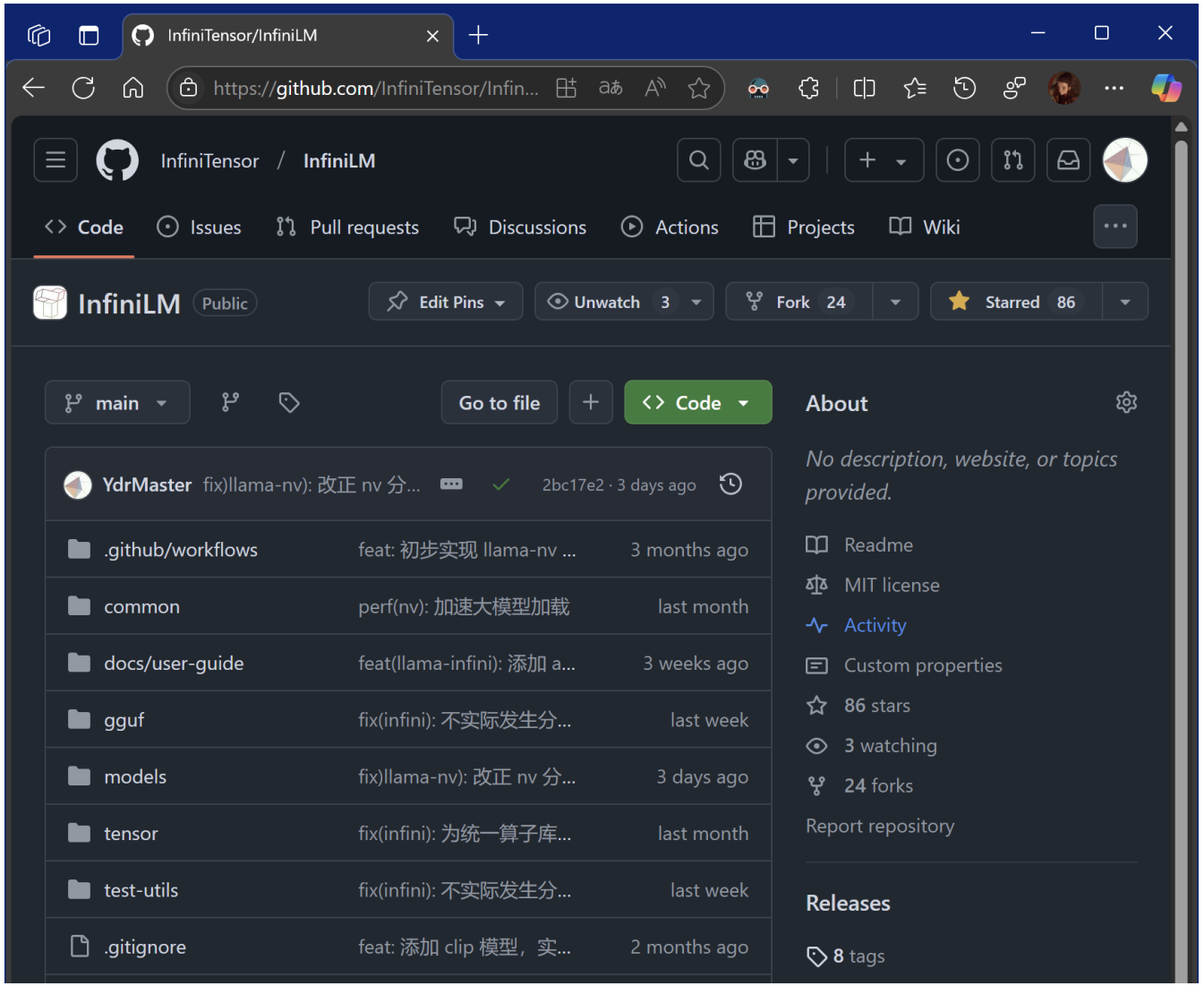
大模型推理引擎 InfiniLM

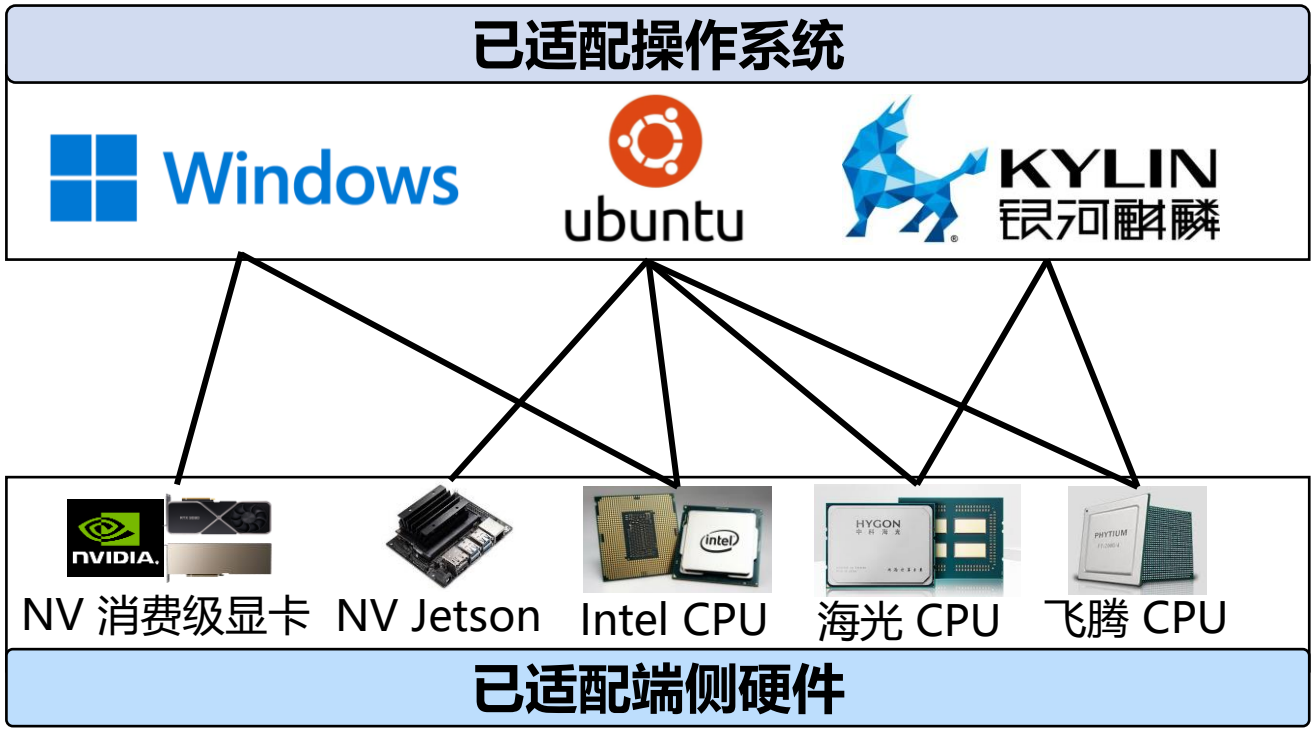
工程概况



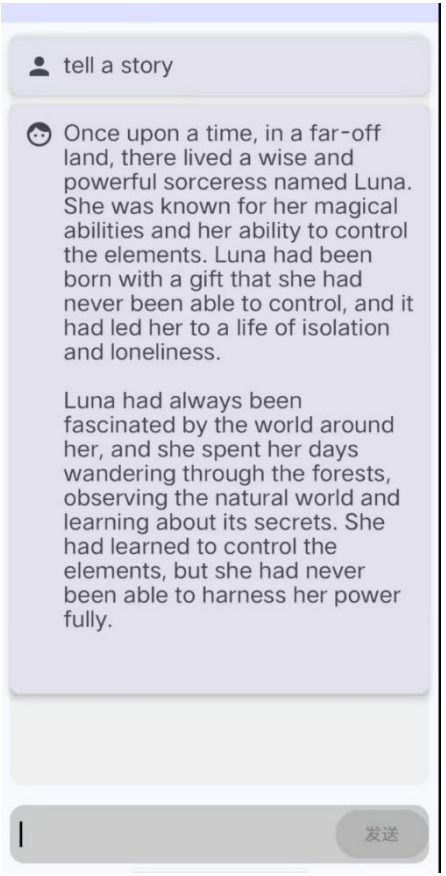
功能	
模型结构	✔ Llama(MiniCPM/qwen2/Mixtral) ✔ GPT2
模型格式	HuggingFace ✔ gguf
精度支持	✔ f32/f16 ggml-quants
硬件支持	✔ x86/Arm CPU ✔ 英伟达 ✔ 华为昇腾 ✔ 寒武纪 ✔ 天数智芯 ✔ 沐曦集成电路 ✔ 摩尔线程 ✔ OpenCL

适配情况

得益于Rust支持能力，项目适用于众多主流操作系统

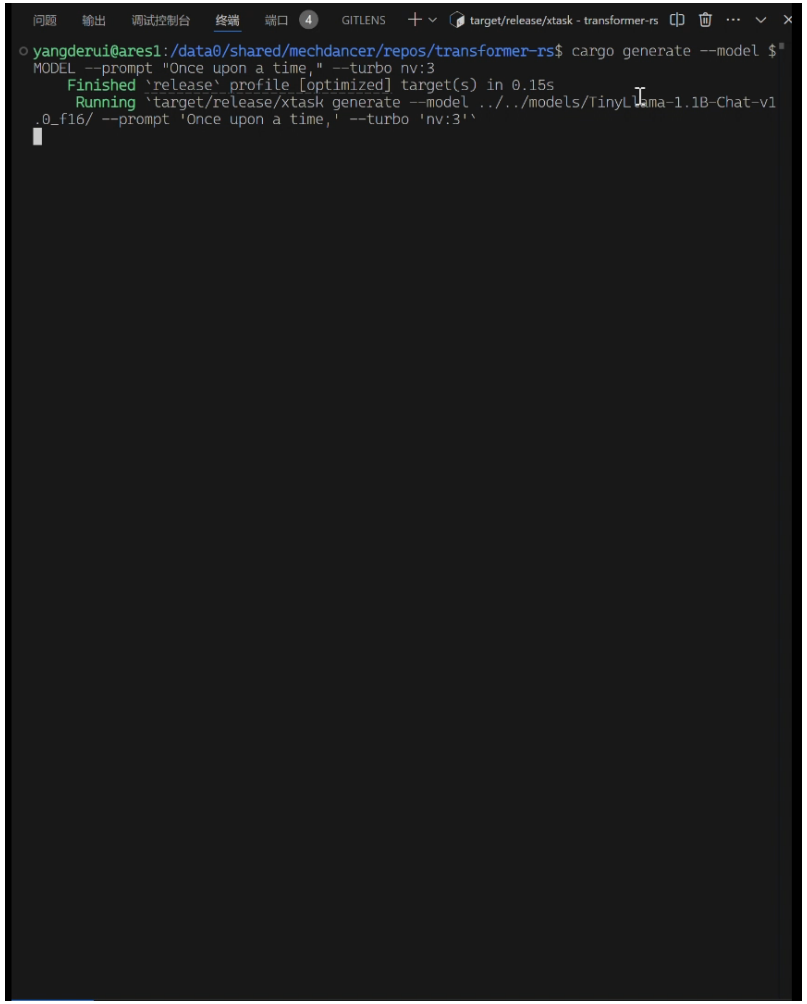


✔桌面/服务器支持矩阵



✔Android+Arm适配

应用案例


问题
输出
调试控制台
终端
端口 4
GITLENS
target/release/xtask - transformer-rs
yangderui@ares1:/data0/shared/mechdancer/repos/transformer-rs$ cargo generate --model $MODEL --prompt "Once upon a time," --turbo nv:3
Finished `release` profile [optimized] target(s) in 0.15s
Running `target/release/xtask generate --model ../../models/TinyLlama-1.1B-Chat-v1.0_f16/ --prompt 'Once upon a time,' --turbo 'nv:3'`


