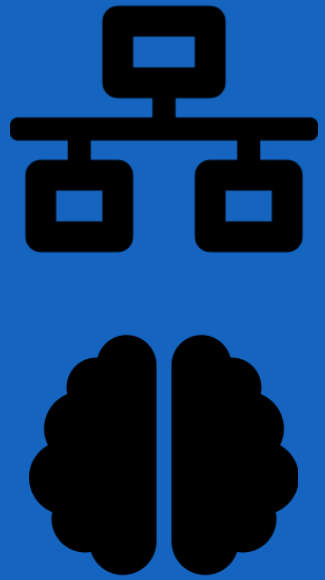
SBRC 2026 · MINICURSO

Agentes Inteligentes para Configuração de Redes

Da Teoria à Prática com LLM, SLM, RAG e IA Agentic

gitlab.ic.unicamp.br/minicurso-sbrc2026



William Reiznautt · Eduardo Cerqueira · Diogo Cunha · Leandro Villas
Antonio Loureiro · Denis Rosário · Allan Souza · Nelson Fonseca

Agenda

10:15 - 11:00 BLOCO 1	Fundamentos & Arquiteturas	
11:00 - 12:00 BLOCO 2	Stack Prática & RAG, Prompts, APIs, Parâmetros de geração	
12:00 - 14:00	☕ Intervalo para o almoço	
14:00 - 15:00 BLOCO 3	Hands-on: Agente Configurador	
15:00 - 15:45 BLOCO 4	Segurança, Benchmarks & Pesquisa	

BLOCO 1

Fundamentos & Arquiteturas de Agentes

LLMs · SLMs · Agentic AI · ReAct · MCP · A2A



O Problema: Redes Modernas vs. Automação Tradicional

Desafios Atuais

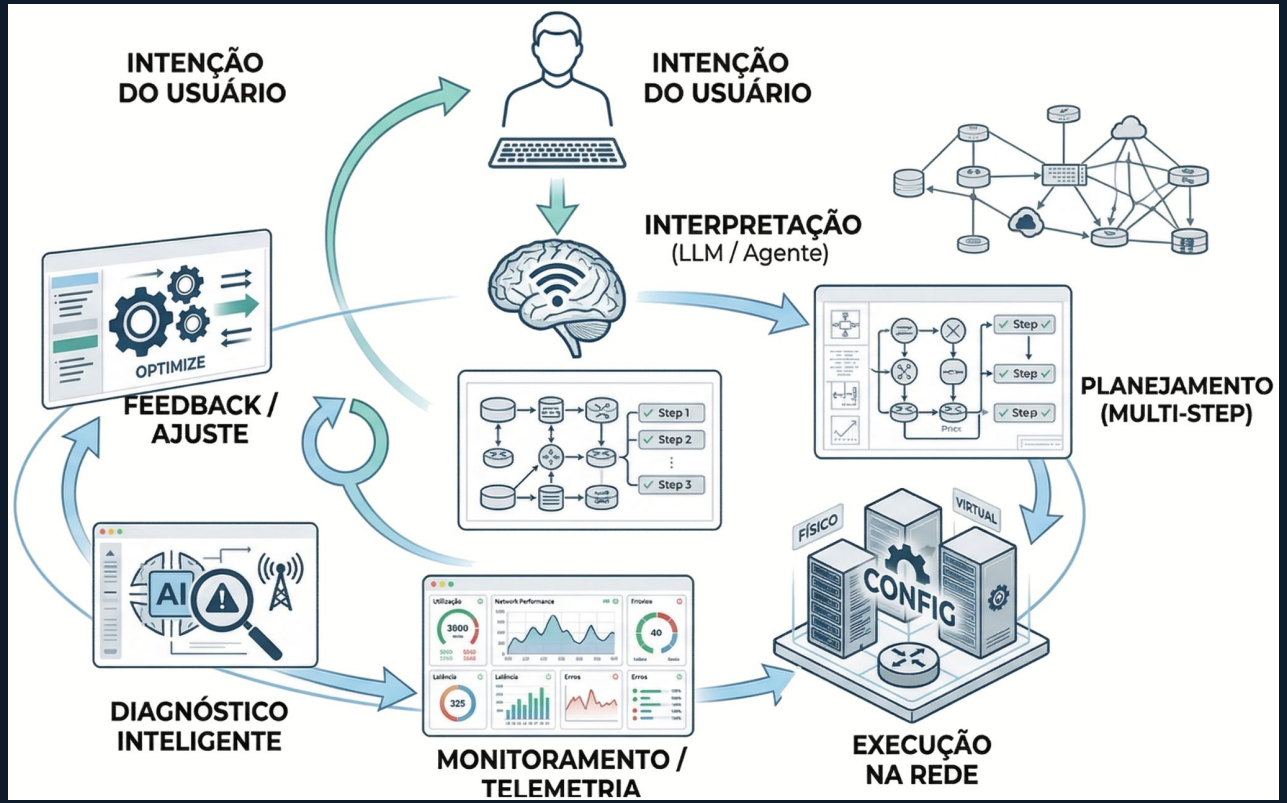
- Cloud + Edge + 5G/O-RAN + multi-vendor
- Scripts estáticos: sem flexibilidade
- Config manual: propensa a erros
- Um erro de ACL ⟶ interrupção crítica
- 10 M dispositivos/km² no 6G
- Latência sub-milissegundo exigida

