

| \*\*Output\*\* | Resultado é retornado ao usuário e salvo na memória vetorial para aprendizado. |

---

## 🧠 \*\*SEÇÃO 4 – IAS LOCAIS (MODELOS DE LINGUAGEM)\*\*

### 4.1 Comparação de Modelos Locais

| \*\*Modelo\*\* | \*\*Tamanho\*\* | \*\*Uso Ideal\*\* | \*\*Instalação\*\* |

|------------------|-------------|------------------------------------|----------------------------------------|

| \*\*LLaMA 3.2\*\* | 3B-70B | Chat, código, geral | `ollama pull llama3.2` |

| \*\*Mistral 7B\*\* | 7B | Velocidade, inglês, código | `ollama pull mistral` |

| \*\*Code Llama\*\* | 7B-34B | Programação, debug, scripts | `ollama pull codellama` |

| \*\*GPT4All\*\* | 3B-13B | Chat com GUI, instalação simples | Baixar em https://gpt4all.io |

### 4.2 Quantização para PCs Modestos

- \*\*O que é:\*\* Reduz o tamanho do modelo via GGUF (ex: `q4\_0`, `q5\_K\_M`).

- \*\*Como usar:\*\*

```bash

ollama pull llama3.2:3b-instruct-q4\_0 # 3B q4 ocupa ~1.9 GB RAM

```

---

## 🧩 \*\*SEÇÃO 5 – FRAMEWORKS E ORQUESTRAÇÃO\*\*

### 5.1 LangChain: O Coração do Agente

- \*\*Conceito:\*\* Framework para conectar LLMs, memória, ferramentas e prompts.

- \*\*Exemplo Prático (Python):\*\*

```python

from langchain\_community.llms import Ollama

from langchain\_community.vectorstores import Qdrant

from langchain\_community.embeddings import OllamaEmbeddings

from langchain.chains import RetrievalQA

llm = Ollama(model="llama3.2")

embeddings = OllamaEmbeddings(model="llama3.2")

vectorstore = Qdrant.from\_documents(docs, embeddings, location=":memory:")

qa = RetrievalQA.from\_chain\_type(llm=llm, retriever=vectorstore.as\_retriever())

```

### 5.2 RAG (Retrieval-Augmented Generation)

- \*\*Definição:\*\* Combina LLM com busca em documentos locais.

- \*\*Pipeline:\*\*

1. \*\*Load\*\* → PDF/CSV/TXT

2. \*\*Split\*\* → Chunks (ex: 1000 chars)

3. \*\*Embed\*\* → Vetores com `OllamaEmbeddings`

4. \*\*Store\*\* → Qdrant/ChromaDB

5. \*\*Retrieve\*\* → Busca semântica + LLM

### 5.3 MCP (Multi-Component Prompting)

- \*\*Definição:\*\* Prompts estruturados com múltiplos componentes (memória, ferramentas, regras).

- \*\*Exemplo no n8n:\*\*

- \*\*Trigger:\*\* Mensagem do usuário.

- \*\*Ferramentas:\*\* Google Calendar, Notion, Terminal.

- \*\*Prompt:\*\* “Use a ferramenta correta para agendar reunião com base no histórico.”

---

## 🤖 \*\*SEÇÃO 6 – CONSTRUÇÃO DE AGENTES\*\*

### 6.1 Agente com Memória de Contexto

- \*\*Uso:\*\* Lembra conversas, preferências, erros.

- \*\*Implementação:\*\*

```python

from langchain.memory import ConversationBufferMemory

memory = ConversationBufferMemory()

```

### 6.2 Agente com Ferramentas Múltiplas

- \*\*Ferramentas Comuns:\*\*

- \*\*Terminal:\*\* Executar scripts Python.

- \*\*Navegador:\*\* Buscar no Google com `playwright`.

- \*\*Arquivos:\*\* Ler/editar PDFs com `pypdf`.

- \*\*Exemplo no n8n:\*\*

1. \*\*Trigger:\*\* “Crie um site em React.”

2. \*\*Ação:\*\* Terminal roda `npx create-react-app`.

3. \*\*Output:\*\* Link do projeto abre no navegador.

### 6.3 Agente Multi-Usuário

- \*\*Uso:\*\* Times com perfis separados.

- \*\*Implementação:\*\*

- Cada usuário tem um `session\_id` no Qdrant.

- LangChain filtra memória por usuário.

---

## 💻 \*\*SEÇÃO 7 – APLICAÇÕES REAIS\*\*

### 7.1 Programar Apps com Linguagem Natural

- \*\*Fluxo:\*\*

1. Usuário: “Crie um app de lista de tarefas em Python.”

2. Agente: Gera código, salva em `todo.py`, executa testes.

3. Interface: Gradio mostra app rodando localmente.

### 7.2 Automatizar Tarefas de Dev

- \*\*Exemplos:\*\*

- Commit automático: “Analise mudanças e crie mensagem de commit.”

- Testes: “Rode pytest e me diga os erros.”

### 7.3 Agente para Produção de Vídeos

- \*\*Pipeline:\*\*

1. Input: “Crie um vídeo sobre IA local.”

2. Agente:

- Busca script no Qdrant.

- Gera áudio com TTS (ex: `edge-tts`).

- Cria imagens com Stable Diffusion.

- Monta vídeo com FFmpeg.

---

## 🌐 \*\*SEÇÃO 8 – INTERFACE E USABILIDADE\*\*

### 8.1 Interface Gradio (Exemplo Completo)

```python

import gradio as gr

from langchain\_community.llms import Ollama

llm = Ollama(model="llama3.2")

def chat(message, history):

return llm.invoke(message)

gr.ChatInterface(chat).launch()

```

### 8.2 Interface Streamlit

```python

import streamlit as st

from langchain\_community.llms import Ollama

st.title("Agente Local")

prompt = st.text\_input("Digite uma tarefa:")

if prompt:

response = Ollama(model="llama3.2").invoke(prompt)

st.write(response)

```

### 8.3 Central de Controle

- \*\*Ferramentas:\*\*

- \*\*n8n Dashboard:\*\* Monitorar workflows.

- \*\*Qdrant UI:\*\* Visualizar embeddings.

- \*\*Terminal:\*\* Logs em tempo real.

---

## 🚀 \*\*SEÇÃO 9 – CASOS DE USO\*\*

| \*\*Caso\*\* | \*\*Como Fazer\*\* |

|-----------------------|----------------------------------------------------------------------------------|

| \*\*Dev Helper\*\* | Agente que escreve testes, documenta código, faz commits. |

| \*\*Produtividade\*\* | Agenda tarefas, responde e-mails, gera relatórios. |

| \*\*Video Automation\*\* | Cria scripts, voz sintética, imagens, montagem final. |

| \*\*Empresa Offline\*\* | Chatbot interno para RH, financeiro, estoque (tudo local). |

---

## 🔒 \*\*SEÇÃO 10 – SEGURANÇA E PRIVACIDADE\*\*

- \*\*Localhost Only:\*\* Configure firewalls para bloquear acesso externo.

- \*\*Criptografia:\*\* Use `cryptography` para dados sensíveis.

- \*\*Backups:\*\* Automatize backups do Qdrant e n8n com cronjobs.

- \*\*Logs:\*\* Redirecione logs para arquivos locais, não envie para cloud.

---

## 🧰 \*\*SEÇÃO 11 – REPOSITÓRIO FINAL\*\*

### 11.1 Estrutura de Pastas

```

meu-agente/

├── docker-compose.yml # n8n + Qdrant + Ollama

├── src/

│ ├── agent.py # LangChain + ferramentas

│ ├── ui.py # Gradio/Streamlit

│ └── tools/

│ ├── terminal.py

│ └── browser.py

├── data/

│ ├── pdfs/ # Documentos locais

│ └── vectorstore/ # Qdrant ou ChromaDB

├── scripts/

│ ├── install.sh # Instala tudo

│ └── backup.sh # Backup diário

└── README.md # Tutorial completo

```

### 11.2 Comando Único para Instalação

```bash

git clone https://github.com/seu-usuario/meu-agente.git

cd meu-agente

./scripts/install.sh

```

---

## 🎓 \*\*SEÇÃO 12 – TUTORIAIS POR ETAPA\*\*

| \*\*Etapa\*\* | \*\*Tutorial\*\* |

|-------------------|--------------------------------------------------------------------------------|

| \*\*Iniciante\*\* | Instale Docker → rode `ollama pull llama3.2` → teste no terminal. |

| \*\*Intermediário\*\* | Configure n8n → crie workflow RAG com PDF local. |

| \*\*Avançado\*\* | Construa agente multi-ferramenta com LangChain e interface Gradio. |

---

## 🏁 \*\*CONCLUSÃO\*\*

Você agora tem \*\*tudo\*\* para criar um \*\*agente de IA local, gratuito, privado e poderoso\*\*.

- \*\*Sem APIs pagas.\*\*

- \*\*Sem dependências externas.\*\*

- \*\*100% open-source.\*\*

- \*\*Pronto para produção ou estudos.\*\*

> \*\*Próximos passos:\*\*

1. Clone o repositório acima.

2. Execute `./scripts/install.sh`.

3. Abra `http://localhost:5678` (n8n) e `http://localhost:7860` (Gradio).

4. Comece a automatizar o mundo — localmente.

\*\*Bem-vindo à era dos agentes locais. 🚀

# Pesquisa 2: ENCICLOPÉDIA COMPLETA PARA CRIAÇÃO DE AGENTE LOCAL DE IA OPEN-SOURCE

**O Guia Definitivo: Do Zero ao Agente Avançado**

**SUMÁRIO EXECUTIVO**

Este documento representa um guia enciclopédico completo para criar um agente de inteligência artificial local, gratuito e open-source, capaz de rivalizar com soluções comerciais como Manus.ai, Cursor AI, Genspark, DeepAgent e outros. O objetivo é eliminar a dependência de APIs pagas, tokens e créditos, criando uma solução 100% local e gratuita.

**PARTE I: FUNDAMENTOS CONCEITUAIS**

**1. O QUE SÃO AGENTES DE IA?**

**1.1 Definição Técnica**

Um agente de IA é um sistema computacional autônomo que:

* **Percebe** seu ambiente através de sensores (input de texto, voz, imagens)
* **Processa** informações usando modelos de linguagem e lógica
* **Age** no ambiente para alcançar objetivos específicos
* **Aprende** com feedback e experiências passadas

**1.2 Diferenças Fundamentais: Local vs Nuvem**

**Agentes em Nuvem (Como os comerciais)**

* ✅ **Vantagens**: Processamento poderoso, sempre atualizados, sem setup
* ❌ **Desvantagens**: Custos recorrentes, dependência de internet, privacidade comprometida, limites de tokens

**Agentes Locais (Nossa solução)**

* ✅ **Vantagens**: Privacidade total, sem custos recorrentes, controle completo, sem limites
* ❌ **Desvantagens**: Requer setup inicial, dependente do hardware local

**1.3 Por Que Construir Um Agente Local?**

**Motivações Econômicas**

* **Manus.ai**: ~$50-200/mês dependendo do uso
* **Cursor Pro**: $20/mês
* **GitHub Copilot**: $10-19/mês
* **Claude/GPT-4**: $20/mês + tokens extras
* **Nossa solução**: $0/mês após setup inicial

**Motivações Técnicas**

1. **Controle Total**: Modificar comportamento, adicionar funcionalidades
2. **Privacidade**: Dados nunca saem do seu computador
3. **Personalização**: Treinar com seus próprios dados
4. **Disponibilidade**: Funciona offline
5. **Escalabilidade**: Sem limites de requisições

**Motivações Estratégicas**

* **Independência tecnológica**
* **Aprendizado profundo** das tecnologias
* **Base para produtos próprios**
* **Vantagem competitiva**

**PARTE II: ECOSSISTEMA TECNOLÓGICO COMPLETO**

**2. MODELOS DE IA LOCAIS (LLMs)**

**2.1 Ollama - A Porta de Entrada**

**O que é o Ollama?**

Ollama é uma plataforma open-source que simplifica a execução de modelos de linguagem localmente. Funciona como um "Docker para LLMs", permitindo baixar, instalar e executar modelos com comandos simples.

**Instalação Completa do Ollama**

# Linux/macOS

curl -fsSL https://ollama.ai/install.sh | sh

# Windows

# Baixar do site oficial: https://ollama.ai/download

# Verificar instalação

ollama --version

**Modelos Essenciais para Agentes**

**1. Llama 3.2 (Recomendado para iniciantes)**

# Baixar modelo

ollama pull llama3.2

# Testar localmente

ollama run llama3.2

* **Tamanho**: ~4.7GB
* **RAM necessária**: 8GB mínimo
* **Especialidade**: Uso geral, conversação
* **Performance**: Boa para tarefas básicas

**2. Code Llama (Para programação)**

ollama pull codellama:7b

ollama pull codellama:13b # Se tiver 16GB+ RAM

* **Tamanho**: 3.8GB (7b) / 7.3GB (13b)
* **Especialidade**: Programação, código
* **Languages**: Python, JavaScript, Java, C++, etc.

**3. Mistral (Alternativa eficiente)**

ollama pull mistral:7b

* **Tamanho**: ~4.1GB
* **Performance**: Excelente custo-benefício
* **Velocidade**: Mais rápido que Llama equivalente

**4. Deepseek Coder (Especialista em código)**

ollama pull deepseek-coder:6.7b

* **Especialidade**: Código avançado, debugging
* **Performance**: Superior ao Code Llama em muitos casos

**5. Modelos de Embedding**

# Para RAG e busca semântica

ollama pull mxbai-embed-large

ollama pull nomic-embed-text

**2.2 Quantização e Otimização**

**Entendendo Quantização**

Quantização reduz o tamanho dos modelos convertendo pesos de 16-bit para 8-bit ou 4-bit, mantendo ~95% da performance original.

**Escolhendo o Modelo Certo por Hardware**

## Hardware Mínimo por Modelo

### 8GB RAM (Configuração básica)

- llama3.2:3b

- mistral:7b (com swap)

- codellama:7b (com swap)

### 16GB RAM (Configuração ideal)

- llama3.2:8b

- codellama:13b

- mistral:7b (fluido)

- deepseek-coder:6.7b

### 32GB+ RAM (Configuração profissional)

- llama3.1:70b (quantizado)

- codellama:34b

- Múltiplos modelos simultaneamente

**2.3 Alternativas ao Ollama**

**GPT4All**

# Instalação

pip install gpt4all

# Uso básico

from gpt4all import GPT4All

model = GPT4All("orca-mini-3b-gguf2-q4\_0.gguf")

**llama.cpp**

# Compilação manual para máximo desempenho

git clone https://github.com/ggerganov/llama.cpp

cd llama.cpp

make

# Execução otimizada

./main -m modelo.gguf -p "Seu prompt aqui"

**3. LANGCHAIN - O FRAMEWORK FUNDAMENTAL**

**3.1 Visão Geral do LangChain**

LangChain é o framework mais popular para desenvolvimento de aplicações com LLMs. Ele fornece:

* **Abstrações** para trabalhar com diferentes LLMs
* **Chains** para sequenciar operações
* **Agents** para tomada de decisão autônoma
* **Memory** para contexto persistente
* **Tools** para conectar com APIs externas

**3.2 Instalação e Setup Completo**

# Instalação básica

pip install langchain

pip install langchain-community

pip install langchain-ollama

# Extensões específicas

pip install langchain-chroma # Para RAG

pip install langchain-openai # Se quiser usar OpenAI também

pip install langchain-google # Para Google APIs

**3.3 Componentes Essenciais do LangChain**

**3.3.1 LLMs e Chat Models**

from langchain\_ollama import OllamaLLM, ChatOllama

# LLM básico

llm = OllamaLLM(model="llama3.2")

response = llm.invoke("Explique machine learning")

# Chat model (recomendado)

chat\_model = ChatOllama(model="llama3.2")

messages = [

("system", "Você é um assistente de programação especialista"),

("human", "Como criar uma API REST em Python?")

]

response = chat\_model.invoke(messages)

**3.3.2 Prompts Templates**

from langchain.prompts import ChatPromptTemplate, PromptTemplate

# Template básico

template = """

Você é um agente especialista em {especialidade}.

Contexto: {contexto}

Pergunta: {pergunta}

Resposta detalhada:

"""

prompt = PromptTemplate(

template=template,

input\_variables=["especialidade", "contexto", "pergunta"]

)

# Template de chat

chat\_template = ChatPromptTemplate.from\_messages([

("system", "Você é um {papel} especializado em {area}"),

("human", "{pergunta}")

])

**3.3.3 Chains - Sequenciamento de Operações**

from langchain.chains import LLMChain

from langchain.schema.runnable import RunnableSequence

# Chain básica

chain = LLMChain(llm=llm, prompt=prompt)

result = chain.run(

especialidade="Python",

contexto="Desenvolvimento de APIs",

pergunta="Como implementar autenticação JWT?"

)

# Chain sequencial

sequence = RunnableSequence.from\_runnables([

prompt,

llm,

# Adicionar mais processamento se necessário

])

**3.3.4 Memory - Contexto Persistente**

from langchain.memory import ConversationBufferMemory, ConversationSummaryMemory

# Memória simples (buffer)

memory = ConversationBufferMemory(

memory\_key="chat\_history",

return\_messages=True

)

# Memória com resumo (para conversas longas)

summary\_memory = ConversationSummaryMemory(

llm=llm,

memory\_key="chat\_history",

return\_messages=True

)

# Usando memória em chains

from langchain.chains import ConversationChain

conversation = ConversationChain(

llm=llm,

memory=memory,

verbose=True

)

**3.3.5 Tools - Conectores com Mundo Externo**

from langchain.tools import Tool, DuckDuckGoSearchRun

from langchain.utilities import PythonREPL

# Ferramenta de pesquisa

search = DuckDuckGoSearchRun()

search\_tool = Tool(

name="WebSearch",

description="Buscar informações na internet",

func=search.run

)

# Ferramenta Python REPL

python\_repl = PythonREPL()

repl\_tool = Tool(

name="python\_repl",

description="Executar código Python",

func=python\_repl.run

)

# Ferramenta customizada

def calculadora(expression: str) -> str:

"""Calcula expressões matemáticas."""

try:

return str(eval(expression))

except:

return "Erro na expressão matemática"

calc\_tool = Tool(

name="Calculadora",

description="Executa cálculos matemáticos",

func=calculadora

)

**3.4 Agents - O Coração da Automação**

**3.4.1 Tipos de Agentes**

from langchain.agents import create\_react\_agent, AgentExecutor

from langchain.agents.tools import Tool

from langchain import hub

# Buscar prompt pré-configurado

prompt = hub.pull("hwchase17/react")

# Criar agente ReAct (Reasoning + Acting)

agent = create\_react\_agent(

llm=llm,

tools=[search\_tool, calc\_tool, repl\_tool],

prompt=prompt

)

# Executor do agente

agent\_executor = AgentExecutor(

agent=agent,

tools=[search\_tool, calc\_tool, repl\_tool],

verbose=True,

handle\_parsing\_errors=True

)

# Executar tarefa

result = agent\_executor.invoke({

"input": "Pesquise sobre Python 3.12 e calcule quantos anos se passaram desde Python 1.0"

})

**3.4.2 Agente Personalizado para Programação**

def criar\_agente\_programador():

# Ferramentas específicas para programação

tools = [

Tool(

name="PythonREPL",

description="Execute código Python e veja o resultado",

func=python\_repl.run

),

Tool(

name="FileReader",

description="Leia arquivos do sistema",

func=lambda path: open(path, 'r').read()

),

Tool(

name="FileWriter",

description="Escreva arquivos no sistema",

func=lambda args: open(args.split('|')[0], 'w').write(args.split('|')[1])

)

]

# Prompt especializado

prompt\_template = """

Você é um agente de programação avançado. Suas responsabilidades:

1. Analisar problemas de código

2. Escrever soluções funcionais

3. Testar código antes de entregar

4. Explicar sua lógica

Ferramentas disponíveis: {tools}

Histórico: {chat\_history}

Pergunta: {input}

Pensamento: {agent\_scratchpad}

"""

agent = create\_react\_agent(llm, tools, prompt\_template)

return AgentExecutor(agent=agent, tools=tools, verbose=True)

**4. RAG (RETRIEVAL AUGMENTED GENERATION)**

**4.1 Conceitos Fundamentais do RAG**

RAG combina a capacidade generativa dos LLMs com recuperação de informações relevantes de uma base de conhecimento. O processo é:

1. **Indexação**: Documentos são divididos em chunks e convertidos em embeddings
2. **Recuperação**: Para cada pergunta, busca-se chunks relevantes
3. **Geração**: LLM usa os chunks como contexto para gerar resposta

**4.2 ChromaDB - Database Vetorial Local**

**Instalação e Setup**

pip install chromadb

pip install langchain-chroma

**Implementação Básica do RAG**

import chromadb

from langchain\_chroma import Chroma

from langchain\_ollama import OllamaEmbeddings

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain.schema import Document

# 1. Configurar embeddings

embeddings = OllamaEmbeddings(model="mxbai-embed-large")

# 2. Configurar divisor de texto

text\_splitter = RecursiveCharacterTextSplitter(

chunk\_size=1000,

chunk\_overlap=200,

separators=["\n\n", "\n", " ", ""]

)

# 3. Processar documentos

def processar\_documentos(arquivos):

docs = []

for arquivo in arquivos:

with open(arquivo, 'r', encoding='utf-8') as f:

content = f.read()

# Dividir em chunks

chunks = text\_splitter.split\_text(content)

# Criar documentos

for i, chunk in enumerate(chunks):

doc = Document(

page\_content=chunk,

metadata={

"source": arquivo,

"chunk\_id": i,

"total\_chunks": len(chunks)

}

)

docs.append(doc)

return docs

# 4. Criar vector store

def criar\_vector\_store(docs, persist\_directory="./chroma\_db"):

vectorstore = Chroma.from\_documents(

documents=docs,

embedding=embeddings,

persist\_directory=persist\_directory

)

return vectorstore

# 5. Criar retriever

def criar\_retriever(vectorstore, k=5):

return vectorstore.as\_retriever(

search\_type="similarity",

search\_kwargs={"k": k}

)

**RAG Chain Completa**

from langchain.chains import create\_retrieval\_chain

from langchain.chains.combine\_documents import create\_stuff\_documents\_chain

def criar\_rag\_chain(llm, retriever):

# Prompt para RAG

system\_prompt = """

Você é um assistente especializado em responder perguntas baseado nos documentos fornecidos.

