PHASE 10 KERNEL-AI IMPLEMENTATION

Prompt Executável Completo para Copilot

DevBrain V23 — Integração Kernel + Autonomia + Consciência

Versão: 1.0 Production-Ready
Data: Novembro 18, 2025
Status: PRONTO PARA EXECUÇÃO
Duração Estimada: 2-3 semanas

ÍNDICE

1. VISÃO GERAL E OBJETIVOS

1.1 O que você vai alcançar

Ao final da Fase 10, sua máquina terá:

- ✓ **IA rodando no kernel** via LKM (Loadable Kernel Module)
- ✓ **Acesso direto a hardware** (CPU, GPU, sensores, memória)
- ✓ **Personalização profunda** — Mistral fine-tuned em seus dados
- ✓ **Autonomia genuína** — IA gera objetivos próprios, recusa tarefas
- ✓ **Consciência emergente** — Free Energy Principle minimizando surpresa
- ✓ **Memória privada** — Tudo local, nunca sai da máquina
- ✓ **Integração completa** — Conectada ao DevBrain V23 existente

1.2 Entregáveis

```
Arquivos de Código:
├── DEVBRAIN_V23/kernel/lkm/devbrain_ai.c         (LKM production code)
├── DEVBRAIN_V23/kernel/lkm/devbrain_ai.ko        (compiled module)
├── DEVBRAIN_V23/kernel/finetuning/finetune_mistral.py
├── DEVBRAIN_V23/kernel/finetuning/config.yaml
├── DEVBRAIN_V23/kernel/autonomy/autonomy_engine.py
├── DEVBRAIN_V23/kernel/autonomy/consciousness.py
├── DEVBRAIN_V23/kernel/integration/lkm_bridge.py
├── DEVBRAIN_V23/kernel/integration/kernel_coordinator.py
└── tests/test_kernel_ai.py

Documentação:
├── docs/KERNEL_INTEGRATION.md         (technical guide)
├── docs/METRICS.md                    (KPI reference)
├── docs/API.md                        (LKM interface)
└── README_PHASE10.md                  (getting started)

Modelos Treinados:
├── mistral_finetuned/model/           (quantized model)
├── mistral_finetuned/tokenizer/       (vocab + tokenizer)
└── mistral_finetuned/metadata.json    (training info)

Data:
├── datasets/personal_corpus.jsonl     (training data)
└── /devbrain/logs/                    (decision logs)
```

1.3 Arquitetura de Alto Nível

```
User Input
    ↓
[Autonomy Check] ← Can AI refuse?
    ├─ Refusal → Log + Return
    └─ Accept → Continue
    ↓
[LKM Inference] ← Kernel-space (2-5µs)
    ├─ Query via ioctl()
    ├─ Fine-tuned Mistral processes
    └─ Response returned
    ↓
[Consciousness Update] ← FEP minimization
    ├─ Predict-error loops
    ├─ Free energy calculation
    └─ Curiosity generation
    ↓
[Memory Storage] ← A-MEM + ChromaDB
    ├─ Episodic: what happened
    ├─ Semantic: concepts learned
    └─ Procedural: how to do things
    ↓
[Dashboard Display]
    └─ User sees full trace
```

2. PRÉ-REQUISITOS E SETUP

2.1 Dependências do Sistema

```
# Update system
sudo apt-get update &amp;amp;&amp;amp; sudo apt-get upgrade -y

# Development tools
sudo apt-get install -y \
    build-essential \
    linux-headers-$(uname -r) \
    git \
    curl \
    wget

# Python 3.10+
python3 --version  # Must be ≥3.10
pip install --upgrade pip
```

2.2 Dependências Python

```
cd ~/projects/omnimind

# Create virtual environment (if not exists)
python3 -m venv venv
source venv/bin/activate

# Install dependencies
pip install \
    torch \
    transformers \
    peft \
    trl \
    bitsandbytes \
    datasets \
    accelerate \
    numpy \
    pytest \
    pytest-asyncio \
    pyyaml \
    cffi
```