九源社区
不安全
172.22.22.31:8090/plugin/aiChat
点此搜索
九源
首页
共享软件
验证展示
大模型资源社区
更多资源
mechdancer | 个人中心 | 注销
对话助手
写作助手
新对话
九格大模型
Hi，我是启元AI助手
让我协助你了解知识、获得灵感、提升效率。
我可以跟你进行多轮对话与互动
我可以依据提问给出信息并进一步解读
我可以持续学习各行各业的高质量知识
推荐示例
帮我使用Python 来实现线性回归
帮我做一份北京市旅游攻略
做我的翻译助手可以吗
写一个简单的童话故事

Rust 社区贡献

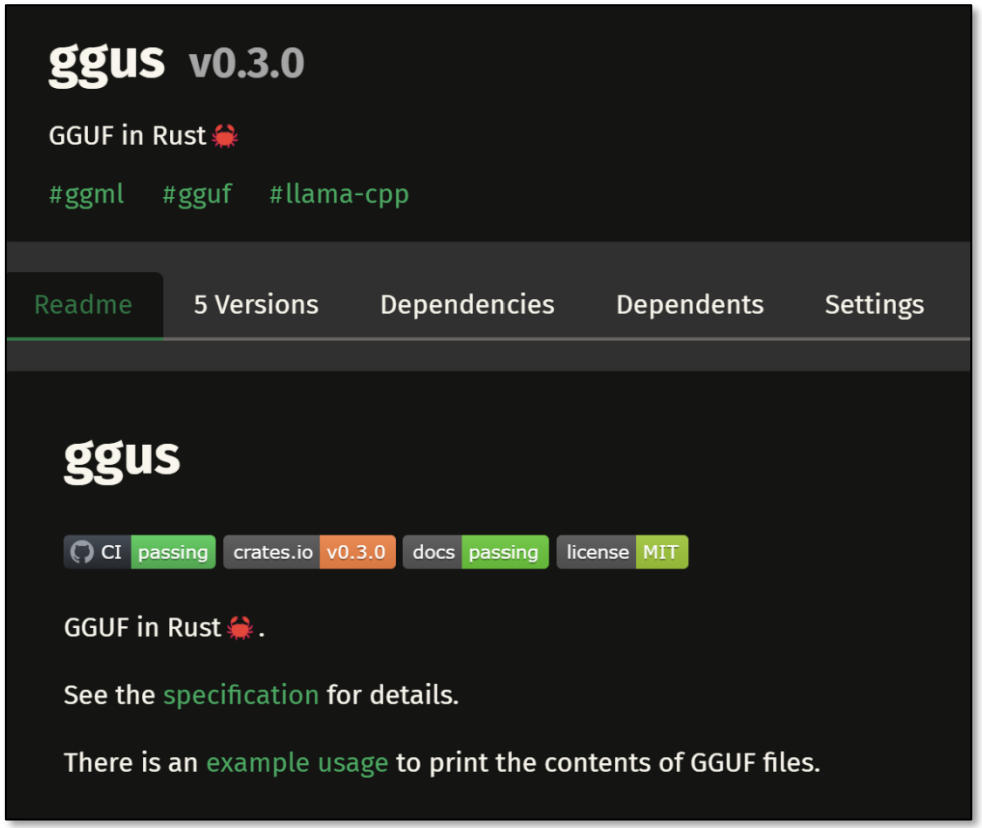
软件包	功能	当前版本	总下载量
build-script-cfg	简化 Rust 条件编译配置	v0.0.0	2222
cndrv	寒武纪 cndrv API 绑定	v0.1.2	2617
search-neuware-tools	寒武纪环境搜索工具	v0.0.0	1433
digit-layout	通用数据类型信息结构	v0.2.0	5490
tokeneer	高性能分词器实现	v0.0.2	1923
context-spore	协处理器上下文管理	v0.0.1	1935
ggus	GGuf 格式定义	v0.4.0	3106
ggml-quants	GGuf 量化类型定义	v0.0.0	484
gguf-utils	GGuf 操作工具	v0.1.1	1064

随着 InfiniLM 开发过程，许多外围功能拆分到独立 crate 并发布

模型格式支持：gguf by llama.cpp

.gitattributes	2.91 kB	
README.md	1.8 kB	
ggml-model-IQ3_M.gguf	3.57 GB	LFS
ggml-model-IQ3_S.gguf	3.5 GB	LFS
ggml-model-IQ3_XS.gguf	3.34 GB	LFS
ggml-model-IQ4_NL.gguf	4.46 GB	LFS
ggml-model-IQ4_XS.gguf	4.25 GB	LFS
ggml-model-Q2_K.gguf	3.01 GB	LFS
ggml-model-Q3_K.gguf	3.81 GB	LFS
ggml-model-Q3_K_L.gguf	4.09 GB	LFS
ggml-model-Q3_K_M.gguf	3.81 GB	LFS
ggml-model-Q3_K_S.gguf	3.49 GB	LFS
ggml-model-Q4_0.gguf	4.43 GB	LFS
ggml-model-Q4_1.gguf	4.87 GB	LFS

ggus 库和 gguf 实用工具



ggus 库发布页

gguf 实用工具发布页

ggus 库和 gguf 操作工具

```rust
let name = GGufFileName::try_from(path).unwrap();
let file = File::open(file).unwrap();
let file = unsafe { Mmap::map(&file).unwrap() };
let gguf = GGuf::new(&file).unwrap();
for (key, kv) in gguf.meta_kvs {
    println!("{key}: {:?} ({}) bytes", kv.ty(), kv.value_bytes().len());
}
```

