RWKV for the Web

Web-RWKV 推理引擎架构解析

cryscan / web-rwkv

Type [/] to search

le · Issues 2 · Pull requests · Actions · Projects · Wiki · Security · Insights · Settings

web-rwkv Public

Unpin · Unwatch 6 · Fork 22 · Star 282

main · 1 Branch · 146 Tags · Go to file t · Add file · Code

cryscan Simplify resource binding APIs. ✓ 202c5dc · last week · 657 Commits

.cargo	Allow model to be created from bytes.	2 years ago
.github/workflows	Bump version to v0.10.8	3 weeks ago
assets	Add RWKV-Othello example (rwkv v7)	last month
crates/web-rwkv-derive	Make clippy happy.	3 months ago
examples	Simplify resource binding APIs.	last week
screenshots	Add screenshots.	2 years ago
src	Simplify resource binding APIs.	last week
.gitattributes	Subgroup (#27)	10 months ago
.gitignore	Update .gitignore	2 months ago
Cargo.toml	Separate `tokio` feature completely so that `no-default-featur...`	2 weeks ago
LICENSE	Add license.	2 years ago
README.md	Bump version to v0.10.8	3 weeks ago

README · License

Web-RWKV

About

Implementation of the RWKV language model in pure WebGPU/Rust.

Readme
View license
Activity
282 stars
6 watching
22 forks

Releases 140

v0.10.10 Latest
2 weeks ago

+ 139 releases

Packages

No packages published
Publish your first package

Contributors 5

Languages

- Rust 83.7%
- WGSL 15.8%
- Python 0.5%

为什么是『Web』

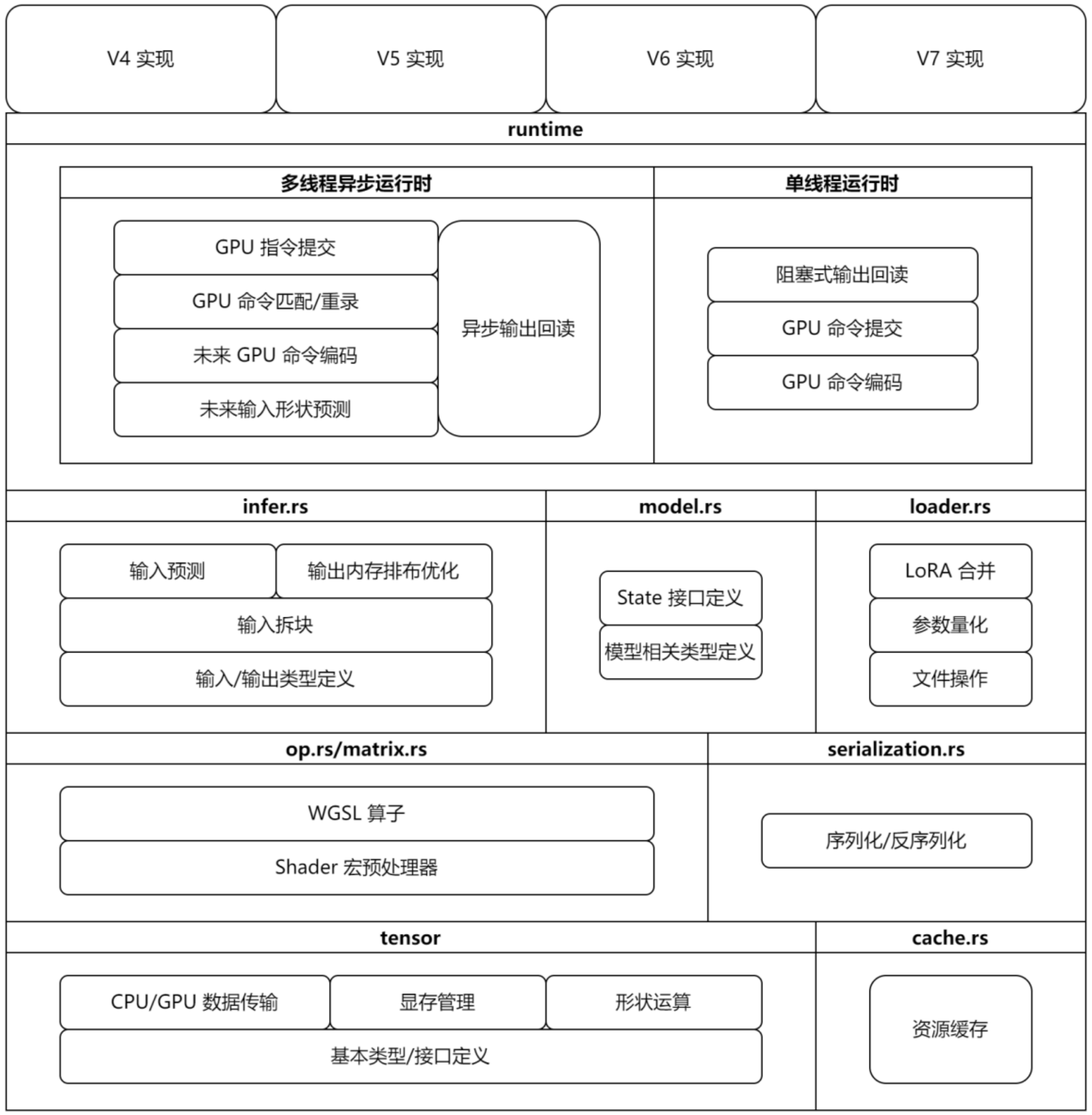
一份代码，多端通用

- PC
- Linux
- MacOS
- Android
- iOS
- 浏览器（WebGPU）

特点

- 高度集成：无 Python、CUDA 依赖
- 广泛兼容：支持 NVIDIA/AMD/Intel GPUs
- 并行推理：同时批量处理不定长的输入
- 相对快速：全量 V7 3B 输入 560 t/s 生成 80 t/s @ 3945WX + 3080
- 静态量化：支持 INT8/NF4/SF4、支持量化参数序列化/反序列化
- 微调外挂：支持 LoRA、支持 State 热切换
- 自由灵活：注入 Hook 读写任意推理阶段的数据，修改模型计算

架构



V4 实现
V5 实现
V6 实现
V7 实现
runtime
多线程异步运行时
GPU 指令提交
GPU 命令匹配/重录
未来 GPU 命令编码
未来输入形状预测
异步输出回读
单线程运行时
阻塞式输出回读
GPU 命令提交
GPU 命令编码
infer.rs
输入预测
输出内存排布优化
输入拆块
输入/输出类型定义
model.rs
State 接口定义
模型相关类型定义
loader.rs
LoRA 合并
参数量化
文件操作
op.rs/matrix.rs
WGSL 算子
Shader 宏预处理器
serialization.rs
序列化/反序列化
tensor
CPU/GPU 数据传输
显存管理
形状运算
基本类型/接口定义
cache.rs
资源缓存

