

# Projeto - Sistema de IA Local com Ollama

---

## Visão Geral

Este projeto propõe a criação de um serviço de inteligência artificial local utilizando a plataforma Ollama no Windows 11 Pro. O objetivo principal é propor um hardware adequado para rodar modelos de linguagem grandes (LLMs) diretamente na máquina, oferecendo recursos como análise de código e processamento de linguagem natural via rede. Isso permite que outros computadores na rede acessem esses modelos sem precisar de hardware dedicado ou conexão constante com a nuvem, o que é ótimo para privacidade, latência e custo.

---

## 1. Análise do Hardware e Viabilidade

**Veremos a seguir uma análise de um hardware recomendado, destacando sua adequação para o Ollama:**

- **CPU Ryzen 7 5700X:**

- Considerada uma "CPU robusta para processamento híbrido CPU/GPU e múltiplas requisições". Isso é crucial porque, embora os LLMs dependam fortemente da GPU, a CPU ainda desempenha um papel importante na coordenação, pré-processamento e pós-processamento de dados, garantindo que o sistema possa lidar com várias solicitações simultaneamente sem gargalos.

- **RTX 5060 Ti (16GB VRAM):**

- Este é o coração do sistema para IA. A generosa quantidade de 16GB de VRAM "permite executar modelos até 70B com quantização adequada" (não recomendado ocupar 100% da VRAM com a carga de modelo de IA, devendo restar VRAM livre para dados de contexto e para outros processos que requeiram uso de GPU). A VRAM é a memória mais crítica para rodar LLMs, pois é onde os pesos do modelo são carregados. Quanto mais VRAM, maiores e mais complexos os modelos que podem ser executados, ou mais modelos podem ser carregados simultaneamente.

- **32GB RAM:**

- Descrita como "suficiente para modelos grandes e múltiplas conexões simultâneas". A RAM principal do sistema complementa a VRAM, sendo essencial para o sistema operacional, outros aplicativos e para o carregamento e descarregamento de partes dos modelos que não cabem inteiramente na VRAM, bem como para o manuseio de contextos longos.

- **SSD 1TB:**

- Este espaço de armazenamento é "adequado para múltiplos modelos (considere reservar 200-300GB para modelos)". Os modelos de IA,

especialmente os grandes, podem ocupar muitos gigabytes, então ter um SSD rápido e espaçoso garante que o carregamento e a troca de modelos sejam ágeis, melhorando a experiência geral.

A combinação desses componentes garante que o sistema não apenas consiga rodar os modelos, mas também os faça de forma eficiente e responsiva, mesmo sob demanda de rede.

---

## 2. Modelos Recomendados para o Projeto

O documento destaca uma escolha principal de modelo e sugere outros complementares, otimizando o sistema para tarefas específicas:

➤ **Modelo Recomendado: qwen2.5-coder:3b**

- ✓ O comando para puxá-lo é `ollama pull qwen2.5-coder:3b` .
- ✓ A escolha é justificada por ser "Especializado em análise e geração de código", o que o torna ideal para o objetivo de um assistente de programação.
- ✓ Mesmo com "3.1 bilhões de parâmetros", ele "oferece excelente capacidade de raciocínio".
- ✓ Importante notar que ele "cabe confortavelmente em 16GB VRAM", o que é perfeito para a GPU especificada.
- ✓ Ele também suporta "contextos longos (Máximo 32768 tokens)", permitindo que o modelo "se lembre" de mais informações durante uma conversa ou análise de código extensa.

• **Modelos Complementares:**

- **ollama pull llama:7b:** Um modelo multimodal, provavelmente para tarefas que envolvem visão (análise de imagens, diagramas, etc.), o que adiciona versatilidade ao sistema.
- **ollama pull llama3.2:3b-instruct-fp16:** Um modelo de linguagem geral, útil para instruções e conversação, oferecendo uma alternativa ao modelo de codificação.
- **ollama pull phi3:medium-4k:** Mais um modelo de linguagem otimizado para instruções e eficiência, complementando as capacidades do sistema.

Essa seleção estratégica de modelos permite que o servidor Ollama atenda a uma variedade de necessidades de IA, com foco especial em desenvolvimento de software.

---

## 3. Parte 1: Preparação do Sistema

A fase de preparação é fundamental para garantir que o hardware e o sistema operacional estejam configurados para o máximo desempenho e compatibilidade com o Ollama e seus modelos.

### 3.1. Ajuste da BIOS

Detalhes de configurações específicas da BIOS que otimizam o desempenho da CPU e da GPU:

- **Acesso à BIOS: Geralmente via teclas DEL, F2 ou F12 durante a inicialização.**
  - **Advanced Mode:** Buscar o modo avançado para acessar todas as opções.
  - **Configurações de CPU (Advanced → CPU Configuration):**
    - **SMT Mode (ou Simultaneous Multi-Threading):** Ativar (Enabled). Isso permite que cada núcleo da CPU execute múltiplos threads, aumentando o paralelismo e a eficiência do processamento.
    - **Precision Boost Overdrive (PBO):** Ativar (Enabled). PBO é uma tecnologia da AMD que permite que a CPU aumente suas frequências além dos limites padrão de forma dinâmica, resultando em melhor desempenho em cargas de trabalho que se beneficiam de maior clock.
  - **Configurações de Memória (Advanced → Memory Configuration):**
    - **Perfil DOCP ou XMP:** Ativar (Profile 1). Estes perfis são padrões de Intel (XMP) e AMD (DOCP) que permitem que a RAM opere em sua velocidade nominal de fábrica, que geralmente é maior do que a velocidade padrão definida pela placa-mãe. Essencial para o desempenho de LLMs que são sensíveis à largura de banda da memória.
  - **Configuração Resizable BAR (ou Smart Access Memory):** Ativar (Enabled) em Advanced → PCI Subsystem Settings. Essa tecnologia permite que a CPU acesse a VRAM da GPU de forma mais eficiente, geralmente resultando em ganhos de desempenho em certas aplicações, incluindo cargas de trabalho de IA.
  - **Salvar e Sair:** Pressionar F10 para salvar as alterações e reiniciar o sistema.

Esses ajustes na BIOS são cruciais para extrair o máximo de performance dos componentes.

### 3.2. Otimizações do Windows 11

Além da BIOS, o Windows 11 também precisa de ajustes para um desempenho ideal:

- **Opções de Energia (Painel de Controle → Hardware e Sons → Opções de Energia):**
  - ✓ Selecionar "Alto Desempenho". Se não estiver visível, clicar em "Mostrar planos adicionais". O plano de alto desempenho evita que o sistema reduza a frequência da CPU ou GPU para economizar energia, garantindo que os componentes estejam sempre prontos para o máximo esforço.
- **Configuração da Memória Virtual (sysdm.cpl → Propriedades do Sistema → Avançado → Desempenho → Configurações → Avançado → Memória virtual → Alterar):**
  - Desmarcar "Gerenciar automaticamente".
  - Selecionar a unidade C: e marcar "Tamanho personalizado".

- Definir "Tamanho inicial (MB)" e "Tamanho máximo (MB)" para 49152 (equivalente a 48GB). Isso aloca uma grande quantidade de memória virtual (arquivo de paginação) no SSD, o que é útil para evitar erros de "memória insuficiente" ao carregar modelos muito grandes ou múltiplos modelos, embora o desempenho seja inferior ao uso direto da RAM.
- **Desativar Compactação de Memória via PowerShell:**
  - Abrir o PowerShell como administrador.
  - Executar Disable-MMAgent -mc. Isso desativa a compactação de memória, que pode consumir ciclos de CPU para economizar RAM. Para um sistema com 32GB de RAM e grande memória virtual, a compactação de memória pode ser desnecessária e até prejudicial ao desempenho.
- **Reiniciar o computador:** Para aplicar todas as otimizações.