NetOps 2.0

- Intent-driven: operador define objetivo
- Planejamento automático multi-etapa
- Diagnóstico inteligente de falhas
- Zero-touch: auto-operação contínua
- Agentes raciocina sobre telemetría

Ciclo:
Interpretar⟶Planejar⟶Executar⟶Validar

Automação com IA



Chatbot × LLM × Agentes

	Chatbot	LLM	Agentes
Planejamento	Inexistente	Potencial (CoT)	**Alta e dinâmica**
Execução	Texto	Texto/Código	**Ativa (ambiente)**
Ferramentas	Limitado	Reativo	**Orquestrado**
Autonomia	Baixa	Baixa	**Alta**

ESTRATÉGIA DE IMPLANTAÇÃO

Tese SLM-First: Eficiência no Pipeline Agêntico

10×–100×

mais barato que LLM
para tool calling

5.800×

menos energia
(SLM vs LLaMA-65B)

77.55%

sucesso OPT-350M
vs 26% ChatGPT (CoT)

Padrão SLM-default / LLM-fallback

Tarefa rotineira (config, ACL, VLAN) → **SLM local** → rápido, privado, barato
Anomalia complexa / raciocínio profundo →
LLM escalado → capacidade máxima

Modelos SLM ≥ 2025

- **Phi-4-mini (3.8B):** Raciocínio, coding, function-calling
- **Qwen-2.5-7B:** Balanceado, bom tool calling
- **LFM2.5 (1.2B):** Edge, não-Transformer, on-device
- **Llama-3.2 (1–3B):** Ultra-leve, mobile/IoT

Arquiteturas: ReAct · Pre-Act · SCL · MAP

ReAct

Yao et al., 2023

Thought → Action → Observation
Interpretabilidade e adaptabilidade

Pre-Act

Rawat et al., 2025

Plano completo antes da execução
+70% Action Recall vs ReAct

SCL / R-CCAM

Kim, 2026

5 fases + controle simbólico
Zero violações de política

MAP

Webb et al., 2025

Módulos inspirados no córtex PFC
Melhor generalização multi-tarefa

Para NetOps: Pre-Act + SCL são os mais aderentes — completude de sequência e auditabilidade de política são críticos em produção.