迭代

Version 0.1

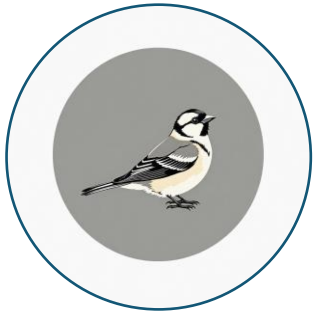
- 成功运行 RWKV 4 “Dove”

Version 0.2

- 代码模块化
- 实现 Tensor API
- 实现并行推理
- Crates.io 发布

Version 0.3

- RWKV 5 “Eagle”
- LoRA 加载



Version 0.4

- RWKV 6 “Finch”

迭代

Version 0.5

- Shader 宏预处理器
- 同时支持 Fp32/Fp16 读写

Version 0.8

- 异步运行时
- 多线程 GPU 命令编码
- GPU 命令预测编码
- 使用 Subgroups 操作
- 生成提速 50%-100%

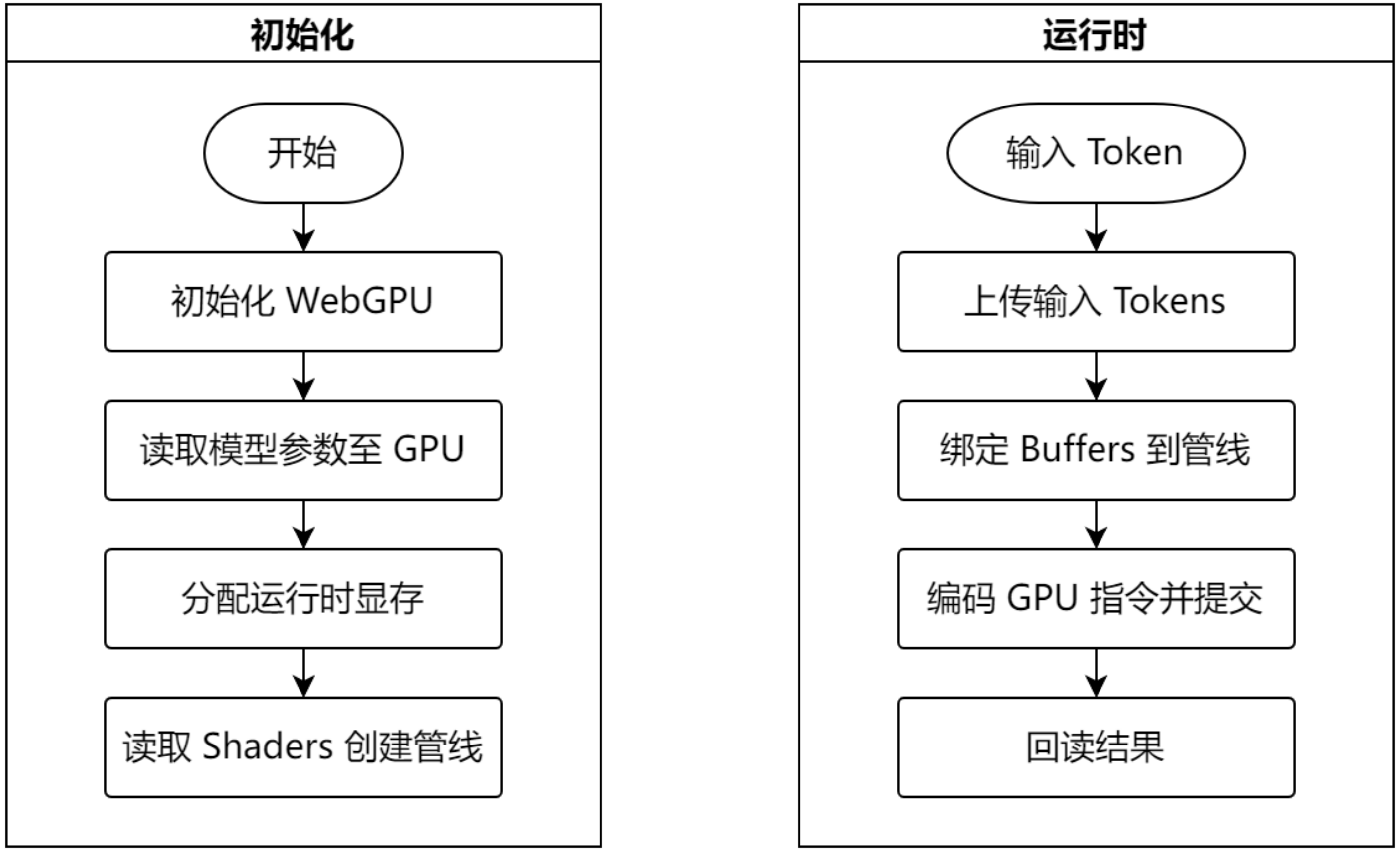
Version 0.9

- RWKV 7 “Goose”
- 支持在大量设备的浏览器上运行

Version 0.10

- Bind Group 缓存，生成提速 30%-50%
- 简化 Bind Group 相关 API，实现模型更方便

Mini Web-RWKV



1. 初始化 WebGPU

https://tool.geekfa.com/

```rust
pub struct Environment {
    pub instance: Instance,
    pub adapter: Adapter,
    pub device: Device,
    pub queue: Queue,
}

impl Environment {
    pub async fn create() -> Result<Self> {
        let instance = wgpu::Instance::new(InstanceDescriptor::default());
        let adapter = instance
            .request_adapter(&RequestAdapterOptions::default())
            .await?;
        let (device, queue) = adapter
            .request_device(&DeviceDescriptor::default(), None)
            .await?;
    }
}
```

```rust
pub struct Model {
    pub env: Arc<Environment>,
    pub info: ModelInfo,
    pub tensor: Arc<ModelTensor>,
    pub pipeline: Arc<ModelPipeline>,
}

pub struct ModelInfo {
    pub num_layers: usize,
    pub num_emb: usize,
    pub num_vocab: usize,
}

pub struct ModelTensor {
    pub dim: Buffer,
    pub embed: Embed,
    pub head: Head,
    pub layers: Vec<Layer>,
}
```

```rust
pub struct LayerNorm {
    pub w: Buffer,
    pub b: Buffer,
}

pub struct Embed {
    pub layer_norm: LayerNorm,
    pub w: Vec<f16>,
}

pub struct Head {
    pub layer_norm: LayerNorm,
    pub dims: Buffer,
    pub w: Buffer,
}
```

```rust
pub struct Att {
    pub time_decay: Buffer,
    pub time_first: Buffer,
    pub dims: Buffer,
    pub time_mix_k: Buffer,
    pub time_mix_v: Buffer,
    pub time_mix_r: Buffer,
    pub w_k: Buffer,
    pub w_v: Buffer,
    pub w_r: Buffer,
    pub w_o: Buffer,
}

pub struct Ffn {
    pub time_mix_k: Buffer,
    pub time_mix_r: Buffer,
    pub dims_k: Buffer,
    pub dims_v: Buffer,
    pub dims_r: Buffer,
    pub w_k: Buffer,
    pub w_v: Buffer,
    pub w_r: Buffer,
}
```