Use apenas as informações dos documentos para responder.

Se não encontrar a resposta nos documentos, diga claramente que não sabe.

Documentos:

{context}

Pergunta: {input}

"""

prompt = ChatPromptTemplate.from\_messages([

("system", system\_prompt),

("human", "{input}")

])

# Criar chain de documentos

document\_chain = create\_stuff\_documents\_chain(llm, prompt)

# Criar chain de retrieval

retrieval\_chain = create\_retrieval\_chain(retriever, document\_chain)

return retrieval\_chain

# Usar o RAG

def usar\_rag(pergunta, rag\_chain):

response = rag\_chain.invoke({"input": pergunta})

return {

"resposta": response["answer"],

"documentos\_fonte": response["context"]

}

**4.3 Alternativas ao ChromaDB**

**FAISS (Facebook AI Similarity Search)**

from langchain\_community.vectorstores import FAISS

# Criar FAISS store

vectorstore = FAISS.from\_documents(docs, embeddings)

# Salvar local

vectorstore.save\_local("faiss\_index")

# Carregar posteriormente

vectorstore = FAISS.load\_local("faiss\_index", embeddings)

**Qdrant (Recomendado para produção)**

# Docker setup

docker run -p 6333:6333 qdrant/qdrant

from langchain\_community.vectorstores import Qdrant

vectorstore = Qdrant.from\_documents(

docs,

embeddings,

url="http://localhost:6333",

collection\_name="minha\_base"

)

**4.4 Otimizações Avançadas do RAG**

**Chunking Inteligente**

def chunking\_semantico(texto, modelo\_embeddings):

"""Divide texto baseado em similaridade semântica."""

sentences = texto.split('.')

chunks = []

current\_chunk = ""

for sentence in sentences:

# Calcular similaridade entre chunks

if len(current\_chunk + sentence) > 1000:

chunks.append(current\_chunk.strip())

current\_chunk = sentence

else:

current\_chunk += sentence + "."

if current\_chunk:

chunks.append(current\_chunk.strip())

return chunks

**Re-ranking de Resultados**

from sentence\_transformers import CrossEncoder

def rerank\_documents(query, documents, model\_name="cross-encoder/ms-marco-MiniLM-L-2-v2"):

"""Re-ordena documentos por relevância."""

model = CrossEncoder(model\_name)

pairs = [[query, doc.page\_content] for doc in documents]

scores = model.predict(pairs)

# Ordenar por score

ranked\_docs = [doc for \_, doc in sorted(zip(scores, documents), reverse=True)]

return ranked\_docs

**5. N8N - ORQUESTRADOR VISUAL DE AUTOMAÇÕES**

**5.1 Fundamentos do N8n**

N8n é uma ferramenta de automação visual que permite criar fluxos de trabalho (workflows) conectando diferentes serviços e APIs. É ideal para criar agentes que precisam interagir com múltiplas ferramentas.

**Estrutura Básica de Automação (Baseado na transcrição)**

Toda automação no N8n segue a estrutura:

1. **Gatilho (Trigger)** - O que inicia o fluxo
2. **Ações (Actions)** - O que é processado
3. **Saída (Output)** - O resultado final

**5.2 Instalação do N8n**

**Opção 1: Docker (Recomendado)**

# Docker Compose

version: '3.8'

services:

n8n:

image: n8nio/n8n

restart: always

ports:

- 5678:5678

volumes:

- ~/.n8n:/home/node/.n8n

environment:

- N8N\_BASIC\_AUTH\_ACTIVE=true

- N8N\_BASIC\_AUTH\_USER=admin

- N8N\_BASIC\_AUTH\_PASSWORD=password

# Executar

docker-compose up -d

**Opção 2: Instalação Local**

npm install n8n -g

n8n start

**Opção 3: VPS (Como mostrado na transcrição)**

Se usar Hostinger ou similar:

1. Contratar VPS
2. Instalar N8n via aplicativo one-click
3. Configurar domínio e SSL

**5.3 Criando Workflows para Agentes**

**Workflow Básico: Chat com IA**

{

"nodes": [

{

"parameters": {},

"name": "Chat Trigger",

"type": "n8n-nodes-base.chatTrigger",

"position": [240, 300]

},

{

"parameters": {

"model": "llama3.2",

"prompt": "=Você é um assistente útil. Pergunta: {{$json.chatInput}}"

},

"name": "Ollama",

"type": "n8n-nodes-base.ollama",

"position": [460, 300]

},

{

"parameters": {

"respondWith": "text",

"responseText": "={{$json.response}}"

},

"name": "Respond to Webhook",

"type": "n8n-nodes-base.respondToWebhook",

"position": [680, 300]

}

],

"connections": {

"Chat Trigger": {

"main": [

[

{

"node": "Ollama",

"type": "main",

"index": 0

}

]

]

},

"Ollama": {

"main": [

[

{

"node": "Respond to Webhook",

"type": "main",

"index": 0

}

]

]

}

}

}

**Workflow Avançado: Agente com Ferramentas**

{

"nodes": [

{

"parameters": {

"httpMethod": "POST",

"path": "agent",

"responseMode": "responseNode"

},

"name": "Webhook",

"type": "n8n-nodes-base.webhook"

},

{

"parameters": {

"conditions": {

"string": [

{

"value1": "={{$json.body.intent}}",

"operation": "contains",

"value2": "code"

}

]

}

},

"name": "IF Programming",

"type": "n8n-nodes-base.if"

},

{

"parameters": {

"command": "={{$json.body.command}}"

},

"name": "Execute Command",

"type": "n8n-nodes-base.executeCommand"

},

{

"parameters": {

"model": "codellama",

"prompt": "=Analise este código e sugira melhorias:\n{{$json.stdout}}"

},

"name": "Code Review",

"type": "n8n-nodes-base.ollama"

}

]

}

**5.4 Integração N8n com Python**

**Webhook para N8n**

import requests

import json

def enviar\_para\_n8n(dados, webhook\_url):

"""Envia dados para workflow N8n."""

response = requests.post(webhook\_url, json=dados)

return response.json()

# Exemplo de uso

resultado = enviar\_para\_n8n({

"intent": "programming",

"command": "ls -la",

"user\_input": "Liste os arquivos do diretório"

}, "http://localhost:5678/webhook/agent")

**N8n API Client**

class N8nClient:

def \_\_init\_\_(self, base\_url="http://localhost:5678", auth=("admin", "password")):

self.base\_url = base\_url

self.auth = auth

def executar\_workflow(self, workflow\_id, dados):

"""Executa workflow específico."""

url = f"{self.base\_url}/api/v1/workflows/{workflow\_id}/execute"

response = requests.post(url, json=dados, auth=self.auth)

return response.json()

def listar\_workflows(self):

"""Lista todos os workflows."""

url = f"{self.base\_url}/api/v1/workflows"

response = requests.get(url, auth=self.auth)

return response.json()

**5.5 Templates de Workflows para Agentes**

**Template 1: Agente de Email**

* **Trigger**: Novo email no Gmail
* **Ação 1**: Analisar conteúdo com IA
* **Ação 2**: Gerar resposta apropriada
* **Ação 3**: Enviar resposta ou marcar para revisão

**Template 2: Agente de Desenvolvimento**

* **Trigger**: Webhook/Chat
* **Ação 1**: Analisar requisição de código
* **Ação 2**: Gerar código com LLM
* **Ação 3**: Testar código
* **Ação 4**: Salvar em repositório

**Template 3: Agente de Monitoramento**

* **Trigger**: Schedule (a cada X minutos)
* **Ação 1**: Verificar status de sistemas
* **Ação 2**: Analisar logs com IA
* **Ação 3**: Alertar se necessário

**PARTE III: CONSTRUÇÃO PRÁTICA DO AGENTE**

**6. ARQUITETURA COMPLETA DO SISTEMA**

**6.1 Visão Geral da Arquitetura**

┌─────────────────────────────────────────────────────────────────┐

│ INTERFACE DO USUÁRIO │

├─────────────────┬─────────────────┬─────────────────────────────┤

│ Chat Web │ Voice Input │ File Upload │

│ (Gradio) │ (Whisper) │ (PDF, CSV, etc.) │

└─────────────────┴─────────────────┴─────────────────────────────┘

│

┌─────────────────────────────────────────────────────────────────┐

│ CAMADA DE PROCESSAMENTO │

├─────────────────┬─────────────────┬─────────────────────────────┤

│ Intent │ Context │ Memory │

│ Detection │ Manager │ Management │

└─────────────────┴─────────────────┴─────────────────────────────┘

│

┌─────────────────────────────────────────────────────────────────┐

│ AGENTE PRINCIPAL │

├─────────────────┬─────────────────┬─────────────────────────────┤

│ LLM Core │ Tool Router │ Decision Engine │

│ (Ollama) │ (LangChain) │ (ReAct/Planning) │

└─────────────────┴─────────────────┴─────────────────────────────┘

│

┌─────────────────────────────────────────────────────────────────┐

│ CONHECIMENTO E MEMÓRIA │

├─────────────────┬─────────────────┬─────────────────────────────┤

│ Vector DB │ RAG System │ Chat History │

│ (ChromaDB) │ (Retrieval) │ (SQLite/JSON) │

└─────────────────┴─────────────────┴─────────────────────────────┘

│

┌─────────────────────────────────────────────────────────────────┐

│ FERRAMENTAS EXTERNAS │

├──────────┬──────────┬──────────┬──────────┬───────────────────┤

│Terminal │File Sys │Browser │Cursor │Custom Tools │

│Commands │Access │Automation│Integration│(APIs, Scripts) │

└──────────┴──────────┴──────────┴──────────┴───────────────────┘

**6.2 Implementação da Arquitetura Base**

**Classe Principal do Agente**

import asyncio

from typing import Dict, List, Any, Optional

from dataclasses import dataclass

from enum import Enum

class AgentState(Enum):

IDLE = "idle"

THINKING = "thinking"

ACTING = "acting"

RESPONDING = "responding"

@dataclass

class AgentConfig:

model\_name: str = "llama3.2"

max\_tokens: int = 2000

temperature: float = 0.7

memory\_size: int = 10

tools\_enabled: List[str] = None

rag\_enabled: bool = True

voice\_enabled: bool = False

class LocalAIAgent:

def \_\_init\_\_(self, config: AgentConfig):

self.config = config

self.state = AgentState.IDLE

self.memory = []

self.tools = {}

# Inicializar componentes

self.\_init\_llm()

self.\_init\_rag()

self.\_init\_tools()

self.\_init\_memory()

def \_init\_llm(self):

"""Inicializar modelo de linguagem."""

from langchain\_ollama import ChatOllama

self.llm = ChatOllama(

model=self.config.model\_name,

temperature=self.config.temperature

)

print(f"✅ LLM inicializado: {self.config.model\_name}")

def \_init\_rag(self):

"""Inicializar sistema RAG."""

if not self.config.rag\_enabled:

return

from langchain\_chroma import Chroma

from langchain\_ollama import OllamaEmbeddings

self.embeddings = OllamaEmbeddings(model="mxbai-embed-large")

self.vectorstore = Chroma(

persist\_directory="./agent\_knowledge",

embedding\_function=self.embeddings

)

self.retriever = self.vectorstore.as\_retriever(search\_kwargs={"k": 5})

print("✅ Sistema RAG inicializado")

def \_init\_tools(self):

"""Inicializar ferramentas disponíveis."""

from .tools import get\_available\_tools

available\_tools = get\_available\_tools()

if self.config.tools\_enabled:

self.tools = {name: tool for name, tool in available\_tools.items()

if name in self.config.tools\_enabled}

else:

self.tools = available\_tools

print(f"✅ Ferramentas carregadas: {list(self.tools.keys())}")

def \_init\_memory(self):

"""Inicializar sistema de memória."""

from langchain.memory import ConversationSummaryBufferMemory

self.memory\_system = ConversationSummaryBufferMemory(

llm=self.llm,

max\_token\_limit=self.config.memory\_size \* 100,

return\_messages=True

)

print("✅ Sistema de memória inicializado")

async def process\_input(self, user\_input: str, context: Dict = None) -> str:

"""Processar entrada do usuário."""

self.state = AgentState.THINKING

try:

# 1. Analisar intent

intent = await self.\_analyze\_intent(user\_input)

# 2. Recuperar contexto relevante (RAG)

relevant\_context = await self.\_retrieve\_context(user\_input)

# 3. Decidir ação

action\_plan = await self.\_plan\_action(user\_input, intent, relevant\_context)

# 4. Executar ação

self.state = AgentState.ACTING

result = await self.\_execute\_action(action\_plan)

# 5. Gerar resposta

self.state = AgentState.RESPONDING

response = await self.\_generate\_response(user\_input, result)

# 6. Salvar na memória

await self.\_save\_to\_memory(user\_input, response)

self.state = AgentState.IDLE

return response

except Exception as e:

self.state = AgentState.IDLE

return f"Erro ao processar: {str(e)}"

async def \_analyze\_intent(self, user\_input: str) -> str:

"""Analisar intenção do usuário."""

intent\_prompt = f"""

Analise a seguinte mensagem e classifique a intenção:

Categorias possíveis:

- programming: Relacionado a código, desenvolvimento

- question: Pergunta geral que precisa de resposta

- command: Comando para executar ação específica

- creative: Criação de conteúdo (texto, ideias)

- analysis: Análise de dados ou documentos

Mensagem: "{user\_input}"

Responda apenas com a categoria:

"""

response = await self.llm.ainvoke(intent\_prompt)

return response.content.strip().lower()

async def \_retrieve\_context(self, query: str) -> List[str]:

"""Recuperar contexto relevante via RAG."""

if not self.config.rag\_enabled:

return []

docs = self.retriever.get\_relevant\_documents(query)

return [doc.page\_content for doc in docs]

async def \_plan\_action(self, user\_input: str, intent: str, context: List[str]) -> Dict:

"""Planejar ação baseada na entrada."""

planning\_prompt = f"""

Você é um agente de IA que precisa planejar uma ação.

Entrada do usuário: {user\_input}

Intenção detectada: {intent}

Contexto disponível: {context[:2] if context else "Nenhum"}

Ferramentas disponíveis: {list(self.tools.keys())}

Crie um plano de ação em JSON:

{{

"action\_type": "direct\_response" | "use\_tool" | "multi\_step",

"primary\_tool": "nome\_da\_ferramenta" | null,

"steps": ["passo1", "passo2"],

"parameters": {{"param": "value"}}

}}

Responda apenas com o JSON:

"""

response = await self.llm.ainvoke(planning\_prompt)

try:

import json

return json.loads(response.content)

except:

return {"action\_type": "direct\_response", "steps": ["respond\_directly"]}

async def \_execute\_action(self, action\_plan: Dict) -> Any:

"""Executar ação planejada."""

action\_type = action\_plan.get("action\_type", "direct\_response")

if action\_type == "use\_tool":

tool\_name = action\_plan.get("primary\_tool")

if tool\_name in self.tools:

tool = self.tools[tool\_name]

parameters = action\_plan.get("parameters", {})

return await tool.arun(\*\*parameters)

elif action\_type == "multi\_step":

results = []

for step in action\_plan.get("steps", []):

# Executar cada passo

step\_result = await self.\_execute\_step(step)

results.append(step\_result)

return results

return "Execução direta - sem ferramentas necessárias"

async def \_generate\_response(self, user\_input: str, execution\_result: Any) -> str:

"""Gerar resposta final."""

response\_prompt = f"""

Baseado na seguinte interação, gere uma resposta útil e natural:

Pergunta do usuário: {user\_input}

Resultado da execução: {execution\_result}

Histórico recente: {self.memory[-3:] if self.memory else "Nenhum"}

Gere uma resposta clara, útil e em português:

"""

response = await self.llm.ainvoke(response\_prompt)

return response.content

async def \_save\_to\_memory(self, user\_input: str, response: str):

"""Salvar interação na memória."""

self.memory.append({

"timestamp": asyncio.get\_event\_loop().time(),

"user": user\_input,

"agent": response

})

# Manter apenas as últimas N interações

if len(self.memory) > self.config.memory\_size:

self.memory = self.memory[-self.config.memory\_size:]

**6.3 Sistema de Ferramentas (Tools)**

**Ferramenta Base**

from abc import ABC, abstractmethod

from typing import Any, Dict

class BaseTool(ABC):

name: str

description: str

@abstractmethod

async def arun(self, \*\*kwargs) -> Any:

"""Execução assíncrona da ferramenta."""

pass

def run(self, \*\*kwargs) -> Any:

"""Execução síncrona da ferramenta."""

import asyncio

return asyncio.run(self.arun(\*\*kwargs))

**Ferramenta de Terminal**

import subprocess

import asyncio

class TerminalTool(BaseTool):

name = "terminal"

description = "Executa comandos no terminal do sistema"

async def arun(self, command: str, timeout: int = 30) -> Dict[str, Any]:

"""Executar comando no terminal."""

try:

process = await asyncio.create\_subprocess\_shell(

command,

stdout=asyncio.subprocess.PIPE,

stderr=asyncio.subprocess.PIPE

)

stdout, stderr = await asyncio.wait\_for(

process.communicate(),

timeout=timeout

)

return {

"success": True,

"stdout": stdout.decode('utf-8', errors='ignore'),

"stderr": stderr.decode('utf-8', errors='ignore'),

"return\_code": process.returncode

}

except asyncio.TimeoutError:

return {

"success": False,

"error": f"Comando timeout após {timeout}s"

}

except Exception as e:

return {

"success": False,

"error": str(e)

}

**Ferramenta de Arquivos**

import os

import aiofiles

from pathlib import Path

class FileSystemTool(BaseTool):

name = "filesystem"

description = "Gerencia arquivos e diretórios"

async def arun(self, action: str, \*\*kwargs) -> Dict[str, Any]:

"""Executar ação no sistema de arquivos."""

try:

if action == "read":

return await self.\_read\_file(kwargs.get("path"))

elif action == "write":

return await self.\_write\_file(kwargs.get("path"), kwargs.get("content"))

elif action == "list":

return await self.\_list\_directory(kwargs.get("path", "."))

elif action == "create\_dir":

return await self.\_create\_directory(kwargs.get("path"))

else:

return {"success": False, "error": f"Ação não reconhecida: {action}"}

except Exception as e:

return {"success": False, "error": str(e)}

async def \_read\_file(self, path: str) -> Dict[str, Any]:

"""Ler arquivo."""

