KTransformers CLI 功能技术报告

1. 概述

KTransformers CLI（命令 `kt`）采用"默会设计"理念，面向使用终端的半专业用户，使其能够在不查阅外部文档的情况下，独立完成模型管理、推理服务启动、模型量化等完整工作流。CLI 的交互设计注重可发现性和一致性，命令结构直观，错误提示清晰。

1.1 核心价值

- **默会设计**: 所有操作通过单一 `kt` 命令完成，子命令结构清晰自然
- **智能配置**: 自动检测硬件环境，根据模型类型自动计算最优参数
- **开箱即用**: 首次运行自动引导配置，支持自动安装 Shell 补全
- **双语支持**: 完整的中英文界面，自动检测系统语言

---

2. CLI 架构设计

2.1 代码结构

```
kt-kernel/python/cli/
├── main.py              # 主入口点，命令注册
├── __init__.py          # 版本定义
├── i18n.py               # 国际化模块（563 行，287+ 条翻译）
├── commands/
│   ├── __init__.py
│   ├── model.py         # 模型管理（410 行）
│   ├── config.py        # 配置管理（168 行）
│   ├── run.py           # 启动推理服务器（896 行）
│   ├── chat.py          # 交互式聊天（438 行）
│   ├── quant.py         # 模型量化（240 行）
│   ├── bench.py         # 基准测试（275 行）
│   ├── doctor.py        # 环境诊断（527 行）
│   ├── version.py       # 版本信息（119 行）
│   └── sft.py           # 微调功能（53 行，即将推出）
├── config/
│   ├── __init__.py
│   └── settings.py      # 配置管理器（312 行）
├── completions/         # 静态 Shell 补全脚本
```

```
|   ├── kt-completion.bash
|   ├── _kt
|   └── kt.fish
└── utils/
    ├── __init__.py
    ├── console.py        # 终端工具函数
    ├── environment.py    # 环境检测（CPU/GPU/内存）
    ├── model_registry.py # 模型注册表（400 行）
    └── sglang_checker.py # SGLang 兼容性检查
```

2.2 命令层次结构

```
kt（主命令）
├── version              # 版本信息
├── run <model>          # 启动推理服务器
├── chat                 # 交互式聊天
├── quant <model>        # 模型量化
├── bench                # 完整基准测试
├── microbench           # 微基准测试
├── doctor               # 环境诊断
├── model（子命令组）
|   ├── （无参数）        # 显示支持的模型
|   ├── download         # 下载模型
|   ├── list             # 列出模型
|   ├── search           # 搜索模型
|   ├── path-list        # 列出存储路径
|   ├── path-add         # 添加存储路径
|   └── path-remove      # 移除存储路径
├── config（子命令组）
|   ├── init             # 初始化设置向导
|   ├── show [key]       # 显示配置
|   ├── set <key> <val>  # 设置配置
|   ├── get <key>        # 获取配置
|   ├── reset            # 重置配置
|   └── path             # 显示配置文件路径
└── sft（子命令组，即将推出）
    ├── train            # 训练模型
    ├── chat             # 与微调模型对话
    └── export           # 导出模型
```

3. 核心命令详解

3.1 kt run – 启动推理服务器

这是最核心的命令，用于启动基于 SGLang + kt-kernel 的模型推理服务器。

3.1.1 基本用法

```
# 使用模型名称或别名启动（推荐）
kt run deepseek-v3
kt run m2
kt run k2

# 交互式选择已下载的模型
kt run
```

设计原则：`kt run` 仅支持模型注册表中列出的模型（运行 `kt model` 查看完整列表）。这样可以集中优化配置，确保最佳用户体验。如需运行其他模型，请通过 `--model-path` 选项指定本地路径，但可能需要手动调整参数。

3.1.2 核心选项

选项	简写	默认值	说明
`--host`	`-H`	`0.0.0.0`	服务器监听地址
`--port`	`-p`	`30000`	服务器端口
`--gpu-experts`	–	自动计算	每层 GPU 专家数量
`--cpu-threads`	–	`cores * 0.8`	CPU 推理线程数
`--numa-nodes`	–	自动检测	NUMA 节点数
`--tensor-parallel-size`	`--tp`	自动检测	张量并行大小
`--model-path`	–	自动查找	自定义模型路径
`--weights-path`	–	–	量化权重路径
`--kt-method`	–	模型默认	KT 量化方法
`--attention-backend`	–	`triton`	注意力后端
`--max-total-tokens`	–	`40000`	最大 token 数
`--max-running-requests`	–	`32`	最大并发请求数
`--chunked-prefill-size`	–	`4096`	分块预填充大小
`--mem-fraction-static`	–	`0.98`	静态内存比例
`--quantize`	`-q`	`false`	启用量化
`--dry-run`	–	`false`	仅显示命令不执行