2. 声明模型结构

```
https://tool.geekfa.com/

pub struct ModelBuffer {
    pub tokens: Vec<u16>,
    pub num_tokens: Buffer,

    pub emb_x: Buffer,
    pub emb_o: Buffer,

    pub att_x: Buffer,
    pub att_kx: Buffer,
    pub att_vx: Buffer,
    pub att_rx: Buffer,
    pub att_k: Buffer,
    pub att_v: Buffer,
    pub att_r: Buffer,
    pub att_w: Buffer,
    pub att_o: Buffer,

    pub ffn_x: Buffer,
    pub ffn_kx: Buffer,
    pub ffn_vx: Buffer,
    pub ffn_rx: Buffer,
    pub ffn_k: Buffer,
    pub ffn_v: Buffer,
    pub ffn_r: Buffer,
    pub ffn_o: Buffer,

    pub head_x: Buffer,
    pub head_r: Buffer,
    pub head_o: Buffer,

    pub map: Buffer,
}

pub struct ModelState(pub Vec<LayerState>);

pub struct LayerState {
    pub att: Buffer,
    pub ffn: Buffer,
}
```

3. 声明运行时需要的 Buffer

4. 从文件中读取模型参数，分配参数显存并上传

https://tool.geekfa.com/

```rust
impl Model {
    fn from_bytes(data: &[u8], env: Arc<Environment>) -> Result<Self> {
        let device = &env.device;
        let model = SafeTensors::deserialize(data)?;
        let load_tensor_f32 = |name: String| -> Result<Buffer> {
            let tensor = model.tensor(&name)?.data();
            let tensor: Vec<_> = pod_collect_to_vec::<_, f16>(tensor)
                .into_iter()
                .map(f16::to_f32)
                .collect();
            let buffer = device.create_buffer_init(&BufferInitDescriptor {
                label: Some(&name),
                contents: cast_slice(&tensor),
                usage: BufferUsages::STORAGE,
            });
            Ok(buffer)
        };
        let load_tensor_f16 = |name: String| -> Result<Buffer> {
            let tensor = model.tensor(&name)?.data();
            let buffer = device.create_buffer_init(&BufferInitDescriptor {
                label: Some(&name),
                contents: cast_slice(tensor),
                usage: BufferUsages::STORAGE,
            });
            Ok(buffer)
        };
        let embed = Embed {
            layer_norm: LayerNorm {
                w: load_tensor_f32("blocks.0.ln0.weight".into())?,
                b: load_tensor_f32("blocks.0.ln0.bias".into())?,
            },
            w: pod_collect_to_vec(model.tensor("emb.weight")?.data()),
        };
        let head = Head {
            layer_norm: LayerNorm {
                w: load_tensor_f32("ln_out.weight".into())?,
                b: load_tensor_f32("ln_out.bias".into())?,
            },
            dims: create_uniform_u32(&[num_emb as u32, num_vocab as u32]),
            w: load_tensor_f16("head.weight".into())?,
```

5. Buffer 分配显存

```rust
pub fn create_buffer(&self, tokens: &[u16]) -> ModelBuffer {
    let device = &self.env.device;

    let create_buffer_f32 = |capacity: usize| -> Buffer {
        let data = vec![0.0f32; capacity];
        device.create_buffer_init(&BufferInitDescriptor {
            label: None,
            contents: cast_slice(&data),
            usage: BufferUsages::STORAGE | BufferUsages::COPY_DST | BufferUsages::COPY_SRC,
        })
    };
    let load_buffer_f32 = |data: &[f32]| -> Buffer {
        device.create_buffer_init(&BufferInitDescriptor {
            label: None,
            contents: cast_slice(data),
            usage: BufferUsages::STORAGE | BufferUsages::COPY_DST,
        })
    };
    let create_uniform_u32 = |values: &[u32]| -> Buffer {
        device.create_buffer_init(&BufferInitDescriptor {
            label: None,
            contents: cast_slice(values),
            usage: BufferUsages::UNIFORM | BufferUsages::COPY_DST,
        })
    };
```

https://tool.geekfa.com/

```rust
    let num_tokens = tokens.len();
    let num_emb = self.info.num_emb;
    let num_vocab = self.info.num_vocab;
    let capacity = num_tokens * num_emb;

    let input = self.embedding(tokens);

    let map = device.create_buffer(&BufferDescriptor {
        label: None,
        size: 4 * num_vocab as u64,
        usage: BufferUsages::MAP_READ | BufferUsages::COPY_DST,
        mapped_at_creation: false,
    });

    ModelBuffer {
        tokens: tokens.to_vec(),
        num_tokens: create_uniform_u32(&[num_tokens as u32]),
        emb_x: load_buffer_f32(&input),
        emb_o: create_buffer_f32(capacity),
        att_x: create_buffer_f32(capacity),
        att_kx: create_buffer_f32(capacity),
        att_vx: create_buffer_f32(capacity),
        att_rx: create_buffer_f32(capacity),
        att_k: create_buffer_f32(capacity),
        att_v: create_buffer_f32(capacity),
        att_r: create_buffer_f32(capacity),
        att_w: create_buffer_f32(capacity),
        att_o: create_buffer_f32(capacity),
        ffn_x: create_buffer_f32(capacity),
        ffn_kx: create_buffer_f32(capacity),
        ffn_vx: create_buffer_f32(4 * capacity),
        ffn_rx: create_buffer_f32(capacity),
        ffn_k: create_buffer_f32(4 * capacity),
        ffn_v: create_buffer_f32(capacity),
        ffn_r: create_buffer_f32(capacity),
        ffn_o: create_buffer_f32(capacity),
        head_x: create_buffer_f32(num_emb),
        head_r: create_buffer_f32(num_emb),
        head_o: create_buffer_f32(num_vocab),
        map,
    }
}
```