Essas otimizações garantem que o sistema operacional não interfira no desempenho máximo dos modelos de IA.

---

## 4. Parte 2: Instalação de Drivers e Software Básico

Com o sistema base otimizado, o próximo passo é instalar os drivers e softwares essenciais para o funcionamento da GPU com IA.

### 4.1. Drivers NVIDIA e CUDA Toolkit

A NVIDIA fornece as ferramentas necessárias para que seus GPUs sejam usados em computação paralela, o que é fundamental para LLMs.

- **Instalar Driver NVIDIA Studio:**
  - Acessar o site de download da NVIDIA.
  - Selecionar: GeForce > GeForce RTX 50 Series > NVIDIA GeForce RTX 5060 TI > Windows 11 > Português (Brazil).
  - **Crucial:** Selecionar "**Studio Driver**", não "Game Ready". Drivers Studio são otimizados para estabilidade e desempenho em aplicações criativas e de computação, como IA.
  - Executar o instalador, escolher "Instalação Personalizada" e marcar "Realizar uma instalação limpa" para evitar conflitos com versões anteriores.
- **Instalar CUDA Toolkit 12.3:**
  - Acessar <https://developer.nvidia.com/cuda-downloads>.
  - Selecionar Windows → x86\_64 → 11 → exe (local).
  - Executar o instalador com as opções padrão. O CUDA Toolkit é uma plataforma de desenvolvimento paralela e uma API para GPUs NVIDIA, essencial para que frameworks de IA se comuniquem com a GPU.

- **Instalar cuDNN:**

- Acessar <https://developer.nvidia.com/cudnn>.
- Criar uma conta NVIDIA Developer (gratuita) e fazer login.
- Baixar a versão compatível com CUDA 12.3. cuDNN (CUDA Deep Neural Network library) é uma biblioteca de primitivas de GPU aceleradas para deep learning, otimizando o desempenho de operações comuns em redes neurais.

**\* Extrair o arquivo e copiar seu conteúdo para as pastas do CUDA Toolkit:**

- ✓ Arquivos da pasta "bin" para C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.3\bin
- ✓ Arquivos da pasta "include" para `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.3\include`
- ✓ Arquivos da pasta "lib" para `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.3\lib\x64`

- **Verificar a Instalação:**

- Abrir o PowerShell.
- Digitar nvidia-smi para verificar informações da GPU e versão do driver.
- Digitar nvcc --version para verificar a versão do CUDA Toolkit instalada.

A correta instalação desses componentes é o alicerce para que o Ollama possa utilizar plenamente a capacidade da GPU.

## 4.2. Python e Ambiente Virtual

Python é a linguagem dominante em IA, e ambientes virtuais são essenciais para gerenciar dependências.

- **Baixar e Instalar Python 3.10:**

- Acessar <https://www.python.org/downloads/release/python-31011/>.
- Baixar o instalador Windows (64-bit).
- **Importante:** Marcar "Add Python 3.10 to PATH" durante a instalação para facilitar o acesso via linha de comando.

- **Criar Ambiente Virtual para o CodeLlama:**

- Abrir o Prompt de Comando.
- Navegar até a pasta do projeto: cd C:\Users\SeuUsuario\Documents
- Criar uma nova pasta para o projeto: mkdir CodeLlama
- Entrar na pasta: cd CodeLlama
- Criar o ambiente virtual: python -m venv venv

- Ativar o ambiente virtual: venv\Scripts\activate (o prompt deve mostrar "(venv)" no início).

- **Instalar Bibliotecas Necessárias:**

**\* Com o ambiente virtual ativado:**

- ✓ **pip install torch torchvision torchaudio --index-url <https://download.pytorch.org/whl/cu121>:** Instala o PyTorch, uma das principais bibliotecas de deep learning, com otimizações para CUDA 12.1.
- ✓ **pip install transformers accelerate sentencepiece protobuf:** Instala bibliotecas essenciais do Hugging Face para trabalhar com modelos Transformer, aceleração de inferência e tokenização.
- ✓ **pip install huggingface\_hub:** Para interagir com o Hugging Face Hub, onde muitos modelos de LLM são hospedados.

A configuração de um ambiente Python isolado garante que as dependências do projeto não entrem em conflito com outras instalações Python no sistema.

---

## 5. Parte 3: Configuração do Ollama como Serviço de Rede no Windows 11 Pro

Esta é a parte central do projeto, transformando o Ollama em um serviço robusto e acessível.

### 5.1. Instalação e Configuração Inicial do Ollama

- **Baixar e Instalar Ollama:**

- Acessar <https://ollama.com/download>.
- Baixar OllamaSetup.exe e executá-lo como administrador para concluir a instalação.

### 5.2. Configuração de Variáveis de Ambiente Permanentes

**Essas variáveis controlam o comportamento do Ollama e o tornam acessível pela rede. O documento oferece dois métodos:**

- **Método 1: Via PowerShell (Recomendado)**

Executar o PowerShell como administrador e rodar os seguintes comandos:

```
[Environment]::SetEnvironmentVariable("OLLAMA_HOST", "0.0.0.0:11434", "Machine")
[Environment]::SetEnvironmentVariable("OLLAMA_MODELS", "C:\OllamaModels", "Machine")
[Environment]::SetEnvironmentVariable("OLLAMA_KEEP_ALIVE", "60m", "Machine")
[Environment]::SetEnvironmentVariable("OLLAMA_NUM_PARALLEL", "4", "Machine")
[Environment]::SetEnvironmentVariable("OLLAMA_MAX_LOADED_MODELS", "2", "Machine")
[Environment]::SetEnvironmentVariable("OLLAMA_ORIGINS", "*", "Machine")
[Environment]::SetEnvironmentVariable("CUDA_VISIBLE_DEVICES", "0", "Machine")
[Environment]::SetEnvironmentVariable("OLLAMA_DEBUG", "0", "Machine")
[Environment]::SetEnvironmentVariable("OLLAMA_FLASH_ATTENTION", "1", "Machine")
```

### **\*\*Explicação das Variáveis:\*\***

- ✓ `OLLAMA\_HOST="0.0.0.0:11434"` : Define que o Ollama escutará em todas as interfaces de rede no IP `0.0.0.0` na porta `11434`, tornando-o acessível de qualquer máquina na rede local.
- ✓ `OLLAMA\_MODELS="C:\OllamaModels"` : Especifica o diretório onde os modelos do Ollama serão armazenados.
- ✓ `OLLAMA\_KEEP\_ALIVE="60m"` : Mantém os modelos carregados na VRAM por 60 minutos após a última requisição, reduzindo o tempo de carregamento para requisições subsequentes.
- ✓ `OLLAMA\_NUM\_PARALLEL="4"` : Permite 4 inferências paralelas, otimizando o uso da GPU e CPU para múltiplas requisições simultâneas.
- ✓ `OLLAMA\_MAX\_LOADED\_MODELS="2"` : Permite que até 2 modelos sejam mantidos na VRAM ao mesmo tempo.
- ✓ `OLLAMA\_ORIGINS="\*"` : Permite acesso CORS de qualquer origem, facilitando a integração com aplicações web.
- ✓ `CUDA\_VISIBLE\_DEVICES="0"` : Garante que o Ollama use a primeira GPU disponível.
- ✓ `OLLAMA\_DEBUG="0"` : Desabilita o modo de depuração.
- ✓ `OLLAMA\_FLASH\_ATTENTION="1"` : Ativa a otimização Flash Attention, que melhora a velocidade e eficiência de modelos baseados em Transformer.