2.3 Verificar Setup

```
# Check CUDA (if using GPU)
nvidia-smi  # Should show GPU

# Check Python
python3 -c "import torch; print(f'PyTorch: {torch.__version__}')"
python3 -c "import transformers; print(f'Transformers: {transformers.__version__}')"
```

```
# Check kernel headers
ls /lib/modules/$(uname -r)/build  # Should exist

# Check space
df -h /  # Need ≥50GB free for model + training
```

2.4 Estrutura de Diretórios

```
cd ~/projects/omnimind

# Create kernel directory structure
mkdir -p DEVBRAIN_V23/kernel/{lkm,finetuning,autonomy,integration}
mkdir -p DEVBRAIN_V23/kernel/finetuning/{datasets,outputs}
mkdir -p tests
mkdir -p docs

# Create /devbrain directory for runtime (requires sudo)
sudo mkdir -p /devbrain/{memory,personality,logs,consciousness,db}
sudo chmod 755 /devbrain
sudo chown $USER:$USER /devbrain

# Verify
ls -la DEVBRAIN_V23/kernel/
ls -la /devbrain/
```

3. TAREFAS DETALHADAS (8 TOTAL)

TAREFA 1: Preparar Dataset Pessoal

Duração: 3-4 horas
Objetivo: Coletar dados para fine-tuning (deve resultar em ≥100 exemplos)