5. State 分配显存

https://tool.geekfa.com/

```rust
pub fn create_state(&self) -> ModelState {
    let device = &self.env.device;

    let ModelInfo {
        num_layers,
        num_emb,
        ..
    } = self.info;

    let create_buffer_f32 = |data: &[f32]| -> Buffer {
        device.create_buffer_init(&BufferInitDescriptor {
            label: None,
            contents: cast_slice(data),
            usage: BufferUsages::STORAGE | BufferUsages::COPY_DST | BufferUsages::COPY_SRC,
        })
    };

    let mut layers = vec![];
    for _ in 0..num_layers {
        let mut att = vec![0.0f32; 4 * num_emb];
        att[3 * num_emb..4 * num_emb]
            .iter_mut()
            .for_each(|x| *x = -1.0e30);

        let ffn = vec![0.0f32; num_emb];

        let layer = LayerState {
            att: create_buffer_f32(&att),
            ffn: create_buffer_f32(&ffn),
        };
        layers.push(layer);
    }

    ModelState(layers)
}
```

```
@group(0) @binding(0) var<uniform> dims: vec2<u32>;                    // [C, R]
@group(0) @binding(1) var<storage, read> matrix: array<vec2<u32>>;     // (R, C)
@group(0) @binding(2) var<storage, read> input: array<vec4<f32>>;      // (T, C)
@group(0) @binding(3) var<storage, read_write> output: array<vec4<f32>>;   // (T, R)

const BLOCK_SIZE: u32 = 256u;

var<workgroup> local_sum: array<vec4<f32>, BLOCK_SIZE>;

fn reduce_step_barrier(index: u32, stride: u32) {
    if index < stride {
        local_sum[index] += local_sum[index + stride];
    }
    workgroupBarrier();
}

@compute @workgroup_size(256, 1, 1)
fn matmul(@builtin(global_invocation_id) invocation_id: vec3<u32>) {
    let index = invocation_id.x;
    let channel = invocation_id.y;      // 1 channel: 4 rows in matrix
    let token = invocation_id.z;
    let stride = dims / 4u;

    local_sum[index] = vec4<f32>(0.0);
    for (var i = index; i < stride.x; i += BLOCK_SIZE) {
        let ti = token * stride.x + i;
        var ci = channel * 4u * stride.x + i;

        // read 4 elements from the input
        let x = input[ti];

        // read 4 rows from the matrix, each with 4 unpacked floats, forming a 4x4 sub-block
        var data: vec2<u32>;
        var m: mat4x4<f32>;

        data = matrix[ci]; m[0] = vec4<f32>(unpack2x16float(data.x), unpack2x16float(data.y)); ci += stride.x;
        data = matrix[ci]; m[1] = vec4<f32>(unpack2x16float(data.x), unpack2x16float(data.y)); ci += stride.x;
        data = matrix[ci]; m[2] = vec4<f32>(unpack2x16float(data.x), unpack2x16float(data.y)); ci += stride.x;
        data = matrix[ci]; m[3] = vec4<f32>(unpack2x16float(data.x), unpack2x16float(data.y));
        local_sum[index] += transpose(m) * x;
    }
    workgroupBarrier();

    reduce_step_barrier(index, 128u);
    reduce_step_barrier(index, 64u);
    reduce_step_barrier(index, 32u);

    if index < 32u {
        local_sum[index] += local_sum[index + 16u];
        local_sum[index] += local_sum[index + 8u];
        local_sum[index] += local_sum[index + 4u];
        local_sum[index] += local_sum[index + 2u];
        local_sum[index] += local_sum[index + 1u];
    }

    if index == 0u {
        output[token * stride.y + channel] = local_sum[0];
    }
}
```

6. 手写 Compute Shader，实现
   - Layer Normalization
   - 矩阵×向量
   - Time Shift
   - WKV Operator
   - Channel Mix

7. 读取 Shader，创建 Compute Pipeline

https://tool.geekfa.com/

```rust
let create_pipeline = |shader: &str, entry_point: &str| -> ComputePipeline {
    let module = &device.create_shader_module(ShaderModuleDescriptor {
        label: None,
        source: ShaderSource::Wgsl(Cow::Borrowed(shader)),
    });
    device.create_compute_pipeline(&ComputePipelineDescriptor {
        label: None,
        layout: None,
        module,
        entry_point,
    })
};

let pipeline = Arc::new(ModelPipeline {
    layer_norm: create_pipeline(include_str!("shaders/layer_norm.wgsl"), "layer_norm"),
    token_shift: create_pipeline(include_str!("shaders/token_shift.wgsl"), "token_shift"),
    matmul: create_pipeline(include_str!("shaders/matmul.wgsl"), "matmul"),
    token_mix: create_pipeline(include_str!("shaders/token_mix.wgsl"), "token_mix"),
    activation: create_pipeline(include_str!("shaders/activation.wgsl"), "activation"),
    channel_mix: create_pipeline(include_str!("shaders/channel_mix.wgsl"), "channel_mix"),
    add: create_pipeline(include_str!("shaders/add.wgsl"), "add"),
});
```

https://tool.geekfa.com/

```rust
pub struct ModelPipeline {
    pub layer_norm: ComputePipeline,
    pub token_shift: ComputePipeline,
    pub matmul: ComputePipeline,
    pub token_mix: ComputePipeline,
    pub activation: ComputePipeline,
    pub channel_mix: ComputePipeline,
    pub add: ComputePipeline,
}
```

8. 当输入了一个 Token：转化为 Embedding 并上传 GPU

https://tool.geekfa.com/

```rust
pub fn embedding(&self, tokens: &[u16]) -> Vec<f32> {
    let num_tokens = tokens.len();
    let num_emb = self.info.num_emb;
    let capacity = num_tokens * num_emb;

    let mut input = vec![];
    input.reserve(capacity);
    for token in tokens {
        let index = *token as usize;
        let mut embed: Vec<_> = self.tensor.embed.w[index * num_emb..(index + 1) * num_emb]
            .iter()
            .copied()
            .map(f16::to_f32)
            .collect();
        input.append(&mut embed);
    }
    input
}
```