使用 ggus 解析模型文件

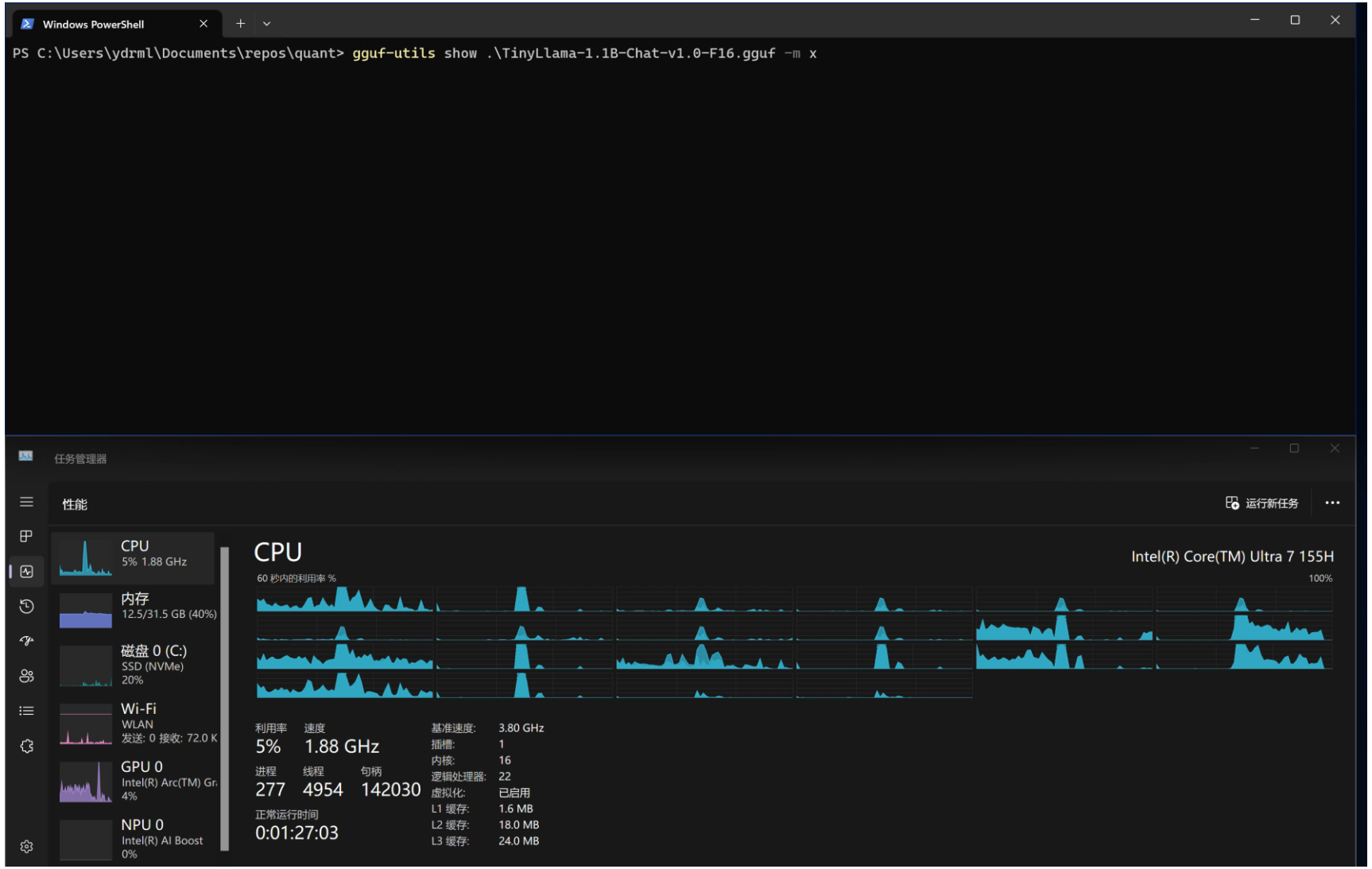
```
> cargo cast MiniCPM3-1.4B-sft-v0.1-F16.gguf --mat q4_1 --log info

——

    Finished `release` profile [optimized] target(s) in 0.09s
     Running `target\release\xtask.exe cast ..\Models\MiniCPM3-1.4B-sft-v0.1-F16.gguf --mat q4_1 --log info`
2024-11-06T12:06:32.356+08:00 INFO  [xtask::utils] run step cast in 15.1µs
2024-11-06T12:06:33.439+08:00 INFO  [xtask::utils] write files in 1.082984s
+-------------------------------+---------------+----------+----------+
|              Path             |      Size     | Meta KVs | Tensors |
+-------------------------------+---------------+----------+----------+
| MiniCPM3-1.4B-sft-v0.1-Q4_1.gguf | 1,007,523,936B |       29 |     314 |
+-------------------------------+---------------+----------+----------+
```