- **Método 2: Via Interface Gráfica (Alternativo)**

- Win + R → sysdm.cpl → Enter → Variáveis de Ambiente....
  - Em "Variáveis do sistema", adicionar cada uma das variáveis mencionadas acima com seus respectivos valores.

Essas variáveis são cruciais para o funcionamento do Ollama como um serviço de rede, com performance otimizada e comportamento desejado.

### **5.3. Configuração do Serviço Windows**

Para que o Ollama funcione como um serviço em segundo plano e inicie automaticamente, é necessário configurá-lo como um serviço do Windows. O documento propõe um script PowerShell para isso.

- **Criar C:\OllamaService\install-service.ps1:**

```
# Script para instalar Ollama como serviço Windows
# Execute como Administrador

param(
    [string]$ServiceName = "OllamaAIService",
    [string]$DisplayName = "Ollama AI Service",
    [string]$Description = "Serviço de IA Local Ollama para Análise de Código"
)

Write-Host "Instalando Ollama como serviço Windows..." -ForegroundColor Green
```

```

# Parar serviço existente se houver
try{
    Stop-Service -Name $ServiceName -Force -ErrorAction SilentlyContinue
    Remove-Service -Name $ServiceName -ErrorAction SilentlyContinue
} catch {
    Write-Host "Nenhum serviço anterior encontrado." -ForegroundColor Yellow
}

# Encontrar executável do Ollama
$ollamaPath = Get-Command ollama -ErrorAction SilentlyContinue
if (-not $ollamaPath) {
    $ollamaPath = "${env:LOCALAPPDATA}\Programs\Ollama\ollama.exe"
    if (-not (Test-Path $ollamaPath)) {
        Write-Host "ERRO: Ollama não encontrado!" -ForegroundColor Red
        exit 1
    }
}
Write-Host "Ollama encontrado em: $($ollamaPath.Source)" -ForegroundColor Green

# Criar diretório de logs
$logDir = "C:\OllamaService\Logs"
if (!(Test-Path $logDir)) {
    New-Item -ItemType Directory -Path $logDir -Force
}

# Usar NSSM para criar o serviço
$nssmPath = "C:\OllamaService\nssm.exe"

# Baixar NSSM se não existir
if (!(Test-Path $nssmPath)) {
    Write-Host "Baixando NSSM..." -ForegroundColor Yellow
    $nssmUrl = "https://nssm.cc/release/nssm-2.24.zip"
    $nssmZip = "C:\OllamaService\nssm.zip"

    New-Item -ItemType Directory -Path "C:\OllamaService" -Force
    Invoke-WebRequest -Uri $nssmUrl -OutFile $nssmZip
    Expand-Archive -Path $nssmZip -DestinationPath "C:\OllamaService\temp"
    Copy-Item "C:\OllamaService\temp\nssm-2.24\win64\nssm.exe" -Destination $nssmPath
    Remove-Item "C:\OllamaService\temp" -Recurse -Force
    Remove-Item $nssmZip -Force
}

# Instalar serviço usando NSSM
& $nssmPath install $ServiceName $ollamaPath.Source "serve"
& $nssmPath set $ServiceName DisplayName $DisplayName
& $nssmPath set $ServiceName Description $Description
& $nssmPath set $ServiceName Start SERVICE_AUTO_START
& $nssmPath set $ServiceName AppStdout "$logDir\ollama-stdout.log"
& $nssmPath set $ServiceName AppStderr "$logDir\ollama-stderr.log"
& $nssmPath set $ServiceName AppRotateFiles 1
& $nssmPath set $ServiceName AppRotateOnline 1
& $nssmPath set $ServiceName AppRotateSeconds 86400
& $nssmPath set $ServiceName AppRotateBytes 10485760

# Configurar variáveis de ambiente para o serviço (essencial para que o NSSM passe as variáveis corretas para o serviço)
& $nssmPath set $ServiceName AppEnvironmentExtra "OLLAMA_HOST=0.0.0.0:11434"
"OLLAMA_MODELS=C:\OllamaModels" "OLLAMA_KEEP_ALIVE=60m" "OLLAMA_NUM_PARALLEL=4"
"OLLAMA_ORIGIN=*" "CUDA_VISIBLE_DEVICES=0"

Write-Host "Serviço instalado com sucesso!" -ForegroundColor Green
Write-Host "Iniciando serviço..." -ForegroundColor Yellow

Start-Service -Name $ServiceName
Get-Service -Name $ServiceName

Write-Host "`nServiço configurado e iniciado!" -ForegroundColor Green
Write-Host "Logs disponíveis em: $logDir" -ForegroundColor Cyan

```

- **Função do Script:**

- Este script utiliza o NSSM (Non-Sucking Service Manager), uma ferramenta que permite encapsular qualquer aplicação como um serviço do Windows.
- Ele primeiro verifica e remove qualquer serviço Ollama existente.
- Localiza o executável do Ollama.
- Cria um diretório para logs.
- Baixa o NSSM se ele não estiver presente e o extrai.
- Instala o Ollama como um serviço (OllamaAIService), define seu nome de exibição e descrição.
- Configura o serviço para iniciar automaticamente com o sistema (SERVICE\_AUTO\_START).
- Redireciona a saída padrão e de erro do Ollama para arquivos de log (ollama-stdout.log, ollama-stderr.log) e configura a rotação de logs.
- **Importante:** Ele define novamente as variáveis de ambiente específicas para o serviço Ollama usando AppEnvironmentExtra, garantindo que o serviço funcione com as configurações desejadas (host, modelos, keep-alive, etc.) independentemente das variáveis de ambiente do sistema.
- Finalmente, inicia o serviço e exibe seu status.

#### 5.4. Configuração de Firewall do Windows

Para permitir que outras máquinas na rede acessem o serviço Ollama, o firewall precisa ser configurado.

- **Criar C:\OllamaService\configure-firewall.ps1:**

```
# Configurar Firewall para Ollama
# Execute como Administrador

Write-Host "Configurando Firewall do Windows para Ollama..." -ForegroundColor Green

# Remover regras existentes
Remove-NetFirewallRule -DisplayName "Ollama AI Service*" -ErrorAction SilentlyContinue

# Criar regras de firewall
New-NetFirewallRule -DisplayName "Ollama AI Service - Inbound" -Direction Inbound -Protocol TCP -LocalPort 11434 -Action Allow -Profile Domain,Private,Public
New-NetFirewallRule -DisplayName "Ollama AI Service - Outbound" -Direction Outbound -Protocol TCP -LocalPort 11434 -Action Allow -Profile Domain,Private,Public

# Regras adicionais para CUDA/GPU
New-NetFirewallRule -DisplayName "Ollama AI Service - NVIDIA" -Direction Inbound -Program "C:\Program Files\NVIDIA Corporation\NVSMI\nvidia-smi.exe" -Action Allow -Profile Domain,Private,Public

Write-Host "Regras de firewall configuradas:" -ForegroundColor Green
Get-NetFirewallRule -DisplayName "Ollama AI Service*" | Select-Object DisplayName, Direction, Action, Enabled

Write-Host "`nFirewall configurado com sucesso!" -ForegroundColor Green
Write-Host "Porta 11434 liberada para acesso de rede" -ForegroundColor Cyan
```

- **Função do Script:**

- Remove quaisquer regras de firewall pré-existentes para o "Ollama AI Service" para evitar duplicação.
- Cria uma regra de **entrada (Inbound)** para permitir conexões TCP na porta 11434 (porta padrão do Ollama) de qualquer perfil de rede (domínio, privado, público), o que é essencial para que outras máquinas consigam se conectar.
- Cria uma regra de **saída (Outbound)** para a mesma porta, embora a regra de entrada seja a mais crítica para acessibilidade.
- Adiciona uma regra específica para o executável nvidia-smi.exe, garantindo que as ferramentas da NVIDIA para monitoramento e gerenciamento da GPU possam operar sem bloqueios de firewall.
- Exibe as regras criadas e confirma que a porta 11434 foi liberada.