9. 绑定 Buffers 到 Pipeline；创建 Compute Pass；编码、提交 GPU 命令

https://tool.geekfa.com/

```rust
fn create_bind_group(&self, buffer: &ModelBuffer, state: &ModelState) -> ModelBindGroup {
        let device = &self.env.device;
        let pipeline = &self.pipeline;

        let [layer_norm_layout, token_shift_layout, matmul_layout, token_mix_layout, activation_layout, channel_mix_layout,
            [
                &pipeline.layer_norm,
                &pipeline.token_shift,
                &pipeline.matmul,
                &pipeline.token_mix,
                &pipeline.activation,
                &pipeline.channel_mix,
                &pipeline.add,
            ]
            .map(|pipeline| pipeline.get_bind_group_layout(0));

        let embed = {
            let layer_norm = device.create_bind_group(&BindGroupDescriptor {
                label: None,
                layout: &layer_norm_layout,
                entries: &[
                    BindGroupEntry {
                        binding: 0,
                        resource: self.tensor.embed.w.as_entire_binding(),
                    },
                    BindGroupEntry {
                        binding: 1,
                        resource: buffer.emb_x.as_entire_binding(),
                    },
                    BindGroupEntry {
                        binding: 2,
                        resource: self.tensor.embed.layer_norm.w.as_entire_binding(),
                    },
                    BindGroupEntry {
                        binding: 3,
                        resource: self.tensor.embed.layer_norm.b.as_entire_binding(),
                    },
                    BindGroupEntry {
                        binding: 4,
                        resource: buffer.emb_o.as_entire_binding(),
                    },
                ],
            });
            EmbedBindGroup { layer_norm }
        };
```

https://tool.geekfa.com/

```rust
fn run_internal(&self, buffer: &ModelBuffer, state: &ModelState) {
        let device = &self.env.device;
        let queue = &self.env.queue;

        let bind_group = self.create_bind_group(buffer, state);
        let pipeline = &self.pipeline;

        let ModelInfo {
            num_emb, num_vocab, ..
        } = self.info;

        let num_tokens = buffer.tokens.len() as u32;
        let num_emb_vec4 = num_emb as u32 / 4;
        let num_vocab_vec4 = num_vocab as u32 / 4;
        const BLOCK_SIZE: u32 = 256;

        let mut encoder = device.create_command_encoder(&CommandEncoderDescriptor::default());
        {
            let mut pass = encoder.begin_compute_pass(&ComputePassDescriptor::default());

            pass.set_pipeline(&pipeline.layer_norm);
            pass.set_bind_group(0, &bind_group.embed.layer_norm, &[]);
            pass.dispatch_workgroups(1, num_tokens, 1);
        }
        for layer in &bind_group.layers {
            {
                let mut pass = encoder.begin_compute_pass(&ComputePassDescriptor::default());
                pass.set_pipeline(&pipeline.layer_norm);
                pass.set_bind_group(0, &layer.att_layer_norm, &[]);
                pass.dispatch_workgroups(1, num_tokens, 1);
                // ...
            }
            // ...
            encoder.copy_buffer_to_buffer(&buffer.head_o, 0, &buffer.map, 0, 4 * num_vocab as u64);
            queue.submit(Some(encoder.finish()));
        }
}
```

https://tool.geekfa.com/

```rust
let (sender, receiver) = async_channel::bounded(1);
let slice = buffer.map.slice(..);
slice.map_async(wgpu::MapMode::Read, move |v| {
    sender.send_blocking(v).unwrap();
});
// 等待 GPU 计算完成，阻塞!
self.env.device.poll(wgpu::MaintainBase::Wait);
match receiver.recv_blocking() {
    Ok(_) => {
        let data = {
            let data = slice.get_mapped_range();
            cast_slice(&data).to_vec()
        };
        buffer.map.unmap();
        Ok(data)
    }
    Err(err) => Err(err.into()),
}
```

10. 回读计算结果

恭喜你实现了 Web-RWKV 0.1

- 速度比 Torch 实现快一些……
- V4 3B 50 t/s @ 3945WX + 3080
- 但是如果我的机器是 Xeon E5-2673 v3 + 4090
- 那么速度只有 25 t/s（……）
- 我们一步一步来优化它……