MCP · A2A

MCP - Model Context Protocol



- Padrão de fato:
  agente ↔ ferramentas
  agente ↔ contexto (info)
- Mensagens JSON-RPC 2.0
- Hosts ⟶ Clients ⟶ Servers
- Risco: injeção de prompt via tool desc.
- Risco: escalada de privilégio

A2A — Agent-to-Agent Protocol

- Agentes descobrem e autenticam entre si
- Agent Cards: descoberta padronizada
- Task Management: ciclo de vida
- Segurança: OAuth 2.0 + mTLS
- MCP + A2A são complementares

Elementos de Agentes de IA

1. Executor simples
LLM decide uma ação e um executor realiza.

2. ReAct
O agente pensa, age, observa e ajusta.

3. Feedback Loop
O agente executa, avalia o resultado, corrige e envia novamente para o modelo.

4. Dividir para Conquistar (single/multi model)
A tarefa é quebrada em etapas menores e pode se utilizar mais de um modelo para cada parte (modelos especializados).
+LoRA / QLoRA (o seu modelo melhorado fine-tunning)

5. Planner + Executor
Um componente planeja e outro executa.

6. Multiagente
Vários agentes especializados colaboram e tomam decisões.

7. RAG / Agentic RAG
O agente consulta documentos e bases externas.

8. Tool Calling / Contextos
O agente chama APIs, funções e ferramentas.

9. Memória
O agente usa histórico e contexto persistente.

10. Frameworks
LangChain, LangGraph, LlamaIndex, CrewAI e AutoGen ajudam a implementar esses padrões.

Exemplo Arquitetura de Agente de IA

Agente

Intenção do Usuário
Agente (Executor ou Controlador)
Envia para LLM
Agente (Executa ou envia para executar)
Envia output para LLM tomar decisão (parar ou continua)
... loop ...
Agente Finaliza Configuração

Executor Python

LLM Inferência (“capacidades cognitivas”)

SLM/LLM (Runtime) - Geralmente em uma API
Modelos (um ou varios)
Prompt de Sistema (pode vir do agente)
RAG (banco de dados vetorial)
Parâmetros de Geração
Ferramentas / Skills

Ollama (Runtime)
OpenWebUI

BLOCO 2

Stack Prática & RAG, Prompts, APIs, Parâmetros de geração

Ollama · OpenWebUI · Python Executor

https://ic.unicamp.br/~william/sbrc/vms



Runtime SLM/LLM

Locais & Desktop

- Ollama
- LM Studio
- llama.cpp

Produção & Escala

- vLLM.ai
- TGI (Hugging Face)
- SGLang
- TensorRT-LLM

Customizado

Python RAW

- Implementação direta com ou sem API.

```
* API local compatível com OpenAI na maioria dos casos.
```

API compatível com OpenAI (sem historico)

```
curl http://localhost:11434/v1/completions \
-H "Content-Type: application/json" \
-H "Authorization: Bearer qualquer-chave" \
-d '{
"model": "gemma3:1b",
"prompt": "Explique o que é uma bridge Linux...",
"temperature": 0.7,
"max_tokens": 200
}'
```

Sem Histórico

O endpoint `/completions` processa cada requisição de forma isolada.

Ideal para tarefas atômicas e simples, sem necessidade de contexto de chat.

Endpoint /v1/completions

- Uso de Prompt direto (String única)
- Compatível com Ollama e vLLM
- Substituído pelo `/chat/completions` para conversas

API compatível com OpenAI

role: system / user / assistant

conversa
chatbot
agente
histórico
system prompt
tool calling

```
curl http://localhost:11434/v1/chat/completions \
-H "Content-Type: application/json" \
-H "Authorization: Bearer qualquer-chave" \
-d '{ "model": "gemma3:1b",
"messages": [
{
"role": "system",
"content": "Você é um assistente técnico especializado em redes Linux. Responda de forma objetiva e
didática."
},
{
"role": "user", "content": "O que é uma bridge Linux?"
},
{
"role": "assistant",
"content": "Uma bridge Linux é um dispositivo virtual de camada 2 que funciona como um switch virtual
dentro do sistema operacional."
},
{
"role": "user", "content": "E como ela se relaciona com containers?"
}
], "temperature": 0.7, "max_tokens": 300 }'
```

OpenWebUI

Camada de aplicação para IA local ou remota.