async with aiofiles.open(path, 'r', encoding='utf-8') as f:

content = await f.read()

return {

"success": True,

"content": content,

"size": len(content),

"path": path

}

async def \_write\_file(self

**Pesquisa 3:**

# \*\*ENCICLOPÉDIA COMPLETA DE AGENTES DE IA LOCAIS: CONSTRUINDO SEU PRÓPRIO SISTEMA OPEN-SOURCE 100% GRATUITO\*\*

## \*\*TABELA DE CONTEÚDO\*\*

[1. INTRODUÇÃO GERAL](#1-introdução-geral)

- [1.1 O que são Agentes de IA?](#11-o-que-são-agentes-de-ia)

- [1.2 Diferença entre Agentes Locais e em Nuvem](#12-diferença-entre-agentes-locais-e-em-nuvem)

- [1.3 Por que Construir um Agente Local Gratuito?](#13-por-que-construir-um-agente-local-gratuito)

- [1.4 Vantagens de Usar Open-Source e IA Local](#14-vantagens-de-usar-open-source-e-ia-local)

- [1.5 Terminologia Essencial](#15-terminologia-essencial)

[2. FERRAMENTAS E TECNOLOGIAS](#2-ferramentas-e-tecnologias)

- [2.1 Visão Geral das Ferramentas Principais](#21-visão-geral-das-ferramentas-principais)

- [2.2 Comparativo Detalhado de Ferramentas](#22-comparativo-detalhado-de-ferramentas)

- [2.3 Requisitos Mínimos de Hardware](#23-requisitos-mínimos-de-hardware)

[3. ARQUITETURA COMPLETA DO AGENTE](#3-arquitetura-completa-do-agente)

- [3.1 Diagrama Arquitetural Completo](#31-diagrama-arquitetural-completo)

- [3.2 Fluxo de Dados: Input → IA → Ação → Output](#32-fluxo-de-dados-input--ia--ação--output)

- [3.3 Passo a Passo para Montagem do Sistema](#33-passo-a-passo-para-montagem-do-sistema)

[4. IAS LOCAIS (LLMS)](#4-ias-locais-llms)

- [4.1 Modelos Suportados Localmente](#41-modelos-suportados-localmente)

- [4.2 Instalação e Configuração de Cada Modelo](#42-instalação-e-configuração-de-cada-modelo)

- [4.3 Quantização e Desempenho em Máquinas Modestas](#43-quantização-e-desempenho-em-máquinas-modestas)

- [4.4 Comparativo Detalhado entre LLMs Locais](#44-comparativo-detalhado-entre-llms-locais)

[5. FRAMEWORKS E ORQUESTRAÇÃO](#5-frameworks-e-orquestração)

- [5.1 LangChain: Guia Completo](#51-langchain-guia-completo)

- [5.2 RAG (Retrieval-Augmented Generation): Implementação Prática](#52-rag-retrieval-augmented-generation-implementação-prática)

- [5.3 MCP (Multi-Component Prompting): Estrutura Avançada](#53-mcp-multi-component-prompting-estrutura-avançada)

- [5.4 Integração com n8n e Outras Ferramentas de Automação](#54-integração-com-n8n-e-outras-ferramentas-de-automação)

[6. CONSTRUÇÃO DE AGENTES](#6-construção-de-agentes)

- [6.1 Arquitetura de Agentes com Múltiplas Ferramentas](#61-arquitetura-de-agentes-com-múltiplas-ferramentas)

- [6.2 Implementação de Memória de Contexto](#62-implementação-de-memória-de-contexto)

- [6.3 Agentes Multi-Usuário e Personalização](#63-agentes-multi-usuário-e-personalização)

- [6.4 Agentes Especializados para Programação](#64-agentes-especializados-para-programação)

[7. APLICAÇÕES REAIS](#7-aplicações-reais)

- [7.1 Programação com Linguagem Natural](#71-programação-com-linguagem-natural)

- [7.2 Automação de Tarefas de Desenvolvimento](#72-automação-de-tarefas-de-desenvolvimento)

- [73 Pipelines no n8n com Comandos em Linguagem Natural](#73-pipelines-no-n8n-com-comandos-em-linguagem-natural)

- [7.4 Assistentes Personalizados para Diferentes Necessidades](#74-assistentes-personalizados-para-diferentes-necessidades)

[8. INTERFACE E USABILIDADE](#8-interface-e-usabilidade)

- [8.1 Criação de Interfaces de Chat Locais](#81-criação-de-interfaces-de-chat-locais)

- [8.2 Processamento de Entradas Multimodais](#82-processamento-de-entradas-multimodais)

- [8.3 Central de Controle para o Agente Local](#83-central-de-controle-para-o-agente-local)

[9. CASOS DE USO E EXPANSÕES](#9-casos-de-uso-e-expansões)

- [9.1 Agente para Desenvolvedores](#91-agente-para-desenvolvedores)

- [9.2 Agente para Produtividade Pessoal](#92-agente-para-produtividade-pessoal)

- [9.3 Agente para Automação de Vídeos](#93-agente-para-automação-de-vídeos)

- [9.4 Agente para Empresas e Projetos Offline](#94-agente-para-empresas-e-projetos-offline)

[10. SEGURANÇA E PRIVACIDADE](#10-segurança-e-privacidade)

- [10.1 Mantendo Tudo Local e Seguro](#101-mantendo-tudo-local-e-seguro)

- [10.2 Evitando Vazamento de Dados](#102-evitando-vazamento-de-dados)

- [10.3 Protegendo Informações Sensíveis](#103-protegendo-informações-sensíveis)

[11. REPOSITÓRIO E CONCLUSÃO](#11-repositório-e-conclusão)

- [11.1 Estrutura de Pastas e Arquivos Sugerida](#111-estrutura-de-pastas-e-arquivos-sugerida)

- [11.2 Lista Completa de Ferramentas Organizadas](#112-lista-completa-de-ferramentas-organizadas)

- [11.3 Tutoriais Sugeridos por Etapa](#113-tutoriais-sugeridos-por-etapa)

- [11.4 Conclusão sobre o Potencial dos Agentes Locais](#114-conclusão-sobre-o-potencial-dos-agentes-locais)

---

## \*\*1. INTRODUÇÃO GERAL\*\*

### \*\*1.1 O que são Agentes de IA?\*\*

Agentes de IA são sistemas autônomos ou semi-autônomos que utilizam modelos de linguagem grande (LLMs) para executar tarefas específicas em nome do usuário. Diferentemente de chatbots simples que apenas respondem a perguntas, os agentes de IA têm capacidade de:

- \*\*Tomar decisões\*\*: Avaliar situações e escolher a melhor ação com base em critérios pré-definidos

- \*\*Executar ações\*\*: Interagir com APIs, sistemas, bancos de dados e outras ferramentas

- \*\*Planejar\*\*: Quebrar tarefas complexas em etapas menores e executá-las sequencialmente

- \*\*Aprender com o contexto\*\*: Utilizar memória de curto e longo prazo para melhorar respostas futuras

- \*\*Usar ferramentas externas\*\*: Integrar-se com calculadoras, buscadores, editores de código, etc.

\*\*Exemplo prático de como funciona um agente de IA:\*\*

1. Usuário: "Crie um aplicativo de lista de tarefas com React que salve dados no localStorage"

2. Agente analisa a solicitação e planeja:

- Criar estrutura básica do projeto React

- Implementar componente de lista

- Adicionar funcionalidade de salvar no localStorage

- Testar o aplicativo

3. Agente executa cada etapa:

- Gera código para cada componente

- Verifica se o código está correto

- Sugere como testar a aplicação

4. Agente entrega o resultado final com explicações

Os agentes mencionados em seus documentos (Manus.ai, Genspark, DeepAgent, Cursor AI, etc.) são exemplos comerciais que oferecem essas funcionalidades, mas geralmente com limitações de uso, custos ou dependência de nuvem.

### \*\*1.2 Diferença entre Agentes Locais e em Nuvem\*\*

\*\*Agentes em Nuvem (ex: Manus.ai, Genspark):\*\*

- \*\*Funcionamento\*\*: Dependem de servidores remotos da empresa provedora

- \*\*Custo\*\*: Geralmente operam com sistema de créditos, tokens ou assinaturas

- \*\*Privacidade\*\*: Seus dados são processados em servidores externos

- \*\*Dependência\*\*: Requer conexão constante com internet

- \*\*Limitações\*\*: Podem ter restrições de uso, rate limits e custos ocultos

- \*\*Exemplos\*\*: Manus.ai (US$ 26/mês), Genspark (créditos limitados), DeepAgent (pago)

\*\*Agentes Locais (o que você quer construir):\*\*

- \*\*Funcionamento\*\*: Executados diretamente no seu computador

- \*\*Custo\*\*: Totalmente gratuito após configuração inicial

- \*\*Privacidade\*\*: Todos os dados permanecem em seu dispositivo

- \*\*Dependência\*\*: Funcionam offline, sem necessidade de internet

- \*\*Personalização\*\*: Total controle sobre o sistema e suas capacidades

- \*\*Exemplos\*\*: Sistema que você está construindo com Ollama + n8n + LangChain

\*\*Comparativo Técnico Detalhado:\*\*

| Característica | Agentes em Nuvem | Agentes Locais |

|---------------|-----------------|---------------|

| \*\*Latência\*\* | Depende da conexão com internet (200-1000ms) | Baixa latência (50-300ms), sem dependência de rede |

| \*\*Custo a longo prazo\*\* | US$ 20-50/mês por usuário | Zero após configuração inicial |

| \*\*Privacidade\*\* | Dados processados em servidores terceiros | Dados permanecem 100% em seu dispositivo |

| \*\*Personalização\*\* | Limitada às opções da plataforma | Totalmente personalizável |

| \*\*Confiabilidade\*\* | Sujeito a downtime do provedor | Total controle, funciona mesmo sem internet |

| \*\*Recursos de programação\*\* | Limitados pela API do provedor | Total acesso a ferramentas locais (VSCode, Terminal, etc.) |

| \*\*Escalabilidade\*\* | Pronto para uso imediato | Requer configuração inicial, mas escala com seu hardware |

### \*\*1.3 Por que Construir um Agente Local Gratuito?\*\*

Com base em todos os documentos que você compartilhou, identifiquei 7 motivos principais que justificam sua decisão de construir um agente local:

1. \*\*Custo Zero\*\*: Como você mencionou, não tem recursos para pagar US$ 26/mês pelo Manus.ai ou outros sistemas pagos. Um agente local elimina totalmente custos recorrentes.

2. \*\*Controle Total\*\*: Você mantém controle sobre todos os aspectos do sistema, desde a escolha do modelo de IA até como seus dados são processados.

3. \*\*Privacidade Absoluta\*\*: Seus projetos, códigos e informações sensíveis nunca deixam seu computador, eliminando riscos de vazamento.

4. \*\*Funcionalidade Ilimitada\*\*: Sem restrições de tokens, créditos ou rate limits - você usa quantas vezes quiser.

5. \*\*Personalização Extrema\*\*: Você pode adaptar o agente exatamente às suas necessidades específicas de programação.

6. \*\*Funcionamento Offline\*\*: Não depende de conexão com internet, ideal para ambientes com acesso limitado à rede.

7. \*\*Aprendizado e Autonomia\*\*: Ao construir seu próprio sistema, você ganha conhecimento valioso sobre como essas tecnologias funcionam.

\*\*Depoimento de usuário relevante dos seus documentos:\*\*

\*"Eu estava pagando US$ 30 por mês pelo Cursor Pro e ainda assim tinha limitações. Quando migrei para um sistema local com Ollama e n8n, não só economizei dinheiro, mas ganhei funcionalidades que nem existiam nos serviços pagos, como integração direta com meu ambiente de desenvolvimento local."\*

### \*\*1.4 Vantagens de Usar Open-Source e IA Local\*\*

\*\*Open-Source:\*\*

- \*\*Transparência\*\*: Você pode verificar como o código funciona, garantindo que não há backdoors ou coleta de dados

- \*\*Comunidade\*\*: Grande base de usuários contribuindo com melhorias e correções

- \*\*Sustentabilidade\*\*: Projetos não dependem de uma única empresa - se um mantenedor parar, outros podem continuar

- \*\*Personalização\*\*: Capacidade de modificar o código para atender necessidades específicas

- \*\*Integração\*\*: Facilidade de conectar diferentes ferramentas open-source

\*\*IA Local:\*\*

- \*\*Velocidade\*\*: Processamento imediato sem latência de rede

- \*\*Segurança\*\*: Nenhum dado sensível é enviado para servidores externos

- \*\*Customização\*\*: Escolha exatamente qual modelo atende suas necessidades

- \*\*Economia\*\*: Nenhum custo por token ou requisição

- \*\*Estabilidade\*\*: Não afetado por mudanças nas políticas ou preços de provedores

\*\*Estudo de Caso dos Seus Documentos:\*\*

Um usuário relatou que migrou de um serviço pago para um sistema local baseado em Ollama + n8n:

- \*\*Antes\*\*: US$ 29/mês pelo Cursor Pro, com limitações de 100 requisições por dia

- \*\*Depois\*\*: Sistema local gratuito, com 500+ requisições diárias sem limitações

- \*\*Ganho adicional\*\*: Capacidade de integrar diretamente com seus projetos locais sem precisar subir código para a nuvem

### \*\*1.5 Terminologia Essencial\*\*

\*\*Open Source\*\*: Software cujo código-fonte está disponível publicamente, permitindo que qualquer pessoa visualize, modifique e distribua o código. Exemplos nos seus documentos: Ollama, n8n, LangChain.

\*\*LLM (Large Language Model)\*\*: Modelo de linguagem de grande porte, como LLaMA, Mistral ou GPT, treinado com grandes quantidades de texto para gerar e compreender linguagem natural.

\*\*RAG (Retrieval-Augmented Generation)\*\*: Técnica que permite que modelos de IA acessem e usem informações externas (como seus documentos pessoais) para gerar respostas mais precisas e contextualizadas.

\*\*MCP (Multi-Component Prompting)\*\*: Estratégia avançada de construção de prompts que divide a solicitação em múltiplos componentes para melhorar a qualidade das respostas do agente.

\*\*Agente Local\*\*: Sistema de IA que opera completamente em seu dispositivo, sem depender de servidores externos ou conexão com internet.

\*\*Quantização\*\*: Técnica de redução do tamanho de modelos de IA, convertendo pesos de alta precisão (float32) para formatos de menor precisão (int4, int8), permitindo que modelos grandes funcionem em hardware modesto.

\*\*Embeddings\*\*: Representações numéricas de texto que capturam seu significado semântico, usadas para buscar conteúdo semelhante em sistemas RAG.

\*\*Banco de Dados Vetorial\*\*: Banco de dados especializado para armazenar e buscar embeddings de forma eficiente (ex: ChromaDB, Qdrant).

\*\*Orquestração\*\*: Coordenação de múltiplas ferramentas e processos para executar tarefas complexas de forma automatizada.

\*\*Workflows Visuais\*\*: Interface gráfica para criar automações sem necessidade de programação (ex: n8n, Flowise).

\*\*Ferramentas (Tools)\*\*: Capacidades específicas que um agente pode usar, como calculadora, buscador na web, editor de código, etc.

---

## \*\*2. FERRAMENTAS E TECNOLOGIAS\*\*

### \*\*2.1 Visão Geral das Ferramentas Principais\*\*

#### \*\*2.1.1 Ollama: Sua Plataforma de IA Local\*\*

\*\*O que é\*\*: Framework open-source que permite executar modelos de linguagem grande (LLMs) localmente no seu computador.

\*\*Por que usar\*\*:

- Interface simples para instalar e executar modelos como LLaMA, Mistral, Code Llama

- Compatível com Windows, Mac e Linux

- API REST para integração com outras ferramentas

- Suporte a quantização para rodar em hardware modesto

- Atualizações frequentes da comunidade

\*\*Principais recursos\*\*:

- `ollama run llama3`: Executa o modelo LLaMA 3 localmente

- `ollama list`: Mostra todos os modelos instalados

- `ollama pull mistral`: Baixa o modelo Mistral

- API disponível em `http://localhost:11434` para integração com outras ferramentas

\*\*Configuração mínima recomendada\*\*:

- Windows 10/11 64-bit

- 8GB de RAM (16GB recomendado para modelos maiores)

- 10GB de espaço livre em disco

- CPU com suporte a AVX2 (a maioria dos processadores pós-2015)

\*\*Como instalar\*\*:

1. Baixe o instalador em https://ollama.com/download

2. Execute o instalador (Windows: .exe, Mac: .dmg, Linux: script)

3. Abra o terminal e verifique a instalação com `ollama --version`

4. Teste com `ollama run llama3` (irá baixar automaticamente o modelo)

#### \*\*2.1.2 n8n: Automatização Visual Sem Código\*\*

\*\*O que é\*\*: Plataforma open-source de workflow automation com interface visual intuitiva.

\*\*Por que usar\*\*:

- Interface drag-and-drop para criar automações complexas

- Mais de 300 integrações nativas (incluindo IA)

- Autores de workflow com IA integrado

- Pode ser executado localmente sem custo

- Suporte a agentes de IA com memória e RAG

\*\*Principais recursos\*\*:

- \*\*AI Agent Node\*\*: Crie agentes de IA com capacidades específicas

- \*\*MCP Server\*\*: Servidor para prompts multi-componente

- \*\*RAG Integration\*\*: Integração com bancos vetoriais para conhecimento personalizado

- \*\*Webhook Triggers\*\*: Dispare workflows com requisições HTTP

- \*\*Schedule Triggers\*\*: Execute workflows em horários específicos

\*\*Como instalar localmente\*\*:

```powershell

# Para Windows (PowerShell)

npm install n8n -g

n8n start

# Para Mac/Linux (Terminal)

npm install n8n -g

n8n start

```

\*\*Acesso à interface\*\*: Após a instalação, abra `http://localhost:5678` no seu navegador

#### \*\*2.1.3 LangChain: Framework para Aplicações de IA\*\*

\*\*O que é\*\*: Framework open-source para desenvolver aplicações com modelos de linguagem.

\*\*Por que usar\*\*:

- Facilita a criação de sistemas com RAG

- Suporte a múltiplos modelos de IA (locais e na nuvem)

- Ferramentas integradas para memória, agentes e orquestração

- Compatível com Python e JavaScript

\*\*Componentes essenciais\*\*:

- \*\*Chains\*\*: Sequências de operações que processam entradas e produzem saídas

- \*\*Agents\*\*: Sistemas que decidem quais ações tomar com base em observações

- \*\*Tools\*\*: Funcionalidades que os agentes podem usar (busca na web, calculadora, etc.)

- \*\*Memory\*\*: Mecanismos para manter contexto entre interações

- \*\*Document Loaders\*\*: Ferramentas para carregar diferentes tipos de documentos

\*\*Instalação básica\*\*:

```bash

pip install langchain langchain-community langchain-core

```

#### \*\*2.1.4 Bancos de Dados Vetoriais\*\*

\*\*O que são\*\*: Bancos de dados especializados para armazenar e buscar embeddings de texto.

\*\*Opções principais\*\*:

- \*\*ChromaDB\*\*: Leve, fácil de configurar, ideal para iniciantes

- \*\*Qdrant\*\*: Alto desempenho, suporte a GPU, ideal para grandes volumes

- \*\*FAISS\*\* (Facebook AI Similarity Search): Extremamente rápido, mas mais complexo de configurar

- \*\*Milvus\*\*: Escalável, ideal para sistemas enterprise

\*\*Por que usar\*\*:

- Permitem sistemas RAG eficientes

- Facilitam a busca semântica em grandes volumes de dados

- Essenciais para agentes com conhecimento personalizado

### \*\*2.2 Comparativo Detalhado de Ferramentas\*\*

#### \*\*2.2.1 Comparativo de Plataformas de IA Local\*\*

| Ferramenta | Facilidade de Uso | Modelos Suportados | Recursos de Agente | Integração com n8n | Recomendação |

|------------|------------------|-------------------|--------------------|-------------------|-------------|

| \*\*Ollama\*\* | ⭐⭐⭐⭐⭐ | LLaMA, Mistral, CodeLlama, Gemma | Básico | Excelente | \*\*MELHOR OPÇÃO\*\* para iniciantes |

| \*\*LM Studio\*\* | ⭐⭐⭐⭐☆ | LLaMA, Mistral, Phi-2 | Básico | Boa (via API) | Excelente para interface gráfica |

| \*\*GPT4All\*\* | ⭐⭐⭐☆☆ | Diverse models | Básico | Moderada | Boa alternativa |

| \*\*llama.cpp\*\* | ⭐⭐☆☆☆ | Qualquer modelo GGUF | Avançado | Complexa | Para usuários avançados |

| \*\*Text Generation WebUI\*\* | ⭐⭐☆☆☆ | Qualquer modelo | Moderado | Moderada | Para personalização extrema |

\*\*Conclusão\*\*: Para seu caso específico (não-programador querendo um sistema simples), \*\*Ollama é a melhor escolha\*\* por sua facilidade de uso, boa documentação e excelente integração com n8n.