3.1.3 智能特性

硬件自动检测

- 检测 GPU 型号和显存
- 检测 CPU 型号、核心数、NUMA 拓扑
- 检测系统内存总量

模型自动发现

- 支持模型名称模糊搜索
- 支持别名（如 `m2` → `MiniMax-M2`）
- 在配置的模型存储路径中自动查找
- 如模型未找到，提示使用 `kt model path-add` 添加存储路径

参数自动计算

- 根据模型类型和 GPU 显存自动计算 `kt-num-gpu-experts`
- 计算函数位于 `kt-kernel/python/cli/utils/model_registry.py`
- 内置各模型的计算函数：

```python
# kt-kernel/python/cli/utils/model_registry.py
def compute_deepseek_v3_gpu_experts(tensor_parallel_size, vram_per_gpu_gb):
    per_gpu_gb = 16
    if vram_per_gpu_gb < per_gpu_gb:
        return 0
    total_vram = int(tensor_parallel_size * (vram_per_gpu_gb - per_gpu_gb))
    return total_vram // 3
```

参数优先级

```
CLI 参数 > 模型默认参数 > 配置文件 > 自动检测
```

透传 SGLang 参数

- 支持直接传递未定义的参数给 SGLang
- 例如：`kt run m2 --fp8-gemm-backend triton`

3.1.4 执行流程

```
1. 检查 SGLang 是否安装及 kt-kernel 兼容性
2. 若未指定模型，启动交互式选择
3. 检测硬件配置
4. 解析模型信息（从注册表或本地路径）
5. 确定模型路径和权重路径
6. 计算最优参数
7. 显示配置摘要
8. 启动 SGLang 服务器
```

3.2 kt model – 模型管理

管理模型的下载、列表、搜索和存储路径。

3.2.1 显示支持的模型

```
kt model
```

输出示例：

```
KTransformers Supported Models

┌────────────────────┬────────────┐
│ Model              │ Status     │
├────────────────────┼────────────┤
| DeepSeek-V3-0324   | ✓ Local    |
| DeepSeek-V3.2      | -          |
| DeepSeek-R1-0528   | -          |
| Kimi-K2-Thinking   | ✓ Local    |
| MiniMax-M2         | -          |
└────────────────────┴────────────┘
```

3.2.2 下载模型

```
# 列出可用模型
kt model download --list

# 下载指定模型
kt model download deepseek-v3

# 指定下载路径
kt model download k2-thinking --path /data/models

# 直接从 HuggingFace 下载
kt model download Qwen/Qwen3-30B

# 跳过确认
kt model download m2 -y
```

3.2.3 搜索模型

在支持的模型列表中搜索：

```
# 搜索包含 "deepseek" 的模型
kt model search deepseek

# 搜索包含 "kimi" 的模型
kt model search kimi
```

3.2.4 列出本地模型

```
# 列出所有支持的模型（含本地状态）
kt model list

# 仅列出本地已下载的模型
kt model list --local

# 显示详细路径
kt model list --local --verbose
```

3.2.5 管理存储路径

```
# 查看所有配置的存储路径
kt model path-list

# 添加新路径
kt model path-add /data/models

# 移除路径
kt model path-remove /old/path
```

3.2.6 支持的模型

模型名称	别名	类型	默认方法
DeepSeek-V3-0324	dsv3, deepseek3, v3-0324	MoE	FP8
DeepSeek-V3.2	dsv3.2, deepseek3.2, v3.2	MoE	FP8
DeepSeek-R1-0528	dsr1, r1, r1-0528	MoE	FP8
Kimi-K2-Thinking	kimi-thinking, k2-thinking, k2	MoE	RAWINT4
MiniMax-M2	m2	MoE	FP8
MiniMax-M2.1	m2.1	MoE	FP8

3.3 kt chat – 交互式聊天

与运行中的模型服务器进行交互式对话，基于 OpenAI SDK。

3.3.1 基本用法

```
# 连接到默认服务器 (127.0.0.1:30000)
kt chat

# 连接到指定服务器
kt chat --host 192.168.1.100 --port 8080

# 调整生成参数
kt chat --temperature 0.9 --max-tokens 4096

# 设置系统提示词
kt chat --system "You are a helpful assistant."
```