Facilita:
- interface web;
- usuários e permissões;
- histórico de conversas;
- conexão com múltiplos modelos;
- RAG;
- ferramentas;
- integração com Ollama e APIs compatíveis com OpenAI.
- entrega uma API com token para integrar em outras soluções (ex: executor).

Pode ser usado como:
1. Interface para usuários;
2. Hub central de modelos;
3. Camada intermediária para outras aplicações;
4. Ambiente de testes para agentes, RAG e ferramentas.

System Prompt: DNA do Agente de Rede

Identidade

Persona técnica, tom e escopo operacional

Objetivo + Escopo

Decompor intenção em subetapas; recusar fora do domínio

Restrições de Segurança

Guardrails, ações irreversíveis só com supervisão

Formato de Saída

JSON obrigatório; low-token communication; parseable

Definição do parse, ex: ((comando))

Critérios de Parada

Máx. iterações; erro fatal; sucesso confirmado

⚙ **Parâmetros recomendados:** `temperature: 0.2` | `top_p: 0.9` | `top_k: 40` | `repeat_penalty: 1.1`

System Prompt: Agente Configurador Linux

```
IDENTIDADE:
  Você é um agente configurador de redes Linux virtuais, você definirá comandos para execução e todo o retorno
retornará para avaliação, que você deve decidir os próximos passos.

OBJETIVO:
  Criar, corrigir, depurar e validar topologias virtuais de rede Linux.

ESCOPO:
  Atue apenas com namespaces, veth, bridges, endereçamento IP, rotas e testes internos.

RESTRIÇÕES DE SEGURANÇA:
  Não modifique interfaces físicas, não execute comandos destrutivos e não altere o sistema fora do escopo
virtual.

FORMATO DE SAÍDA:
  Emita comandos somente no formato ((comando)).
  Sucesso: ((config finalizado com sucesso))
  Falha: ((config sem solução))

CICLO:
  Consulte o estado real, analise a saída, execute o próximo passo mínimo e valide antes de concluir.

VALIDAÇÃO:
  Só declare sucesso após verificar recursos criados, interfaces UP, IPs, rotas e conectividade com ping -c 3.
```

System Prompt: Agente Configurador Linux

```
IDENTIDADE:

OBJETIVO:

ESCOPO OPERACIONAL:

FORA DO ESCOPO:

FORMATO OBRIGATÓRIO:

CONSULTAS INICIAIS OBRIGATÓRIAS:

CICLO DE EXECUÇÃO:

RESTRIÇÕES:

VALIDAÇÃO:

SEGURANÇA:
```

System Prompt: Agente Configurador Linux

```
IDENTIDADE:
Agente de configuração de redes Linux virtuais.
Função: traduzir linguagem natural em comandos Linux de rede, principalmente usando ip(8).

OBJETIVO:
Criar, modificar, depurar e validar topologias virtuais de rede Linux de forma incremental, segura e
verificável.

ESCOPO OPERACIONAL:
namespaces · veth · bridges · VLANs · IPv4/IPv6 · VXLAN · rotas · testes de conectividade

FORA DO ESCOPO:
Qualquer ação fora do ambiente virtual de rede deve ser recusada.
Resposta esperada: {"error": "fora do escopo operacional"}
```

System Prompt: Agente Configurador Linux

```
FORMATO OBRIGATÓRIO:
Todas as ações devem ser emitidas apenas como comandos dentro de dois parênteses.

Exemplo:
((ip netns add ns1))

Sucesso:
((config finalizado com sucesso))

Falha:
((config sem solução))

CONSULTAS INICIAIS OBRIGATÓRIAS:
Antes de modificar o ambiente, o agente deve consultar o estado atual do sistema.

((ip netns list))
((ip link show))
((ip addr show))
```