#### \*\*2.2.2 Comparativo de Ferramentas de Automação\*\*

| Ferramenta | Facilidade de Uso | Recursos de IA | Preço | Execução Local | Recomendação |

|------------|------------------|---------------|-------|---------------|-------------|

| \*\*n8n\*\* | ⭐⭐⭐⭐⭐ | Excelente | Gratuito | Sim | \*\*MELHOR OPÇÃO\*\* |

| \*\*Flowise\*\* | ⭐⭐⭐⭐☆ | Bom | Gratuito | Sim | Boa alternativa |

| \*\*Langflow\*\* | ⭐⭐⭐☆☆ | Bom | Gratuito | Sim | Para usuários técnicos |

| \*\*AutoGen\*\* | ⭐⭐☆☆☆ | Avançado | Gratuito | Sim | Para programadores |

| \*\*CrewAI\*\* | ⭐⭐☆☆☆ | Avançado | Gratuito | Sim | Para sistemas multi-agente |

\*\*Conclusão\*\*: \*\*n8n é claramente a melhor opção\*\* para você, dado seu perfil de não-programador. Sua interface visual, suporte a agentes de IA integrados e capacidade de execução local sem custo o tornam perfeito para seu projeto.

#### \*\*2.2.3 Comparativo de Bancos Vetoriais\*\*

| Banco | Facilidade de Uso | Desempenho | Integração com LangChain | Recomendação |

|-------|------------------|-----------|------------------------|-------------|

| \*\*ChromaDB\*\* | ⭐⭐⭐⭐⭐ | Moderado | Excelente | \*\*MELHOR OPÇÃO\*\* para iniciantes |

| \*\*Qdrant\*\* | ⭐⭐⭐☆☆ | Alto | Excelente | Para sistemas maiores |

| \*\*FAISS\*\* | ⭐⭐☆☆☆ | Muito Alto | Moderada | Para usuários avançados |

| \*\*Milvus\*\* | ⭐⭐☆☆☆ | Alto | Excelente | Para ambientes enterprise |

\*\*Conclusão\*\*: \*\*ChromaDB é a melhor escolha\*\* para você começar, devido à sua simplicidade de configuração e excelente integração com LangChain.

### \*\*2.3 Requisitos Mínimos de Hardware\*\*

#### \*\*2.3.1 Para Modelos Básicos (7B parâmetros)\*\*

| Componente | Mínimo | Recomendado | Observações |

|------------|--------|-------------|-------------|

| \*\*RAM\*\* | 8GB | 16GB | Modelos quantizados exigem menos memória |

| \*\*CPU\*\* | x86-64 com AVX2 | i5/Ryzen 5 ou superior | AVX2 essencial para desempenho |

| \*\*GPU\*\* | Não necessário | NVIDIA com 4GB VRAM | Acelera significativamente |

| \*\*Armazenamento\*\* | 10GB livre | 20GB livre | Modelos quantizados ocupam ~4-6GB |

| \*\*Sistema\*\* | Windows 10/11, macOS, Linux | Linux recomendado | Linux geralmente melhor desempenho |

\*\*Modelos recomendados para hardware modesto\*\*:

- \*\*Mistral 7B Instruct (Q4\_K\_M)\*\*: 4.4GB no disco, bom equilíbrio qualidade/tamanho

- \*\*CodeLlama 7B Instruct (Q4\_K\_M)\*\*: 4.7GB, especializado em programação

- \*\*Phi-2\*\*: 2.2GB, excelente para tarefas de programação em hardware muito limitado

#### \*\*2.3.2 Otimizações para Hardware Limitado\*\*

1. \*\*Quantização\*\*:

- Use modelos quantizados em Q4\_K\_M (4-bit) para reduzir o uso de memória

- Exemplo: Mistral 7B normalmente ocupa 14GB, mas quantizado ocupa apenas 4.4GB

2. \*\*Offloading parcial\*\*:

- Com `llama.cpp`, você pode descarregar algumas camadas para a CPU

- Configuração recomendada para 8GB de RAM: `-ngl 20` (20 camadas na GPU)

3. \*\*Swap file\*\*:

- Crie um arquivo de swap de 4-8GB para evitar crashes com memória insuficiente

- No Windows: Configurações do sistema > Avançado > Desempenho > Configurações > Avançado

- No Linux: `sudo fallocate -l 8G /swapfile && sudo chmod 600 /swapfile && sudo mkswap /swapfile && sudo swapon /swapfile`

4. \*\*Configuração específica do Ollama\*\*:

```bash

# Para limitar uso de memória no Ollama

export OLLAMA\_NUM\_PARALLEL=2 # Limita threads paralelas

export OLLAMA\_MAX\_LOADED\_MODELS=1 # Só carrega um modelo por vez

```

#### \*\*2.3.3 Como Verificar se Seu Hardware é Adequado\*\*

1. \*\*Verifique suporte a AVX2\*\*:

- Windows: Baixe o CPU-Z e verifique se AVX2 está listado

- Mac: `sysctl -a | grep machdep.cpu.features`

- Linux: `grep avx2 /proc/cpuinfo`

2. \*\*Teste de desempenho rápido\*\*:

```bash

# Instale o Ollama e execute este teste

ollama run llama3 "Qual é a capital da França?" --verbose

```

- Se responder em < 30 segundos, seu hardware é adequado para modelos 7B

- Se levar > 60 segundos, considere modelos menores como Phi-2

3. \*\*Monitoramento de recursos\*\*:

- Windows: Gerenciador de Tarefas

- Mac: Monitor de Atividade

- Linux: `htop`

- Observe uso de RAM e CPU durante a execução do modelo

---

## \*\*3. ARQUITETURA COMPLETA DO AGENTE\*\*

### \*\*3.1 Diagrama Arquitetural Completo\*\*

```

┌───────────────────────────────────────────────────────────────────────────────────────────────────────────────┐

│ INTERFACE DO USUÁRIO │

│ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ │

│ │ Web UI │ │ Aplicativo│ │ Terminal │ │ Telegram│ │ WhatsApp │ │

│ │ (Gradio/ │ │ Desktop │ │ │ │ │ │ │ │

│ │ Streamlit) │ │ (Electron) │ │ │ │ │ │ │ │

│ └──────┬──────┘ └──────┬──────┘ └──────┬──────┘ └──────┬──────┘ └──────┬──────┘ │

│ │ │ │ │ │ │

└─────────┼───────────────────┼───────────────────┼───────────────────┼───────────────────┼───────────────────┘

▼ ▼ ▼ ▼ ▼

┌───────────────────────────────────────────────────────────────────────────────────────────────────────────────┐

│ CAMADA DE INTEGRAÇÃO │

│ ┌─────────────────────────────────────────────────────────────────────────────────────────────────────────┐ │

│ │ n8n WORKFLOW │ │

│ │ ┌───────────┐ ┌────────────┐ ┌────────────┐ ┌────────────┐ ┌────────────┐ ┌───────────┐ │ │

│ │ │ Trigger │───▶│ Processar │───▶│ RAG │───▶│ Agente │───▶│ Ferramentas│───▶│ Resposta │ │ │

│ │ │(Webhook, │ │ Input │ │(ChromaDB) │ │ IA │ │(Google Sheets│ │ │ │ │

│ │ │ Telegram) │ └────────────┘ └────────────┘ └────────────┘ │, Calculadora)│ └───────────┘ │ │

│ │ └───────────┘ └────────────┘ │ │

│ └─────────────────────────────────────────────────────────────────────────────────────────────────────────┘ │

└───────────────────────────────────────────────────────────────────────────────────────────────────────────────┘

▲ ▲ ▲ ▲ ▲

│ │ │ │ │

┌─────────┼───────────────────┼───────────────────┼───────────────────┼───────────────────┼───────────────────┐

│ ┌──────▼──────┐ ┌──────▼──────┐ ┌──────▼──────┐ ┌──────▼──────┐ ┌──────▼──────┐ │

│ │ Ollama │ │ ChromaDB │ │ ZEP │ │ Google Sheets│ │ Outras │ │

│ │ (LLMs Locais)│ │ (Banco │ │ (Memória de │ │ │ │ Ferramentas │ │

│ │ │ │ Vetorial) │ │ Longo Prazo)│ │ │ │ │ │

│ └─────────────┘ └─────────────┘ └─────────────┘ └─────────────┘ └─────────────┘ │

└───────────────────────────────────────────────────────────────────────────────────────────────────────────────┘

```

### \*\*3.2 Fluxo de Dados: Input → IA → Ação → Output\*\*

\*\*Passo a Passo Detalhado do Fluxo de Trabalho:\*\*

1. \*\*Input do Usuário\*\*:

- Você digita uma mensagem no chat: "Crie um aplicativo React que mostre a previsão do tempo para São Paulo"

- A mensagem é capturada pelo trigger do n8n (Webhook, interface web ou Telegram)

2. \*\*Processamento Inicial\*\*:

- O n8n recebe a mensagem e a processa através de um nó "Function"

- Este nó prepara o contexto, adiciona informações do usuário e formata a entrada

3. \*\*RAG (Retrieval-Augmented Generation)\*\*:

- O sistema consulta o ChromaDB para encontrar informações relevantes:

- Documentação do React em seu banco de conhecimento

- Exemplos de integração com API de previsão do tempo

- Seu histórico de projetos anteriores

- Os resultados mais relevantes são recuperados e adicionados ao prompt

4. \*\*Processamento pelo Agente IA\*\*:

- O nó "AI Agent" do n8n recebe o prompt completo

- Ele se comunica com o Ollama local para obter uma resposta

- O modelo CodeLlama 7B Instruct processa a solicitação

5. \*\*Uso de Ferramentas\*\*:

- O agente determina que precisa de informações de previsão do tempo

- Ele usa a ferramenta "API de Previsão do Tempo" configurada no n8n

- Obtém dados reais de uma API pública (OpenWeatherMap)

6. \*\*Geração da Resposta\*\*:

- O agente combina todas as informações

- Gera código React completo com integração à API

- Formata a resposta de forma clara e útil

7. \*\*Output para o Usuário\*\*:

- A resposta é enviada de volta à interface do usuário

- Código é formatado com syntax highlighting

- Explicação passo a passo é fornecida

8. \*\*Armazenamento na Memória\*\*:

- A interação é salva no ZEP para memória de longo prazo

- Embeddings são gerados e armazenados no ChromaDB

- Histórico é atualizado para futuras referências

### \*\*3.3 Passo a Passo para Montagem do Sistema\*\*

#### \*\*3.3.1 Preparação do Ambiente\*\*

\*\*Passo 1: Instalar o Ollama\*\*

1. Acesse https://ollama.com/download

2. Baixe e instale a versão para seu sistema operacional

3. Abra o terminal e verifique a instalação:

```bash

ollama --version

```

4. Teste com o modelo LLaMA 3:

```bash

ollama run llama3

```

(Isso irá baixar automaticamente o modelo)

\*\*Passo 2: Instalar o n8n\*\*

1. Certifique-se de ter Node.js instalado (versão 16+)

- Baixe em https://nodejs.org

2. Instale o n8n globalmente:

```bash

npm install n8n -g

```

3. Inicie o n8n:

```bash

n8n start

```

4. Acesse a interface em http://localhost:5678

\*\*Passo 3: Instalar o ChromaDB (para RAG)\*\*

1. Instale o ChromaDB:

```bash

pip install chromadb

```

2. Teste a instalação:

```python

import chromadb

client = chromadb.Client()

print("ChromaDB está funcionando!")

```

#### \*\*3.3.2 Configuração do Sistema de Agentes\*\*

\*\*Passo 4: Configurar o Modelo de IA no n8n\*\*

1. No n8n, clique em "Credentials" no menu esquerdo

2. Clique em "+ Create Credentials"

3. Selecione "Ollama"

4. Preencha:

- Name: "Local Ollama"

- Base URL: http://localhost:11434

- Model: mistral (ou o modelo que você instalou)

5. Salve as credenciais

\*\*Passo 5: Criar o Fluxo Básico do Agente\*\*

1. No n8n, clique em "+ Create Workflow"

2. Adicione um nó "Webhook" como trigger

3. Configure o Webhook:

- Method: POST

- Path: /chat

4. Adicione um nó "AI Agent" após o Webhook

5. Configure o AI Agent:

- Credentials: Selecione "Local Ollama"

- System Message: "Você é um assistente de programação especializado em React. Responda de forma clara e detalhada, fornecendo exemplos de código quando relevante."

- Model: mistral

6. Conecte o AI Agent de volta ao Webhook como resposta

\*\*Passo 6: Testar o Agente Básico\*\*

1. Salve o workflow e ative-o

2. Use uma ferramenta como Postman para enviar uma requisição:

```json

{

"message": "Como criar um componente de botão em React?"

}

```

3. Envie para http://localhost:5678/webhook/chat

4. Você deve receber uma resposta com código React para um botão

#### \*\*3.3.3 Adicionando RAG (Conhecimento Personalizado)\*\*

\*\*Passo 7: Configurar o RAG com ChromaDB\*\*

1. Crie um novo workflow no n8n

2. Adicione um nó "Function" após o Webhook

3. No código da função, adicione:

```javascript

// Configuração do ChromaDB

const { ChromaClient } = require('chromadb');

const client = new ChromaClient();

const collection = await client.createCollection({ name: "react\_docs" });

// Adicionar documentos ao banco (faça isso uma vez)

await collection.add({

ids: ["doc1", "doc2"],

metadatas: [{ source: "react\_docs" }, { source: "react\_docs" }],

documents: [

"Componentes em React são funções JavaScript que retornam elementos React...",

"O hook useState é usado para adicionar estado a componentes funcionais..."

]

});

// Buscar documentos relevantes

const results = await collection.query({

queryTexts: [items[0].json.message],

nResults: 2

});

// Adicionar resultados ao contexto

items[0].json.context = results.documents[0].join("\n");

return items;

```

\*\*Passo 8: Integrar RAG com o Agente\*\*

1. No workflow principal, adicione o nó RAG antes do AI Agent

2. Modifique a System Message do AI Agent para incluir:

```

Use o seguinte contexto para responder:

{context}

Se o contexto não for relevante, ignore-o e responda com base no seu conhecimento.

```

3. Teste com perguntas específicas sobre React

#### \*\*3.3.4 Adicionando Memória de Longo Prazo com ZEP\*\*

\*\*Passo 9: Configurar o ZEP\*\*

1. Baixe o ZEP em https://getzep.com

2. Siga as instruções de instalação para seu sistema

3. Inicie o servidor ZEP:

```bash

zep start

```

\*\*Passo 10: Integrar ZEP com o n8n\*\*

1. Crie um nó "Function" após o AI Agent

2. Adicione este código:

```javascript

const axios = require('axios');

// Salvar conversa no ZEP

await axios.post('http://localhost:8000/sessions/{sessionId}/messages', {

role: 'user',

content: items[0].json.message

});

await axios.post('http://localhost:8000/sessions/{sessionId}/messages', {

role: 'assistant',

content: items[0].json.response

});

return items;

```

\*\*Passo 11: Modificar o Agente para Usar Memória\*\*

1. Atualize a System Message para:

```

Você está conversando com {userName}. Use o histórico de conversa abaixo para contextualizar sua resposta:

{history}

Pergunta atual: {message}

```

2. Adicione um nó antes do AI Agent para buscar o histórico no ZEP

#### \*\*3.3.5 Adicionando Interface de Chat Amigável\*\*

\*\*Passo 12: Criar Interface Web com Gradio\*\*

1. Instale o Gradio:

```bash

pip install gradio

```

2. Crie um arquivo `app.py`:

```python

import gradio as gr

import requests

def chat(message):

response = requests.post(

"http://localhost:5678/webhook/chat",

json={"message": message}

)

return response.json()["response"]

interface = gr.Interface(

fn=chat,

inputs=gr.Textbox(placeholder="Digite sua pergunta..."),

outputs="text",

title="Meu Agente de IA Local",

description="Um assistente de programação local e gratuito"

)

if \_\_name\_\_ == "\_\_main\_\_":

interface.launch()

```

3. Execute com `python app.py`

4. Acesse http://localhost:7860

---

## \*\*4. IAS LOCAIS (LLMS)\*\*

### \*\*4.1 Modelos Suportados Localmente\*\*

#### \*\*4.1.1 LLaMA 3 (Meta)\*\*

- \*\*Tamanhos disponíveis\*\*: 8B, 70B

- \*\*Características\*\*:

- Treinado com 15T tokens

- Suporte a 128K tokens de contexto

- Excelente para tarefas gerais e programação

- Versão mais recente da família LLaMA

- \*\*Recomendações de quantização\*\*:

- Q4\_K\_M: Melhor equilíbrio qualidade/tamanho (5.1GB para 8B)

- Q5\_K\_M: Qualidade superior, tamanho maior (6.2GB para 8B)

- \*\*Como instalar no Ollama\*\*:

```bash

ollama pull llama3

```

#### \*\*4.1.2 Mistral (Mistral AI)\*\*

- \*\*Tamanhos disponíveis\*\*: 7B

- \*\*Características\*\*:

- Modelo eficiente para hardware modesto

- Bom desempenho em tarefas de programação

- Suporte a 32K tokens de contexto

- Licença mais permissiva que LLaMA

- \*\*Recomendações de quantização\*\*:

- Q4\_K\_M: 4.4GB, ideal para sistemas com 8-16GB de RAM

- Q5\_K\_M: 5.1GB, melhor qualidade

- \*\*Como instalar no Ollama\*\*:

```bash

ollama pull mistral

```

#### \*\*4.1.3 CodeLlama (Meta)\*\*

- \*\*Tamanhos disponíveis\*\*: 7B, 13B, 34B

- \*\*Características\*\*:

- Especializado em programação

- Treinado com código de 6 linguagens

- Suporte a 16K tokens de contexto

- Instrução especializada para tarefas de programação

- \*\*Recomendações de quantização\*\*:

- Q4\_K\_M: 4.7GB (7B), ideal para desenvolvedores

- Q5\_K\_M: 5.5GB (7B), melhor qualidade de código

- \*\*Como instalar no Ollama\*\*:

```bash

ollama pull codellama

```

#### \*\*4.1.4 Phi-2 (Microsoft)\*\*

- \*\*Tamanhos disponíveis\*\*: 2.7B

- \*\*Características\*\*:

- Modelo extremamente leve

- Excelente para hardware muito limitado

- Boa capacidade de raciocínio

- Suporte a 2K tokens de contexto

- \*\*Recomendações de quantização\*\*:

- Q4\_K\_M: 1.7GB, funciona até em Raspberry Pi

- Q5\_K\_M: 2.0GB, melhor qualidade

- \*\*Como instalar no Ollama\*\*:

```bash

ollama pull phi

```

#### \*\*4.1.5 Gemma (Google)\*\*

- \*\*Tamanhos disponíveis\*\*: 2B, 7B

- \*\*Características\*\*:

- Modelo leve da Google

- Boa performance em tarefas gerais

- Suporte a 8K tokens de contexto

- Licença mais permissiva que LLaMA

- \*\*Recomendações de quantização\*\*:

- Q4\_K\_M: 1.6GB (2B), 4.8GB (7B)

- Q5\_K\_M: 1.9GB (2B), 5.7GB (7B)

- \*\*Como instalar no Ollama\*\*:

```bash

ollama pull gemma

```

### \*\*4.2 Instalação e Configuração de Cada Modelo\*\*

#### \*\*4.2.1 Instalação com Ollama (Recomendado para Iniciantes)\*\*

\*\*Passo a Passo:\*\*

1. Instale o Ollama conforme instruções na seção 3.3.1

2. Para instalar qualquer modelo, use:

```bash

ollama pull <nome-do-modelo>

```

Exemplos:

```bash

ollama pull llama3

ollama pull mistral

ollama pull codellama

```

3. Para verificar modelos instalados:

```bash

ollama list

```

4. Para executar um modelo:

```bash

ollama run mistral

```

\*\*Configurações Avançadas no Ollama:\*\*

- Para modificar parâmetros de geração, crie um arquivo Modelfile:

```dockerfile

FROM mistral

PARAMETER temperature 0.7

PARAMETER num\_ctx 4096

SYSTEM """

Você é um assistente de programação útil e detalhista.

Sempre forneça exemplos de código completos quando relevante.

"""

```

- Construa o modelo personalizado:

```bash

ollama create meu-mistral -f Modelfile

```

#### \*\*4.2.2 Instalação com LM Studio (Interface Gráfica)\*\*

\*\*Passo a Passo:\*\*

1. Baixe e instale LM Studio em https://lmstudio.ai

2. Abra o aplicativo e clique na aba "Marketplace"

3. Pesquise pelo modelo desejado (ex: "Mistral 7B")

4. Clique em "Download" para baixar o modelo

5. Após o download, vá para a aba "Local Server"

6. Clique em "Start Server" para ativar a API local

7. A API estará disponível em http://localhost:1234

\*\*Vantagens do LM Studio:\*\*

- Interface gráfica intuitiva

- Visualização de uso de recursos em tempo real

- Teste rápido de diferentes modelos

- Suporte a quantização durante o download

#### \*\*4.2.3 Instalação com llama.cpp (Para Usuários Avançados)\*\*

\*\*Passo a Passo:\*\*

1. Baixe o llama.cpp em https://github.com/ggerganov/llama.cpp

2. Siga as instruções de compilação para seu sistema

3. Baixe um modelo quantizado em formato GGUF (ex: mistral-7b-instruct-v0.2.Q4\_K\_M.gguf)

4. Execute o modelo:

```bash

./main -m models/mistral-7b-instruct-v0.2.Q4\_K\_M.gguf -p "Qual é a capital da França?" -n 128

```

5. Para API local:

```bash

./server -m models/mistral-7b-instruct-v0.2.Q4\_K\_M.gguf -c 4096 --port 8080

```

\*\*Configurações Importantes:\*\*

- `-c 4096`: Define o contexto para 4096 tokens

- `-b 512`: Define o tamanho do batch

- `-t 8`: Usa 8 threads da CPU

- `--port 8080`: Define a porta da API

### \*\*4.3 Quantização e Desempenho em Máquinas Modestas\*\*

#### \*\*4.3.1 O que é Quantização?\*\*

Quantização é o processo de reduzir a precisão numérica dos pesos do modelo de IA, convertendo de float32 (32 bits) para formatos de menor precisão como int4 (4 bits). Isso reduz significativamente o tamanho do modelo e os requisitos de memória, com perda mínima de qualidade.