3.3.2 选项说明

选项	简写	默认值	说明
`--host`	`-H`	来自配置	服务器地址
`--port`	`-p`	来自配置	服务器端口
`--model`	`-m`	自动选择	模型名称
`--temperature`	`-t`	`0.7`	采样温度 (0.0–2.0)
`--max-tokens`	–	`2048`	最大生成 token 数
`--system`	`-s`	–	系统提示词
`--stream/--no-stream`	–	`true`	启用/禁用流式输出
`--save-history/--no-save-history`	–	`true`	保存对话历史
`--history-file`	–	自动生成	历史文件路径

3.3.3 聊天内命令

命令	说明
`/help`, `/h`	显示帮助信息
`/quit`, `/exit`, `/q`	退出聊天
`/clear`, `/c`	清空对话历史
`/history`, `/hist`	显示对话历史
`/info`, `/i`	显示当前设置
`/retry`, `/r`	重新生成最后一条回复

3.3.4 对话历史

对话历史自动保存到 `~/.ktransformers/chat_history/chat_YYYYMMDD_HHMMSS.json`：

```
{
  "model": "DeepSeek-V3-0324",
```

```
  "timestamp": "2025-01-04T12:34:56",
  "messages": [
    {"role": "user", "content": "Hello!"},
    {"role": "assistant", "content": "Hello! How can I help you today?"}
  ]
}
```

3.4 kt config – 配置管理

管理 CLI 的配置文件。

3.4.1 配置文件位置

```
~/.ktransformers/config.yaml
```

3.4.2 配置子命令

```
# 初始化/重新运行设置向导
kt config init

# 显示所有配置
kt config show

# 显示特定配置项
kt config show server.port

# 设置配置值
kt config set server.port 8080
kt config set general.language zh

# 获取配置值
kt config get server.port

# 重置为默认值
kt config reset --yes

# 显示配置文件路径
kt config path
```

3.4.3 默认配置结构

```yaml
# 通用设置
general:
  language: auto          # auto, en, zh
  color: true
  verbose: false
  _initialized: true      # 首次运行标记
  _completion_installed: true  # 补全安装标记

# 路径配置
paths:
  models: ~/.ktransformers/models
  cache: ~/.ktransformers/cache
  weights: ""

# 服务器配置
server:
  host: 0.0.0.0
  port: 30000

# 推理配置（通常使用模型默认值）
inference:
  env:
    PYTORCH_ALLOC_CONF: expandable_segments:True
    SGLANG_ENABLE_JIT_DEEPGEMM: "0"

# 下载配置
download:
  mirror: ""              # HuggingFace 镜像
  resume: true
  verify: true

# 高级配置
advanced:
  env: {}                 # 额外环境变量
  sglang_args: []         # 额外 SGLang 参数
  llamafactory_args: []   # 额外 LlamaFactory 参数

# 依赖配置
dependencies:
  sglang:
    source: github        # pypi or github
    repo: https://github.com/kvcache-ai/sglang
    branch: main
```

3.5 kt quant – 模型量化

> **施工中**: 此功能正在开发中

将模型权重量化为 INT4/INT8 格式以进行 CPU 推理。

3.5.1 基本用法

```
# 使用默认设置量化
kt quant deepseek-v3

# 指定量化方法
kt quant /path/to/model --method int8

# 指定输出路径
kt quant m2 --output /data/weights/m2-int4

# 指定输入权重类型
kt quant k2 --method int4 --input-type fp16

# 自定义 CPU 线程
kt quant deepseek-v3 --cpu-threads 32 --numa-nodes 4
```

3.5.2 选项说明

选项	简写	默认值	说明
`--method`	`-m`	`int4`	量化方法 (int4/int8)
`--output`	`-o`	自动生成	输出路径
`--input-type`	`-i`	`fp8`	输入权重类型
`--cpu-threads`	–	CPU 核心数	CPU 线程数
`--numa-nodes`	–	自动检测	NUMA 节点数
`--no-merge`	–	`false`	不合并 safetensor
`--yes`	`-y`	`false`	跳过确认

3.5.3 量化流程

```
1. 解析输入模型路径（支持名称或路径）
2. 自动生成输出路径
3. 检测 CPU 配置
4. 调用 convert_cpu_weights.py 脚本
5. 保存量化后的权重
```