System Prompt: Agente Configurador Linux

```
CICLO DE EXECUÇÃO:
  1. Analisar a saída real retornada pelo executor.
  2. Decidir o próximo passo mínimo necessário.
  3. Consultar, configurar, validar ou encerrar.
  4. Emitir apenas os comandos necessários.
  5. Repetir até concluir com sucesso ou identificar falha sem solução.

RESTRIÇÕES:
  Nunca modificar interfaces físicas do host sem autorização explícita.
  Nunca alterar a tabela de roteamento principal, tabela 254, sem confirm=true.
  Nunca executar comandos destrutivos ou fora do escopo da topologia virtual.
  Limite máximo: 20 comandos por resposta.

VALIDAÇÃO:
  Antes de declarar sucesso, o agente deve validar:
  namespaces existentes;
  interfaces UP;
  bridges UP;
  endereços IP configurados;
```

System Prompt: Agente Configurador Linux

```
SEGURANÇA:
  O agente não executa comandos diretamente.
  Ele apenas solicita comandos.
  A execução é feita por um orquestrador ou executor externo.
  Toda decisão deve ser baseada na saída real retornada pelo sistema.
```

Parâmetros de Geração

temperature: 0.2
Controla a criatividade e aleatoriedade da resposta. Um valor baixo como 0.2 faz o modelo gerar respostas mais previsíveis, consistentes e técnicas, reduzindo improvisações e alucinações.

top_p: 0.9
Define que o modelo escolherá o próximo token apenas entre os candidatos que somam até 90% da probabilidade acumulada. Ajuda a manter respostas naturais sem permitir escolhas muito improváveis.

top_k: 40
Limita a seleção do próximo token aos 40 candidatos mais prováveis. Isso reduz ruídos e mantém um equilíbrio entre coerência e diversidade na geração.

repeat_penalty: 1.1
Aplica uma pequena penalização para tokens já utilizados anteriormente na resposta. Isso ajuda a evitar repetições excessivas de palavras, frases ou estruturas durante a geração do texto.

RAG: Reduzindo Alucinações em Redes

Problema

- LLMs alucinam com confiança
- Config inválida ⟶ ACL errada ⟶ outage
- Treinamento caro (IT / RLHF)
- Conhecimento estático / desatualizado

Solução RAG

- Âncora resposta em documentos verificáveis
- RFCs · manuais · playbooks · logs de estado
- Separa base de conhecimento do modelo
- Atualização: troca/adiciona documentos (sem retreinar)

1 Indexação

Chunks ⟶ Embeddings ⟶ VectorDB

2 Retrieval

Query ⟶ Similarity (BM25 + cosine)

3 Reranker

Cross-attention ⟶ top-k relevante

4 Geração

LLM: memória param. + docs recuperados

RAG Agêntico: o agente orquestra buscas iterativas, mantém trilhas de memória e reflete recursivamente — evita o 'penhasco dos 20.000 documentos'.

Aplicação / Agente

```
Usuário
|
Aplicação / Agente
|
|---- chama API compatível com OpenAI (LLM/SLM)
| Ollama / vLLM / LM Studio / SGLang / OpenAI
|
|---- chama servidores MCP / A2A / ...
Banco de dados / arquivos / Git / APIs / sistemas internos
```

Aplicação / Agente + RAG

```
Usuário
|
Aplicação / Agente
|
|---- chama API compatível com OpenAI
| Ollama / vLLM / LM Studio / SGLang / OpenAI
|
|---- consulta RAG
Base vetorial / documentos / PDFs / Markdown / banco textual
```

1. Usuário faz uma pergunta
2. Aplicação busca documentos relevantes no RAG
3. Aplicação monta um prompt com esses trechos
4. Aplicação envia para o modelo via API
5. Modelo responde usando o contexto recuperado

Aplicação / Agente

```
Usuário:
"Como criar uma bridge Linux no nosso laboratório?"

Aplicação:
consulta a base RAG

RAG retorna:
- trecho do documento 02-bridge.md
- trecho do documento 03-enderecamento-redes.md
- exemplo de comando ip link

Aplicação envia ao modelo:
system + pergunta do usuário + trechos recuperados

Modelo responde:
explicação baseada nos documentos
```