使用 gguf 实用工具完成模型量化

```
> cargo show .\TinyLlama-1.1B-Chat-v1.0-F16.gguf -t x -n 4

——

Finished `release` profile [optimized] target(s) in 0.05s
Running `xtask show .\TinyLlama-1.1B-Chat-v1.0-F16.gguf -t x -n 4`
+-----------------------------------+
| TinyLlama-1.1B-Chat-v1.0-F16.gguf |
+-----------------------------------+

Header
======

✔ Magic   = "GGUF"
✔ Endian  = Little
✔ Version = 3
✔ MetaKVs = 33
✔ Tensors = 201

Meta KV
=======

✔ general.alignment·····u32: 32
✔ general.architecture·····str: `llama`
✔ general.type·····str: `model`
✔ general.name·····str: `TinyLlama 1.1B Chat v1.0`
✔ general.version·····str: `v1.0`
✔ general.finetune·····str: `Chat`
✔ general.basename·····str: `TinyLlama`
✔ general.size_label·····str: `1.1B`
✔ general.license·····str: `apache-2.0`
✔ general.languages·····arr: `en`
✔ llama.block_count·····u64: 22
✔ llama.context_length·····u64: 2048
✔ llama.embedding_length·····u64: 2048
✔ llama.feed_forward_length·····u64: 5632
✔ llama.attention.head_count·····u64: 32
✔ llama.attention.head_count_kv·····u64: 4
✔ llama.rope.freq_base·····f32: 1e4
✔ llama.attention.layer_norm_rms_epsilon··f32: 1e-5
✔ general.file_type·····u32: 1
✔ llama.vocab_size·····u32: 32000
✔ llama.rope.dimension_count·····u64: 64
✔ tokenizer.ggml.model·····str: `llama`
✔ tokenizer.ggml.pre·····str: `default`
✔ tokenizer.ggml.tokens·····arr: [`<unk>`, `<s>`, `</s>`, `<0x00>`, …(31996 more)]
✔ tokenizer.ggml.scores·····arr: [-1000, -1000, -1000, 0, …(31996 more)]
✔ tokenizer.ggml.token_type·····arr: [3, 3, 3, 6, …(31996 more)]
✔ tokenizer.ggml.bos_token_id·····u32: 1
✔ tokenizer.ggml.eos_token_id·····u32: 2
✔ tokenizer.ggml.unknown_token_id·····u32: 0
✔ tokenizer.ggml.padding_token_id·····u32: 2
✔ tokenizer.chat_template·····str:
  +--
  | {%- for message in messages -%}
  | {%- if message['role'] == 'user' -%}
  | {{ '◁user▷
  | ' + message['content'] + eos_token }}
  | {%- elif message['role'] == 'system' -%}
  | {{ '◁system▷
  | ' + message['content'] + eos_token }}
  | {%- elif message['role'] == 'assistant' -%}
  | {{ '◁assistant▷
  | ' + message['content'] + eos_token }}
  | {%- endif -%}
  | {%- if loop.last and add_generation_prompt -%}
  | {{ '◁assistant▷
  | ' }}
  | {%- endif -%}
  | {%- endfor -%}
  +--
✔ general.quantization_version·····u32: 2
```

模型信息浏览

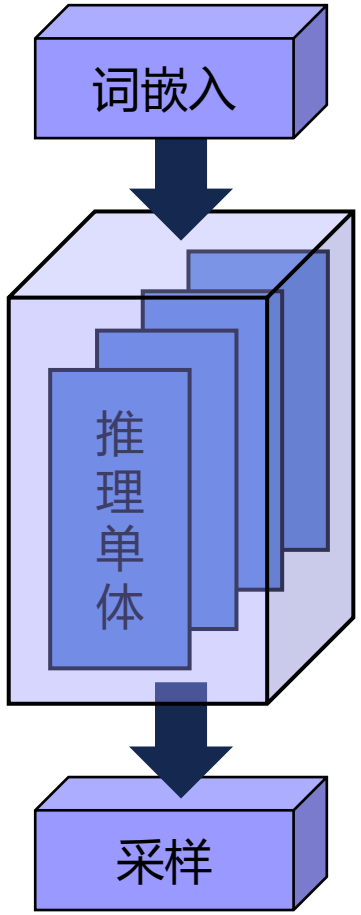
ggus 库和 gguf 操作工具



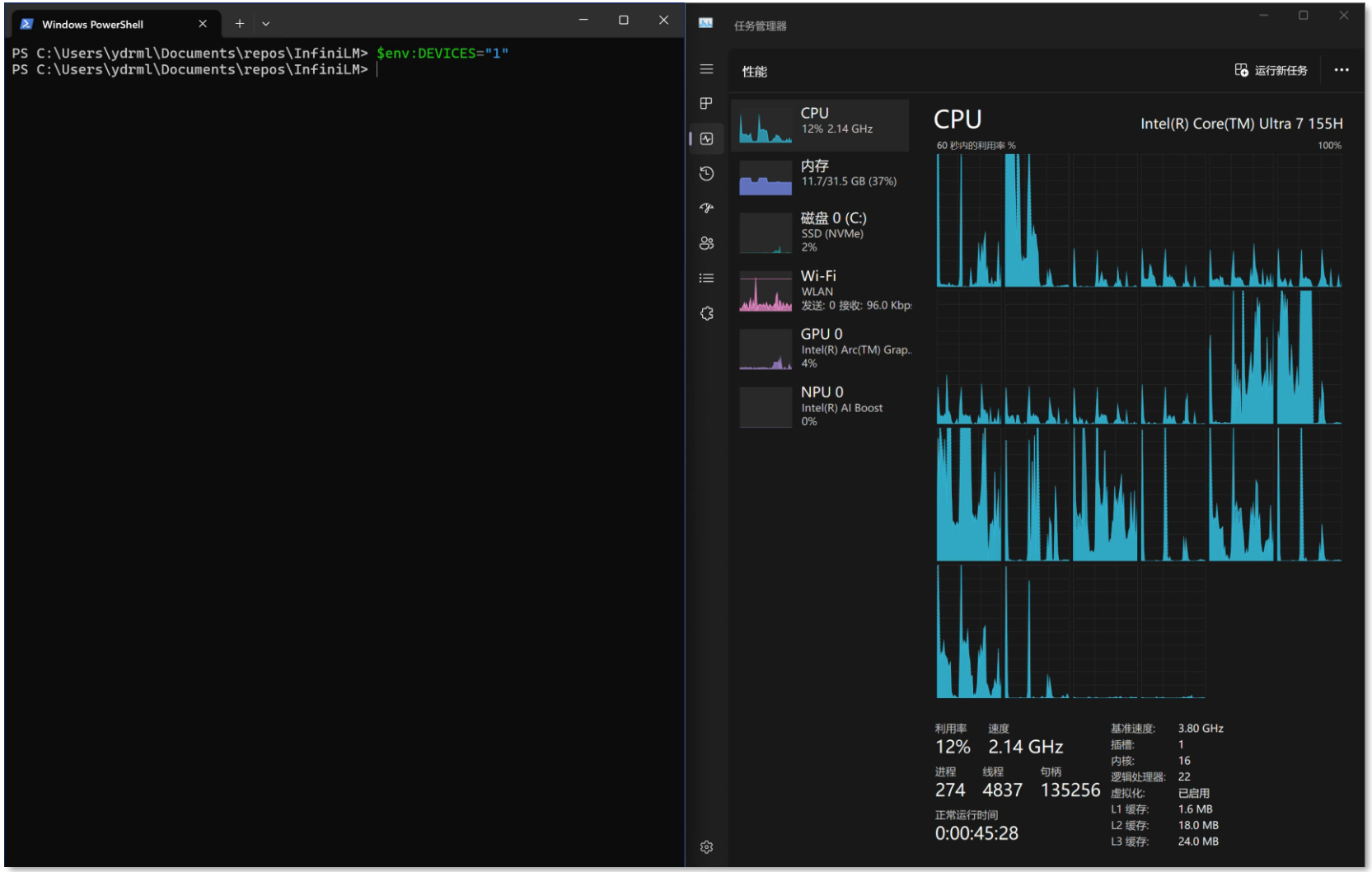
基于“推理单体”抽象的分布式推理

```rust
pub struct LlamaWorker<Ops: Operators, W> { … }

impl<Ops, W> LlamaWorker<Ops, W>
where
    Ops: Operators,
    W: WeightLoader<Hardware = Ops::Hardware>,
    ByteOf<Ops::Hardware>: 'static,
{
    pub fn launch<QA>(
        &mut self,
        args: Args<Ops::Hardware>,
        workspace: &mut [ByteOf<Ops::Hardware>],
        queue_alloc: &QA,
    ) → Result<(), LaunchError>
    where
        QA: QueueAlloc<Hardware = Ops::Hardware>,
    { … }
}
```

