~~SETTING FOUNDATIONS~~

Implementing SWARM (finally)

MIKA
DENGHAAS

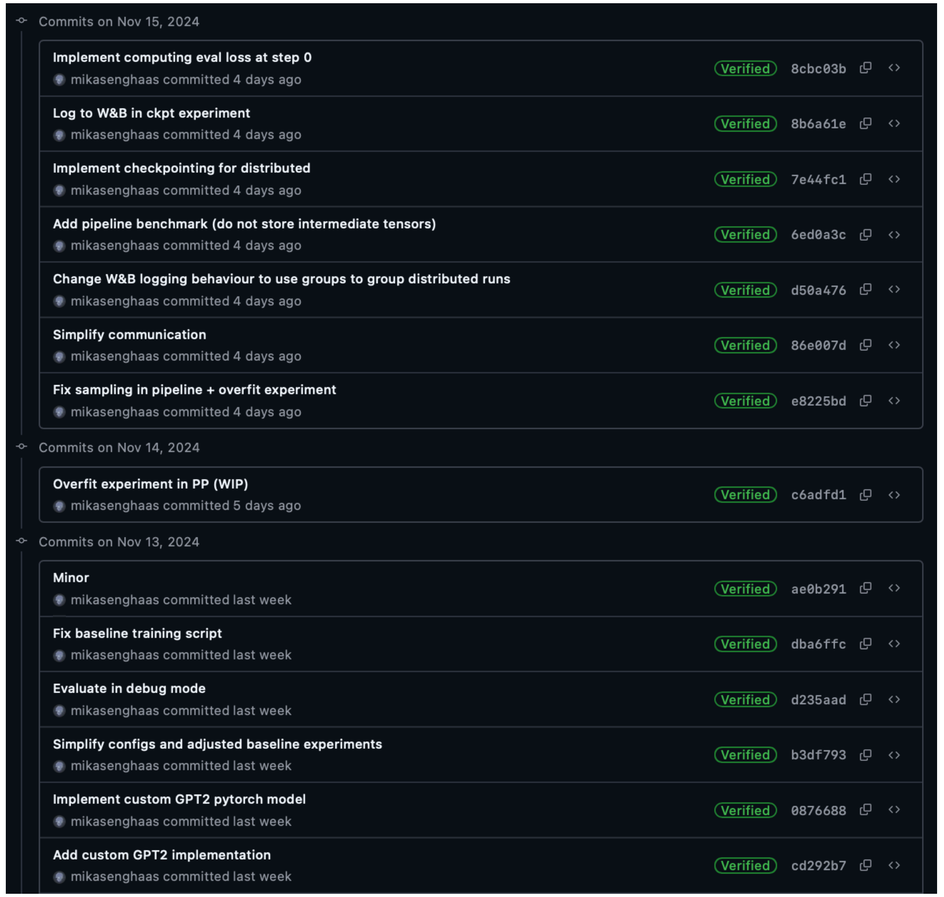
EPFL

Next Steps

1. ~~Verify 1 GPU baseline~~
2. Implement traditional PP → Minimal (200ish) PP Gist
3. Implement SWARM → Yandex SWARM + PI ZeroBand

NanoGPT setup → (GPT-2 (140m) on FineWebEdu on A100)

- Allow for truly distributed workers (IPv4+port, instead of local cuda worker -> might have to switch communication backedn)
- Implement missing functionality (remote logging, sampling, MP)
- Benchmark speed and convergence