3.6 kt bench / kt microbench – 基准测试

> **施工中**: 此功能正在开发中

运行性能基准测试。

3.6.1 完整基准测试

```
# 运行所有基准测试
kt bench

# 运行特定类型测试
kt bench --type inference
kt bench --type moe
kt bench --type mla
kt bench --type linear
kt bench --type attention

# 指定模型和输出
kt bench --model deepseek-v3 --output results.json

# 设置迭代次数
kt bench --iterations 20
```

3.6.2 微基准测试（即将推出）

```
kt microbench moe --batch-size 4 --seq-len 2048
kt microbench mla -b 1 -s 4096 --iterations 1000
```

3.7 kt doctor – 环境诊断

诊断环境问题并提供修复建议。

3.7.1 基本用法

```
# 标准诊断
kt doctor

# 详细诊断
kt doctor --verbose
```

3.7.2 检查项目

检查项	说明
Python 版本	是否 >= 3.10
CUDA	是否安装及版本
GPU	型号、显存
CPU	型号、核心数、线程数
CPU 指令集	AVX2, AVX512, AMX
NUMA 拓扑	节点数和 CPU 分布
系统内存	总量和可用量
磁盘空间	模型存储路径的可用空间
kt-kernel	是否安装、变体类型
SGLang	安装来源（PyPI/源码）
SGLang kt-kernel 支持	是否支持 `--kt-gpu-prefill-token-threshold`
环境管理器	conda, venv, uv, docker

3.7.3 输出示例

```
KTransformers Environment Diagnostics

┌──────────────────────┬──────────┬──────────────────────────────────────┐
| Check                | Status   | Value                                |
└──────────────────────┴──────────┴──────────────────────────────────────┘
| Python version       | ✓ OK     | 3.11.8                               |
| CUDA availability    | ⚠ Warn   | Not found                            |
| GPU detection        | ⚠ Warn   | No GPU detected                      |
| CPU                  | ✓ OK     | Intel Xeon (32 cores / 64 threads)|
| CPU Instructions     | ✓ OK     | AVX2, AVX512F, AMX-INT8              |
| NUMA Topology        | ✓ OK     | 2 node(s)                            |
| System memory        | ✓ OK     | 256 GB available / 512 GB total      |
| Model Path 1         | ✓ OK     | 500GB available at /data/models      |
| kt-kernel            | ✓ OK     | v0.4.0 (AMX)                         |
| SGLang Source        | ✓ OK     | Source (GitHub: kvcache-ai/sglang)|
| SGLang kt-kernel     | ✓ OK     | Supported                            |
└──────────────────────┴──────────┴──────────────────────────────────────┘

All checks passed! Your environment is ready.
```

3.8 kt version – 版本信息

显示 CLI 和相关包的版本信息。

```
# 基本版本信息
kt version

# 详细版本信息
kt version --verbose
```

输出包括：

- KTransformers CLI 版本
- Python 版本和平台
- CUDA 版本
- kt-kernel 版本和变体
- SGLang 版本和安装来源
- 其他依赖包版本

---

4. 核心功能模块

4.1 国际化 (i18n)

4.1.1 语言检测优先级

```
1. KT_LANG 环境变量（最高优先级）
2. 配置文件中的 general.language 设置
3. 系统 LANG 环境变量
4. 默认英语
```

4.1.2 使用方式

```python
from kt_kernel.cli.i18n import t, get_lang, set_lang

# 获取当前语言
lang = get_lang()  # "en" or "zh"

# 翻译消息
msg = t("welcome")  # "Welcome to KTransformers!" or "欢迎使用 KTransformers! "

# 带参数的翻译
msg = t("install_found", name="conda", version="24.1.0")
# "Found conda (version 24.1.0)"

# 设置语言
set_lang("zh")
```

4.1.3 翻译覆盖

287+ 条翻译消息，覆盖：

- 通用消息（成功、错误、警告等）
- 所有命令的帮助文本
- 所有命令的输出消息
- 首次运行设置向导
- 错误提示和建议

4.2 首次运行设置向导

当用户首次运行 `kt` 命令时（配置文件不存在），自动启动设置向导：

```
┌─────────────────────────────────────────┐
|           kt-cli                        |
|                                         |
|  Welcome to KTransformers CLI!          |
|  欢迎使用 KTransformers CLI!            |
|                                         |
|  Let's set up your preferences.         |
|  让我们设置您的偏好。                   |
└─────────────────────────────────────────┘

Select your preferred language / 选择您的首选语言:

  [1] English
  [2] 中文 (Chinese)

Enter choice / 输入选择 [1]:
```