## 5.5. Modelfile Personalizado para Análise de Código

Um dos pontos mais interessantes é a criação de um Modelfile customizado, que define como o modelo qwen2.5-coder:3b se comportará.

- **Criar C:\OllamaModels\Modelfiles\CodeAnalyzer.modelfile:**

```
FROM qwen2.5-coder:3b

# Parâmetros de temperatura e sampling otimizados para programação
PARAMETER temperature 0.1
PARAMETER top_p 0.9
PARAMETER top_k 40
PARAMETER repeat_penalty 1.1
PARAMETER num_ctx 4096
PARAMETER num_predict 2048

# Template de sistema especializado para programação
TEMPLATE """{{- if .Suffix }}<|fim_prefix|>{{ .Prompt }}<|fim_suffix|>{{ .Suffix }}<|fim_middle|>
{{- else if .Messages }}
{{- if or .System .Tools }}<|im_start|>system
{{- if .System }}
{{ .System }}
{{- end }}
{{- if .Tools }}

# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools>:
<tools>
{{- range .Tools }}
{"type": "function", "function": "{{ .Function }}"
{{- end }}
</tools>

For each function call, return a json object with function name and arguments within <tool_call></tool_call> with NO
other text. Do not include any backticks or `` ` json.
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>
{{- end }}<|im_end|>
{{ end }}
{{- range $i, $_ := .Messages }}
{{- $last := eq (len (slice $.Messages $i)) 1 -}}
{{- if eq .Role "user" }}<|im_start|>user
```

```

{{ .Content }}<|im_end|>
{{ else if eq .Role "assistant" }}<|im_start|>assistant
{{ if .Content }}{{ .Content }}
{{- else if .ToolCalls }}<tool_call>
{{ range .ToolCalls }}{"name": "{{ .Function.Name }}", "arguments": {{ .Function.Arguments }}}
{{ end }}</tool_call>
{{- end }}{{ if not $last }}<|im_end|>
{{ end }}
{{- else if eq .Role "tool" }}<|im_start|>user
<tool_response>
{{ .Content }}
</tool_response><|im_end|>
{{ end }}
{{- if and (ne .Role "assistant") $last }}<|im_start|>assistant
{{ end }}
{{- end }}
{{- else }}
{{- if .System }}<|im_start|>system
{{ .System }}<|im_end|>
{{ end }}{{ if .Prompt }}<|im_start|>user
{{ .Prompt }}<|im_end|>
{{ end }}<|im_start|>assistant
{{ end }}{{ if .Response }}{{ if .Response }}<|im_end|>{{ end }}"""

```

#### **# System prompt otimizado para desenvolvimento de software**

SYSTEM """"Você é um assistente de programação especializado, focado em fornecer código de alta qualidade, explicações claras e soluções eficientes. Suas diretrizes são:

##### **DESENVOLVIMENTO DE CÓDIGO:**

- Escreva código limpo, bem estruturado e seguindo boas práticas
- Use nomes de variáveis e funções descritivos
- Inclua comentários relevantes e documentação
- Priorize legibilidade e manutenibilidade
- Implemente tratamento de erros adequado

##### **LINGUAGENS E TECNOLOGIAS:**

- Python, JavaScript, TypeScript, Java, C++, C#, Go, Rust, PHP
- Frameworks web: React, Vue, Angular, FastAPI, Django, Flask
- Banco de dados: SQL, MongoDB, PostgreSQL, MySQL
- DevOps: Docker, Kubernetes, CI/CD
- Ferramentas: Git, Linux, APIs REST

##### **METODOLOGIA DE RESPOSTA:**

1. Analise o problema completamente
2. Forneça uma solução direta e funcional
3. Explique o código quando necessário
4. Sugira melhorias ou alternativas
5. Inclua exemplos de uso quando apropriado

##### **FORMATO DE CÓDIGO:**

- Use blocos de código com sintaxe highlighting
- Separe código principal de exemplos
- Inclua imports/dependências necessárias
- Forneça instruções de execução quando relevante

##### **DEBUGGING E OTIMIZAÇÃO:**

- Identifique possíveis problemas no código
- Sugira otimizações de performance
- Explique complexidade algorítmica quando relevante
- Ofereça soluções para casos edge

Seja prestativo, preciso, eficiente e mantenha foco em soluções práticas que funcionem.""""

```

# Configurações de stop tokens para melhor controle
PARAMETER stop "<|im_start|>"
PARAMETER stop "<|im_end|>"
PARAMETER stop "<|endoftext|>"

```

- **Detalhes do Modelfile:**

- **FROM qwen2.5-coder:3b:** Indica que este Modelfile é baseado no modelo Qwen 2.5 Coder de 3 bilhões de parâmetros.

- **Parâmetros Otimizados para Programação:**

- **temperature 0.1:** Controla a aleatoriedade da saída. Um valor baixo (0.1) faz com que o modelo seja mais determinístico e focado, ideal para tarefas de codificação onde a precisão é crucial.
- **top\_p 0.9, top\_k 40:** Parâmetros de amostragem que influenciam a diversidade e relevância das palavras escolhidas pelo modelo.
- **repeat\_penalty 1.1:** Desencoraja o modelo a repetir frases ou conceitos, promovendo respostas mais variadas e concisas.
- **num\_ctx 4096:** Define o tamanho da janela de contexto do modelo. No Modelfile em si, 4096 é o padrão, mas o cliente Python pode sobreescrivê-lo para 16384, indicando suporte a contextos mais longos para análise de código.
- **num\_predict 2048:** O número máximo de tokens a serem gerados em uma resposta.
- **TEMPLATE:** Define a estrutura da conversa e como o modelo deve interpretar diferentes papéis (usuário, assistente, sistema). Este template é específico para o Qwen 2.5 e inclui suporte para ferramentas (<tools>), o que é avançado e permite que o modelo chame funções externas.
- **SYSTEM Prompt Otimizado:** Esta é a parte mais "instrutiva" do Modelfile. Ela define o persona do modelo ("assistente de programação especializado") e suas diretrizes:
  - **Desenvolvimento de Código:** Foca em código limpo, bem estruturado, com boas práticas, documentação e tratamento de erros.
  - **Linguagens e Tecnologias:** Lista uma vasta gama de tecnologias que o modelo deve ser capaz de lidar (Python, JS, React, Docker, SQL, etc.).
  - **Metodologia de Resposta:** Guia o modelo para analisar problemas, fornecer soluções diretas, explicar o código, sugerir melhorias e incluir exemplos.
  - **Formato de Código:** Orienta o uso de blocos de código com highlighting, separação de exemplos e inclusão de dependências.
  - **Debugging e Otimização:** Incentiva o modelo a identificar problemas, sugerir otimizações e explicar a complexidade algorítmica.
  - O prompt finaliza com uma instrução clara de ser "**prestativo, preciso, eficiente e manter foco em soluções práticas que funcionem**".
- **Stop Tokens:** "<|im\_start|>", "<|im\_end|>", "<|endoftext|>" são tokens especiais que sinalizam ao modelo quando parar de gerar texto. Essenciais para controlar o formato da saída e evitar "alucinações" ou continuações indesejadas.

Este Modelfile personaliza o comportamento do qwen2.5-coder:3b, tornando-o uma ferramenta de análise e geração de código altamente direcionada e útil.

---

## 6. Passo-a-Passo de Implementação Completa (Fases)

Visão sintetizada dos passos anteriores em um guia prático, dividindo a implementação em fases.