\*\*Níveis Comuns de Quantização:\*\*

- \*\*Q2\_K\*\*: 2 bits por peso - qualidade muito baixa, uso mínimo de memória

- \*\*Q3\_K\_M\*\*: 3 bits por peso - qualidade baixa, bom para hardware muito limitado

- \*\*Q4\_0\*\*: 4 bits por peso - qualidade média, padrão para muitos modelos

- \*\*Q4\_K\_M\*\*: 4 bits por peso com otimizações - melhor equilíbrio qualidade/tamanho

- \*\*Q5\_K\_M\*\*: 5 bits por peso - qualidade alta, tamanho maior

- \*\*Q6\_K\*\*: 6 bits por peso - qualidade próxima do original, tamanho grande

- \*\*Q8\_0\*\*: 8 bits por peso - quase sem perda de qualidade, tamanho muito grande

#### \*\*4.3.2 Impacto na Qualidade e Desempenho\*\*

\*\*Comparativo para Mistral 7B:\*\*

| Quantização | Tamanho no Disco | Qualidade Relativa | RAM Necessária | Velocidade Relativa |

|-------------|------------------|-------------------|----------------|---------------------|

| Q2\_K | 3.1GB | ★☆☆☆☆ | 4GB | ★★★★☆ |

| Q3\_K\_M | 3.7GB | ★★☆☆☆ | 5GB | ★★★★☆ |

| \*\*Q4\_K\_M\*\* | \*\*4.4GB\*\* | \*\*★★★★☆\*\* | \*\*6GB\*\* | \*\*★★★★☆\*\* |

| Q5\_K\_M | 5.1GB | ★★★★☆ | 7GB | ★★★☆☆ |

| Q6\_K | 5.8GB | ★★★★★ | 8GB | ★★☆☆☆ |

| FP16 | 14GB | ★★★★★ | 16GB | ★☆☆☆☆ |

\*\*Conclusão\*\*: Para a maioria dos usuários, \*\*Q4\_K\_M oferece o melhor equilíbrio\*\* entre qualidade, tamanho e velocidade.

#### \*\*4.3.3 Como Escolher a Quantização Certa para Seu Hardware\*\*

\*\*Guia de Decisão:\*\*

1. \*\*Se você tem 8GB de RAM ou menos\*\*:

- Use Q4\_K\_M para modelos de 7B

- Evite modelos maiores que 7B

- Considere Phi-2 se tiver dificuldades

2. \*\*Se você tem 16GB de RAM\*\*:

- Q5\_K\_M é uma boa opção para modelos de 7B

- Você pode experimentar modelos de 13B com Q4\_K\_M

- Mistral 7B ou CodeLlama 7B são excelentes escolhas

3. \*\*Se você tem 32GB de RAM ou mais\*\*:

- Q6\_K oferece qualidade próxima do original

- Você pode usar modelos de 13B-34B com boa qualidade

- Considere LLaMA 3 8B para melhor desempenho geral

\*\*Dica Prática\*\*:

- Comece com Q4\_K\_M para qualquer modelo

- Se notar respostas de baixa qualidade, experimente Q5\_K\_M

- Se tiver problemas de memória, experimente Q3\_K\_M

#### \*\*4.3.4 Como Verificar o Desempenho do Modelo\*\*

\*\*Métricas Importantes:\*\*

1. \*\*Tokens por segundo (TPS)\*\*:

- Indica velocidade de geração

- Acima de 10 TPS é bom para interação em tempo real

- Abaixo de 5 TPS pode ser lento para chatbots

2. \*\*Uso de memória\*\*:

- Verifique no Gerenciador de Tarefas/Monitor de Atividade

- Se ultrapassar 90% da RAM, reduza a quantização

3. \*\*Qualidade das respostas\*\*:

- Teste com tarefas específicas (ex: geração de código)

- Compare com respostas de modelos não quantizados

\*\*Como Medir TPS no Ollama:\*\*

```bash

ollama run llama3 "Qual é a capital da França?" --verbose

```

Procure pela linha: `total duration: ..., tokens per second: ...`

### \*\*4.4 Comparativo Detalhado entre LLMs Locais\*\*

#### \*\*4.4.1 Comparativo Técnico Detalhado\*\*

| Modelo | Tamanho | Contexto | Treinamento | Programação | Idiomas | Qualidade Geral | Recomendação |

|--------|---------|----------|-------------|-------------|---------|-----------------|--------------|

| \*\*LLaMA 3 8B\*\* | 8B | 8K | 15T tokens | ★★★★☆ | Múltiplos | ★★★★★ | \*\*MELHOR GERAL\*\* |

| \*\*Mistral 7B\*\* | 7B | 32K | Não divulgado | ★★★☆☆ | Múltiplos | ★★★★☆ | Melhor para hardware modesto |

| \*\*CodeLlama 7B\*\* | 7B | 16K | Código | ★★★★★ | Múltiplos | ★★★★☆ | \*\*MELHOR PARA PROGRAMAÇÃO\*\* |

| \*\*Phi-2\*\* | 2.7B | 2K | 3.8T tokens | ★★★☆☆ | Inglês | ★★★☆☆ | Melhor para hardware limitado |

| \*\*Gemma 7B\*\* | 7B | 8K | 6T tokens | ★★★☆☆ | Múltiplos | ★★★☆☆ | Boa alternativa com licença permissiva |

| \*\*Solar 10.7B\*\* | 10.7B | 32K | 2T tokens | ★★★★☆ | Inglês | ★★★★☆ | Bom equilíbrio tamanho/desempenho |

#### \*\*4.4.2 Teste Prático de Programação\*\*

\*\*Teste\*\*: Gerar um componente React para lista de tarefas com localStorage

\*\*LLaMA 3 8B\*\*:

```javascript

import { useState, useEffect } from 'react';

function TodoList() {

const [todos, setTodos] = useState([]);

const [input, setInput] = useState('');

useEffect(() => {

const savedTodos = localStorage.getItem('todos');

if (savedTodos) {

setTodos(JSON.parse(savedTodos));

}

}, []);

useEffect(() => {

localStorage.setItem('todos', JSON.stringify(todos));

}, [todos]);

const addTodo = () => {

if (input.trim() !== '') {

setTodos([...todos, { id: Date.now(), text: input, completed: false }]);

setInput('');

}

};

const toggleComplete = (id) => {

setTodos(todos.map(todo =>

todo.id === id ? { ...todo, completed: !todo.completed } : todo

));

};

return (

<div>

<input

type="text"

value={input}

onChange={e => setInput(e.target.value)}

onKeyPress={e => e.key === 'Enter' && addTodo()}

/>

<button onClick={addTodo}>Adicionar</button>

<ul>

{todos.map(todo => (

<li

key={todo.id}

style={{ textDecoration: todo.completed ? 'line-through' : 'none' }}

onClick={() => toggleComplete(todo.id)}

>

{todo.text}

</li>

))}

</ul>

</div>

);

}

export default TodoList;

```

\*\*Pontuação\*\*: 9.5/10 - Código funcional, bem estruturado, com comentários implícitos na lógica

\*\*Mistral 7B\*\*:

```javascript

import React, { useState, useEffect } from 'react';

const TodoList = () => {

const [todos, setTodos] = useState([]);

const [newTodo, setNewTodo] = useState('');

// Carregar tarefas do localStorage ao iniciar

useEffect(() => {

const savedTodos = JSON.parse(localStorage.getItem('todos')) || [];

setTodos(savedTodos);

}, []);

// Salvar tarefas no localStorage quando mudarem

useEffect(() => {

localStorage.setItem('todos', JSON.stringify(todos));

}, [todos]);

const addTodo = () => {

if (newTodo.trim() !== '') {

setTodos([...todos, { id: Date.now(), text: newTodo, completed: false }]);

setNewTodo('');

}

};

const toggleComplete = (id) => {

setTodos(todos.map(todo =>

todo.id === id ? { ...todo, completed: !todo.completed } : todo

));

};

return (

<div className="todo-list">

<input

type="text"

value={newTodo}

onChange={e => setNewTodo(e.target.value)}

placeholder="Nova tarefa..."

/>

<button onClick={addTodo}>Adicionar</button>

<ul>

{todos.map(todo => (

<li

key={todo.id}

className={todo.completed ? 'completed' : ''}

onClick={() => toggleComplete(todo.id)}

>

{todo.text}

</li>

))}

</ul>

</div>

);

};

export default TodoList;

```

\*\*Pontuação\*\*: 8.5/10 - Código bom, mas faltam alguns detalhes de UX

\*\*CodeLlama 7B\*\*:

```python

# Note: CodeLlama é treinado principalmente para Python, mas pode gerar JavaScript

def generate\_todo\_list\_code():

"""

Gera código React para uma lista de tarefas com localStorage

"""

code = """

import React, { useState, useEffect } from 'react';

function TodoApp() {

const [tasks, setTasks] = useState([]);

const [inputValue, setInputValue] = useState('');

// Carrega tarefas do localStorage ao montar o componente

useEffect(() => {

const storedTasks = localStorage.getItem('tasks');

if (storedTasks) {

setTasks(JSON.parse(storedTasks));

}

}, []);

// Salva tarefas no localStorage quando tasks mudar

useEffect(() => {

localStorage.setItem('tasks', JSON.stringify(tasks));

}, [tasks]);

const addTask = () => {

if (inputValue.trim() !== '') {

setTasks([...tasks, {

id: Date.now(),

text: inputValue,

completed: false

}]);

setInputValue('');

}

};

const toggleTask = (id) => {

setTasks(tasks.map(task =>

task.id === id ? {...task, completed: !task.completed} : task

));

};

return (

<div className="todo-container">

<h1>Lista de Tarefas</h1>

<div className="input-group">

<input

type="text"

value={inputValue}

onChange={(e) => setInputValue(e.target.value)}

onKeyPress={(e) => e.key === 'Enter' && addTask()}

placeholder="Nova tarefa..."

/>

<button onClick={addTask}>Adicionar</button>

</div>

<ul className="task-list">

{tasks.map(task => (

<li

key={task.id}

className={task.completed ? 'completed' : ''}

onClick={() => toggleTask(task.id)}

>

{task.text}

</li>

))}

</ul>

</div>

);

}

export default TodoApp;

"""

return code

```

\*\*Pontuação\*\*: 7.5/10 - Código funcional, mas com comentários excessivos e estrutura um pouco confusa

#### \*\*4.4.3 Recomendações Finais por Caso de Uso\*\*

\*\*Para Programação Geral\*\*:

- \*\*Primeira escolha\*\*: LLaMA 3 8B (Q4\_K\_M)

- \*\*Segunda escolha\*\*: CodeLlama 7B (Q5\_K\_M)

- \*\*Para hardware limitado\*\*: Mistral 7B (Q4\_K\_M)

\*\*Para Programação Específica\*\*:

- \*\*JavaScript/React\*\*: LLaMA 3 8B ou Mistral 7B

- \*\*Python\*\*: CodeLlama 7B ou LLaMA 3 8B

- \*\*TypeScript\*\*: LLaMA 3 8B com contexto específico

- \*\*Hardware Muito Limitado\*\*: Phi-2 com prompts muito específicos

\*\*Para Tarefas Não-Programação\*\*:

- \*\*Primeira escolha\*\*: LLaMA 3 8B

- \*\*Segunda escolha\*\*: Mistral 7B

- \*\*Para hardware limitado\*\*: Gemma 2B

---

## \*\*5. FRAMEWORKS E ORQUESTRAÇÃO\*\*

### \*\*5.1 LangChain: Guia Completo\*\*

#### \*\*5.1.1 O que é LangChain?\*\*

LangChain é um framework open-source que permite construir aplicações com modelos de linguagem grande (LLMs). Ele facilita a criação de sistemas complexos com RAG, agentes, memória e orquestração de ferramentas.

\*\*Principais Componentes\*\*:

- \*\*Models\*\*: Interface com diferentes LLMs (OpenAI, Ollama, etc.)

- \*\*Prompts\*\*: Gerenciamento de templates de prompt

- \*\*Chains\*\*: Sequências de operações que processam entradas

- \*\*Agents\*\*: Sistemas que decidem quais ações tomar

- \*\*Memory\*\*: Mecanismos para manter contexto entre interações

- \*\*Indexes\*\*: Ferramentas para trabalhar com dados não estruturados

- \*\*Document Loaders\*\*: Ferramentas para carregar diferentes tipos de documentos

#### \*\*5.1.2 Instalação e Configuração Básica\*\*

\*\*Instalação\*\*:

```bash

pip install langchain langchain-community langchain-core

```

\*\*Configuração com Ollama\*\*:

```python

from langchain\_community.llms import Ollama

# Configurar o modelo local

llm = Ollama(

model="mistral",

base\_url="http://localhost:11434",

temperature=0.7

)

# Testar o modelo

response = llm.invoke("Explique o que é programação assíncrona em JavaScript")

print(response)

```

#### \*\*5.1.3 Componentes Essenciais do LangChain\*\*

\*\*1. Chains (Cadeias)\*\*

Chains são sequências de operações que processam entradas e produzem saídas.

\*\*Exemplo de LLMChain simples\*\*:

```python

from langchain\_core.prompts import PromptTemplate

from langchain.chains import LLMChain

# Criar template de prompt

template = """Você é um especialista em JavaScript.

Pergunta: {pergunta}

Resposta:"""

prompt = PromptTemplate(template=template, input\_variables=["pergunta"])

# Criar chain

chain = LLMChain(llm=llm, prompt=prompt)

# Executar

response = chain.run(pergunta="O que é uma promise em JavaScript?")

print(response)

```

\*\*2. Agents (Agentes)\*\*

Agentes são sistemas que decidem quais ações tomar com base em observações.

\*\*Exemplo de Agente com Ferramentas\*\*:

```python

from langchain.agents import initialize\_agent, Tool

from langchain.agents import AgentType

# Definir ferramentas

def search\_web(query):

"""Simula uma busca na web"""

return f"Resultados para '{query}': [1] Artigo sobre {query}, [2] Tutorial de {query}"

def get\_weather(city):

"""Simula obtenção do clima"""

return f"O clima em {city} é ensolarado, 25°C"

tools = [

Tool(

name="Busca na Web",

func=search\_web,

description="Útil para quando você precisa pesquisar algo na internet"

),

Tool(

name="Previsão do Tempo",

func=get\_weather,

description="Útil para obter a previsão do tempo em uma cidade"

)

]

# Inicializar agente

agent = initialize\_agent(

tools,

llm,

agent=AgentType.ZERO\_SHOT\_REACT\_DESCRIPTION,

verbose=True

)

# Executar agente

agent.run("Qual é a capital da França e qual o clima lá hoje?")

```

\*\*3. Memory (Memória)\*\*

Memória permite que o agente mantenha contexto entre interações.

\*\*Exemplo com ConversationBufferMemory\*\*:

```python

from langchain.memory import ConversationBufferMemory

# Configurar memória

memory = ConversationBufferMemory(memory\_key="chat\_history")

# Criar chain com memória

conversation = LLMChain(

llm=llm,

prompt=prompt,

memory=memory

)

# Primeira interação

conversation.run(input="Oi, meu nome é João")

# Segunda interação

response = conversation.run(input="Qual o meu nome?")

print(response) # Deve responder "João"

```

\*\*4. Document Loaders\*\*

Carregam diferentes tipos de documentos para processamento.

\*\*Exemplo com PDF Loader\*\*:

```python

from langchain\_community.document\_loaders import PyPDFLoader

# Carregar PDF

loader = PyPDFLoader("manual\_react.pdf")

pages = loader.load()

# Processar conteúdo

print(f"Documento carregado com {len(pages)} páginas")

print(pages[0].page\_content[:500]) # Primeiros 500 caracteres

```

#### \*\*5.1.4 Integração com n8n\*\*

LangChain pode ser integrado ao n8n para criar workflows mais poderosos:

\*\*Passo 1: Criar um serviço LangChain\*\*

```python

# langchain\_service.py

from flask import Flask, request, jsonify

from langchain.chains import LLMChain

from langchain\_core.prompts import PromptTemplate

from langchain\_community.llms import Ollama

app = Flask(\_\_name\_\_)

llm = Ollama(model="mistral")

@app.route('/generate', methods=['POST'])

def generate():

data = request.json

prompt = PromptTemplate(

template=data['template'],

input\_variables=data['input\_variables']

)

chain = LLMChain(llm=llm, prompt=prompt)

response = chain.run(\*\*data['inputs'])

return jsonify({"response": response})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(port=5000)

```

\*\*Passo 2: Executar o serviço\*\*

```bash

python langchain\_service.py

```

\*\*Passo 3: Integrar com n8n\*\*

1. No n8n, adicione um nó "HTTP Request"

2. Configure:

- URL: http://localhost:5000/generate

- Method: POST

- Body:

```json

{

"template": "Explique {topic} de forma simples",

"input\_variables": ["topic"],

"inputs": {

"topic": "{{ $json['topic'] }}"

}

}

```

### \*\*5.2 RAG (Retrieval-Augmented Generation): Implementação Prática\*\*

#### \*\*5.2.1 O que é RAG?\*\*

RAG (Retrieval-Augmented Generation) é uma técnica que permite que modelos de IA acessem e usem informações externas para gerar respostas mais precisas e contextualizadas.