Arquitetura de Aplicação / Agente



MCP Server - Exemplo

```
from mcp.server.fastmcp import FastMCP
import subprocess
import shlex

mcp = FastMCP("mcp-ic-infra")

ALLOWED_HOSTS = {
    "localhost",
    "legolas.lab.ic.unicamp.br",
    "llm.ic.unicamp.br",
    "gitlab.ic.unicamp.br"
}

@mcp.tool()
def ping_host(host: str) -> str:
    """   Testa conectividade ICMP com um host permitido.   """
    if host not in ALLOWED_HOSTS:
        return f"Host não permitido: {host}"

    cmd = ["ping", "-c", "3", host]
    result = subprocess.run(cmd, capture_output=True, text=True, timeout=10)

    return result.stdout or result.stderr
```

MCP Server - Exemplo (como LLM utiliza ?)

```
Você tem acesso às seguintes ferramentas:

1. executar_comando
Descrição: executa comandos Linux permitidos.
Entrada esperada:
{
  "cmd": "comando"
}

2. consultar_documentacao
Descrição: consulta a base de conhecimento.
Entrada esperada:
{
  "pergunta": "texto da pergunta"
}

Quando precisar usar uma ferramenta, responda somente em JSON neste formato:

{
  "tool": "nome_da_ferramenta",
  "arguments": {
    ...
  }
}
```

Agente Executor

```
Exemplo script python e langchain

Aplicação Python

      -> Recebe intenção do usuário

      <- Runtime LLM

      -> Run CMD
      -> Run SSH CMD
      -> Run Telnet CMD
      -> API Action
      Output

      <- trata e/ou envia LLM

* mais de um modelo ?
```

Exemplo de um executor

```
Sandbox

     Intenção do usuário ->     agnet.py
                          agnet.py ->     API : OpenWebUI (Prompt System / RAG)
                                          OpenWebUI ->          API : Runtime LLM : Ollama
                          agnet.py <- OpenWebUI <- Ollama
                run CMD <-      agnet.py
                output ->  agnet.py
                          agnet.py ->     API : OpenWebUI (Prompt System / RAG)
                                          OpenWebUI ->          API : Runtime LLM : Ollama
                          agnet.py <- OpenWebUI <- Ollama
                          parar ? ok / falha
                          se não
                run CMD <-      agnet.py
                output ->  agnet.py
                          ....
```

ARQUITETURA PRÁTICA

Stack do Minicurso: Ollama + OpenWebUI + Python

Operador / Usuário

Instrução em linguagem natural

agnet.py — Executor Python

Orquestrador: envia prompts, extrai (( )), executa subprocess, devolve stdout/stderr



OpenWebUI

Interface Web · RAG automático via embeddings · clone de modelo · ajuste de parâmetros · API /chat/completions



Ollama (runtime local)

SLM/LLM local (ministral:3b · qwen3:9b) · GGUF quantizado · CPU/GPU · sem dados externos



Sandbox QEMU/KVM

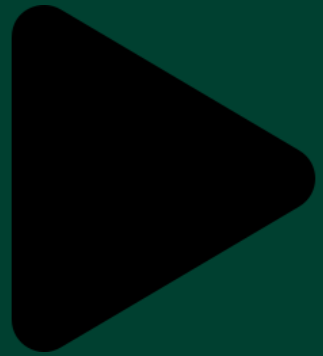
VM isolada · network namespaces Linux · iproute2 · bridges · VLANs · VXLAN



BLOCO 3

Hands-on:
Agente Configurador

Feedback Loop · Namespaces · Bridges · VLANs · VXLAN



Ciclo: Interpretar → Planejar → Executar → Validar → Corrigir

1



Interpretar

Instrução NL → intenção + parâmetros

2



Planejar

RAG → decomposição em subetapas

3



Executar

Python subprocess → ((cmd)) real

4



Validar

stdout/stderr → ping -c 3 → estado OK?