### 6.1. Fase 1: Preparação do Ambiente

- **Criar Estrutura de Diretórios:**

```
New-Item -ItemType Directory -Path "C:\OllamaModels" -Force  
New-Item -ItemType Directory -Path "C:\OllamaModels\Modelfiles" -Force  
New-Item -ItemType Directory -Path "C:\OllamaService" -Force  
New-Item -ItemType Directory -Path "C:\OllamaService\Logs" -Force  
New-Item -ItemType Directory -Path "C:\OllamaService\Scripts" -Force  
New-Item -ItemType Directory -Path "C:\OllamaService\Config" -Force
```

Isso organiza os arquivos do Ollama, modelos, logs, scripts e configurações em locais dedicados.

- **Instalar Ollama:**

- Baixar e executar **OllamaSetup.exe** como administrador.
- Verificar a instalação com **ollama --version**.

- **Configurar Variáveis de Ambiente Permanentes:**

- Executar os comandos PowerShell ou usar a interface gráfica como descrito anteriormente.

### 6.2. Fase 2: Configuração do Serviço

- **Instalar Ollama como Serviço Windows:**

- Executar o script **install-service.ps1** criado na Parte 3.3.

- **Configurar Firewall:**

- Executar o script **configure-firewall.ps1** criado na Parte 3.4.

- **Verificar Serviço:**

- **Get-Service -Name "OllamaAIService":** Para checar se o serviço está rodando.
- **Test-NetConnection -ComputerName localhost -Port 11434:** Para confirmar que a porta do Ollama está acessível localmente.

### 6.3. Fase 3: Download e Configuração dos Modelos

- **Baixar Modelo Principal:**

- **ollama pull qwen2.5-coder:3b**

- **Baixar Modelos Complementares:**
  - ollama pull llava:7b
  - ollama pull llama3.2:3b-instruct-fp16
  - ollama pull phi3:medium-4k
- **Criar Modelo Personalizado:**
  - Navegar para cd C:\OllamaModels\Modelfiles.
  - **ollama create code-analyzer -f CodeAnalyzer.modelfile:** Este comando usa o Modelfile para criar um novo modelo chamado code-analyzer baseado no qwen2.5-coder:3b.
- **Testar Modelo Localmente:**
  - **ollama run code-analyzer:** Para interagir com o modelo recém-criado diretamente no servidor.

#### **6.4. Fase 4: Scripts de Gerenciamento do Serviço**

Para facilitar a administração do serviço Ollama, o documento fornece dois scripts PowerShell muito úteis.

- **Script de Monitoramento (C:\OllamaService\Scripts\monitor-service.ps1):**

```
# Monitor do Serviço Ollama
param(
    [int]$RefreshInterval = 30
)

function Show-OllamaStatus {
    Clear-Host
    Write-Host "===== -ForegroundColor Cyan
    Write-Host " OLLAMA AI SERVICE - MONITOR" -ForegroundColor Cyan
    Write-Host "===== -ForegroundColor Cyan
    Write-Host "Atualizado: $(Get-Date)" -ForegroundColor Yellow
    Write-Host ""

# Status do Serviço
Write-Host "STATUS DO SERVIÇO:" -ForegroundColor Green
$service = Get-Service -Name "OllamaAIService" -ErrorAction SilentlyContinue
if ($service) {
    Write-Host " Nome: $($service.DisplayName)" -ForegroundColor White
    Write-Host " Status: $($service.Status)" -ForegroundColor $((if($service.Status -eq 'Running'){'Green'}else{'Red'}))
    Write-Host " Tipo de Início: $($service.StartType)" -ForegroundColor White
} else {
    Write-Host " SERVIÇO NÃO ENCONTRADO!" -ForegroundColor Red
}
Write-Host ""

# Teste de Conectividade
Write-Host "CONECTIVIDADE:" -ForegroundColor Green
$connection = Test-NetConnection -ComputerName "localhost" -Port 11434 -WarningAction SilentlyContinue
Write-Host " Porta 11434: $($if($connection.TcpTestSucceeded){'ABERTA'}else{'FECHADA'})" -ForegroundColor $((if($connection.TcpTestSucceeded){'Green'}else{'Red'}))
Write-Host ""

# Modelos Carregados
Write-Host "MODELOS CARREGADOS:" -ForegroundColor Green
try {
    $response = Invoke-RestMethod -Uri "http://localhost:11434/api/ps" -Method GET -TimeoutSec 5
    if ($response -and $response.models) {
        foreach ($model in $response.models) {
```

```

        Write-Host " $($model.name) - $($model.size_vram)MB VRAM" -ForegroundColor White
    }
} else {
    Write-Host " Nenhum modelo carregado" -ForegroundColor Yellow
}
} catch {
    Write-Host " Erro ao consultar modelos: $($_.Exception.Message)" -ForegroundColor Red
}
Write-Host ""

# Uso de GPU
Write-Host "USO DE GPU:" -ForegroundColor Green
try{
    $gpuInfo = nvidia-smi --query-gpu=name,memory.used,memory.total,utilization.gpu --
format=csv,noheader,nounits 2>$null
    if ($gpuInfo) {
        $gpu = $gpuInfo.Split(',')
        Write-Host " GPU: $($gpu[0].Trim())" -ForegroundColor White
        Write-Host " Memória: $($gpu[1].Trim())MB / $($gpu[2].Trim())MB" -ForegroundColor White
        Write-Host " Utilização: $($gpu[3].Trim())%" -ForegroundColor White
    }
} catch {
    Write-Host " Informações de GPU não disponíveis" -ForegroundColor Yellow
}
Write-Host ""

# Logs Recentes
Write-Host "LOGS RECENTES (últimas 5 linhas):" -ForegroundColor Green
$logFile = "C:\OllamaService\Logs\ollama-stdout.log"
if (Test-Path $logFile) {
    Get-Content $logFile -Tail 5 | ForEach-Object {
        Write-Host " $_" -ForegroundColor Gray
    }
} else {
    Write-Host " Arquivo de log não encontrado" -ForegroundColor Yellow
}
Write-Host ""
Write-Host "Pressione Ctrl+C para sair. Próxima atualização em $RefreshInterval segundos..." -ForegroundColor Cyan
}

```

#### **# Loop de monitoramento**

```

while ($true){
    Show-OllamaStatus
    Start-Sleep -Seconds $RefreshInterval
}

```

- **Funcionalidade:** Este script fornece um monitoramento em tempo real do serviço Ollama, exibindo:

- Status do serviço ("Ollama AI Service").
- Conectividade da porta 11434.
- Modelos Ollama atualmente carregados (via API /api/ps).
- Uso de GPU (memória e utilização) via nvidia-smi.
- As últimas 5 linhas do log padrão do Ollama.

Ele atualiza automaticamente a cada RefreshInterval segundos, sendo uma ferramenta valiosa para diagnosticar o estado do servidor.