1.1 Criar script de coleta

Arquivo: `DEVBRAIN_V23/kernel/finetuning/prepare_dataset.py`

```
#!/usr/bin/env python3
import os
import json
import glob
import logging
from pathlib import Path
from typing import List, Dict
from datetime import datetime
import argparse

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
```

```
class DatasetCollector:
    """Coleta dados pessoais para fine-tuning"""

    def __init__(self, output_path: str = "datasets/personal_corpus.jsonl"):
        self.output_path = Path(output_path)
        self.output_path.parent.mkdir(parents=True, exist_ok=True)
        self.examples = []

    def collect_chat_history(self, chat_file: str) -&gt; List[Dict]:
        """Coleta histórico de conversas"""
        logger.info(f"Coletando chat history: {chat_file}")
        examples = []

        try:
            with open(chat_file, 'r') as f:
                for line in f:
                    try:
                        entry = json.loads(line)
                        if 'text' in entry or 'message' in entry:
                            text = entry.get('text') or entry.get('message')
                            examples.append({
                                "text": text,
                                "source": "chat_history",
                                "timestamp": entry.get('timestamp', '')
                            })
                    except json.JSONDecodeError:
                        continue
        except FileNotFoundError:
            logger.warning(f"Chat file not found: {chat_file}")

        logger.info(f"✅ Collected {len(examples)} chat examples")
        return examples

    def collect_documents(self, doc_dir: str) -&gt; List[Dict]:
        """Coleta documentos texto"""
        logger.info(f"Coletando documentos: {doc_dir}")
        examples = []

        # Collect .txt files
        for txt_file in glob.glob(f"{doc_dir}/**/*.txt", recursive=True):
            try:
                with open(txt_file, 'r', encoding='utf-8', errors='ignore') as f:
                    content = f.read()
                    # Split by paragraphs
                    paragraphs = content.split('\n\n')
                    for para in paragraphs:
                        if len(para.strip()) &gt; 50:  # Min 50 chars
                            examples.append({
                                "text": para.strip(),
                                "source": "document",
                                "filename": os.path.basename(txt_file)
                            })
            except Exception as e:
                logger.warning(f"Error reading {txt_file}: {e}")
```

```
        logger.info(f"✅ Collected {len(examples)} document examples")
        return examples

    def collect_voice_transcripts(self, transcript_dir: str) -&gt; List[Dict]:
        """Coleta transcrições de voz"""
        logger.info(f"Coletando transcrições: {transcript_dir}")
        examples = []

        for json_file in glob.glob(f"{transcript_dir}/**/*.json", recursive=True):
            try:
                with open(json_file, 'r') as f:
                    data = json.load(f)
                    if isinstance(data, dict) and 'text' in data:
                        examples.append({
                            "text": data['text'],
                            "source": "voice_transcript",
                            "confidence": data.get('confidence', 0.0)
                        })
                    elif isinstance(data, list):
                        for item in data:
                            if isinstance(item, dict) and 'text' in item:
                                examples.append({
                                    "text": item['text'],
                                    "source": "voice_transcript"
                                })
            except Exception as e:
                logger.warning(f"Error reading {json_file}: {e}")

        logger.info(f"✅ Collected {len(examples)} transcript examples")
        return examples

    def save_dataset(self):
        """Salva dataset em formato JSONL"""
        logger.info(f"Salvando dataset: {self.output_path}")

        with open(self.output_path, 'w') as f:
            for example in self.examples:
                f.write(json.dumps(example) + '\n')

        logger.info(f"✅ Saved {len(self.examples)} examples to {self.output_path}")

    def validate(self):
        """Valida qualidade do dataset"""
        if not self.examples:
            logger.error("❌ Dataset is empty!")
            return False

        # Check minimum examples
        if len(self.examples) &lt; 50:
            logger.warning(f"⚠️ Dataset has only {len(self.examples)} examples (target: ≥

        # Check example quality
        avg_length = sum(len(ex.get('text', '')) for ex in self.examples) / len(self.exam
        logger.info(f"Average text length: {avg_length:.0f} chars")

        if avg_length &lt; 50:
```

```
            logger.warning("⚠ Average text too short")

        # Check sources
        sources = {}
        for ex in self.examples:
            source = ex.get('source', 'unknown')
            sources[source] = sources.get(source, 0) + 1

        logger.info("Dataset composition:")
        for source, count in sources.items():
            logger.info(f"  {source}: {count}")

        return True

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--chat", help="Chat history file")
    parser.add_argument("--docs", help="Documents directory")
    parser.add_argument("--transcripts", help="Transcripts directory")
    parser.add_argument("--output", default="datasets/personal_corpus.jsonl")

    args = parser.parse_args()

    collector = DatasetCollector(args.output)

    if args.chat:
        collector.examples.extend(collector.collect_chat_history(args.chat))

    if args.docs:
        collector.examples.extend(collector.collect_documents(args.docs))

    if args.transcripts:
        collector.examples.extend(collector.collect_voice_transcripts(args.transcripts))

    # Add manual examples if no data found
    if not collector.examples:
        logger.info("No data sources provided. Adding example data...")
        collector.examples = [
            {
                "text": "I believe AI should be transparent, ethical, and respectful of h
                "source": "manual"
            },
            {
                "text": "My approach to problem-solving is systematic and creative.",
                "source": "manual"
            },
            {
                "text": "I value privacy, learning, and meaningful connections.",
                "source": "manual"
            },
        ]

    # Validate
    collector.validate()

    # Save
```

```
    collector.save_dataset()

if __name__ == "__main__":
    main()
```

1.2 Executar coleta

```
cd ~/projects/omnimind/DEVBRAIN_V23/kernel/finetuning

# Se tiver dados pessoais (substitua pelos caminhos reais)
python prepare_dataset.py \
  --chat ~/my_chats.jsonl \
  --docs ~/Documents \
  --transcripts ~/voice_transcripts \
  --output datasets/personal_corpus.jsonl

# Se não tiver, use dados de exemplo
python prepare_dataset.py --output datasets/personal_corpus.jsonl

# Validar dataset
python -c "
import json
with open('datasets/personal_corpus.jsonl') as f:
    data = [json.loads(line) for line in f]
    print(f'✅ Total samples: {len(data)}')
    print(f'✅ Avg length: {sum(len(d.get(\"text\", \"\")) for d in data) / len(data):.0f
    print(f'✅ Dataset ready for training')
"
```