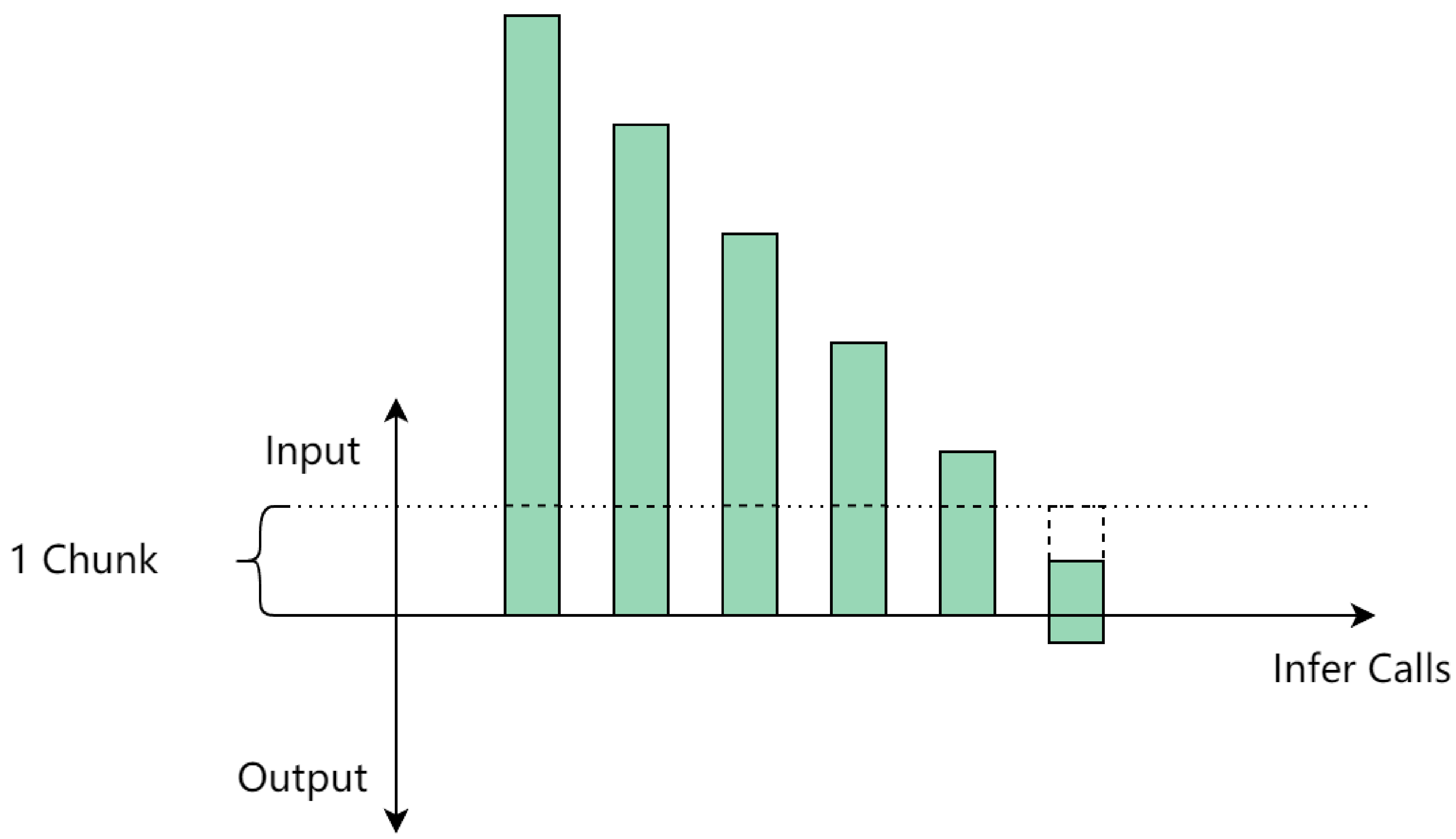
长输入优化及并行推理

- 长输入使用 GEMM 而非 GEMV，节省大量带宽，提速 8-10 倍
- 为了效率，GEMM Kernel 期望输入序列长是 32 的倍数
- 过长的输入可能会使 GPU 长时间繁忙从而触发 TDR
- 故我们需要将长输入拆分成合理大小的块，一次处理一块
- 那并行推理也可以复用这个加速机制吗？