- Script de Gerenciamento do Serviço (C:\OllamaService\Scripts\manage-service.ps1):

```

# Gerenciador do Serviço Ollama
param(
    [Parameter(Mandatory=$true)]
    [ValidateSet("start", "stop", "restart", "status", "logs", "install", "uninstall")]
    [string]$Action
)

$serviceName = "OllamaAIService"

function Start-OllamaService {
    Write-Host "Iniciando serviço Ollama..." -ForegroundColor Yellow
    Start-Service -Name $serviceName
    Start-Sleep -Seconds 5
    Get-Service -Name $serviceName
}

function Stop-OllamaService {
    Write-Host "Parando serviço Ollama..." -ForegroundColor Yellow
    Stop-Service -Name $serviceName -Force
    Get-Service -Name $serviceName
}

function Restart-OllamaService {
    Write-Host "Reiniciando serviço Ollama..." -ForegroundColor Yellow
    Restart-Service -Name $serviceName -Force
    Start-Sleep -Seconds 5
    Get-Service -Name $serviceName
}

function Show-ServiceStatus {
    $service = Get-Service -Name $serviceName -ErrorAction SilentlyContinue
    if ($service) {
        Write-Host "Status do Serviço:" -ForegroundColor Green
        $service | Format-Table Name, Status, StartType, DisplayName -AutoSize

        # Teste de conectividade
        $connection = Test-NetConnection -ComputerName "localhost" -Port 11434 -WarningAction SilentlyContinue
        Write-Host "Conectividade na porta 11434: $($connection.TcpTestSucceeded)?OK?else{FALHA})" -
        ForegroundColor $($connection.TcpTestSucceeded)?('Green'):('Red'))
    } else {
        Write-Host "Serviço não encontrado!" -ForegroundColor Red
    }
}

function Show-ServiceLogs {
    $logFile = "C:\OllamaService\Logs\ollama-stdout.log"
    $errorLogFile = "C:\OllamaService\Logs\ollama-stderr.log"

    Write-Host "==== LOGS PADRÃO (últimas 20 linhas) ====" -ForegroundColor Green
    if (Test-Path $logFile) {
        Get-Content $logFile -Tail 20
    } else {
        Write-Host "Arquivo de log não encontrado: $logFile" -ForegroundColor Yellow
    }

    Write-Host "`n==== LOGS DE ERRO (últimas 10 linhas) ====" -ForegroundColor Red
    if (Test-Path $errorLogFile) {
        Get-Content $errorLogFile -Tail 10
    } else {
        Write-Host "Arquivo de log de erro não encontrado: $errorLogFile" -ForegroundColor Yellow
    }
}

# Executar ação
switch ($Action) {
    "start" { Start-OllamaService }
    "stop" { Stop-OllamaService }
}

```

```

"restart" { Restart-OllamaService }
"status" { Show-ServiceStatus }
"logs" { Show-ServiceLogs }
"install" {
    Write-Host "Execute o script install-service.ps1 para instalar o serviço" -ForegroundColor Cyan
}
"uninstall" {
    Write-Host "Removendo serviço..." -ForegroundColor Yellow
    Stop-Service -Name $ServiceName -Force -ErrorAction SilentlyContinue
    & "C:\OllamaService\nssm.exe" remove $ServiceName confirm
    Write-Host "Serviço removido!" -ForegroundColor Green
}

```

- **Funcionalidade:** Este script unifica várias ações de gerenciamento do serviço Ollama:

- **start:** Inicia o serviço.
- **stop:** Para o serviço.
- **restart:** Reinicia o serviço.
- **status:** Exibe o status atual do serviço e a conectividade da porta.
- **logs:** Mostra as últimas linhas dos logs padrão e de erro.
- **install:** Instruções para usar o install-service.ps1.
- **uninstall:** Para o serviço e o remove completamente do sistema.

Essa ferramenta simplifica muito a manutenção do servidor Ollama.

## 6.5. Fase 5: Scripts de Cliente para Outras Máquinas

Para que outras máquinas possam interagir com o serviço Ollama configurado, o documento oferece um exemplo de cliente Python e exemplos de cURL.

- **Cliente Python para Outras Máquinas (ollama\_client.py):**

```

import requests
import json
import sys
import argparse
from typing import Optional, Dict, Any

class OllamaNetworkClient:
    def __init__(self, server_ip: str, server_port: int = 11434):
        self.base_url = f"http://{server_ip}:{server_port}"
        self.model = "code-analyzer"

    def test_connection(self) -> bool:
        """Testa conectividade com o servidor Ollama"""
        try:
            response = requests.get(f"{self.base_url}/api/tags", timeout=10)
            return response.status_code == 200
        except requests.exceptions.RequestException:
            return False

    def list_models(self) -> Dict[str, Any]:
        """Lista modelos disponíveis no servidor"""
        try:
            response = requests.get(f"{self.base_url}/api/tags", timeout=10)
            response.raise_for_status()
            return response.json()
        except requests.exceptions.RequestException as e:
            return {"error": f"Erro ao listar modelos: {str(e)}"}

```

```

def analyze_code(self, code_content: str, analysis_type: str = "full") -> Dict[str, Any]:
    """Analisa código usando o modelo Ollama remoto"""
    url = f"{self.base_url}/api/chat"

    prompt = f"""
Analise o seguinte código legado e forneça:
1. Análise detalhada dos problemas identificados
2. Proposta de refatoração
3. Código melhorado
4. Explicação das mudanças

Tipo de análise: {analysis_type}

Código:

{code_content}
"""

    payload = {
        "model": self.model,
        "messages": [
            {"role": "user", "content": prompt}
        ],
        "stream": False,
        "options": {
            "temperature": 0.1,
            "num_ctx": 16384,
            "num_predict": 2048
        }
    }

    try:
        response = requests.post(url, json=payload, timeout=300)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        return {"error": f"Erro na requisição: {str(e)}"}

def analyze_file(self, file_path: str) -> Dict[str, Any]:
    """Analisa um arquivo de código"""
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            code_content = file.read()

        return self.analyze_code(code_content)
    except FileNotFoundError:
        return {"error": f"Arquivo não encontrado: {file_path}"}
    except Exception as e:
        return {"error": f"Erro ao ler arquivo: {str(e)}"}

def get_server_status(self) -> Dict[str, Any]:
    """Obtém status do servidor Ollama"""
    try:
        response = requests.get(f"{self.base_url}/api/ps", timeout=10)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        return {"error": f"Erro ao obter status: {str(e)}"}

def main():
    parser = argparse.ArgumentParser(description='Cliente Ollama para Análise de Código')
    parser.add_argument('server_ip', help='IP do servidor Ollama')
    parser.add_argument('--port', type=int, default=11434, help='Porta do servidor (padrão: 11434)')
    parser.add_argument('--file', help='Arquivo de código para analisar')
    parser.add_argument('--test', action='store_true', help='Testar conectividade')
    parser.add_argument('--status', action='store_true', help='Mostrar status do servidor')
    parser.add_argument('--models', action='store_true', help='Listar modelos disponíveis')

    args = parser.parse_args()

    client = OllamaNetworkClient(args.server_ip, args.port)

```

```

if args.test:
    print(f"Testando conectividade com {args.server_ip}:{args.port}...")
    if client.test_connection():
        print(" ✅ Conectividade OK!")
    else:
        print(" ❌ Falha na conectividade!")
        sys.exit(1)

elif args.status:
    print(f"Status do servidor {args.server_ip}:{args.port}:")
    status = client.get_server_status()
    if "error" in status:
        print(f" ❌ {status['error']}")
    else:
        print(json.dumps(status, indent=2))

elif args.models:
    print(f"Modelos disponíveis em {args.server_ip}:{args.port}:")
    models = client.list_models()
    if "error" in models:
        print(f" ❌ {models['error']}")
    else:
        for model in models.get('models', []):
            print(f" - {model.get('name', 'N/A')}")

elif args.file:
    print(f"Analisando arquivo: {args.file}")
    print(f"Servidor: {args.server_ip}:{args.port}")
    print("-" * 50)

    result = client.analyze_file(args.file)
    if "error" in result:
        print(f" ❌ {result['error']}")
    else:
        print(result['message']['content'])

else:
    # Modo interativo
    print(f"Cliente Ollama conectado a {args.server_ip}:{args.port}")
    print("Digite 'quit' para sair")
    print("-" * 50)

    while True:
        try:
            code_input = input("\nCole seu código aqui (ou 'quit' para sair):\n")
            if code_input.lower() == 'quit':
                break

            if code_input.strip():
                print("Analizando código...")
                result = client.analyze_code(code_input)
                if "error" in result:
                    print(f" ❌ {result['error']}")
                else:
                    print("\n" + "="*50)
                    print("ANÁLISE:")
                    print("="*50)
                    print(result['message']['content'])
        except KeyboardInterrupt:
            print("\nSaindo...")
            break

if __name__ == "__main__":
    main()

```

## **\*\*Funcionalidade:\*\***

Este script Python atua como um cliente de linha de comando para o servidor Ollama. Ele permite:

- `test\_connection()` : Verificar se o servidor Ollama está acessível.
- `list\_models()` : Listar os modelos disponíveis no servidor.
- `analyze\_code(code\_content)` : Enviar um trecho de código para análise, com um prompt específico para análise de código legado.
- `analyze\_file(file\_path)` : Ler um arquivo de código e enviá-lo para análise.
- `get\_server\_status()` : Obter informações sobre o status do servidor Ollama.