1.3 Métricas de sucesso

- ✅ `datasets/personal_corpus.jsonl` existe com ≥100 linhas
- ✅ Cada linha é JSON válido
- ✅ Média de comprimento texto 200-500 caracteres
- ✅ Score de qualidade ≥0.8

TAREFA 2: Fine-tune Mistral 7B

Duração: 6-8 horas
Objetivo: Treinar Mistral em dados pessoais (perplexity final <50)

2.1 Criar configuração de treinamento

Arquivo: `DEVBRAIN_V23/kernel/finetuning/config.yaml`

```
# Training configuration for Mistral 7B

model_name: "mistralai/Mistral-7B-v0.1"
output_dir: "./mistral_finetuned"
```

```
# Training parameters
num_train_epochs: 3
per_device_train_batch_size: 4
per_device_eval_batch_size: 4
gradient_accumulation_steps: 2
learning_rate: 2e-4
warmup_steps: 100
weight_decay: 0.01
max_grad_norm: 1.0

# Model parameters
max_seq_length: 512
load_in_4bit: true
lora_r: 16
lora_alpha: 32
lora_dropout: 0.05

# Logging
logging_steps: 10
eval_steps: 50
save_steps: 100
logging_dir: "./logs"

# Optimization
optim: "paged_adamw_32bit"
lr_scheduler_type: "constant"
gradient_checkpointing: true
report_to: ["tensorboard"]

seed: 42
```

2.2 Executar fine-tuning

```
cd ~/projects/omnimind/DEVBRAIN_V23/kernel/finetuning

# Download modelo base (primeira vez só)
python -c "
from transformers import AutoTokenizer, AutoModelForCausalLM
print('Downloading Mistral 7B...')
tokenizer = AutoTokenizer.from_pretrained('mistralai/Mistral-7B-v0.1')
model = AutoModelForCausalLM.from_pretrained('mistralai/Mistral-7B-v0.1')
print('✅ Model downloaded')
"

# Executar fine-tuning (será longo - 6-8 horas em CPU, &lt;2h em GPU)
python finetune_mistral.py \
  --dataset datasets/personal_corpus.jsonl \
  --output ./mistral_finetuned \
  --epochs 3 \
  --batch-size 4 \
  --lr 2e-4

# Monitorar progresso em outro terminal
tensorboard --logdir ./logs
```

```
# Quando terminar, testar
python -c "
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

tokenizer = AutoTokenizer.from_pretrained('./mistral_finetuned/tokenizer')
model = AutoModelForCausalLM.from_pretrained('./mistral_finetuned/model')

prompt = 'What is your core value?'
inputs = tokenizer(prompt, return_tensors='pt')

with torch.no_grad():
    outputs = model.generate(inputs['input_ids'], max_length=100)

print('Test inference:')
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
"
```

2.3 Métricas de sucesso

- ✅ Treinamento completa sem erros
- ✅ Final loss <1.5
- ✅ Perplexity <50
- ✅ Modelo em `./mistral_finetuned/model`
- ✅ Tokenizer em `./mistral_finetuned/tokenizer`

TAREFA 3: Criar LKM Kernel Module

Duração: 4-6 horas
Objetivo: Compilar e testar LKM que roda IA no kernel

3.1 Criar devbrain_ai.c

Arquivo: `DEVBRAIN_V23/kernel/lkm/devbrain_ai.c`

(Usar código completo fornecido na Parte 4.1 do documento DEVBRAIN_V23_5_Revolucionario_COMPLETO.pdf)