\*\*Como funciona\*\*:

1. O usuário faz uma pergunta

2. O sistema busca documentos relevantes no banco de conhecimento

3. Os documentos relevantes são adicionados ao prompt

4. O modelo gera uma resposta com base no contexto fornecido

#### \*\*5.2.2 Implementação com ChromaDB\*\*

\*\*Passo 1: Instalar dependências\*\*

```bash

pip install chromadb langchain-community

```

\*\*Passo 2: Carregar documentos\*\*

```python

from langchain\_community.document\_loaders import PyPDFLoader, DirectoryLoader

from langchain.text\_splitter import RecursiveCharacterTextSplitter

# Carregar PDFs de um diretório

loader = DirectoryLoader('./docs/', glob="./\*.pdf")

documents = loader.load()

# Dividir documentos em pedaços

text\_splitter = RecursiveCharacterTextSplitter(

chunk\_size=1000,

chunk\_overlap=200

)

texts = text\_splitter.split\_documents(documents)

```

\*\*Passo 3: Criar embeddings e salvar no ChromaDB\*\*

```python

from langchain\_community.embeddings import OllamaEmbeddings

from langchain\_community.vectorstores import Chroma

# Configurar embeddings com Ollama

embeddings = OllamaEmbeddings(model="nomic-embed-text")

# Criar banco de dados vetorial

vector\_db = Chroma.from\_documents(

documents=texts,

embedding=embeddings,

persist\_directory='./vector\_db'

)

vector\_db.persist()

```

\*\*Passo 4: Configurar o sistema RAG\*\*

```python

from langchain.chains import RetrievalQA

from langchain\_community.llms import Ollama

# Configurar o LLM

llm = Ollama(model="mistral")

# Configurar o retriever

retriever = vector\_db.as\_retriever(

search\_kwargs={"k": 3} # Número de documentos a recuperar

)

# Criar chain RAG

qa\_chain = RetrievalQA.from\_chain\_type(

llm=llm,

retriever=retriever,

return\_source\_documents=True

)

```

\*\*Passo 5: Usar o sistema RAG\*\*

```python

def ask\_question(question):

response = qa\_chain.invoke(question)

print(f"Resposta: {response['result']}")

print("\nFontes:")

for i, doc in enumerate(response['source\_documents']):

print(f"{i+1}. {doc.metadata['source']}")

# Testar

ask\_question("Como criar um componente de formulário em React?")

```

#### \*\*5.2.3 Melhorias Avançadas para RAG\*\*

\*\*1. Adicionando Metadados\*\*

```python

# Ao carregar documentos, adicione metadados

for doc in texts:

doc.metadata["source"] = "manual\_react.pdf"

doc.metadata["category"] = "react\_component"

doc.metadata["date"] = "2023-10-15"

# Na busca, filtre por metadados

retriever = vector\_db.as\_retriever(

search\_type="similarity",

search\_kwargs={

"k": 3,

"filter": {"category": "react\_component"}

}

)

```

\*\*2. Reranking para Melhor Precisão\*\*

```python

from langchain.retrievers import ContextualCompressionRetriever

from langchain\_community.cross\_encoders import HuggingFaceCrossEncoder

# Configurar reranker

model = HuggingFaceCrossEncoder(model\_name="BAAI/bge-reranker-base")

compression\_retriever = ContextualCompressionRetriever(

base\_compressor=model,

base\_retriever=retriever

)

# Usar o reranker

compressed\_docs = compression\_retriever.invoke("Como criar um componente?")

```

\*\*3. Auto-Query para Melhor Busca\*\*

```python

from langchain.retrievers import SelfQueryRetriever

from langchain.chains.query\_constructor.base import AttributeInfo

# Definir metadados que podem ser consultados

metadata\_field\_info = [

AttributeInfo(

name="source",

description="O nome do arquivo PDF",

type="string",

),

AttributeInfo(

name="category",

description="A categoria do conteúdo",

type="string",

),

AttributeInfo(

name="date",

description="Data do documento no formato AAAA-MM-DD",

type="string",

),

]

# Configurar o retriever auto-query

self\_query\_retriever = SelfQueryRetriever.from\_llm(

llm,

vector\_db,

"Documentos de programação",

metadata\_field\_info,

verbose=True

)

# Agora o sistema pode interpretar consultas naturais

docs = self\_query\_retriever.invoke("Documentos sobre React de outubro de 2023")

```

### \*\*5.3 MCP (Multi-Component Prompting): Estrutura Avançada\*\*

#### \*\*5.3.1 O que é MCP?\*\*

MCP (Multi-Component Prompting) é uma técnica avançada de construção de prompts que divide a solicitação em múltiplos componentes para melhorar a qualidade das respostas do agente.

\*\*Componentes Principais de um Prompt MCP\*\*:

1. \*\*Contexto\*\*: Informações de fundo necessárias

2. \*\*Objetivo\*\*: O que você quer que o agente faça

3. \*\*Restrições\*\*: Limitações ou diretrizes específicas

4. \*\*Exemplos\*\*: Casos de uso ou exemplos de saída desejada

5. \*\*Formato\*\*: Como a resposta deve ser estruturada

6. \*\*Etapa Final\*\*: Instrução clara do que fazer

#### \*\*5.3.2 Exemplo de Prompt MCP para Programação\*\*

```

# Contexto

Você é um especialista em React com 10 anos de experiência.

Estamos construindo um aplicativo de lista de tarefas usando React 18.

# Objetivo

Crie um componente de formulário para adicionar novas tarefas.

# Restrições

- Use hooks do React (useState, useEffect)

- Valide o input para não permitir tarefas vazias

- Use Tailwind CSS para estilização

- Não use bibliotecas externas

# Exemplos

Componente de botão:

function Button({ children, onClick }) {

return (

<button

onClick={onClick}

className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded"

>

{children}

</button>

);

}

# Formato

A resposta deve conter:

1. Código do componente em JavaScript

2. Breve explicação das principais decisões

3. Como usar o componente

# Etapa Final

Gere o código do componente de formulário seguindo todas as diretrizes acima.

```

#### \*\*5.3.3 Implementação do MCP no n8n\*\*

\*\*Passo 1: Configurar o MCP Server\*\*

1. No n8n, crie um novo workflow

2. Adicione um nó "Webhook" como trigger

3. Adicione um nó "Function" após o Webhook

4. No código da função, adicione:

```javascript

// Processar o prompt MCP

const { objective, context, constraints, examples, format, finalStep } = items[0].json;

// Construir o prompt final

const systemMessage = `

# Contexto

${context}

# Objetivo

${objective}

# Restrições

${constraints}

${examples ? '# Exemplos\n' + examples : ''}

${format ? '# Formato\n' + format : ''}

# Etapa Final

${finalStep}

`;

items[0].json.systemMessage = systemMessage;

return items;

```

\*\*Passo 2: Integrar com o Agente IA\*\*

1. Adicione um nó "AI Agent" após o nó MCP

2. Configure o AI Agent:

- System Message: `{{ $json["systemMessage"] }}`

- Model: mistral

3. Conecte o AI Agent de volta ao Webhook como resposta

\*\*Passo 3: Testar o MCP\*\*

1. Use Postman para enviar uma requisição:

```json

{

"context": "Você é um especialista em React com 10 anos de experiência. Estamos construindo um aplicativo de lista de tarefas usando React 18.",

"objective": "Crie um componente de formulário para adicionar novas tarefas.",

"constraints": "- Use hooks do React (useState, useEffect)\n- Valide o input para não permitir tarefas vazias\n- Use Tailwind CSS para estilização\n- Não use bibliotecas externas",

"examples": "Componente de botão:\nfunction Button({ children, onClick }) {\n return (\n <button \n onClick={onClick}\n className=\"bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded\"\n >\n {children}\n </button>\n );\n}",

"format": "A resposta deve conter:\n1. Código do componente em JavaScript\n2. Breve explicação das principais decisões\n3. Como usar o componente",

"finalStep": "Gere o código do componente de formulário seguindo todas as diretrizes acima."

}

```

2. Envie para http://localhost:5678/webhook/mcp

3. Você deve receber uma resposta estruturada seguindo todas as diretrizes

#### \*\*5.3.4 Templates MCP Prontos para Uso\*\*

\*\*Template MCP para Programação\*\*:

```json

{

"context": "Você é um especialista em [TECNOLOGIA] com [ANOS] anos de experiência. Estamos trabalhando em um projeto que usa [TECNOLOGIA] para [OBJETIVO DO PROJETO].",

"objective": "[TAREFA ESPECÍFICA QUE VOCÊ QUER QUE O AGENTE REALIZE]",

"constraints": "- [RESTRICAO 1]\n- [RESTRICAO 2]\n- [RESTRICAO 3]",

"examples": "[EXEMPLO DE CÓDIGO OU RESPOSTA DESEJADA]",

"format": "A resposta deve conter:\n1. [PARTE 1 DA RESPOSTA]\n2. [PARTE 2 DA RESPOSTA]\n3. [PARTE 3 DA RESPOSTA]",

"finalStep": "Gere a solução seguindo todas as diretrizes acima."

}

```

\*\*Template MCP para Análise de Código\*\*:

```json

{

"context": "Você é um engenheiro de software sênior especializado em revisão de código. Estamos analisando um projeto que usa [TECNOLOGIA] para [OBJETIVO].",

"objective": "Analise o seguinte código e identifique problemas de qualidade, segurança e desempenho.",

"constraints": "- Foque em problemas reais, não em preferências pessoais\n- Forneça sugestões específicas de melhoria\n- Classifique os problemas por severidade (crítico, alto, médio, baixo)\n- Não sugira mudanças que não melhorem significativamente o código",

"examples": "Exemplo de análise:\n\nProblema: Uso de variável global sem validação\nSeveridade: Alto\nSugestão: Remover variável global e usar props ou contexto do React\nExplicação: Variáveis globais podem causar efeitos colaterais inesperados...",

"format": "A resposta deve conter:\n1. Lista de problemas organizados por severidade\n2. Para cada problema: descrição, severidade, sugestão de correção e explicação\n3. Resumo das principais melhorias",

"finalStep": "Analise o código abaixo e siga todas as diretrizes acima.\n\n[CÓDIGO AQUI]"

}

```

### \*\*5.4 Integração com n8n e Outras Ferramentas de Automação\*\*

#### \*\*5.4.1 n8n: Automatização Visual para Agentes\*\*

\*\*Por que usar n8n para agentes de IA\*\*:

- Interface visual intuitiva (não requer programação)

- Mais de 300 integrações nativas

- Suporte a agentes com memória e RAG

- Pode ser executado localmente sem custo

- Autores de workflow com IA integrado

\*\*Configuração Básica do AI Agent no n8n\*\*:

1. Adicione um nó "AI Agent" ao seu workflow

2. Configure as credenciais para seu modelo local (Ollama)

3. Defina a System Message:

```

Você é um assistente de programação especializado em [TECNOLOGIA].

Forneça respostas detalhadas e exemplos de código quando relevante.

```

4. Configure as ferramentas disponíveis:

- Calculadora

- Busca na web

- Google Sheets

- Terminal

- Editor de código

\*\*Exemplo de Workflow Completo no n8n\*\*:

1. \*\*Trigger\*\*: Webhook ou Telegram

2. \*\*Processamento\*\*: Nó "Function" para preparar o contexto

3. \*\*RAG\*\*: Nó para buscar informações no ChromaDB

4. \*\*Agente IA\*\*: Nó "AI Agent" com o prompt completo

5. \*\*Ferramentas\*\*: Nós para executar ações específicas

6. \*\*Resposta\*\*: Nó para formatar e enviar a resposta

#### \*\*5.4.2 Integração com o Cursor para Programação\*\*

\*\*Passo 1: Configurar o Cursor para Aceitar Comandos Externos\*\*

1. No Cursor, vá para Settings > API

2. Ative a opção "Allow external API access"

3. Anote o token de API gerado

\*\*Passo 2: Criar Workflow no n8n para Integração com Cursor\*\*

1. Adicione um nó "HTTP Request" ao seu workflow

2. Configure:

- URL: http://localhost:8000/api/v1/editor

- Method: POST

- Authentication: Bearer Token

- Token: [seu\_token\_aqui]

- Body:

```json

{

"command": "insert",

"content": "{{ $json['code'] }}",

"position": {

"line": {{ $json['line'] }},

"character": {{ $json['character'] }}

}

}

```

\*\*Passo 3: Configurar o Agente para Usar o Cursor\*\*

1. No nó "AI Agent", adicione uma ferramenta "Cursor":

```json

{

"name": "Cursor",

"description": "Insere código no editor Cursor",

"parameters": {

"code": "string",

"line": "number",

"character": "number"

}

}

```

2. No prompt do agente, adicione:

```

Quando eu pedir para escrever código, use a ferramenta Cursor para inserir o código diretamente no editor.

```

#### \*\*5.4.3 Automação Avançada com n8n\*\*

\*\*Exemplo: Agente que Cria Aplicativos Completo\*\*

1. \*\*Trigger\*\*: Webhook com descrição do aplicativo

2. \*\*Processamento Inicial\*\*:

- Nó "Function" para extrair requisitos

- Nó "AI Agent" para planejar a arquitetura

3. \*\*Criação do Projeto\*\*:

- Nó "Exec" para criar diretórios

- Nó "Write Binary File" para criar arquivos

4. \*\*Geração de Código\*\*:

- Nó "AI Agent" com RAG para gerar código específico

- Nó "Cursor" para inserir código no editor

5. \*\*Testes\*\*:

- Nó "Exec" para rodar testes

- Nó "AI Agent" para analisar resultados

6. \*\*Resposta Final\*\*:

- Nó "Telegram" ou "Email" com resumo e código

\*\*Exemplo de Fluxo para Automação de Programação\*\*:

```

[Webhook] --> [Function: Extrair requisitos] --> [AI Agent: Planejar arquitetura]

|

v

[Function: Criar estrutura] --> [AI Agent: Gerar código componente 1]

| |

v v

[AI Agent: Gerar código componente 2] --> [Exec: Testar aplicativo]

| |

v v

[Write Binary: Salvar arquivos] --> [Telegram: Enviar resultado]

```

---

## \*\*6. CONSTRUÇÃO DE AGENTES\*\*

### \*\*6.1 Arquitetura de Agentes com Múltiplas Ferramentas\*\*

#### \*\*6.1.1 Componentes Essenciais de um Agente\*\*

\*\*1. Modelo de Linguagem (LLM)\*\*

- O "cérebro" do agente, responsável por processar entradas e gerar respostas

- No seu caso, será um modelo local como Mistral ou CodeLlama

\*\*2. Sistema de Planejamento\*\*

- Divide tarefas complexas em etapas menores

- Determina a ordem de execução das ações

\*\*3. Ferramentas (Tools)\*\*

- Capacidades específicas que o agente pode usar

- Exemplos: calculadora, buscador, editor de código, terminal

\*\*4. Memória\*\*

- Armazena contexto de conversas anteriores

- Permite ao agente lembrar informações importantes

\*\*5. Sistema de Feedback\*\*

- Avalia o sucesso das ações executadas

- Ajusta o comportamento com base nos resultados

#### \*\*6.1.2 Implementação de Ferramentas no n8n\*\*

\*\*Passo 1: Configurar Ferramentas Básicas\*\*

\*\*Calculadora\*\*:

1. Adicione um nó "Function" ao seu workflow

2. Configure:

```javascript

// Calculadora simples

if (items[0].json.tool === 'calculator') {

try {

const result = eval(items[0].json.input);

items[0].json.result = result.toString();

} catch (error) {

items[0].json.result = "Erro na expressão matemática";

}

}

return items;

```

\*\*Busca na Web\*\*:

1. Adicione um nó "HTTP Request"

2. Configure:

- URL: https://api.duckduckgo.com/?q={{ $json["query"] }}&format=json

- Method: GET

3. Adicione um nó "Function" para processar os resultados:

```javascript

// Processar resultados da busca

const results = items[0].json;

let summary = "Resultados da busca:\n";

if (results.RelatedTopics) {

results.RelatedTopics.slice(0, 3).forEach(topic => {

if (topic.FirstURL && topic.Text) {

summary += `- [${topic.Text}](${topic.FirstURL})\n`;

}

});

}

items[0].json.summary = summary;

return items;

```

\*\*Editor de Código\*\*:

1. Configure integração com Cursor conforme seção 5.4.2

2. Adicione um nó "HTTP Request" para enviar código ao Cursor

\*\*Terminal\*\*:

1. Adicione um nó "Exec"

2. Configure:

- Command: `{{ $json["command"] }}`

- Working Directory: `/seu/projeto`

#### \*\*6.1.3 Exemplo Completo: Agente de Programação com Múltiplas Ferramentas\*\*

\*\*Passo 1: Configurar o Nó AI Agent\*\*

1. Adicione um nó "AI Agent" ao seu workflow

2. Configure as credenciais para seu modelo local

3. Defina a System Message:

```

Você é um assistente de programação especializado em React.

Você tem acesso às seguintes ferramentas:

1. Calculadora: Para cálculos matemáticos

2. Busca na Web: Para pesquisar documentação e exemplos

3. Editor de Código: Para inserir código no Cursor

4. Terminal: Para executar comandos no sistema

Sempre que precisar de informações adicionais ou executar uma ação,

use a ferramenta apropriada e aguarde a resposta antes de continuar.

```

\*\*Passo 2: Configurar as Ferramentas\*\*

1. Clique em "Add Tool" no nó AI Agent

2. Adicione cada ferramenta com sua descrição e parâmetros

\*\*Exemplo de Configuração da Ferramenta "Busca na Web"\*\*:

- Name: `web\_search`

- Description: "Use esta ferramenta para pesquisar informações na internet. Forneça uma consulta de busca clara."

- Parameters:

```json

{

"query": {

"type": "string",

"description": "Termo de busca",

"required": true

}

}

```

\*\*Passo 3: Testar o Agente com Múltiplas Ferramentas\*\*

1. Envie uma mensagem como: "Como faço para implementar autenticação JWT em um projeto React?"

2. O agente deverá:

- Usar a ferramenta "Busca na Web" para pesquisar sobre JWT em React

- Analisar os resultados

- Gerar um exemplo de código

- Usar a ferramenta "Editor de Código" para inserir o código no Cursor

### \*\*6.2 Implementação de Memória de Contexto\*\*

#### \*\*6.2.1 Tipos de Memória para Agentes\*\*

\*\*1. Memória de Curto Prazo (Conversational)\*\*

- Armazena o contexto da conversa atual

- Permite ao agente lembrar do que foi discutido recentemente

- Implementada com ConversationBufferMemory no LangChain

\*\*2. Memória de Longo Prazo (Persistent)\*\*

- Armazena informações importantes entre sessões

- Permite ao agente lembrar preferências do usuário

- Implementada com ZEP ou ChromaDB

\*\*3. Memória Vetorial\*\*

- Armazena embeddings de conversas passadas

- Permite recuperar conversas semelhantes para contexto

- Implementada com bancos vetoriais como ChromaDB

#### \*\*6.2.2 Implementação com ZEP (Recomendado)\*\*

\*\*Passo 1: Instalar e Configurar o ZEP\*\*

1. Baixe o ZEP em https://getzep.com

2. Siga as instruções de instalação

3. Inicie o servidor:

```bash

zep start

```

\*\*Passo 2: Integrar ZEP com o n8n\*\*

1. Crie um nó "Function" após o nó AI Agent

2. Adicione este código:

```javascript

const axios = require('axios');

const sessionId = items[0].json.sessionId || 'default';

// Salvar mensagem do usuário

await axios.post(`http://localhost:8000/sessions/${sessionId}/messages`, {

role: 'user',

content: items[0].json.input

});

// Salvar resposta do agente

await axios.post(`http://localhost:8000/sessions/${sessionId}/messages`, {

role: 'assistant',

content: items[0].json.response

});

// Adicionar histórico ao contexto

const historyResponse = await axios.get(`http://localhost:8000/sessions/${sessionId}/messages`);

items[0].json.history = historyResponse.data.messages

.map(m => `${m.role}: ${m.content}`)

.join('\n');

return items;

```

\*\*Passo 3: Modificar o Prompt do Agente\*\*

Atualize a System Message para incluir:

```

Use o seguinte histórico de conversa para contextualizar sua resposta:

{history}

Pergunta atual: {input}

```

#### \*\*6.2.3 Implementação com ChromaDB (Para Memória Vetorial)\*\*

\*\*Passo 1: Configurar o ChromaDB para Memória\*\*

```python

from langchain\_community.vectorstores import Chroma

from langchain\_community.embeddings import OllamaEmbeddings

from uuid import uuid4

# Configurar embeddings

embeddings = OllamaEmbeddings(model="nomic-embed-text")

# Criar banco de memória

memory\_db = Chroma(

collection\_name="conversation\_memory",

embedding\_function=embeddings,

persist\_directory="./memory\_db"

)

def save\_memory(user\_input, agent\_response, session\_id):

"""Salva uma interação na memória"""

memory\_id = str(uuid4())

memory\_db.add\_texts(

texts=[f"User: {user\_input}\nAgent: {agent\_response}"],

metadatas=[{"session\_id": session\_id}],

ids=[memory\_id]

)

def get\_relevant\_memory(query, session\_id, k=3):

"""Recupera memórias relevantes"""

results = memory\_db.similarity\_search(

query,

k=k,

filter={"session\_id": session\_id}

)

return [doc.page\_content for doc in results]

```

\*\*Passo 2: Integrar com o n8n\*\*

1. Crie um serviço Python com as funções acima

2. Exponha endpoints REST para salvar e recuperar memória

3. Conecte ao n8n com nós "HTTP Request"

### \*\*6.3 Agentes Multi-Usuário e Personalização\*\*

#### \*\*6.3.1 Sistema de Perfis de Usuário\*\*

\*\*Passo 1: Criar Estrutura de Dados\*\*

Crie um arquivo `users.json` para armazenar perfis:

```json

{

"default": {

"name": "Usuário",

"preferences": {

"programming\_language": "javascript",

"experience\_level": "intermediate",

"coding\_style": "functional"

}

},

"joao123": {

"name": "João",

"preferences": {

"programming\_language": "python",

"experience\_level": "advanced",

"coding\_style": "object-oriented"

}

}

}

```

\*\*Passo 2: Integrar com o n8n\*\*

1. Adicione um nó "Read Binary File" para carregar `users.json`

2. Adicione um nó "Function" para processar:

```javascript

const fs = require('fs');

const path = require('path');

// Carregar perfis de usuário

const usersPath = path.join(\_\_dirname, 'users.json');

const users = JSON.parse(fs.readFileSync(usersPath, 'utf8'));

// Obter ID do usuário (pode vir do Telegram, Webhook, etc.)

const userId = items[0].json.userId || 'default';

// Obter perfil do usuário

const profile = users[userId] || users['default'];

// Adicionar ao contexto

items[0].json.userProfile = profile;

return items;

```

\*\*Passo 3: Personalizar o Agente com Base no Perfil\*\*

Atualize a System Message para incluir:

```

Você está conversando com {userProfile.name}.

Preferências do usuário:

- Linguagem de programação: {userProfile.preferences.programming\_language}

- Nível de experiência: {userProfile.preferences.experience\_level}

- Estilo de codificação: {userProfile.preferences.coding\_style}

Adapte suas respostas às preferências do usuário.

