

BATTLE-TESTING SWARM

The SWARM learns

MIKA
SENGHAAS

EPFL

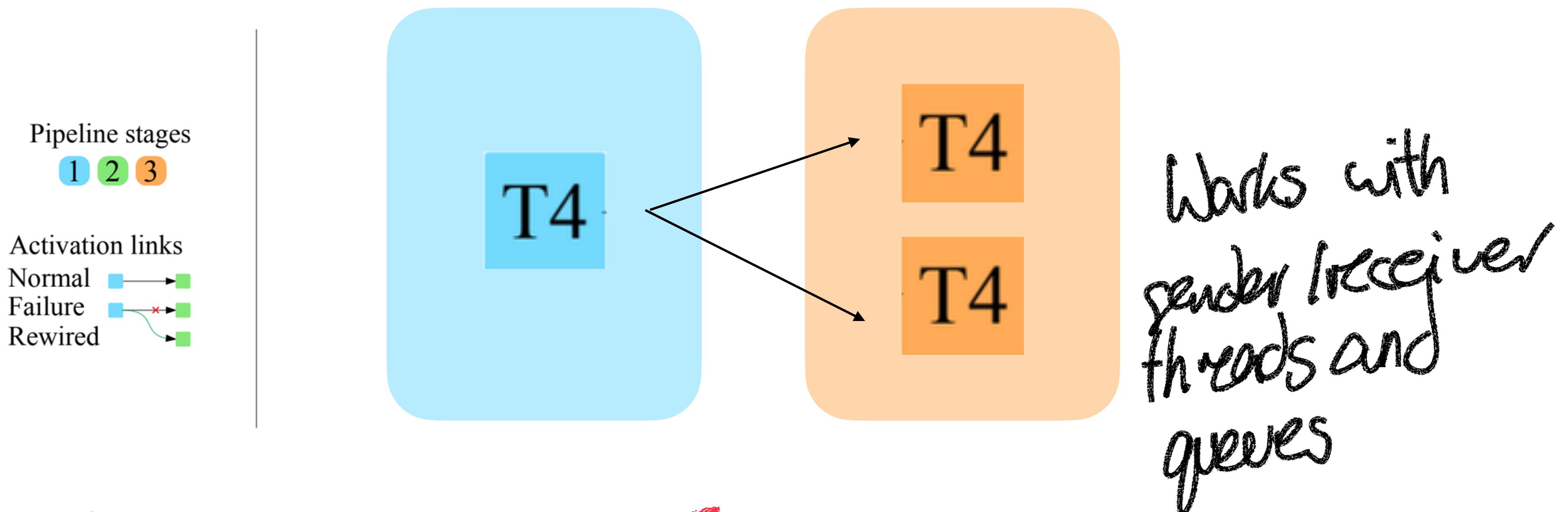
Next Steps

- ① ~~Design first prototype~~
- ② ~~Implement traditional API~~
- ③ Implement SWARM
- ④ Experiments

Where we start: Forward pass

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

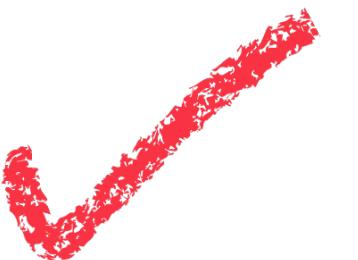
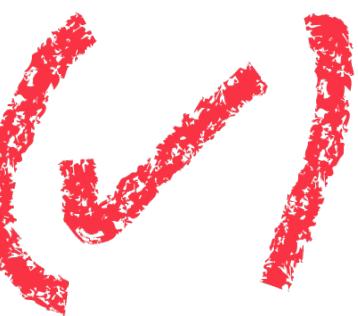
Last
Week



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 sample disjoint data shards, s.t. at end of step each sample is backpropagated 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*