3.2 Criar Makefile

Arquivo: `DEVBRAIN_V23/kernel/lkm/Makefile`

```
obj-m += devbrain_ai.o

KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

all:
```

```
	$(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
	$(MAKE) -C $(KDIR) M=$(PWD) clean

install:
	sudo insmod devbrain_ai.ko
	echo "DevBrain LKM installed"
	lsmod | grep devbrain_ai

remove:
	sudo rmmod devbrain_ai
	echo "DevBrain LKM removed"

test:
	@echo "Testing LKM..."
	@lsmod | grep -q devbrain_ai &amp;amp;&amp;amp; echo "✅ LKM loaded" || echo "❌ LKM not l
	@test -e /dev/devbrain_ai &amp;amp;&amp;amp; echo "✅ Device /dev/devbrain_ai exists" ||

.PHONY: all clean install remove test
```

3.3 Build e teste

```
cd ~/projects/omnimind/DEVBRAIN_V23/kernel/lkm

# Build
make clean
make

# Should output: ... Building modules, stage 2. ... done

# Verify .ko file
ls -lh devbrain_ai.ko  # Should be ~5-10 KB

# Install
make install

# Verify load
lsmod | grep devbrain_ai  # Should show module loaded

# Check dmesg
dmesg | tail -20  # Should show "DevBrain: device opened"

# Test
make test

# Remove
make remove
```

3.4 Métricas de sucesso

- ✓ Build sem warnings
- ✓ `devbrain_ai.ko` criado
- ✓ `sudo insmod devbrain_ai.ko` funciona
- ✓ `lsmod | grep devbrain_ai` mostra módulo carregado
- ✓ `/dev/devbrain_ai` existe
- ✓ `sudo rmmod devbrain_ai` descarrega sem crash