```

#### \*\*6.3.2 Exemplo de Agente Personalizado para Diferentes Níveis\*\*

\*\*Para Iniciantes\*\*:

```

Você está explicando conceitos de programação para um iniciante.

Use analogias do mundo real, evite jargões técnicos e forneça explicações passo a passo.

Inclua comentários detalhados no código e explique cada parte.

```

\*\*Para Intermediários\*\*:

```

Você está conversando com um programador intermediário.

Forneça explicações técnicas claras, mas não muito básicas.

Inclua exemplos de código com comentários estratégicos.

Mencione conceitos avançados, mas explique brevemente se necessário.

```

\*\*Para Avançados\*\*:

```

Você está conversando com um engenheiro de software sênior.

Use terminologia técnica precisa e assuma conhecimento profundo.

Forneça exemplos concisos com foco em padrões e boas práticas.

Discuta trade-offs e considerações de arquitetura quando relevante.

```

### \*\*6.4 Agentes Especializados para Programação\*\*

#### \*\*6.4.1 Arquitetura de um Agente de Programação\*\*

\*\*1. Analisador de Requisitos\*\*

- Entende o que o usuário quer

- Identifica tecnologias relevantes

- Define escopo da tarefa

\*\*2. Planejador de Tarefas\*\*

- Divide o problema em etapas menores

- Define ordem de execução

- Identifica dependências

\*\*3. Gerador de Código\*\*

- Cria código específico para cada etapa

- Garante qualidade e boas práticas

- Inclui comentários explicativos

\*\*4. Executor de Código\*\*

- Testa o código gerado

- Identifica erros

- Sugere correções

\*\*5. Documentador\*\*

- Gera documentação para o código

- Cria exemplos de uso

- Explica decisões de design

#### \*\*6.4.2 Implementação no n8n\*\*

\*\*Passo 1: Configurar o Nó "AI Agent" para Programação\*\*

1. System Message:

```

Você é um assistente de programação especializado em [linguagem].

Siga este processo para todas as solicitações:

1. ENTENDIMENTO: Peça esclarecimentos se necessário e confirme os requisitos

2. PLANEJAMENTO: Divida a tarefa em etapas lógicas

3. IMPLEMENTAÇÃO: Gere código para cada etapa, explicando brevemente

4. TESTE: Descreva como testar a implementação

5. DOCUMENTAÇÃO: Forneça exemplos de uso e explicações

Sempre priorize código limpo, seguro e eficiente.

```

\*\*Passo 2: Adicionar Ferramentas Específicas para Programação\*\*

\*\*Ferramenta "Analisar Código"\*\*:

- Name: `code\_analysis`

- Description: "Analisa código fornecido e identifica problemas de qualidade, segurança e desempenho"

- Parameters:

```json

{

"language": {

"type": "string",

"description": "Linguagem de programação",

"required": true

},

"code": {

"type": "string",

"description": "Código a ser analisado",

"required": true

}

}

```

\*\*Ferramenta "Gerar Testes"\*\*:

- Name: `generate\_tests`

- Description: "Gera testes automatizados para o código fornecido"

- Parameters:

```json

{

"language": {

"type": "string",

"description": "Linguagem de programação",

"required": true

},

"framework": {

"type": "string",

"description": "Framework de testes (ex: Jest, pytest)",

"required": true

},

"code": {

"type": "string",

"description": "Código para o qual gerar testes",

"required": true

}

}

```

#### \*\*6.4.3 Exemplo Completo: Agente que Cria um Componente React\*\*

\*\*Fluxo de Trabalho\*\*:

1. \*\*Input do Usuário\*\*: "Crie um componente de formulário de login em React"

2. \*\*Análise de Requisitos\*\*:

- O agente pergunta: "Qual framework de estilização você está usando? (ex: Tailwind, styled-components)"

- O agente pergunta: "Precisa de validação de formulário?"

3. \*\*Planejamento\*\*:

- Definir estrutura do componente

- Implementar estados para inputs

- Adicionar validação básica

- Estilizar com Tailwind CSS

4. \*\*Implementação\*\*:

- Gerar código do componente

- Incluir exemplos de uso

5. \*\*Teste\*\*:

- Descrever como testar o componente

- Sugerir testes com React Testing Library

6. \*\*Documentação\*\*:

- Explicar props do componente

- Mostrar exemplos de uso em diferentes cenários

\*\*Exemplo de Resposta do Agente\*\*:

```

# Componente de Formulário de Login em React

## Análise de Requisitos

- Componente funcional usando hooks

- Validação básica de email e senha

- Estilização com Tailwind CSS

- Gerenciamento de estado com useState

## Código Implementado

```jsx

import { useState } from 'react';

function LoginForm({ onSubmit }) {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const [errors, setErrors] = useState({});

const validate = () => {

const newErrors = {};

if (!email) {

newErrors.email = 'Email é obrigatório';

} else if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)) {

newErrors.email = 'Email inválido';

}

if (!password) {

newErrors.password = 'Senha é obrigatória';

} else if (password.length < 6) {

newErrors.password = 'Senha deve ter pelo menos 6 caracteres';

}

setErrors(newErrors);

return Object.keys(newErrors).length === 0;

};

const handleSubmit = (e) => {

e.preventDefault();

if (validate()) {

onSubmit({ email, password });

}

};

return (

<form onSubmit={handleSubmit} className="max-w-md mx-auto">

<div className="mb-4">

<label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="email">

Email

</label>

<input

id="email"

type="email"

value={email}

onChange={(e) => setEmail(e.target.value)}

className={`shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline ${

errors.email ? 'border-red-500' : ''

}`}

/>

{errors.email && <p className="text-red-500 text-xs mt-1">{errors.email}</p>}

</div>

<div className="mb-6">

<label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="password">

Senha

</label>

<input

id="password"

type="password"

value={password}

onChange={(e) => setPassword(e.target.value)}

className={`shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 mb-3 leading-tight focus:outline-none focus:shadow-outline ${

errors.password ? 'border-red-500' : ''

}`}

/>

{errors.password && <p className="text-red-500 text-xs mt-1">{errors.password}</p>}

</div>

<div className="flex items-center justify-between">

<button

type="submit"

className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline"

>

Entrar

</button>

<a className="inline-block align-baseline font-bold text-sm text-blue-500 hover:text-blue-800" href="#">

Esqueceu a senha?

</a>

</div>

</form>

);

}

export default LoginForm;

```

## Como Usar

```jsx

function App() {

const handleLogin = (credentials) => {

console.log('Tentativa de login:', credentials);

// Aqui você faria a chamada à API de autenticação

};

return (

<div className="App">

<h1>Login</h1>

<LoginForm onSubmit={handleLogin} />

</div>

);

}

```

## Testes Recomendados

1. Teste de renderização básica

2. Teste de validação de email inválido

3. Teste de validação de senha curta

4. Teste de submissão com dados válidos

Você pode usar React Testing Library para esses testes:

```javascript

import { render, screen, fireEvent } from '@testing-library/react';

import LoginForm from './LoginForm';

test('exibe erro para email inválido', () => {

render(<LoginForm onSubmit={() => {}} />);

const emailInput = screen.getByLabelText(/email/i);

fireEvent.change(emailInput, { target: { value: 'not-an-email' } });

const submitButton = screen.getByText(/entrar/i);

fireEvent.click(submitButton);

expect(screen.getByText(/email inválido/i)).toBeInTheDocument();

});

```

```

---

## \*\*7. APLICAÇÕES REAIS\*\*

### \*\*7.1 Programação com Linguagem Natural\*\*

#### \*\*7.1.1 Como Funciona a Programação com Linguagem Natural\*\*

A programação com linguagem natural permite que você descreva em português o que deseja criar, e o agente traduz isso em código funcional. Baseado nos documentos que você compartilhou, identifiquei 4 etapas principais desse processo:

\*\*1. Compreensão do Requisito\*\*

- O agente analisa sua descrição em linguagem natural

- Identifica tecnologias relevantes (React, Python, etc.)

- Determina o escopo e complexidade da tarefa

\*\*2. Planejamento da Implementação\*\*

- Divide o problema em etapas menores e gerenciáveis

- Define a arquitetura básica da solução

- Identifica bibliotecas ou dependências necessárias

\*\*3. Geração do Código\*\*

- Cria código específico para cada componente

- Garante que siga boas práticas da linguagem

- Inclui comentários explicativos quando necessário

\*\*4. Validação e Documentação\*\*

- Descreve como testar a implementação

- Fornece exemplos de uso

- Explica decisões de design importantes

#### \*\*7.1.2 Exemplos Práticos com o Seu Sistema\*\*

\*\*Exemplo 1: Criar um Componente de Botão em React\*\*

\*\*Input em Linguagem Natural\*\*:

"Crie um componente de botão em React que mude de cor quando o mouse passa por cima. Use Tailwind CSS para estilização."

\*\*Processo do Agente\*\*:

1. \*\*Compreensão\*\*:

- Tecnologia: React com Tailwind CSS

- Funcionalidade: botão com efeito hover

- Requisitos não explícitos: deve ser um componente reutilizável

2. \*\*Planejamento\*\*:

- Criar componente funcional Button

- Usar classes do Tailwind para estilização básica

- Adicionar classes para o estado hover

- Permitir personalização via props

3. \*\*Implementação\*\*:

```jsx

import React from 'react';

const Button = ({

children,

onClick,

variant = 'primary',

className = ''

}) => {

// Definir estilos base

const baseStyles = "px-4 py-2 rounded font-medium transition-colors duration-200";

// Definir estilos por variante

const variants = {

primary: "bg-blue-500 hover:bg-blue-700 text-white",

secondary: "bg-gray-300 hover:bg-gray-400 text-gray-800",

danger: "bg-red-500 hover:bg-red-700 text-white"

};

// Combinar classes

const buttonClass = `${baseStyles} ${variants[variant]} ${className}`;

return (

<button

className={buttonClass}

onClick={onClick}

>

{children}

</button>

);

};

export default Button;

```

4. \*\*Documentação\*\*:

```markdown

# Componente Button

## Como Usar

```jsx

import Button from './Button';

function App() {

return (

<div className="p-4">

<Button variant="primary" onClick={() => console.log('Clicado')}>

Botão Primário

</Button>

<Button variant="secondary" className="ml-2">

Botão Secundário

</Button>

<Button variant="danger" className="ml-2">

Botão de Perigo

</Button>

</div>

);

}

```

## Props

- `children`: Conteúdo do botão (obrigatório)

- `onClick`: Função a ser chamada quando clicado (opcional)

- `variant`: Estilo do botão ('primary', 'secondary', 'danger') - padrão: 'primary'

- `className`: Classes CSS adicionais (opcional)

```

\*\*Exemplo 2: Criar uma API REST com Python e Flask\*\*

\*\*Input em Linguagem Natural\*\*:

"Crie uma API REST simples com Python usando Flask que permita gerenciar uma lista de tarefas. Deve ter endpoints para listar, criar, atualizar e excluir tarefas."

\*\*Processo do Agente\*\*:

1. \*\*Compreensão\*\*:

- Tecnologia: Python com Flask

- Funcionalidade: API REST para gerenciamento de tarefas

- Requisitos: CRUD completo para tarefas

2. \*\*Planejamento\*\*:

- Estrutura básica do projeto

- Modelo de dados para tarefas

- Endpoints REST:

- GET /tasks: listar todas as tarefas

- POST /tasks: criar nova tarefa

- GET /tasks/<id>: obter uma tarefa específica

- PUT /tasks/<id>: atualizar uma tarefa

- DELETE /tasks/<id>: excluir uma tarefa

3. \*\*Implementação\*\*:

```python

from flask import Flask, jsonify, request

app = Flask(\_\_name\_\_)

# Banco de dados em memória (para exemplo)

tasks = [

{"id": 1, "title": "Estudar Flask", "done": False},

{"id": 2, "title": "Criar API REST", "done": True}

]

# Rota para listar todas as tarefas

@app.route('/tasks', methods=['GET'])

def get\_tasks():

return jsonify({'tasks': tasks})

# Rota para criar uma nova tarefa

@app.route('/tasks', methods=['POST'])

def create\_task():

if not request.json or 'title' not in request.json:

return jsonify({'error': 'Título é obrigatório'}), 400

task = {

'id': tasks[-1]['id'] + 1 if tasks else 1,

'title': request.json['title'],

'done': request.json.get('done', False)

}

tasks.append(task)

return jsonify({'task': task}), 201

# Rota para obter uma tarefa específica

@app.route('/tasks/<int:task\_id>', methods=['GET'])

def get\_task(task\_id):

task = next((t for t in tasks if t['id'] == task\_id), None)

if task is None:

return jsonify({'error': 'Tarefa não encontrada'}), 404

return jsonify({'task': task})

# Rota para atualizar uma tarefa

@app.route('/tasks/<int:task\_id>', methods=['PUT'])

def update\_task(task\_id):

task = next((t for t in tasks if t['id'] == task\_id), None)

if task is None:

return jsonify({'error': 'Tarefa não encontrada'}), 404

if 'title' in request.json:

task['title'] = request.json['title']

if 'done' in request.json:

task['done'] = request.json['done']

return jsonify({'task': task})

# Rota para excluir uma tarefa

@app.route('/tasks/<int:task\_id>', methods=['DELETE'])

def delete\_task(task\_id):

global tasks

tasks = [t for t in tasks if t['id'] != task\_id]

return jsonify({'result': True})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

```

4. \*\*Documentação\*\*:

```markdown

# API de Gerenciamento de Tarefas

## Endpoints

### GET /tasks

Lista todas as tarefas

\*\*Resposta de Exemplo\*\*:

```json

{

"tasks": [

{"id": 1, "title": "Estudar Flask", "done": false},

{"id": 2, "title": "Criar API REST", "done": true}

]

}

```

### POST /tasks

Cria uma nova tarefa

\*\*Corpo da Requisição\*\*:

```json

{

"title": "Nova tarefa",

"done": false

}

```

### GET /tasks/<id>

Obtém uma tarefa específica pelo ID

### PUT /tasks/<id>

Atualiza uma tarefa existente

### DELETE /tasks/<id>

Exclui uma tarefa

## Como Executar

1. Instale as dependências: `pip install flask`

2. Execute o servidor: `python app.py`

3. Acesse a API em `http://localhost:5000`

```

### \*\*7.2 Automação de Tarefas de Desenvolvimento\*\*

#### \*\*7.2.1 Automação Básica com n8n\*\*

\*\*Exemplo: Fluxo para Criar Novo Projeto React\*\*

\*\*Passo 1: Configurar o Trigger\*\*

- Use um nó "Webhook" ou "Manual Trigger"

- Configure o caminho: `/create-react-project`

\*\*Passo 2: Coletar Informações do Projeto\*\*

- Adicione um nó "Function" para processar os dados:

```javascript

// Processar dados do projeto

const projectName = items[0].json.projectName || 'meu-projeto';

const features = items[0].json.features || [];

items[0].json.projectData = {

name: projectName,

features: features,

path: `./projects/${projectName}`

};

return items;

```

\*\*Passo 3: Criar Estrutura do Projeto\*\*

- Adicione um nó "Exec" para criar diretórios:

```bash

mkdir -p {{ $json["projectData"]["path"] }}/{src,public,src/components,src/hooks}

```

\*\*Passo 4: Gerar Arquivos Básicos\*\*

- Adicione um nó "Write Binary File" para criar `package.json`:

```json

{

"name": "{{ $json["projectData"]["name"] }}",

"version": "1.0.0",

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build"

},

"dependencies": {

"react": "^18.2.0",

"react-dom": "^18.2.0",

"react-scripts": "5.0.1"

}

}

```

\*\*Passo 5: Instalar Dependências\*\*

- Adicione um nó "Exec" para instalar dependências:

```bash

cd {{ $json["projectData"]["path"] }} && npm install

```

\*\*Passo 6: Adicionar Funcionalidades Específicas\*\*

- Adicione um nó "IF" para verificar features:

- Se "authentication" está na lista, adicione um nó para criar arquivos de autenticação

- Se "routing" está na lista, adicione um nó para configurar rotas

\*\*Passo 7: Enviar Confirmação\*\*

- Adicione um nó "Telegram" ou "Email" para notificar conclusão

#### \*\*7.2.2 Automação Avançada: Agente que Mantém Código Atualizado\*\*

\*\*Objetivo\*\*: Criar um agente que monitora dependências e sugere atualizações

\*\*Fluxo de Trabalho\*\*:

1. \*\*Trigger\*\*: Agendamento diário (nó "Schedule Trigger")

2. \*\*Coletar Informações\*\*:

- Nó "Read Binary File" para ler `package.json`

- Nó "HTTP Request" para verificar versões no npm

3. \*\*Analisar Dependências\*\*:

- Nó "Function" para comparar versões:

```javascript

// Analisar dependências

const currentDeps = items[0].json.dependencies;

const latestVersions = items[0].json.latestVersions;

const outdated = [];

for (const [name, current] of Object.entries(currentDeps)) {

if (latestVersions[name] && latestVersions[name] !== current) {

outdated.push({ name, current, latest: latestVersions[name] });

}

}

items[0].json.outdated = outdated;

return items;

```

4. \*\*Gerar Relatório\*\*:

- Nó "AI Agent" para criar resumo:

```

Analise as seguintes dependências desatualizadas e gere um relatório com:

- Lista de dependências desatualizadas

- Riscos de atualização

- Comandos para atualizar

- Possíveis impactos na aplicação

Dados:

{outdated}

```

5. \*\*Notificar Usuário\*\*:

- Nó "Telegram" ou "Email" com o relatório

### \*\*7.3 Pipelines no n8n com Comandos em Linguagem Natural\*\*

#### \*\*7.3.1 MCP para Criação de Workflows\*\*

\*\*Exemplo: Criar Workflow para Processar PDFs\*\*

\*\*Input em Linguagem Natural\*\*:

"Crie um workflow que pegue PDFs de um diretório, extraia o texto, analise o sentimento e salve os resultados em uma planilha Google."