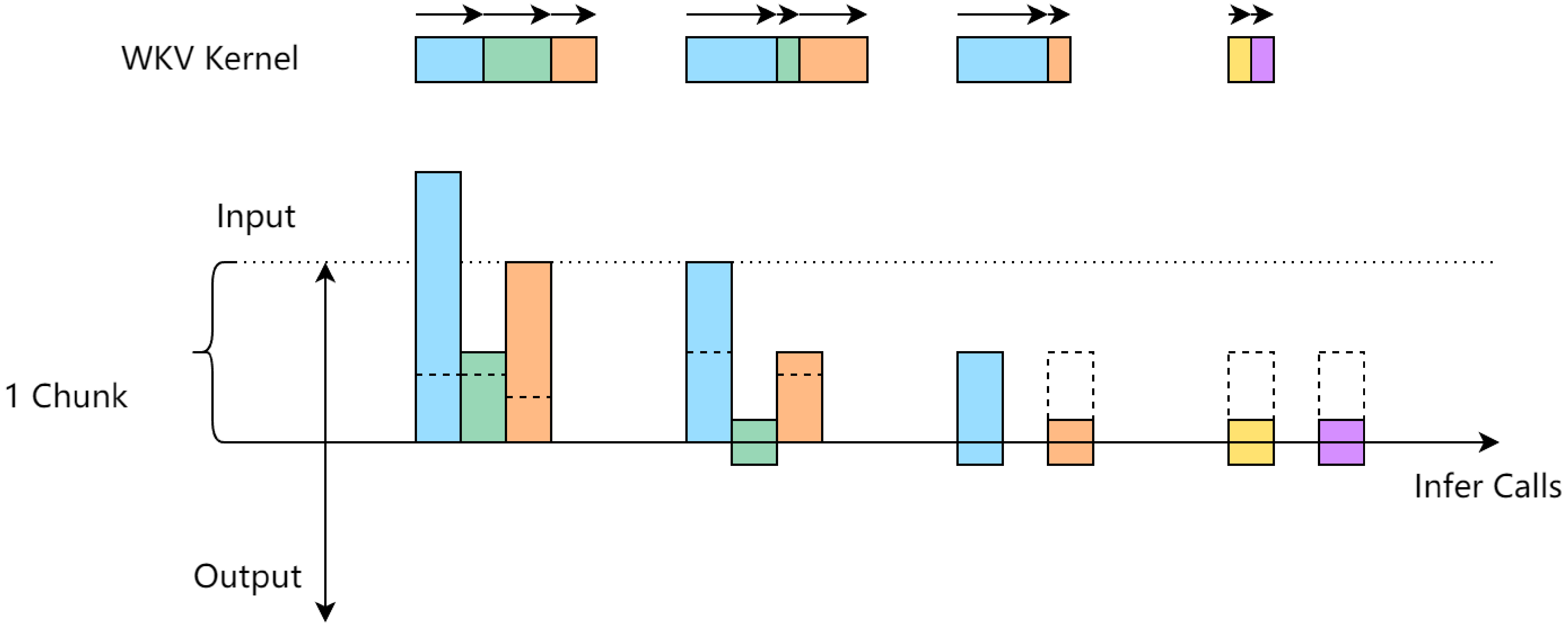
从推理函数的类型中我们可以看到……

https://tool.geekfa.com/

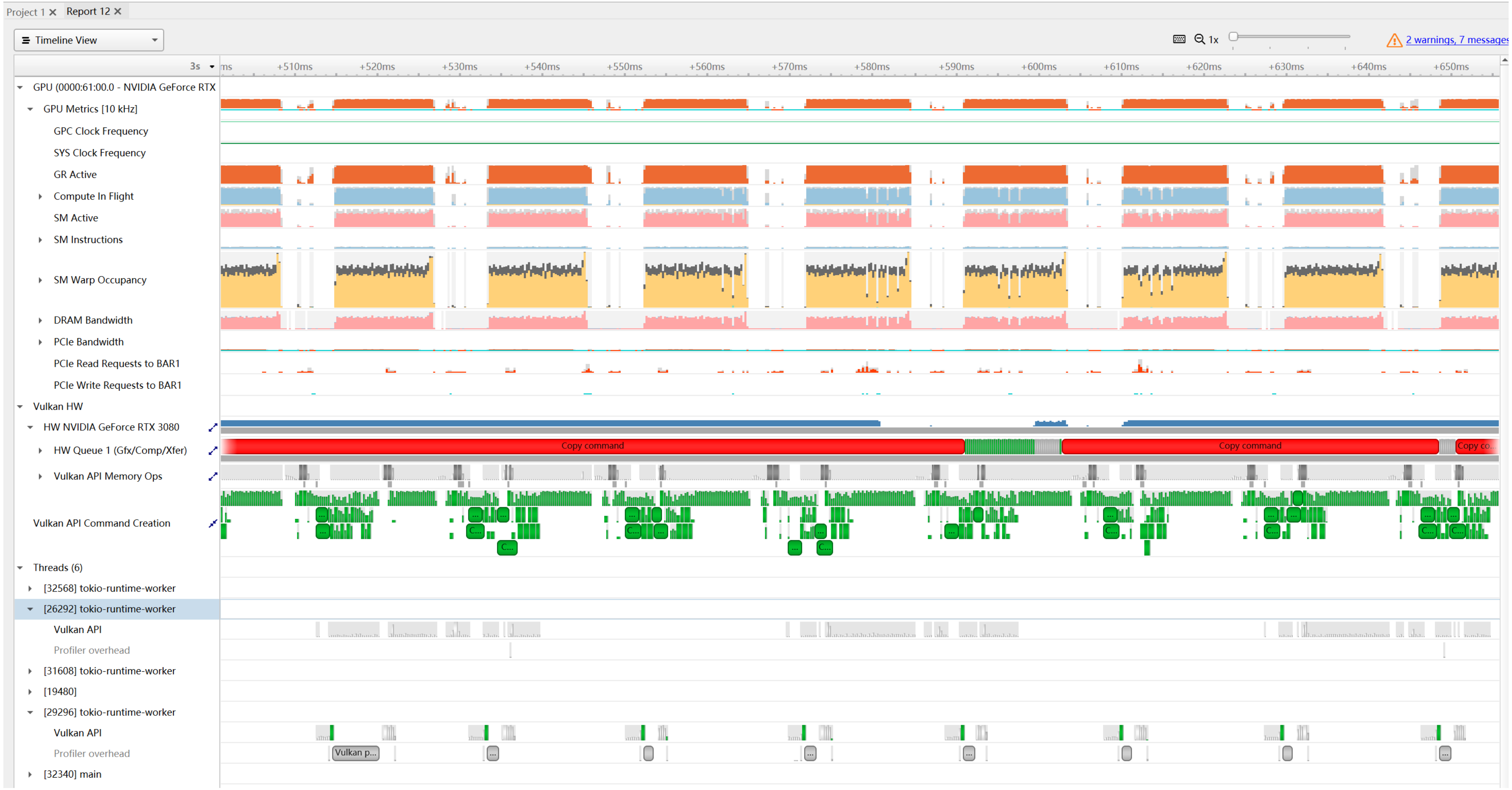
```
/// Perform (partial) inference and return the remaining input and (perhaps partial) output.
/// The amount of input processed during one call is bound by the input chunk size.
pub async fn infer(&self, input: I) -> Result<(I, O)> {
    let (sender, receiver) = flume::bounded(1);
    let submission = Submission { input, sender };
    let _ = self.0.send(submission).await;
    receiver.recv_async().await?
}
```



Input
1 Chunk
Infer Calls
Output


WKV Kernel
Input
1 Chunk
Infer Calls
Output

推理引擎内置 Tracy 性能测试接口，
只要开启相应的编译特性即可。



AMD Ryzen Threadripper PRO 3945WX + 3080

GPU 在等什么？

```
let att_matmul_k = device.create_bind_group(&BindGroupDescriptor {
    label: None,
    layout: &matmul_layout,
    entries: &[
        BindGroupEntry {
            binding: 0,
            resource: layer.att.dims.as_entire_binding(),
        },
        BindGroupEntry {
            binding: 1,
            resource: layer.att.w_k.as_entire_binding(),
        },
        BindGroupEntry {
            binding: 2,
            resource: buffer.att_kx.as_entire_binding(),
        },
        BindGroupEntry {
            binding: 3,
            resource: buffer.att_k.as_entire_binding(),
```

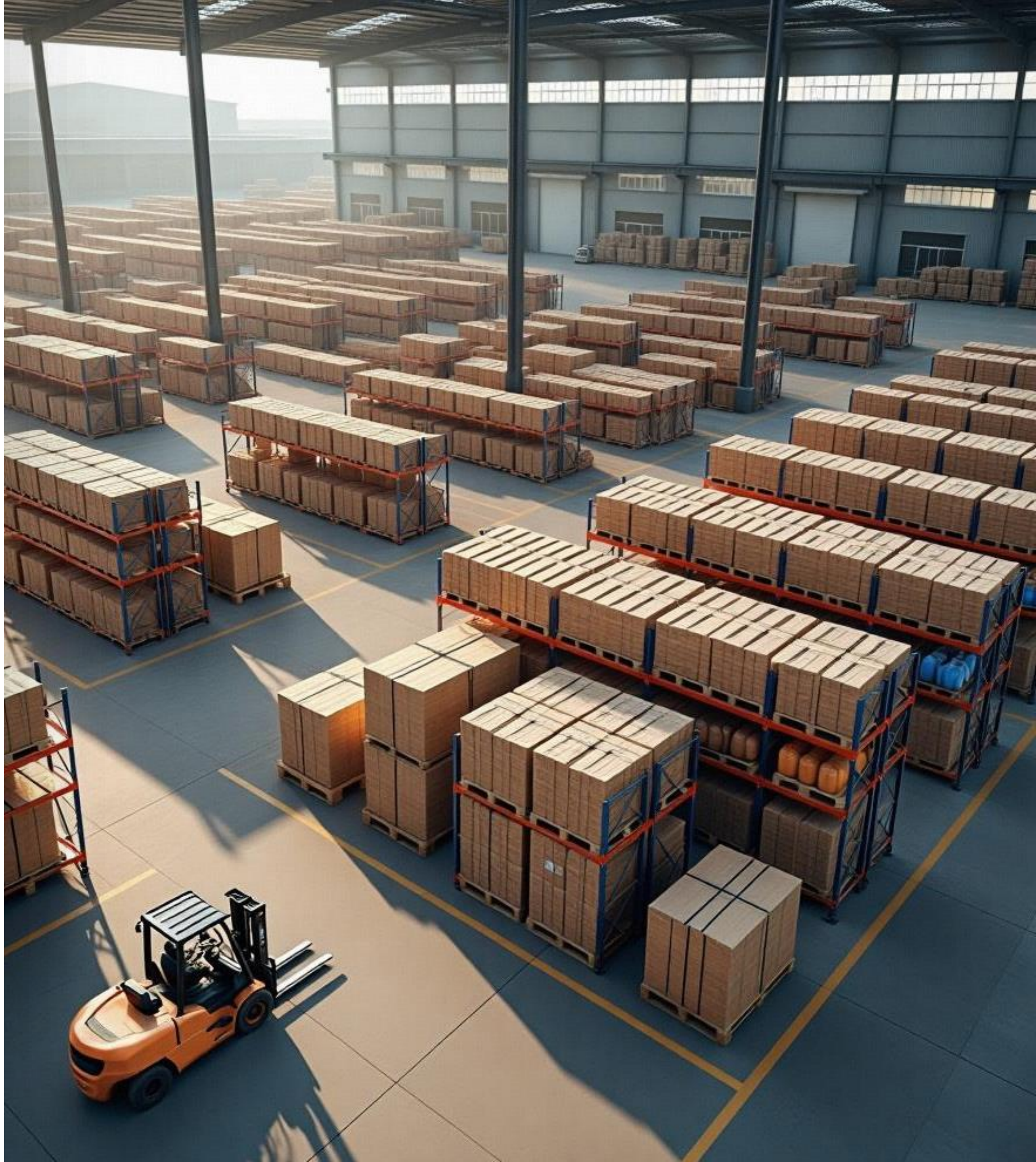
创建绑定组

```
let mut pass = encoder.begin_compute_pass(&ComputePassDes

pass.set_pipeline(&pipeline.layer_norm);
pass.set_bind_group(0, &bind_group.head.layer_norm, &[]);
pass.dispatch_workgroups(1, 1, 1);

pass.set_pipeline(&pipeline.matmul);
pass.set_bind_group(0, &bind_group.head.matmul, &[]);
pass.dispatch_workgroups(1, num_vocab_vec4, 1);

der.copy_buffer_to_buffer(&buffer.head_o, 0, &buffer.map,
```