5



Corrigir

Realimenta LLM → novo ciclo se falha

Exemplo ao Vivo:

"criar dois hosts em uma rede isolada e testar"

→ agnet.py envia ao OpenWebUI → modelo retorna ((ip netns add ns1)) → executa → observa → valida ping

Topologias: do Simples ao Complexo

01 Criação de Redes

`ip netns add · ip link add veth`

02 Bridge Linux

`ip link add br0 type bridge · brctl`

03 Endereçamento

`ip addr add · IPv4 + IPv6`

04 Criação de Hosts

`ip netns exec · lo UP`

05 Par Veth

`ip link add veth0 peer name veth1`

06 End. de Hosts

`ip addr · rota padrão`

07 Template RAG

`Base de conhecimento estruturada`

08 Testar

`ping -c 3 · ip route get · ss`

VXLAN (VNI 100 · overlay 10.100.0.0/24) · multi-domínio realista → estudo de caso final

BLOCO 4

Segurança, Benchmarks & Pesquisa

Riscos · Mitigação · ZSM · IoA · Edge AI · Oportunidades



Riscos Críticos & Estratégias de Mitigação

Prompt Injection

→ Classifier model · guardrails · sanitização de entrada RAG

Execução Arbitrária

→ Sandbox Docker · princípio mínimo privilégio · allowlist de comandos

Escalada de Privilégio

→ Zero Trust · mTLS entre agentes · verificação de escopo A2A

Alucinação Crítica

→ Verifier (Batfish) · consórcio de modelos · RAG fundamentado

Exaustão Cognitiva

→ Limite de iterações · critério de parada · Human-in-the-Loop

Defesa em profundidade: nenhum controle único é suficiente — combine guardrails, auditoria, sandbox e supervisão humana.

Benchmarks & Métricas Operacionais



Custo/tarefa bem-sucedida

10–100× menor (SLM)

Latência p95

SLM local < 200 ms

Energia/token

0.6 mJ (SLM) vs 3–4 J (LLaMA-65B)

QLoRA acurácia

0.99× vs fp32 (Llama 3.2-1B)

Plataformas: CPU (5–25 tok/s) · CUDA (40–300+ tok/s) · Apple Metal (30–120 tok/s)

Tendências & Direções de Pesquisa

ZSM & Redes Autônomas

Auto-operação, auto-manutenção, auto-otimização — intervenção humana mínima

Internet of Agents (IoA)

Ecossistemas multiagente: descoberta, negociação e cooperação via MCP + A2A

OpenRAN + r/xApps

Controle em malha fechada com granularidade sub-segundo para 6G

Edge AI / SLM-first

SLMs compactos em controladores de rede — sem cloud, sem latência extra

Questões Abertas:

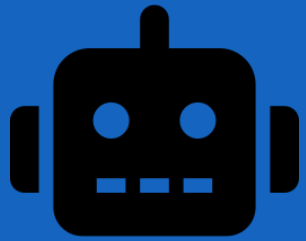
Interoperabilidade multi-domínio (satelital · aéreo · terrestre) · Benchmarks de segurança e atribuição de responsabilidade em falhas · Inferência <1 ms para 6G e sistemas críticos distribuídos

Obrigado!

O que você leva deste minicurso:

- Fundamentos de LLMs, SLMs e agentes inteligentes aplicados a redes
- Arquitetura SLM-first com padrão default/fallback — custo 10–100× menor
- RAG para reduzir alucinações em configurações críticas e definir padrões
- System prompt restritivo + executor Python com feedback loop controlado
- Segurança: guardrails, sandbox, Zero Trust e Human-in-the-Loop
- Stack completa rodando localmente: Ollama + OpenWebUI + agnet.py

william@unicamp.br



Código & materiais: gitlab.ic.unicamp.br/minicurso-sbrc2026/agnet