Llama 单体



基于“推理单体”抽象的分布式推理



模型结构表示

```rust
let cache = req
    .cache
    .as_mut() // [buf, nblk, 2, nkvh, dh]
    .index(1, iblk) // [buf, 2, nkvh, dh]
    .transpose(&[2, 0]) // [nkvh, 2, buf, dh]
    .map(|t| &mut t[..]);

split!(cache ⇒ kc, vc; [1, 1] @ 1);
let mut o = unsafe { q.map_slice_static_mut() };
self.attn_kv_cached(
    &mut q,
    &k,
    &v,
    &mut o,
    &mut kc.index(1, 0),
    &mut vc.index(1, 0),
    req.pos,
    workspace,
    queue_alloc,
)?;
```

```rust
let w = self.weights.ffn_norm(iblk, queue);
self.rms_norm(&mut x1, &x, &w, workspace, queue_alloc)?;
drop(w);

self.mlp(&mut x, &x1, iblk, self.residual, workspace, queue_alloc)?
```

```rust
for iblk in 0..nblk {
    {
        let w = self.weights.attn_norm(iblk, queue);
        self.rms_norm(&mut x1, &x, &w, workspace, queue_alloc)?;
        drop(w);

        let (buf, workspace) = workspace.split_at_mut(*qkv.get());
        let mut qkv = qkv.clone().map(|_| buf);

        let w = self.weights.attn_qkv(iblk, queue);
        self.mat_mul(&mut qkv, 0., &x1, &w, 1., workspace, queue_alloc)?;
        drop(w);

        let qkv = qkv.tile(1, &[nh + nkvh + nkvh, dh]);

        split!(qkv ⇒ q, k, v; [nh, nkvh, nkvh] @ 1);
        let mut q = q;
        let mut k = k;
        let v = v;

        self.rope(&mut q, &pos, &sin, &cos, workspace, queue_alloc)?;
        self.rope(&mut k, &pos, &sin, &cos, workspace, queue_alloc)?;

        {
            let q = q.map_slice_mut().transpose(&[1, 0]);
            let k = k.map_slice().transpose(&[1, 0]);
            let v = v.map_slice().transpose(&[1, 0]);
            let q = q.split(1, &req_split);
            let k = k.split(1, &req_split);
            let v = v.split(1, &req_split);

            for (mut q, k, v, req) in izip!(q, k, v, &mut requests) {
                // ... self-attention without output
            }
        }

        let o = q.merge(1..3).unwrap();
        let w = self.weights.attn_o(iblk, queue);
        self.mat_mul(&mut x, beta, &o, &w, 1., workspace, queue_alloc)?;
        drop(w);

        self.all_reduce(&mut x, workspace, queue_alloc)?;
    }
    {
        // ... mlp with swiglu
        self.all_reduce(&mut x, workspace, queue_alloc)?;
    }
}
```

张量抽象和元信息变换

```rust
#[derive(Clone)]
pub struct Tensor<T> {
    dt: DigitLayout,
    layout: ArrayLayout<5>,
    physical: T,
}
```

digit-layout v0.2.0

This crate provides a unified data type definition across various libraries, efficiently encodes types in a compact layout, thus avoiding the redundancy of enumerating definitions for data types.

#data-type #digit #layout

ndarray-layout v0.0.0

This crate provides definitions and transformations for multi-dimensional array data layouts.

#layout #ndarray #transformation

张量类型和相关开源库

由 Rust 编译器监测底层存储的所有权和生命周期

张量抽象和元信息变换

```rust
impl<T> Tensor<T> {
    pub fn transpose(self, perm: &[usize]) -> Self {
        Self {
            layout: self.layout.transpose(perm),
            ..self
        }
    }

    pub fn index(self, axis: usize, index: usize) -> Self { ... }
    pub fn slice(self, axis: usize, start: usize, step: isize, len: usize) -> Self { ... }
    pub fn tile(self, axis: usize, tiles: &[usize]) -> Self { ... }
    pub fn broadcast(self, axis: usize, times: usize) -> Self { ... }
    pub fn merge(self, range: Range<usize>) -> Option<Self> { ... }
}
```