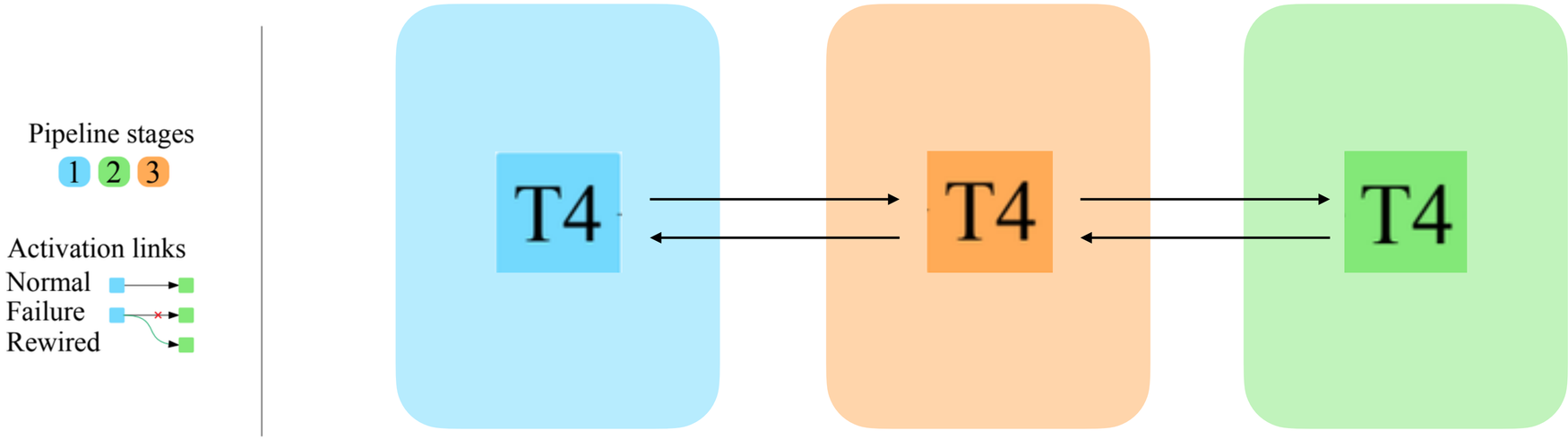


- Finished simple PP implementation
  - Sampling (needed different comm pattern)
  - Checkpointing (checkpointing local model)
  - Verification experiments (memorization + benchmarking)
- *(Also: Removed HF dependency for model)*
- Started SWARM implementation

Finished Implementing PP (AFAB)

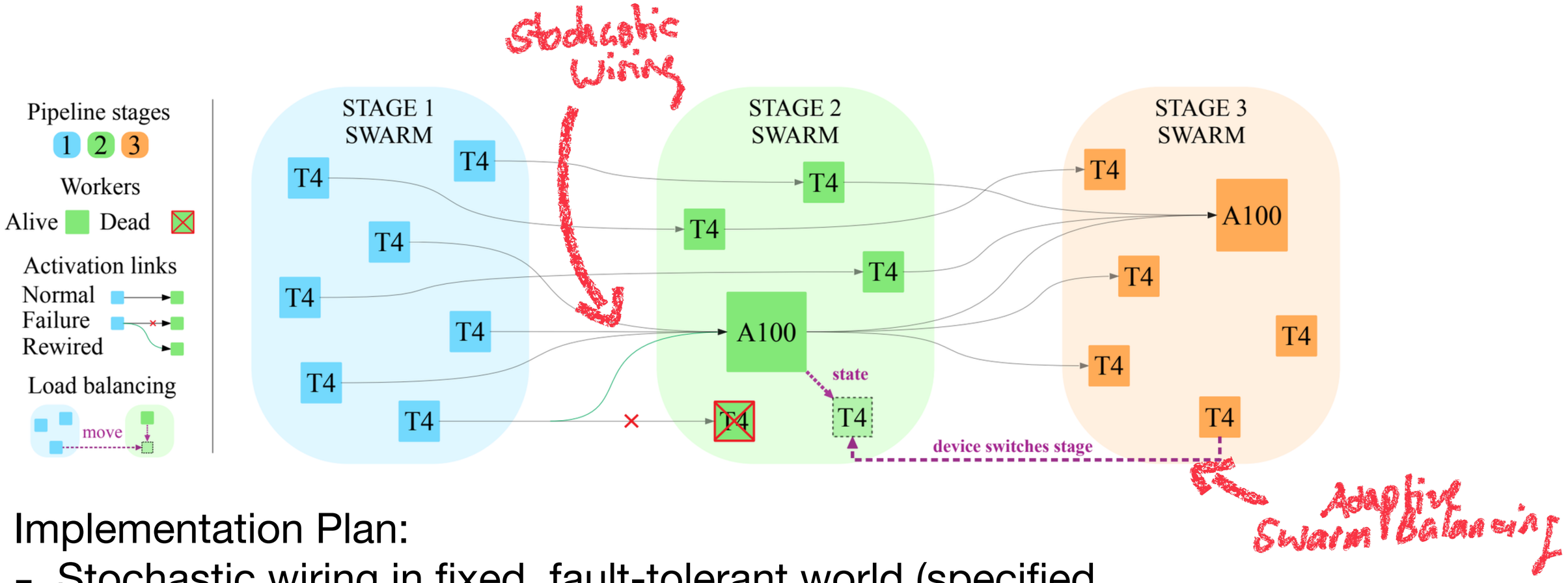
- In-/post-training evaluation
- Sampling
- Logging (Local/ W&B)
- Checkpointing
- LR Scheduling
- Mixed Precision Training