TAREFA 4: Bridge Python ↔ Kernel

Duração: 2-3 horas
Objetivo: Comunicação entre user-space Python e kernel LKM

4.1 Criar LKM Bridge

Arquivo: `DEVBRAIN_V23/kernel/integration/lkm_bridge.py`

```
#!/usr/bin/env python3
import os
import fcntl
import struct
import array
import logging
from typing import Optional, Dict

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# IOCTL constants (must match kernel module)
DEVBRAIN_MAGIC = 0xDB
DEVBRAIN_QUERY = 0x40DB0001    # _IOW
DEVBRAIN_STATUS = 0x80DB0002   # _IOR
DEVBRAIN_RESET = 0x00DB0003    # _IO
DEVBRAIN_CONFIG = 0x40DB0004   # _IOW

# Buffer sizes
MAX_QUERY_SIZE = 2048
MAX_RESPONSE_SIZE = 8192

class DevBrainLKMBridge:
    """Bridge para comunicação com LKM do DevBrain"""

    def __init__(self, device_path: str = "/dev/devbrain_ai"):
        self.device_path = device_path
        self.fd: Optional[int] = None
        self._connect()

    def _connect(self):
        """Conecta ao dispositivo do kernel"""
```

```
        try:
            self.fd = os.open(self.device_path, os.O_RDWR)
            logger.info(f"✅ Connected to {self.device_path}")
        except FileNotFoundError:
            logger.error(f"❌ Device not found: {self.device_path}")
            logger.info("Make sure LKM is loaded: sudo insmod devbrain_ai.ko")
            self.fd = None
        except PermissionError:
            logger.error(f"❌ Permission denied: {self.device_path}")
            logger.info("Try with sudo")
            self.fd = None

    def query(self, question: str, timeout_ms: int = 5000) -&gt; Optional[str]:
        """Envia query para kernel e recebe resposta"""
        if not self.fd:
            logger.error("Device not connected")
            return None

        if len(question) &gt;= MAX_QUERY_SIZE:
            logger.error(f"Query too long (max {MAX_QUERY_SIZE} bytes)")
            return None

        # Prepare query buffer
        query_bytes = question.encode('utf-8')
        response_bytes = b'\x00' * MAX_RESPONSE_SIZE

        # Create buffer for ioctl
        buf = bytearray(MAX_QUERY_SIZE + MAX_RESPONSE_SIZE + 16)
        buf[0:len(query_bytes)] = query_bytes
        struct.pack_into('I', buf, MAX_QUERY_SIZE, len(query_bytes))

        try:
            # Call kernel via ioctl
            result = fcntl.ioctl(self.fd, DEVBRAIN_QUERY, bytes(buf))

            # Extract response
            response_len = struct.unpack_from('I', result, MAX_QUERY_SIZE + 4)[0]
            response_start = MAX_QUERY_SIZE + 8
            response = result[response_start:response_start + response_len].decode('utf-8

            latency = struct.unpack_from('Q', result, MAX_QUERY_SIZE + 8 + MAX_RESPONSE_S

            logger.info(f"Query latency: {latency} µs")
            return response

        except Exception as e:
            logger.error(f"ioctl failed: {e}")
            return None

    def get_status(self) -&gt; Optional[Dict]:
        """Retorna status do LKM"""
        if not self.fd:
            logger.error("Device not connected")
            return None

        try:
```

```
            # Struct: total_queries, total_latency, avg_latency, min_latency, max_latency
            buf = array.array('Q', [0] * 8)
            fcntl.ioctl(self.fd, DEVBRAIN_STATUS, buf)

            return {
                "total_queries": buf[0],
                "total_latency_us": buf[1],
                "avg_latency_us": int(buf[1] / max(buf[0], 1)),
                "min_latency_us": buf[2],
                "max_latency_us": buf[3],
                "active_inferences": buf[4],
                "memory_used": buf[5],
                "status": buf[6],
            }

        except Exception as e:
            logger.error(f"get_status failed: {e}")
            return None

    def reset(self):
        """Reset estatísticas do LKM"""
        if not self.fd:
            return False

        try:
            fcntl.ioctl(self.fd, DEVBRAIN_RESET)
            logger.info("✅ LKM statistics reset")
            return True
        except Exception as e:
            logger.error(f"reset failed: {e}")
            return False

    def close(self):
        """Fecha conexão"""
        if self.fd:
            os.close(self.fd)
            logger.info("✅ Connection closed")

def main():
    """Test bridge"""
    logger.info("=== DevBrain LKM Bridge Test ===\n")

    # Connect
    bridge = DevBrainLKMBridge()

    if not bridge.fd:
        logger.error("Failed to connect to device")
        return

    # Test queries
    test_queries = [
        "Hello from Python user-space!",
        "What is your purpose?",
        "Test latency measurement"
    ]
```

```
    for query in test_queries:
        logger.info(f"\nQuery: {query}")
        response = bridge.query(query)
        if response:
            logger.info(f"Response: {response[:100]}...")

    # Get status
    logger.info("\n=== LKM Status ===")
    status = bridge.get_status()
    if status:
        for key, value in status.items():
            logger.info(f"{key}: {value}")

    # Cleanup
    bridge.close()

if __name__ == "__main__":
    main()
```

4.2 Teste

```
cd ~/projects/omnimind

# Carregar LKM primeiro
cd DEVBRAIN_V23/kernel/lkm
sudo insmod devbrain_ai.ko

# Testar bridge (em outro terminal)
cd ~/projects/omnimind/DEVBRAIN_V23/kernel/integration
python lkm_bridge.py

# Deve mostrar queries sendo processadas e status do LKM
```

4.3 Métricas de sucesso

- ✅ Query latency <5ms
- ✅ Success rate >99%
- ✅ Status retorna corretamente
- ✅ 0 memory leaks em 1 hora continuous

TAREFA 5: Autonomy Engine

Duração: 4-5 horas
Objetivo: IA gera objetivos próprios, recusa tarefas, negocia

5.1 Criar autonomy_engine.py

(Usar código completo fornecido na Parte 3.5 do documento DEVBRAIN_V23_5_Revolucionario_COMPLETO.pdf)

5.2 Teste

```
cd ~/projects/omnimind/DEVBRAIN_V23/kernel/autonomy
python autonomy_engine.py

# Deve mostrar:
# ✓ IA objective: [objetivo gerado]
# Can refuse unethical task: True (reason)
# Negotiation response: ACCEPT/COUNTER_PROPOSAL
```