O script também inclui um `main` que usa `argparse` para permitir diferentes modos de operação (testar conexão, ver status, listar modelos, analisar arquivo ou modo interativo), tornando-o uma ferramenta flexível para usuários.

- **Cliente via cURL (para testes):**

- **Testar Conectividade:** curl -X GET http://192.168.1.100:11434/api/tags
- **Analizar Código:**

```
curl -X POST http://192.168.1.100:11434/api/chat \
-H "Content-Type: application/json" \
-d'{
  "model": "code-analyzer",
  "messages": [
    {"role": "user", "content": "Analise este código: def test(): pass"}
  ],
  "stream": false
}'
```

Esses comandos cURL são exemplos rápidos para testar a comunicação com o servidor Ollama diretamente do terminal, validando a acessibilidade da API.

## **6.6. Fase 6: Configuração de Inicialização Automática**

Para garantir que todas as configurações estejam aplicadas e o serviço Ollama inicie corretamente após reinicializações, segue um script de inicialização completa.

- **Criar C:\OllamaService\Scripts\startup-complete.ps1:**

```
# Script de inicialização completa do Ollama
# Execute uma vez após instalação para configurar tudo

param(
    [switch]$Force
)

Write-Host "======" -ForegroundColor Cyan
Write-Host " CONFIGURAÇÃO COMPLETA OLLAMA SERVICE" -ForegroundColor Cyan
Write-Host "======" -ForegroundColor Cyan

# Verificar privilégios de administrador
if (-NOT ([Security.Principal.WindowsPrincipal] [Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole] "Administrator"))
{
    Write-Host "X Este script deve ser executado como Administrador!" -ForegroundColor Red
    exit 1
}
```

### # 1. Configurar variáveis de ambiente

```
Write-Host "1. Configurando variáveis de ambiente..." -ForegroundColor Yellow
$envVars = @{
    "OLLAMA_HOST" = "0.0.0.0:11434"
    "OLLAMA_MODELS" = "C:\OllamaModels"
    "OLLAMA_KEEP_ALIVE" = "60m"
    "OLLAMA_NUM_PARALLEL" = "4"
    "OLLAMA_MAX_LOADED_MODELS" = "2"
    "OLLAMA_ORIGINS" = "*"
    "CUDA_VISIBLE_DEVICES" = "0"
    "OLLAMA_FLASH_ATTENTION" = "1"
}

foreach ($var in $envVars.GetEnumerator()) {
    [Environment]::SetEnvironmentVariable($var.Key, $var.Value, "Machine")
    Write-Host " ✅ $($var.Key) = $($var.Value)" -ForegroundColor Green
}
```

### # 2. Criar estrutura de diretórios

```
Write-Host "2. Criando estrutura de diretórios..." -ForegroundColor Yellow
$directories = @(
    "C:\OllamaModels",
    "C:\OllamaModels\Modelfiles",
    "C:\OllamaService",
    "C:\OllamaService\Logs",
    "C:\OllamaService\Scripts",
    "C:\OllamaService\Config",
    "C:\OllamaService\Backup"
)

foreach ($dir in $directories) {
    if (!(Test-Path $dir)) {
        New-Item -ItemType Directory -Path $dir -Force | Out-Null
        Write-Host " ✅ Criado: $dir" -ForegroundColor Green
    } else {
        Write-Host " 🛡 Já existe: $dir" -ForegroundColor Cyan
    }
}
```

### # 3. Configurar firewall

```
Write-Host "3. Configurando firewall..." -ForegroundColor Yellow
try{
    Remove-NetFirewallRule -DisplayName "Ollama AI Service*" -ErrorAction SilentlyContinue
    New-NetFirewallRule -DisplayName "Ollama AI Service - Inbound" -Direction Inbound -Protocol TCP -LocalPort
    11434 -Action Allow -Profile Domain,Private,Public | Out-Null
    New-NetFirewallRule -DisplayName "Ollama AI Service - Outbound" -Direction Outbound -Protocol TCP -LocalPort
    11434 -Action Allow -Profile Domain,Private,Public | Out-Null
    Write-Host " ✅ Firewall configurado" -ForegroundColor Green
} catch{
    Write-Host " 🚨 Erro ao configurar firewall: $($_.Exception.Message)" -ForegroundColor Yellow
}
```

### # 4. Configurar exclusões do Windows Defender

```
Write-Host "4. Configurando Windows Defender..." -ForegroundColor Yellow
try{
    Add-MpPreference -ExclusionPath "C:\OllamaModels" -ErrorAction SilentlyContinue
    Add-MpPreference -ExclusionPath "C:\OllamaService" -ErrorAction SilentlyContinue
    Add-MpPreference -ExclusionProcess "ollama.exe" -ErrorAction SilentlyContinue
    Write-Host " ✅ Exclusões do Defender configuradas" -ForegroundColor Green
} catch{
    Write-Host " 🚨 Erro ao configurar Defender: $($_.Exception.Message)" -ForegroundColor Yellow
}
```

### # 5. Instalar serviço se não existir

```
Write-Host "5. Verificando serviço..." -ForegroundColor Yellow
$service = Get-Service -Name "OllamaAIService" -ErrorAction SilentlyContinue
if (-not $service -or $Force){
    Write-Host " Instalando serviço..." -ForegroundColor Cyan
    & "C:\OllamaService\Scripts\install-service.ps1"
```

```
    } else {
        Write-Host " ✅ Serviço já instalado" -ForegroundColor Green
    }


```

#### # 6. Baixar modelos essenciais (correção no nome do modelo e adicionando o qwen2.5-coder:3b)

```
Write-Host "6. Verificando modelos..." -ForegroundColor Yellow
try{
    $models = & ollama list 2>$null
    # Verificando se o modelo principal (qwen2.5-coder:3b) está presente
    if ($models -notmatch "qwen2.5-coder:3b"){
        Write-Host " Baixando qwen2.5-coder:3b (isso pode demorar)..." -ForegroundColor Cyan
        & ollama pull qwen2.5-coder:3b
        Write-Host " ✅ qwen2.5-coder:3b baixado" -ForegroundColor Green
    } else {
        Write-Host " ✅ qwen2.5-coder:3b já disponível" -ForegroundColor Green
    }
} catch{
    Write-Host " ⚠️ Erro ao verificar/baixar modelos: $($_.Exception.Message)" -ForegroundColor Yellow
}
```

#### # 7. Criar modelo personalizado

```
Write-Host "7. Criando modelo personalizado..." -ForegroundColor Yellow
$modelfilePath = "C:\OllamaModels\Modelfiles\CodeAnalyzer.modelfile"
if (Test-Path $modelfilePath){
    try{
        & ollama create code-analyzer -f $modelfilePath
        Write-Host " ✅ Modelo code-analyzer criado" -ForegroundColor Green
    } catch{
        Write-Host " ⚠️ Erro ao criar modelo personalizado: $($_.Exception.Message)" -ForegroundColor Yellow
    }
} else {
    Write-Host " ⚠️ Modelfile não encontrado em $modelfilePath" -ForegroundColor Yellow
}
```