What we have



No fault tolerance !!

What we want

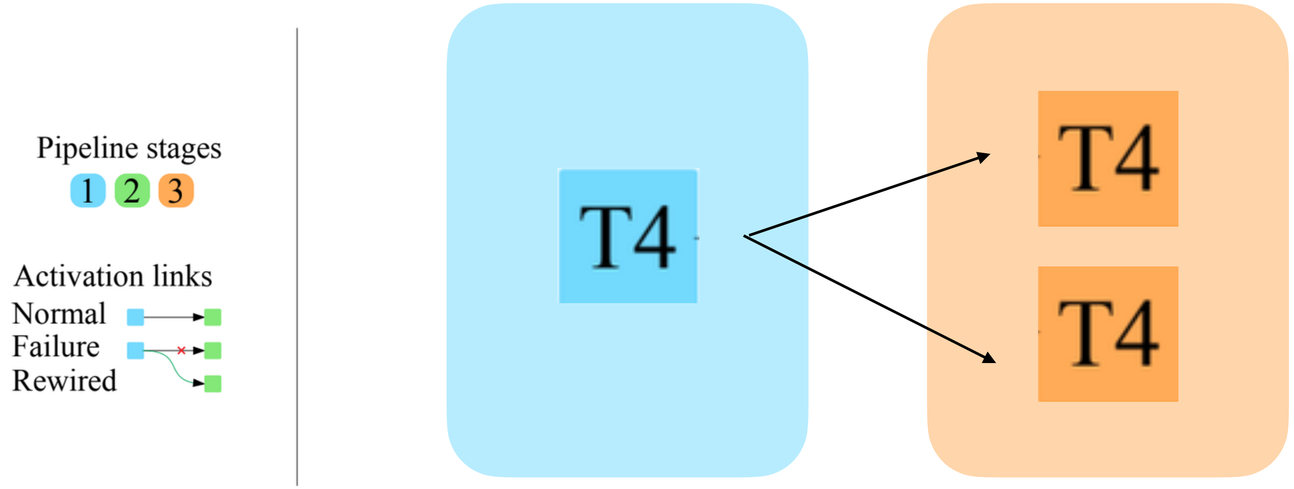


Implementation Plan:

- Stochastic wiring in fixed, fault-tolerant world (specified num_stages, ranks, world_size and uniform distribution of workers across stages -> less sync and headache)
- Fault tolerance via banning servers + periodic swarm balancing
- Move out of sandbox and to internet

Where we start: Forward pass

Send hidden states from one worker to one of two in next stage



Current problem: torch.dist.recv() is blocking and torch.dist.irecv() cannot be cancelled leading to memory errors for wrongly allocated buffers on step > 1

Idea: Pre-compute full computation graph per step? (but this is sync)

Later

- Backward pass (Backtrace computation graph to send gradients)
- Distributed data loader (Ensure that workers in first stage saple disjoint data shards, st. at end of step each sample is backpropped exactly once)
- Optimize with buffers (have to be adjusted per device VRAM)
- Make work over public IP (Tailscale)
- *Dynamic resizing of world (reinitializing process group?, …) and swarm balancing*