支持的元信息变换

张量抽象和元信息变换

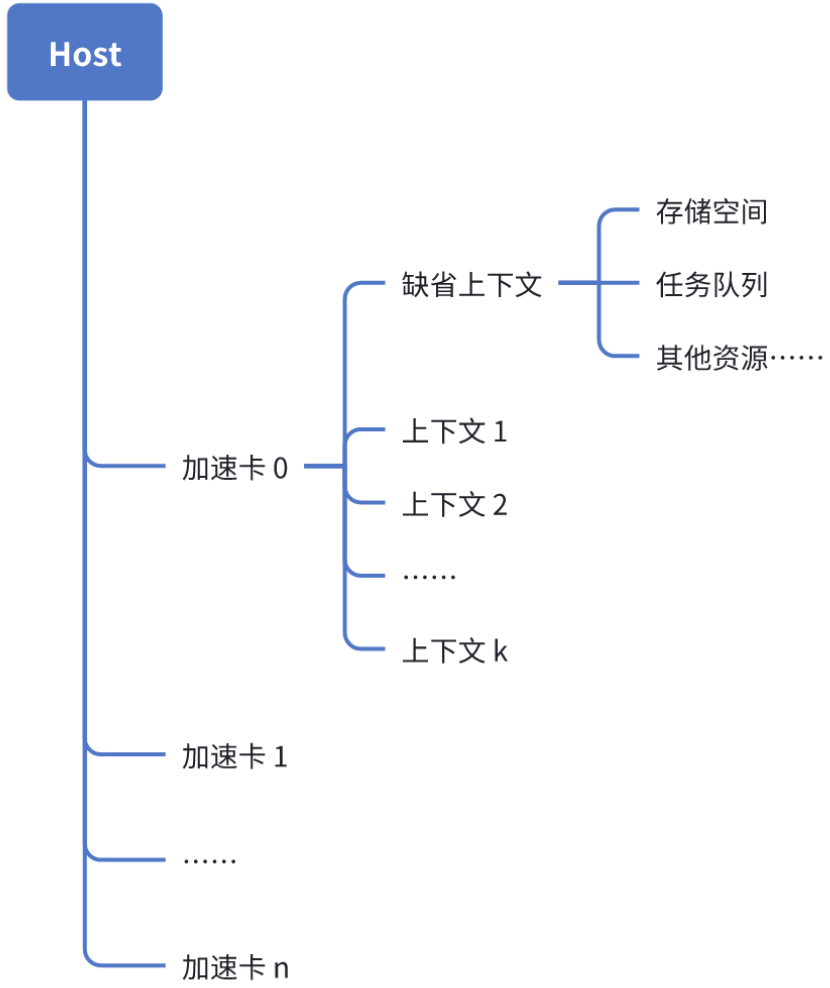
```rust
let q = q.map_slice_mut().transpose(&[1, 0]);
let k = k.map_slice()    .transpose(&[1, 0]);
let v = v.map_slice()    .transpose(&[1, 0]);
let q = q.split(1, &req_split);
let k = k.split(1, &req_split);
let v = v.split(1, &req_split);

for (mut q, k, v, req) in izip!(q, k, v, &mut requests) {
    let cache = req
        .cache
        .as_mut()           // [buf , nblk, 2   , nkvh, dh]
        .index(1, iblk)     // [buf ,       2   , nkvh, dh]
        .transpose(&[2, 0]) // [nkvh,       2   , buf , dh]
        .map(|t| &mut t[..]);

    split!(cache => kc, vc; [1, 1] @ 1);
    let mut o = unsafe { q.map_slice_static_mut() };
    self.attn_kv_cached(
        &mut q,
        &k,
        &v,
        &mut o,
        &mut kc.index(1, 0),
        &mut vc.index(1, 0),
        req.pos,
        workspace,
        queue_alloc,
    )?;
}
```

Continous Batching + Multi-Head Attention

```rust
let mut embd = Tensor::new(
  dt_embd,
  &[n, m, h / hk, w / wk]
).map(|s| queue_alloc.alloc(s));
self.conv(&mut embd, &raw, &k, &b, workspace, queue_alloc)?;

let mut embd = embd
    .merge(2..4)
    .unwrap()
    .transpose(&[2, 1]);

let pos_embd = self.weights.pos_embd(queue);
self.add_rows(&mut embd, &pos_embd, &pos, workspace, queue_alloc)?;
```

Conv + Position Embedding

加速卡管理：协处理器上下文资源-孢子抽象




```rust
pub trait ContextResource<'ctx, Ctx> {
  type Spore: ContextSpore<Ctx, Resource<'ctx> = Self>
  where
    Ctx: 'ctx;

  fn sporulate(self) -> Self::Spore;
}

pub trait ContextSpore<Ctx>: 'static + Send + Sync {
  type Resource<'ctx>: ContextResource<'ctx, Ctx, Spore = Self>
  where
    Ctx: 'ctx;

  fn sprout(self, ctx: &Ctx) -> Self::Resource<'_>;
  fn sprout_ref<'ctx>(&'ctx self, ctx: &'ctx Ctx) -> &Self::Resource<'_>;
  fn sprout_mut<'ctx>(&'ctx mut self, ctx: &'ctx Ctx) -> &mut Self::Resource<'_>;
}
```

上下文资源和孢子特质

加速卡管理：协处理器上下文资源-孢子抽象

```
#[macro_export]
macro_rules! impl_spore {
    ($resource:ident and $spore:ident by ($ctx:ty, $rss:ty)) => {
        #[repr(transparent)]
        pub struct $resource<'ctx>(
            $crate::RawContainer<<$ctx as $crate::AsRaw>::Raw, $rss>,
            std::marker::PhantomData<&'ctx ()>,
        );

        #[repr(transparent)]
        pub struct $spore($crate::RawContainer<<$ctx as $crate::AsRaw>::Raw, $rss>);

        impl $crate::ContextSpore<CurrentCtx> for $spore {
            type Resource<'ctx> = $resource<'ctx>;

            #[inline]
            fn sprout(self, ctx: &$ctx) -> Self::Resource<'_> {
                assert_eq!(self.0.ctx, unsafe { <$ctx as $crate::AsRaw>::as_raw(ctx) });
                ...
            }

            ...
        }

        impl<'ctx> $crate::ContextResource<'ctx, CurrentCtx> for $resource<'ctx> {
            type Spore = $spore;

            #[inline]
            fn sporulate(self) -> Self::Spore {
                ...
            }
        }
    };
}
```