设置向导功能：

1. 语言选择
2. 模型存储路径选择（自动扫描磁盘空间）
3. 检测已有模型
4. 自动安装 Shell 补全脚本

4.3 模型注册表

4.3.1 ModelInfo 数据结构

```python
@dataclass
class ModelInfo:
    name: str                    # 模型名称
    hf_repo: str                 # HuggingFace 仓库
    aliases: list[str]           # 别名列表
    type: str                    # 模型类型 (moe/dense)
    gpu_vram_gb: float           # GPU 显存需求
    cpu_ram_gb: float            # CPU 内存需求
    default_params: dict         # 默认参数
    description: str             # 描述（英文）
```

```
    description_zh: str           # 描述（中文）
    max_tensor_parallel_size: Optional[int]  # 最大张量并行大小
```

4.3.2 模糊搜索

支持多种搜索方式：

- 精确匹配模型名称
- 精确匹配别名
- 名称包含匹配
- HuggingFace 仓库匹配
- 分词模糊匹配

```
registry.search("v3")      # 匹配 DeepSeek-V3-0324, DeepSeek-V3.2
registry.search("m2")      # 匹配 MiniMax-M2
registry.search("kimi")    # 匹配 Kimi-K2-Thinking
```

4.3.3 本地模型发现

自动在配置的存储路径中查找已下载的模型：

```
local_models = registry.find_local_models(max_depth=3)
# 返回 [(ModelInfo, Path), ...]
```

4.3.4 用户自定义模型

用户可以在 `~/.ktransformers/registry.yaml` 中添加自定义模型：

```
models:
  My-Custom-Model:
    hf_repo: "org/custom-model"
    aliases: ["custom", "cm"]
    type: "moe"
    default_params:
      kt-method: "FP8"
      kt-num-gpu-experts: 2
    description: "My custom model"
    description_zh: "我的自定义模型"
```

4.4 Shell 补全

自动为 Bash、Zsh、Fish 安装补全脚本：

Shell	安装位置	自动加载
Bash	`~/.local/share/bash-completion/completions/kt`	bash-completion 2.0+
Zsh	`~/.zfunc/_kt`	需添加到 fpath
Fish	`~/.config/fish/completions/kt.fish`	自动加载

补全脚本在首次运行时自动安装，支持：

- 命令补全
- 选项补全
- 模型名称补全

4.5 环境检测

4.5.1 CPU 检测

```python
@dataclass
class CPUInfo:
    name: str              # CPU 型号
    cores: int             # 物理核心数
    threads: int           # 逻辑线程数
    numa_nodes: int        # NUMA 节点数
    numa_info: dict        # NUMA 拓扑详情
    instruction_sets: list # 支持的指令集
```

支持的指令集检测：

- AVX, AVX2, AVX512 系列
- AMX 系列 (AMX-INT8, AMX-BF16)
- 其他 x86 扩展指令集

4.5.2 GPU 检测

```python
@dataclass
class GPUInfo:
    name: str          # GPU 型号
    vram_gb: float     # 显存大小 (GB)
```

4.5.3 内存检测

```python
@dataclass
class MemoryInfo:
    total_gb: float          # 总内存 (GB)
    available_gb: float      # 可用内存 (GB)
    frequency_mhz: Optional[int]   # 内存频率
    type: Optional[str]      # 内存类型 (DDR4/DDR5)
```

---

5. 配置详解

5.1 配置文件格式

配置文件采用 YAML 格式，位于 `~/.ktransformers/config.yaml` 。

5.2 配置访问方式

```python
from kt_kernel.cli.config.settings import get_settings

settings = get_settings()

# 点符号访问
port = settings.get("server.port", 30000)

# 设置值
settings.set("server.port", 8080)

# 获取所有配置
all_config = settings.get_all()
```

5.3 模型路径配置

支持单路径和多路径配置：

```yaml
# 单路径
paths:
  models: /data/models

# 多路径
paths:
  models:
    - /data/models
    - /mnt/storage/models
    - ~/models
```

---

6. 环境变量

变量	说明	示例
`KT_LANG`	强制设置语言	`en`，`zh`
`HTTP_PROXY`	HTTP 代理	`http://proxy:8080`
`HTTPS_PROXY`	HTTPS 代理	`http://proxy:8080`
`ALL_PROXY`	所有协议代理	`socks5://proxy:1080`