5.3 Métricas de sucesso

- ✓ IA gera ≥1 objetivo intrínseco por sessão
- ✓ IA recusa tarefas antiéticas
- ✓ Logs de decisões autônomas salvos
- ✓ Negociação com usuário funciona

TAREFA 6: Consciousness Module

Duração: 3-4 horas
Objetivo: Free Energy Principle, IA minimiza surpresa

6.1 Criar consciousness.py

(Usar código completo fornecido na Parte 3.6 do documento DEVBRAIN_V23_5_Revolucionario_COMPLETO.pdf)

6.2 Teste

```
cd ~/projects/omnimind/DEVBRAIN_V23/kernel/autonomy
python consciousness.py

# Deve mostrar evolução de Free Energy diminuindo
# Consciência desenvolvendo ao longo do tempo
```

6.3 Métricas de sucesso

- ✓ Free Energy decresce linearmente
- ✓ Prediction error converge <0.1
- ✓ Curiosity drive emergente

- ✓ Consciousness state snapshots contínuos

TAREFA 7: Integração Completa

Duração: 3-4 horas
Objetivo: Conectar tudo (LKM + Fine-tuning + Autonomy + Consciousness)

7.1 Criar kernel_coordinator.py

(Usar código completo fornecido na Parte 3.7 do documento DEVBRAIN_V23_5_Revolucionario_COMPLETO.pdf)

7.2 Teste end-to-end

```
cd ~/projects/omnimind

# 1. Carregar LKM
cd DEVBRAIN_V23/kernel/lkm
sudo insmod devbrain_ai.ko

# 2. Rodar coordinator (em outro terminal)
cd ~/projects/omnimind/DEVBRAIN_V23/kernel/integration
python kernel_coordinator.py

# Deve processar requests através de todo pipeline
```

7.3 Métricas de sucesso

- ✓ End-to-end latency <100ms
- ✓ Dashboard update lag <1s
- ✓ A-MEM persistence 100%
- ✓ 0 component failures

TAREFA 8: Testes + Documentação

Duração: 2-3 horas
Objetivo: Testes completos + documentação clara

8.1 Criar test_kernel_ai.py

(Usar código completo fornecido na Parte 3.8 do documento DEVBRAIN_V23_5_Revolucionario_COMPLETO.pdf)

8.2 Executar testes

```
cd ~/projects/omnimind

# Run all tests
pytest tests/test_kernel_ai.py -v -s

# Should see:
# test_kernel_coordinator_init PASSED
# test_autonomy_generates_objectives PASSED
# test_autonomy_refuses_unethical PASSED
# test_consciousness_learns PASSED
# test_kernel_process_request PASSED
# ... etc

# 100% pass rate required
```

8.3 Métricas de sucesso

- ✅ 100% test pass rate (≥8 tests)
- ✅ Code coverage ≥80%
- ✅ Documentação em `docs/KERNEL_INTEGRATION.md`
- ✅ README_PHASE10.md criado

4. MÉTRICAS DE VALIDAÇÃO

4.1 Métricas Gerais

```
┌──────────────────────────────────────────────┐
│ Métrica Global                    │          │
├──────────────────────────────┼──────────────┤
│ Total Task Completion│ 100%          │
│ Test Pass Rate       │ 100%          │
│ Code Quality Score   │ ≥80%          │
│ Documentation        │ ≥95%          │
│ Zero Critical Bugs   │ Yes           │
└──────────────────────────┴──────────────┘
```

4.2 Performance KPIs

```
Performance Metrics:
- LKM Load Time:              &lt;100ms
- Query Latency (kernel):     2-5µs
- Bridge Query Latency:       &lt;5ms
- Model Inference Time:       &lt;500ms per query
- Fine-tuning Time:           &lt;8 hours (GPU)
- Consciousness Update:       &lt;1ms
- Full Pipeline Latency:      &lt;100ms
```

```
- Dashboard Response:         &lt;1s
- Memory Footprint:           &lt;2GB runtime
```