#### # 8. Testar configuração

```
Write-Host "8. Testando configuração..." -ForegroundColor Yellow
Start-Sleep -Seconds 5

$connection = Test-NetConnection -ComputerName "localhost" -Port 11434 -WarningAction SilentlyContinue
if ($connection.TcpTestSucceeded){
    Write-Host " ✅ Servidor respondendo na porta 11434" -ForegroundColor Green
} else {
    Write-Host " ❌ Servidor não está respondendo" -ForegroundColor Red
}

Write-Host "`n======" -ForegroundColor Cyan
Write-Host " CONFIGURAÇÃO CONCLUÍDA!" -ForegroundColor Cyan
Write-Host "======" -ForegroundColor Cyan
Write-Host "Servidor Ollama configurado como serviço de rede" -ForegroundColor Green
Write-Host "Acesso: http://[IP_DO_SERVIDOR]:11434" -ForegroundColor Cyan
Write-Host "`nPróximos passos:" -ForegroundColor Yellow
Write-Host "1. Reinicie o computador para garantir todas as configurações" -ForegroundColor White
Write-Host "2. Distribua o cliente Python para outras máquinas" -ForegroundColor White
Write-Host "3. Use o monitor: .\monitor-service.ps1" -ForegroundColor White
Write-Host "4. Gerencie o serviço: .\manage-service.ps1 [start|stop|restart|status]" -ForegroundColor White
```

- **Funcionalidade:** Este script é um "tudo-em-um" para configurar o ambiente do Ollama do zero ou verificar/reaplicar configurações. Ele executa em sequência:

1. Verifica privilégios de administrador.
2. Configura as variáveis de ambiente permanentes.
3. Cria a estrutura de diretórios necessária.
4. Configura o firewall do Windows.

5. **Configura exclusões do Windows Defender:** Este é um passo importante que pode evitar problemas de desempenho ou bloqueios, adicionando C:\OllamaModels, C:\OllamaService e o processo ollama.exe às exclusões.
6. Instala o serviço Ollama se ele ainda não estiver instalado.
7. Baixa o modelo principal (qwen2.5-coder:3b) se ainda não estiver presente.
8. Cria o modelo personalizado code-analyzer.
9. Testa a conectividade com o servidor Ollama na porta 11434.

Ao final, ele fornece instruções claras para os próximos passos, como reiniciar, distribuir o cliente e usar os scripts de monitoramento/gerenciamento.

---

## 7. Configurações de Rede e Segurança (Avançadas)

O arquivo **network-config.json** conter configurações mais detalhadas, embora o script de inicialização já defina muitas delas via variáveis de ambiente.

- **C:\OllamaService\Config\network-config.json:**

```
{  
  "server": {  
    "host": "0.0.0.0",  
    "port": 11434,  
    "max_connections": 10,  
    "timeout": 300,  
    "keep_alive": "60m"  
  },  
  "security": {  
    "allowed_origins": "*",  
    "rate_limiting": {  
      "enabled": true,  
      "requests_per_minute": 60,  
      "burst_size": 10  
    }  
  },  
  "performance": {  
    "num_parallel": 4,  
    "max_loaded_models": 2,  
    "gpu_batch_size": 512,  
    "context_size": 16384  
  }  
}
```

Este arquivo centraliza configurações de rede, segurança e desempenho. Embora o Ollama use variáveis de ambiente para muitas dessas configurações, um arquivo JSON como este seria ideal para uma gestão mais legível e versionável em um cenário de produção.

- **Destaques:**

- **max\_connections:** Limite de conexões simultâneas ao servidor.
- **timeout:** Tempo limite para requisições.
- **rate\_limiting:** Uma configuração de segurança importante para evitar abuso do serviço, limitando requisições por minuto.

- **gpu\_batch\_size:** Tamanho do lote de inferência na GPU, impactando diretamente a eficiência.
  - **context\_size:** Tamanho máximo do contexto que o modelo pode processar.
- 

## 8. Estimativa de Performance para Uso em Rede

Segue estimativas realistas de desempenho para o sistema configurado, o que é valioso para definir expectativas.

- **Velocidade de Inferência:** ~15-25 tokens/segundo por requisição
    - Esta é a velocidade com que o modelo gera texto. Um bom range para um modelo de 3B com 16GB VRAM.
  - **Requisições Simultâneas:** 2-4 (dependendo do tamanho do contexto)
    - Indica a capacidade do servidor de processar várias solicitações de IA ao mesmo tempo. A VRAM e a RAM desempenham um papel crucial aqui.
  - **Contexto Máximo:** 32K tokens por requisição
    - A capacidade do modelo de "se lembrar" de uma grande quantidade de informações na conversa ou no código que está sendo analisado. Isso é excelente para análises de código extensas.
  - **Tempo de Carregamento:** ~30-60 segundos (apenas na primeira requisição)
    - O tempo que leva para o modelo ser carregado da SSD para a VRAM na primeira vez que é acessado. Graças ao OLLAMA\_KEEP\_ALIVE, esse tempo é evitado para requisições subsequentes dentro do período definido.
  - **Uso de VRAM:** ~14-15GB (modelo permanece carregado)
    - Confirma que o modelo qwen2.5-coder:3b se encaixa perfeitamente nos 16GB de VRAM da RTX 5060 Ti.
  - **Latência de Rede:** +5-50ms dependendo da rede local
    - O tempo adicional de atraso introduzido pela rede. Em uma rede local, esse valor é geralmente baixo, garantindo uma experiência de uso responsiva para os clientes.
- 

## 9. Comandos de Gerenciamento Úteis

Para o dia a dia, segue alguns comandos PowerShell e Linux (embora o projeto seja para Windows) que são indispensáveis para monitorar e gerenciar o serviço Ollama:

- **Verificar status do serviço:** Get-Service -Name "OllamaAIService"
- **Reiniciar serviço:** Restart-Service -Name "OllamaAIService"

- **Ver logs em tempo real:** Get-Content "C:\OllamaService\Logs\ollama-stdout.log" -Wait (ótimo para depuração)
  - **Monitorar conexões de rede:** netstat -an | findstr ":11434" (para ver quem está conectado na porta do Ollama)
  - **Verificar uso de GPU:** nvidia-smi -l 5 (atualiza o status da GPU a cada 5 segundos)
  - **Testar API remotamente:** Test-NetConnection -ComputerName [IP\_SERVIDOR] -Port 11434 (para verificar a conectividade de outra máquina).
- 

### **Conclusão e Benefícios deste Setup**

Este projeto é um guia abrangente e detalhado para configurar um servidor de IA local com Ollama no Windows 11. Ele aproveita ao máximo o hardware disponível, otimiza o sistema operacional, instala os componentes necessários e configura o Ollama como um serviço de rede. A criação de um Modelfile personalizado para análise de código e a inclusão de scripts de gerenciamento e clientes garantem que o sistema seja não apenas poderoso, mas também fácil de implantar e manter.

O resultado final é um "serviço robusto e permanente de IA para análise de código, acessível por toda a rede, com configurações que persistem após reinicializações e otimizado para múltiplas conexões simultâneas". Isso representa uma solução de IA local de alto desempenho, ideal para equipes de desenvolvimento ou usuários individuais que precisam de capacidades de LLM sem depender de serviços em nuvem, garantindo privacidade, baixa latência e controle total sobre os modelos. É realmente impressionante a quantidade de detalhes e o cuidado em cada etapa do processo!

---

**Uriel Gusmão Apolônio**