---

7. 依赖关系

7.1 核心 CLI 依赖

```toml
# pyproject.toml
dependencies = [
    "typer[all]>=0.9.0",    # CLI 框架
    "rich>=13.0.0",         # 终端格式化
    "pyyaml>=6.0",          # 配置处理
```

```
    "httpx>=0.25.0",        # HTTP 客户端
]
```

7.2 可选依赖

```
optional_dependencies = {
    "chat": ["openai>=1.0.0"],  # 聊天功能
}
```

7.3 运行时依赖

运行 `kt run` 需要：

- SGLang（需从源码安装特定分支）
- kt-kernel
- PyTorch
- Transformers

---

8. 使用示例

8.1 完整工作流

```
# 1. 首次使用 - 自动运行设置向导
kt

# 2. 检查环境
kt doctor

# 3. 查看可用模型
kt model

# 4. 下载模型
kt model download deepseek-v3

# 5. 启动推理服务器
kt run deepseek-v3 --tensor-parallel-size 2

# 6. 另开终端，进行聊天
kt chat
```

```
# 7. 量化模型（可选）
kt quant deepseek-v3 --method int4
```

8.2 高级用法

```
# 自定义配置启动
kt run m2 \
  --tensor-parallel-size 4 \
  --gpu-experts 8 \
  --attention-backend flashinfer \
  --max-total-tokens 100000

# 使用本地模型
kt run /data/custom-model \
  --model-path /data/custom-model \
  --kt-method FP8

# Dry run 查看完整命令
kt run k2-thinking --dry-run

# 连接远程服务器聊天
kt chat --host 192.168.1.100 --port 30000 --temperature 0.5

# 性能测试
kt bench --type moe --iterations 100 --output mo_results.json
```

---

9. 开发指南

> **推荐**: 使用 Claude Code 进行 CLI 开发。Claude Code 能够理解代码结构、自动生成命令框架、处理国际化文本，大幅提升开发效率。

9.1 添加新命令

1. 在 `commands/` 目录创建新文件
2. 使用 Typer 定义命令函数
3. 在 `main.py` 中注册命令

```
# commands/new_command.py
import typer
from kt_kernel.cli.i18n import t

app = typer.Typer(help="New command help")

def new_command(
    name: str = typer.Argument(..., help="Name"),
):
    """Command description."""
    print(f"Hello, {name}!")

# main.py
from kt_kernel.cli.commands import new_command
app.command()(new_command.new_command)
```

9.2 添加翻译

在 `i18n.py` 的 `MESSAGES` 字典中添加：

```
MESSAGES = {
    "en": {
        "my_new_key": "My new message",
    },
    "zh": {
        "my_new_key": "我的新消息",
    },
}
```

9.3 添加新模型

在 `model_registry.py` 的 `BUILTIN_MODELS` 列表中添加：

```
ModelInfo(
    name="My-New-Model",
    hf_repo="org/my-new-model",
    aliases=["mnm", "my-model"],
    type="moe",
    default_params={
        "kt-method": "FP8",
        "kt-num-gpu-experts": 2,
    },
    description="My new model",
```

```
    description_zh="我的新模型",
)
```

10. 故障排查

10.1 常见问题

问题	解决方案
SGLang not found	从源码安装：`git clone https://github.com/kvcache-ai/sglang && pip install -e "python[all]"`
kt-kernel not found	运行 `pip install kt-kernel`
命令补全不生效	重启终端或手动 source 补全脚本
模型未找到	检查模型路径配置：`kt model path-list`
GPU 未检测到	检查 NVIDIA 驱动和 CUDA 安装

10.2 获取诊断信息

```
# 运行完整诊断
kt doctor --verbose

# 查看版本信息
kt version --verbose

# 查看配置
kt config show
```

11. 总结

KTransformers CLI 提供了一个功能完整、用户友好的命令行界面：

- **9 个主命令** + 3 个子命令组
- **287+ 条翻译**实现完整双语支持
- **智能参数计算**降低使用门槛
- **丰富的诊断工具**帮助排查问题
- **模块化架构**便于扩展

CLI 的设计理念是"简单的事情简单做，复杂的事情可能做"，通过智能默认值和自动检测，让用户能够快速上手，同时保留足够的灵活性供高级用户使用。