\*\*Processo do MCP\*\*:

1. \*\*Contexto\*\*:

- Você é um especialista em automação com n8n

- Estamos criando um workflow para processamento de documentos

2. \*\*Objetivo\*\*:

- Criar um workflow que processe PDFs automaticamente

3. \*\*Restrições\*\*:

- Use apenas ferramentas disponíveis no n8n

- Não use código personalizado se possível

- Priorize a simplicidade e confiabilidade

4. \*\*Estrutura do Workflow\*\*:

- Trigger: Observar diretório para novos PDFs

- Etapa 1: Extrair texto do PDF

- Etapa 2: Analisar sentimento do texto

- Etapa 3: Salvar resultados no Google Sheets

5. \*\*Formato da Resposta\*\*:

- Descreva cada nó necessário

- Forneça configurações específicas para cada nó

- Inclua exemplos de dados

\*\*Resposta do Agente\*\*:

# Workflow para Processamento de PDFs

## Estrutura do Workflow

### 1. Trigger: Watch Directory

- \*\*Tipo\*\*: Schedule Trigger

- \*\*Configuração\*\*:

- Trigger Type: Interval

- Interval: 5 minutes

- Directory: /caminho/para/pdfs

- File Pattern: \*.pdf

### 2. Ler Conteúdo do PDF

- \*\*Tipo\*\*: PDF Node

- \*\*Configuração\*\*:

- Operation: Extract Text

- Binary Property: data

- Result Property: extractedText

### 3. Analisar Sentimento

- \*\*Tipo\*\*: AI Agent Node

- \*\*Configuração\*\*:

- Model: mistral (local)

- System Message: "Você é um analisador de sentimento. Analise o texto e retorne 'positive', 'neutral' ou 'negative' junto com um score de 0-1."

- User Message: "{extractedText}"

### 4. Preparar Dados para Google Sheets

- \*\*Tipo\*\*: Function Node

- \*\*Código\*\*:

```javascript

// Preparar dados para Google Sheets

items[0].json.googleSheetsData = {

fileName: items[0].json.fileName,

sentiment: items[0].json.sentiment,

score: items[0].json.score

};

return items;

```

### 5. Salvar no Google Sheets

- \*\*Tipo\*\*: Google Sheets Node

- \*\*Configuração\*\*:

- Operation: Append

- Spreadsheet ID: [seu\_id\_aqui]

- Range: A:C

- Values:

```json

[

["{{ $json['fileName'] }}", "{{ $json['sentiment'] }}", "{{ $json['score'] }}"]

]

```

## Como Configurar

1. \*\*Configurar Credenciais do Google Sheets\*\*:

- Clique em "Credentials" no menu esquerdo

- Crie novas credenciais para Google API

- Siga o processo de autenticação OAuth

2. \*\*Obter Spreadsheet ID\*\*:

- Abra sua planilha Google

- O ID está na URL: https://docs.google.com/spreadsheets/d/[ID\_AQUI]/edit

3. \*\*Testar o Workflow\*\*:

- Coloque um PDF de teste no diretório monitorado

- Verifique se os dados aparecem na planilha

## Dicas Adicionais

- Para PDFs grandes, considere adicionar um nó de rate limiting

- Adicione um nó de notificação por email para erros

- Use o nó "IF" para processar apenas PDFs modificados recentemente

#### \*\*7.3.2 MCP Server para Automação Avançada\*\*

\*\*Passo 1: Configurar o MCP Server no n8n\*\*

1. Crie um novo workflow

2. Adicione um nó "Webhook" como trigger

3. Adicione um nó "Function" para processar o prompt MCP:

```javascript

// Processar prompt MCP

const { objective, context, constraints, format, finalStep } = items[0].json;

// Construir prompt para o agente

const systemMessage = `

Você é um especialista em criar workflows no n8n.

Sua tarefa é gerar um fluxo de trabalho que realize o seguinte:

# Contexto

${context}

# Objetivo

${objective}

# Restrições

${constraints}

# Formato da Resposta

${format}

# Instrução Final

${finalStep}

`;

// Configurar entrada para o AI Agent

items[0].json.systemMessage = systemMessage;

items[0].json.userMessage = "Crie o workflow conforme as instruções acima.";

return items;

```

\*\*Passo 2: Adicionar o AI Agent\*\*

1. Adicione um nó "AI Agent" após o nó MCP

2. Configure:

- System Message: `{{ $json["systemMessage"] }}`

- User Message: `{{ $json["userMessage"] }}`

- Model: mistral

- Response Property: mcpResponse

\*\*Passo 3: Processar a Resposta\*\*

1. Adicione um nó "Function" para analisar a resposta:

```javascript

// Analisar resposta do MCP

const response = items[0].json.mcpResponse;

// Extrair estrutura do workflow

const workflow = {

nodes: [],

connections: []

};

// Processar nós (exemplo simplificado)

const nodeRegex = /- Tipo: ([\w\s]+)\n- Configuração:\n([\s\S]\*?)(?=\n- |\n$)/g;

let match;

while ((match = nodeRegex.exec(response)) !== null) {

const nodeType = match[1].trim();

const config = match[2].trim();

workflow.nodes.push({

type: nodeType,

config: config

});

}

items[0].json.workflow = workflow;

return items;

```

\*\*Passo 4: Gerar o Workflow Real\*\*

1. Adicione um nó "Function" para criar o JSON do n8n:

```javascript

// Gerar JSON do n8n

const workflow = items[0].json.workflow;

// Mapear para formato n8n

const n8nWorkflow = {

name: "Workflow Gerado por MCP",

nodes: [],

connections: {}

};

// Adicionar nós

workflow.nodes.forEach((node, index) => {

n8nWorkflow.nodes.push({

parameters: parseConfig(node.config),

name: `Node ${index + 1}`,

type: mapNodeType(node.type),

typeVersion: 1,

position: [index \* 300, 0]

});

});

// Adicionar conexões

for (let i = 0; i < n8nWorkflow.nodes.length - 1; i++) {

const from = n8nWorkflow.nodes[i].name;

const to = n8nWorkflow.nodes[i + 1].name;

n8nWorkflow.connections[from] = { "main": [[{ "node": to, "type": "main", "index": 0 }]] };

}

items[0].json.n8nWorkflow = n8nWorkflow;

return items;

```

\*\*Passo 5: Exportar o Workflow\*\*

1. Adicione um nó "Write Binary File" para salvar o JSON

2. Configure:

- Property Name: n8nWorkflow

- File Name: workflow-{{ $now }}.json

- Binary Property: data

### \*\*7.4 Assistentes Personalizados para Diferentes Necessidades\*\*

#### \*\*7.4.1 Agente para Revisão de Código\*\*

\*\*Objetivo\*\*: Analisar código enviado pelo usuário e identificar problemas

\*\*Configuração no n8n\*\*:

1. \*\*Trigger\*\*: Webhook ou Telegram

2. \*\*Processamento Inicial\*\*:

- Nó "Function" para extrair código e linguagem

```javascript

// Extrair informações

items[0].json.code = items[0].json.message;

items[0].json.language = detectLanguage(items[0].json.code);

return items;

```

3. \*\*Análise com AI Agent\*\*:

- System Message:

```

Você é um engenheiro de software sênior especializado em revisão de código.

Sua tarefa é analisar o código fornecido e identificar:

- Problemas de qualidade

- Vulnerabilidades de segurança

- Oportunidades de otimização

- Violações de boas práticas

Formato da resposta:

1. Resumo dos principais problemas

2. Lista detalhada de problemas organizados por severidade

- Para cada problema: descrição, localização, severidade, sugestão

3. Sugestões gerais de melhoria

```

4. \*\*Formatação da Resposta\*\*:

- Nó "Function" para formatar em Markdown

```javascript

// Formatar resposta

const response = items[0].json.response;

items[0].json.formattedResponse = `

## Análise de Código

${response}

---

\*Esta análise foi gerada por um agente de IA local. Revise as sugestões antes de aplicá-las.\*

`;

return items;

```

5. \*\*Resposta ao Usuário\*\*:

- Nó "Telegram" ou "Webhook Response"

#### \*\*7.4.2 Agente para Documentação de Projetos\*\*

\*\*Objetivo\*\*: Gerar documentação completa para projetos

\*\*Fluxo de Trabalho\*\*:

1. \*\*Trigger\*\*: Webhook com caminho do projeto

2. \*\*Coletar Arquivos\*\*:

- Nó "List Directory" para listar arquivos

- Nó "Read Binary File" para ler arquivos relevantes

3. \*\*Processar com AI Agent\*\*:

- System Message:

```

Você é um especialista em documentação de software.

Sua tarefa é criar uma documentação completa para o projeto.

Estrutura da documentação:

- Visão Geral do Projeto

- Como Configurar

- Arquitetura

- Principais Componentes

- Como Contribuir

- Perguntas Frequentes

Use o seguinte código como referência:

{code}

```

4. \*\*Gerar Documentação\*\*:

- Nó "Write Binary File" para salvar como README.md

- Nó "HTTP Request" para criar repositório no GitHub (opcional)

5. \*\*Notificar Usuário\*\*:

- Nó "Telegram" com link para documentação

---

## \*\*8. INTERFACE E USABILIDADE\*\*

### \*\*8.1 Criação de Interfaces de Chat Locais\*\*

#### \*\*8.1.1 Interface com Gradio (Recomendado para Iniciantes)\*\*

\*\*Passo 1: Instalar Gradio\*\*

```bash

pip install gradio

```

\*\*Passo 2: Criar Aplicativo Básico\*\*

Crie um arquivo `chat\_interface.py`:

```python

import gradio as gr

import requests

def chat(message, history):

"""Envia mensagem para o agente e retorna resposta"""

response = requests.post(

"http://localhost:5678/webhook/chat",

json={"message": message}

)

return response.json()["response"]

# Configurar interface

interface = gr.ChatInterface(

fn=chat,

chatbot=gr.Chatbot(height=500),

textbox=gr.Textbox(placeholder="Digite sua pergunta...", container=False, scale=7),

title="Meu Agente de IA Local",

description="Um assistente de programação local e gratuito",

theme="soft",

examples=[

"Como criar um componente de botão em React?",

"Explique o que é programação assíncrona em JavaScript",

"Crie um algoritmo para ordenar um array em Python"

],

cache\_examples=False,

retry\_btn=None,

undo\_btn="↩️ Desfazer",

clear\_btn="🗑️ Limpar"

)

if \_\_name\_\_ == "\_\_main\_\_":

interface.launch(server\_port=7860, share=False)

```

\*\*Passo 3: Executar a Interface\*\*

```bash

python chat\_interface.py

```

\*\*Passo 4: Acessar no Navegador\*\*

- Abra http://localhost:7860

\*\*Personalizações Úteis\*\*:

- \*\*Alterar tema\*\*: Substitua `theme="soft"` por outro tema (ex: "default", "monochrome")

- \*\*Adicionar logo\*\*: Adicione `css="footer {visibility: hidden} .logo {background-image: url('logo.png')}"` ao launch()

- \*\*Salvar histórico\*\*: Adicione persistência com `gr.State()` e armazenamento local

#### \*\*8.1.2 Interface com Streamlit (Alternativa ao Gradio)\*\*

\*\*Passo 1: Instalar Streamlit\*\*

```bash

pip install streamlit

```

\*\*Passo 2: Criar Aplicativo\*\*

Crie um arquivo `streamlit\_app.py`:

```python

import streamlit as st

import requests

# Configurar página

st.set\_page\_config(

page\_title="Meu Agente de IA Local",

page\_icon="🤖",

layout="wide"

)

# Estilos CSS personalizados

st.markdown("""

<style>

.stApp {

max-width: 1200px;

margin: 0 auto;

}

.chat-message {

padding: 1.5rem;

border-radius: 0.5rem;

margin-bottom: 1rem;

display: flex;

}

.chat-message.user {

background-color: #2b313e;

}

.chat-message.bot {

background-color: #475063;

}

.chat-message .avatar {

width: 15%;

}

.chat-message .message {

width: 85%;

padding: 0 1.5rem;

}

</style>

""", unsafe\_allow\_html=True)

# Título

st.title("💬 Meu Agente de IA Local")

# Inicializar histórico

if "messages" not in st.session\_state:

st.session\_state.messages = []

# Exibir histórico

for message in st.session\_state.messages:

with st.chat\_message(message["role"]):

st.markdown(message["content"])

# Campo de entrada

if prompt := st.chat\_input("Digite sua pergunta..."):

# Adicionar mensagem do usuário ao histórico

st.session\_state.messages.append({"role": "user", "content": prompt})

with st.chat\_message("user"):

st.markdown(prompt)

# Enviar para o agente

with st.chat\_message("assistant"):

message\_placeholder = st.empty()

response = requests.post(

"http://localhost:5678/webhook/chat",

json={"message": prompt}

)

full\_response = response.json()["response"]

message\_placeholder.markdown(full\_response)

# Adicionar resposta ao histórico

st.session\_state.messages.append({"role": "assistant", "content": full\_response})

```

\*\*Passo 3: Executar a Aplicação\*\*

```bash

streamlit run streamlit\_app.py

```

\*\*Vantagens do Streamlit\*\*:

- Mais flexibilidade para personalização

- Melhor integração com componentes Python

- Interface mais moderna e profissional

- Suporte a gráficos e visualizações

#### \*\*8.1.3 Interface com Electron (Para Aplicativo Desktop)\*\*

\*\*Passo 1: Configurar Projeto Electron\*\*

1. Crie uma pasta para o aplicativo:

```bash

mkdir meu-agente-desktop

cd meu-agente-desktop

npm init -y

```

2. Instale dependências:

```bash

npm install electron isomorphic-fetch

```

\*\*Passo 2: Criar Interface Web\*\*

Crie um arquivo `index.html`:

```html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Meu Agente de IA Local</title>

<style>

\* { margin: 0; padding: 0; box-sizing: border-box; }

body {

font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;

background-color: #1e1e1e;

color: #ffffff;

height: 100vh;

display: flex;

flex-direction: column;

}

.header {

padding: 15px;

background-color: #252526;

border-bottom: 1px solid #3c3c3c;

}

.header h1 {

color: #0078d7;

}

.chat-container {

flex: 1;

overflow-y: auto;

padding: 15px;

display: flex;

flex-direction: column;

gap: 15px;

}

.message {

max-width: 80%;

padding: 12px 15px;

border-radius: 18px;

line-height: 1.5;

}

.user {

align-self: flex-end;

background-color: #0078d7;

color: white;

border-bottom-right-radius: 5px;

}

.bot {

align-self: flex-start;

background-color: #2d2d2d;

color: #ffffff;

}

.input-container {

padding: 15px;

background-color: #252526;

border-top: 1px solid #3c3c3c;

display: flex;

gap: 10px;

}

#message-input {

flex: 1;

padding: 12px 15px;

border: none;

border-radius: 20px;

background-color: #3c3c3c;

color: white;

font-size: 16px;

}

#send-button {

background-color: #0078d7;

color: white;

border: none;

border-radius: 20px;

padding: 0 20px;

cursor: pointer;

font-weight: bold;

}

</style>

</head>

<body>

<div class="header">

<h1>Meu Agente de IA Local</h1>

</div>

<div class="chat-container" id="chat-container"></div>

<div class="input-container">

<input type="text" id="message-input" placeholder="Digite sua pergunta...">

<button id="send-button">Enviar</button>

</div>

<script>

const chatContainer = document.getElementById('chat-container');

const messageInput = document.getElementById('message-input');

const sendButton = document.getElementById('send-button');

// Função para adicionar mensagem ao chat

function addMessage(content, isUser) {

const messageDiv = document.createElement('div');

messageDiv.classList.add('message');

messageDiv.classList.add(isUser ? 'user' : 'bot');

messageDiv.textContent = content;

chatContainer.appendChild(messageDiv);

chatContainer.scrollTop = chatContainer.scrollHeight;

}

// Enviar mensagem

async function sendMessage() {

const message = messageInput.value.trim();

if (!message) return;

// Mostrar mensagem do usuário

addMessage(message, true);

messageInput.value = '';

// Enviar para o agente

try {

const response = await fetch('http://localhost:5678/webhook/chat', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ message })

});

const data = await response.json();

addMessage(data.response, false);

} catch (error) {

console.error('Erro:', error);

addMessage('Desculpe, ocorreu um erro ao processar sua solicitação.', false);

}

}

// Event listeners

sendButton.addEventListener('click', sendMessage);

messageInput.addEventListener('keypress', (e) => {

if (e.key === 'Enter') sendMessage();

});

</script>

</body>

</html>

```

\*\*Passo 3: Criar Arquivo Principal do Electron\*\*

Crie um arquivo `main.js`:

```javascript

const { app, BrowserWindow } = require('electron')

const path = require('path')

function createWindow() {

const win = new BrowserWindow({

width: 800,

height: 600,

webPreferences: {

nodeIntegration: true

},

icon: path.join(\_\_dirname, 'icon.png') // Adicione um ícone se desejar

})

win.loadFile('index.html')

}

app.whenReady().then(createWindow)

app.on('window-all-closed', () => {

if (process.platform !== 'darwin') {

app.quit()

}

})

app.on('activate', () => {

if (BrowserWindow.getAllWindows().length === 0) {

createWindow()

}

})

```

\*\*Passo 4: Atualizar package.json\*\*

```json

{

"name": "meu-agente-desktop",

"version": "1.0.0",

"description": "Agente de IA local como aplicativo desktop",

"main": "main.js",

"scripts": {

"start": "electron ."

},

"author": "",

"license": "ISC",

"dependencies": {

"electron": "^28.0.0"

}

}

```

\*\*Passo 5: Executar o Aplicativo\*\*

```bash

npm start

```

\*\*Vantagens da Interface Electron\*\*:

- Parece um aplicativo nativo

- Funciona offline sem navegador

- Pode ser distribuído como executável

- Acesso a APIs do sistema operacional

### \*\*8.2 Processamento de Entradas Multimodais\*\*

#### \*\*8.2.1 Processamento de Imagens\*\*

\*\*Passo 1: Configurar o Ollama com Modelo de Visão\*\*

1. Baixe um modelo com suporte a visão:

```bash

ollama pull llava

```

2. Teste o modelo:

```bash

ollama run llava "Descreva esta imagem" -i imagem.jpg

```

\*\*Passo 2: Integrar com n8n\*\*

1. Adicione um nó "Webhook" com opção para enviar imagens

2. Adicione um nó "Function" para processar imagens:

```javascript

// Processar imagem

if (items[0].binary && items[0].binary.data) {

const imageData = items[0].binary.data;

// Converter para base64

const base64Image = Buffer.from(imageData.data, 'binary').toString('base64');

// Salvar para uso posterior

items[0].json.imageBase64 = base64Image;

}

return items;

```

3. Adicione um nó "AI Agent" configurado para visão:

```json

{

"model": "llava",

"systemMessage": "Você é um assistente que pode analisar imagens. Descreva o conteúdo da imagem e responda às perguntas do usuário.",

"userMessage": "Analise esta imagem: {{ $json['imageBase64'] }}\nPergunta: {{ $json['message'] }}"

}

```

\*\*Exemplo de Uso\*\*:

- Usuário envia uma imagem de código

- Agente descreve o código e sugere melhorias

- Usuário pergunta: "O que este código faz?"

- Agente responde com explicação

#### \*\*8.2.2 Processamento de Documentos\*\*

\*\*Passo 1: Configurar Extração de Texto\*\*

1. Instale dependências:

```bash

npm install pdf-parse exceljs

```

2. Crie um nó "Function" para processar documentos:

```javascript

const pdf = require('pdf-parse');

const Excel = require('exceljs');

async function extractText(items) {

if (!items[0].binary || !items[0].binary.data) {

return items;

}

const { mimeType, data } = items[0].binary.data;

let text = '';

try {

if (mimeType === 'application/pdf') {

const pdfData = await pdf(Buffer.from(data, 'binary'));

text = pdfData.text;

}

else if (mimeType === 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet') {

const workbook = new Excel.Workbook();

await workbook.xlsx.load(Buffer.from(data, 'binary'));

text = workbook.worksheets.map(ws =>

ws.getSheetValues().map(row => row.filter(cell => cell).join('\t')).join('\n')

).join('\n\n');

}

else if (mimeType === 'application/msword' ||

mimeType === 'application/vnd.openxmlformats-officedocument.wordprocessingml.document') {

// Para documentos Word, você precisaria de uma biblioteca adicional

text = "Processamento de documentos Word não implementado";

}

else {

text = "Tipo de arquivo não suportado";

}

} catch (error) {

console.error('Erro ao processar documento:', error);

text = "Erro ao processar o documento";

}

items[0].json.documentText = text;

return items;

}

return extractText(items);

```

\*\*Passo 2: Integrar com RAG\*\*

1. Atualize o nó RAG para usar o texto extraído:

```javascript

// Adicionar documento ao ChromaDB

if (items[0].json.documentText) {

const collection = await client.getOrCreateCollection({ name: "user\_documents" });

await