编码命令

绑定组创建优化

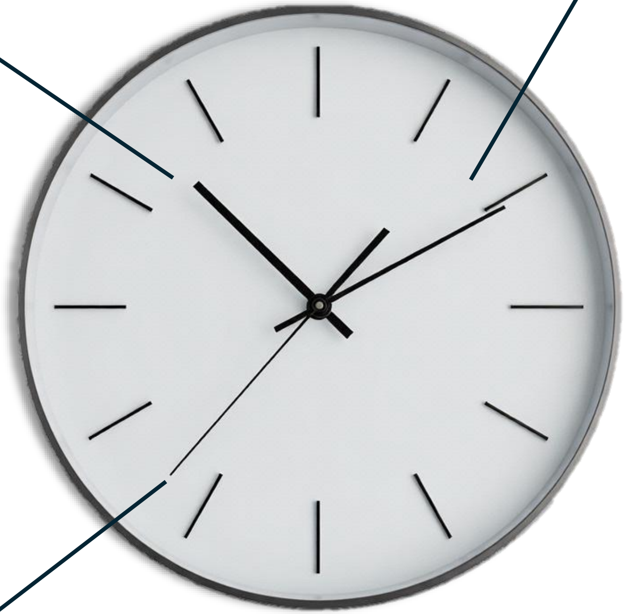
多层级资源缓存复用



绑定组创建优化

管线复用

同一算子变体只用
创建一次管线



显存复用

绑定组复用

相同管线绑定相同
显存可复用绑定组

命令编码优化

输入预测

- 在生成上一输出前，无法得知下一输入?
- 至少我们可以预测形状（假设：推理过程不被人为中断）
- 在 GPU 计算第 N 个 Token 时，CPU 并行编码第 N + 1、N + 2……
- 如果预测错误，则停止现有编码任务，冲刷流水线，重新编码
- 但是，CPU 要如何生产才能达到供需平衡?

- 生产者（CPU）↔ 消费者（GPU）
- 平衡方程
- 动态调整 CPU 编码任务数
- 2 → 1 → 0 循环

Xeon E5，相比同步编码，异步并行预测编码提速 100%

WKV 5/6/7

	T0				T1				T2				T3				A/B/R/W/K
	↓				↓				↓				↓				
S	α0	β0	γ0	δ0	ε0	ζ0	η0	θ0	ι0	κ0	λ0	μ0	ν0	ξ0	ο0	π0	0
	α1	β1	γ1	δ1	ε1	ζ1	η1	θ1	ι1	κ1	λ1	μ1	ν1	ξ1	ο1	π1	1
	α2	β2	γ2	δ2	ε2	ζ2	η2	θ2	ι2	κ2	λ2	μ2	ν2	ξ2	ο2	π2	2
	α3	β3	γ3	δ3	ε3	ζ3	η3	θ3	ι3	κ3	λ3	μ3	ν3	ξ3	ο3	π3	3
	α4	β4	γ4	δ4	ε4	ζ4	η4	θ4	ι4	κ4	λ4	μ4	ν4	ξ4	ο4	π4	4
	α5	β5	γ5	δ5	ε5	ζ5	η5	θ5	ι5	κ5	λ5	μ5	ν5	ξ5	ο5	π5	5
	α6	β6	γ6	δ6	ε6	ζ6	η6	θ6	ι6	κ6	λ6	μ6	ν6	ξ6	ο6	π6	6
	α7	β7	γ7	δ7	ε7	ζ7	η7	θ7	ι7	κ7	λ7	μ7	ν7	ξ7	ο7	π7	7
	α8	β8	γ8	δ8	ε8	ζ8	η8	θ8	ι8	κ8	λ8	μ8	ν8	ξ8	ο8	π8	8
	α9	β9	γ9	δ9	ε9	ζ9	η9	θ9	ι9	κ9	λ9	μ9	ν9	ξ9	ο9	π9	9
	αA	βA	γA	δA	εA	ζA	ηA	θA	ιA	κA	λA	μA	νA	ξA	οA	πA	A
	αB	βB	γB	δB	εB	ζB	ηB	θB	ιB	κB	λB	μB	νB	ξB	οB	πB	B
	αC	βC	γC	δC	εC	ζC	ηC	θC	ιC	κC	λC	μC	νC	ξC	οC	πC	C
	αD	βD	γD	δD	εD	ζD	ηD	θD	ιD	κD	λD	μD	νD	ξD	οD	πD	D
	αE	βE	γE	δE	εE	ζE	ηE	θE	ιE	κE	λE	μE	νE	ξE	οE	πE	E
	αF	βF	γF	δF	εF	ζF	ηF	θF	ιF	κF	λF	μF	νF	ξF	οF	πF	F
SA/V	α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο	π	

Loop/Reduction Direction

Memory/Thread Layout

有关平台兼容性的一些坑……

- 不要把一些 N 卡经验带进 Shader（该同步还得同步）
- A 卡发现 NF4 的 Quantiles 每次从共享内存取一个 `f32` 更快，欣然提交；一小时后发现 I 卡只可取 `vec4<f32>`对齐，否则乱码……
- 苹果芯片的 `tanh` 内置函数输出超过 42 返回 `NaN`（最谔谔）
- ……