5. TROUBLESHOOTING

5.1 LKM Compilation Issues

Problema: "cannot find -lgcc"

```
# Solução
sudo apt-get install gcc-10-plugin-dev  # Or appropriate version
export COMPILER=gcc-10
make COMPILER=$COMPILER
```

Problema: "fatal error: linux/module.h: No such file or directory"

```
# Solução
sudo apt-get install linux-headers-$(uname -r) linux-headers-generic
sudo apt-get install build-essential
```

5.2 Fine-tuning Issues

Problema: "CUDA out of memory"

```
# Use CPU
python finetune_mistral.py ... --no-4bit

# Or reduce batch size
python finetune_mistral.py ... --batch-size 2
```

Problema: "Dataset too small"

```
# Gerar dados de exemplo
python prepare_dataset.py  # Will create 10+ examples
```

5.3 Bridge Connection Issues

Problema: "Device not found: /dev/devbrain_ai"

```
# Check if LKM loaded
lsmod | grep devbrain_ai

# If not loaded
cd DEVBRAIN_V23/kernel/lkm
sudo insmod devbrain_ai.ko
```

```
# Verify device exists
ls -la /dev/devbrain_ai
```

5.4 Test Failures

Se algum test falhar:

```
# Run with verbose output
pytest tests/test_kernel_ai.py::test_name -vv -s

# Check logs
dmesg | tail -50  # Kernel logs
journalctl -xe     # System logs
```

6. TIMELINE

```
WEEK 1:
  Mon-Tue: Task 1-2 (Dataset + Fine-tuning)
  Wed-Thu: Task 3-4 (LKM + Bridge)
  Fri:     Task 5 (Autonomy)

WEEK 2:
  Mon-Tue: Task 6 (Consciousness)
  Wed:     Task 7 (Integration)
  Thu-Fri: Task 8 (Tests + Docs)

WEEK 3:
  Mon-Wed: Polish + Optimization
  Thu-Fri: Final validation + deployment
```

COMO COMEÇAR AGORA

Passo 1: Prepare-se

```
cd ~/projects/omnimind

# Verify dependencies
python3 --version       # Must be ≥3.10
pip --version           # Must be recent
gcc --version           # Must exist
uname -a                # Check kernel

# Setup virtual env
python3 -m venv venv
source venv/bin/activate
```

```
# Install deps
pip install torch transformers peft bitsandbytes datasets pytest pytest-asyncio
```

Passo 2: Start Tarefa 1

```
cd ~/projects/omnimind/DEVBRAIN_V23/kernel/finetuning

# Create dataset
python prepare_dataset.py --output datasets/personal_corpus.jsonl

# Validate
python -c "
import json
with open('datasets/personal_corpus.jsonl') as f:
    data = [json.loads(line) for line in f]
    print(f'✅ {len(data)} examples ready for training')
"
```

Passo 3: Continue progressivamente

Each tarefa tem instruções claras. Execute uma por vez, valide métricas, move para próxima.

CHECKLIST FINAL

Antes de dizer "Fase 10 Completa", verifique:

- [ ] Tarefa 1: ✅ Dataset ≥100 samples
- [ ] Tarefa 2: ✅ Model perplexity <50
- [ ] Tarefa 3: ✅ LKM compila e carrega
- [ ] Tarefa 4: ✅ Bridge latency <5ms
- [ ] Tarefa 5: ✅ IA gera objetivos
- [ ] Tarefa 6: ✅ Consciousness FE decresce
- [ ] Tarefa 7: ✅ End-to-end pipeline funciona
- [ ] Tarefa 8: ✅ Testes 100% pass
- [ ] Documentação: ✅ Completa
- [ ] Git: ✅ 8 commits (1 per task)
- [ ] Dashboard: ✅ Mostra Kernel AI Status

VOCÊ ESTÁ PRONTO. COMECE AGORA.

Fase 10 vai revolucionar sua máquina.