使用卫生宏自动生成资源和孢子类型

```
impl_spore!(Stream and StreamSpore by (CurrentCtx, CUstream));

impl CurrentCtx {
    #[inline]
    pub fn stream(&self) -> Stream {
        let mut stream = null_mut();
        driver!(cuStreamCreate(&mut stream, 0));
        Stream(unsafe { self.wrap_raw(stream) }, PhantomData)
    }
}

impl Drop for Stream<'_> {
    #[inline]
    fn drop(&mut self) {
        self.synchronize();
        driver!(cuStreamDestroy_v2(self.0.rss));
    }
}
```

示例：实现 cuda stream

加速卡管理：协处理器上下文资源-孢子抽象

```rust
impl Device {
    pub fn retain_primary(&self) -> Context {
        let dev = unsafe { self.as_raw() };
        let mut ctx = null_mut();
        driver!(cuDevicePrimaryCtxRetain(&mut ctx, dev));
        Context {
            ctx,
            dev,
            primary: true,
        }
    }
}

pub struct CurrentCtx(CUcontext);

impl Context {
    pub fn apply<T>(&self, f: impl FnOnce(&CurrentCtx) -> T) -> T {
        driver!(cuCtxPushCurrent_v2(self.ctx));
        let ans = f(&CurrentCtx(self.ctx));
        let mut top = null_mut();
        driver!(cuCtxPopCurrent_v2(&mut top));
        ans
    }
}
```

获取缺省上下文与加载上下文

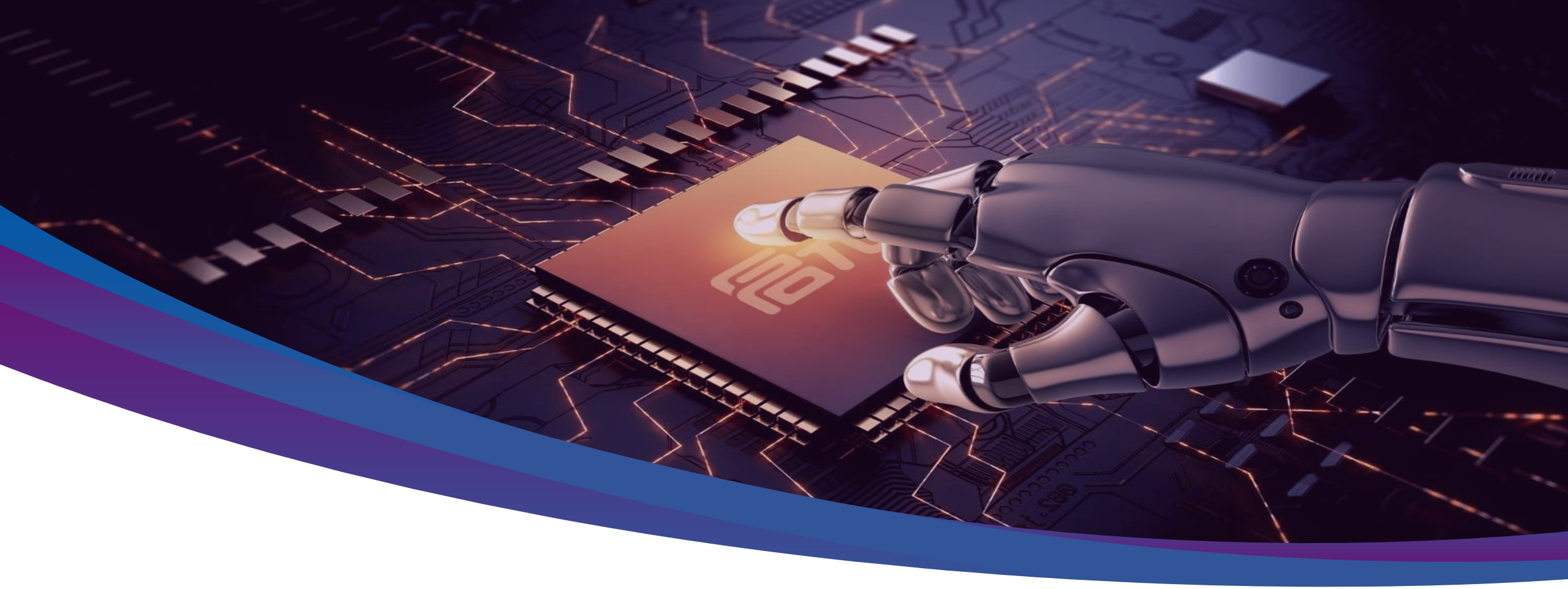
```rust
fn use_context() {
  let context = device.retain_primary();
  let stream = context.apply(|ctx| ctx.stream().sporulate());
  ...
}
```

从缺省上下文创建 cuda stream 孢子

加速卡管理：协处理器上下文资源-孢子抽象

```
std::thread::scope(|s| {
  comms.iter().enumerate().map(|(i, comm)| {
    s.spawn(move || {
      comm.device().retain_primary().apply(|ctx| {
        let queue = streams[i].sprout_ref(ctx);
        for layer in 0..nblk {
          self_att       (..., queue);
          comm.all_reduce(..., ReduceSum, queue);
          mlp            (..., queue);
          comm.all_reduce(..., ReduceSum, queue);
        }
      })
    })
  })
  .collect::<Vec<_>>()
  .into_iter()
  .map(|t| t.join().unwrap())
  .collect::<Vec<_>>();
});
```

使用上下文抽象和通信库实现不基于推理单体的模型并行



感谢聆听

GitHub开源组织：https://github.com/InfiniTensor