Work < V ⌂

github.com

ChatGPT Perplexity EPFL CS450 EPFL CS451 EPFL CS524 ed Ed

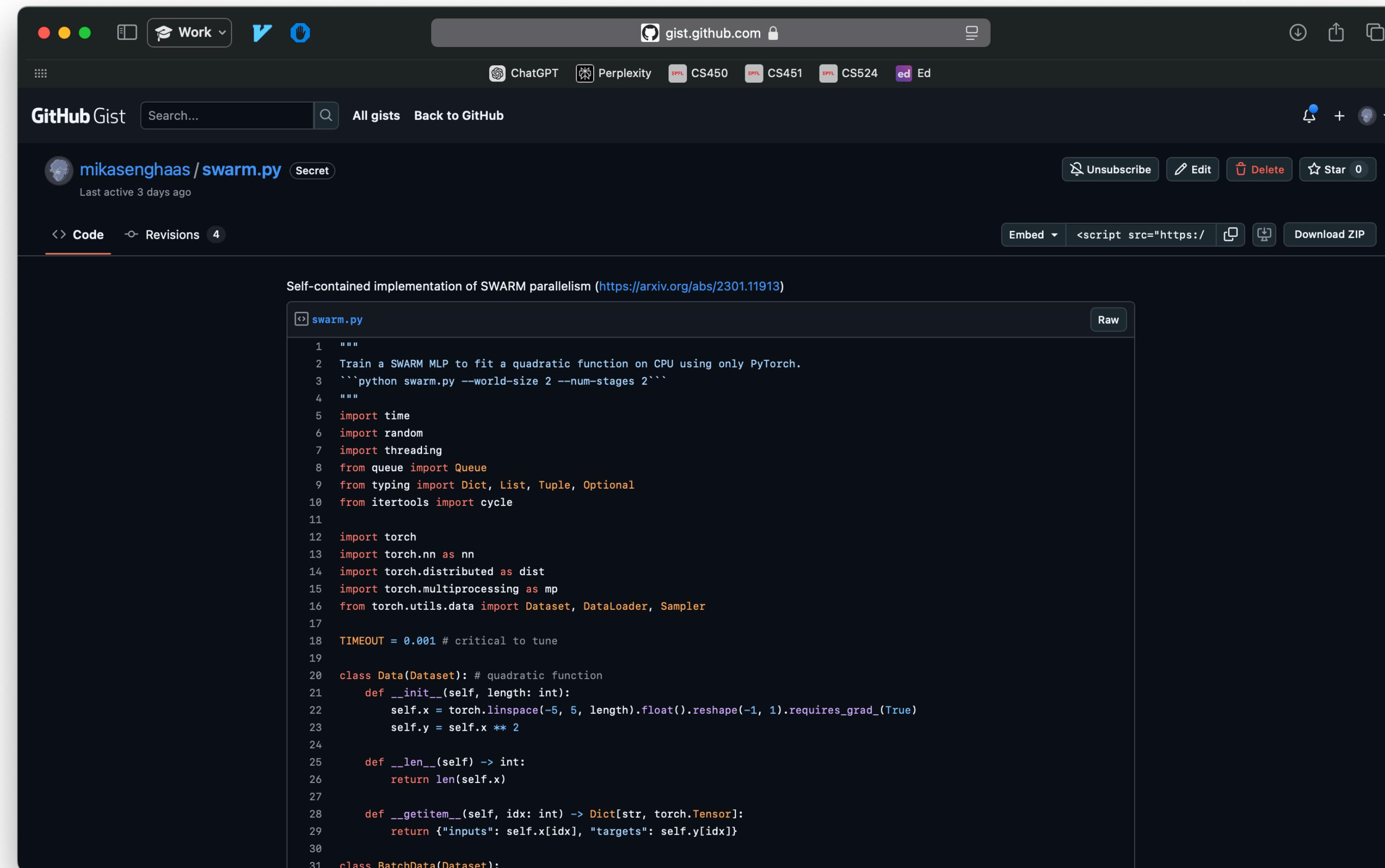
Commits

master All users All time

- o- Commits on Nov 24, 2024
 - Integrate SWARM into experiment code**
mikasenghaas committed 2 days ago Verified 03defd5 ⌂ ↗
- o- Commits on Nov 23, 2024
 - Add CLI args and some other minor improv**
mikasenghaas committed 3 days ago Verified 77bd099 ⌂ ↗
 - Refactored to 361 loc**
mikasenghaas committed 3 days ago Verified ff9ac90 ⌂ ↗
 - Cleanup and fix CPU thrashing in send loop**
mikasenghaas committed 3 days ago Verified 4191cb8 ⌂ ↗
 - Refactor minimal implementation of SWARM**
mikasenghaas committed 3 days ago Verified 79f01d0 ⌂ ↗
 - Implement arbitrarily large stages**
mikasenghaas committed 3 days ago Verified ee644ac ⌂ ↗
- o- Commits on Nov 22, 2024
 - Visualize learning quadratic function (slightly broken for microbatch != batch size != num_samples)**
mikasenghaas committed 4 days ago Verified 6dfb6e0 ⌂ ↗
 - Implement rudimentary backward pass for any 2-stage swarm**
mikasenghaas committed 4 days ago Verified 7c69719 ⌂ ↗
 - Ignore remote saved checkpoints**
mikasenghaas committed 4 days ago Verified 1483f78 ⌂ ↗
 - Fix uneven (micro-) batches bug**
mikasenghaas committed 4 days ago Verified 8ac62ae ⌂ ↗
 - Implement distributed sampler for multiple workers in first stage**
mikasenghaas committed 4 days ago Verified ac2ce5e ⌂ ↗
 - Implement stochastic forward pass**
mikasenghaas committed 4 days ago Verified 6b6149b ⌂ ↗

<https://gist.github.com/mikasenghaas/5fa1aa77ea69f187f531a5889983c249>

Pure PyTorch



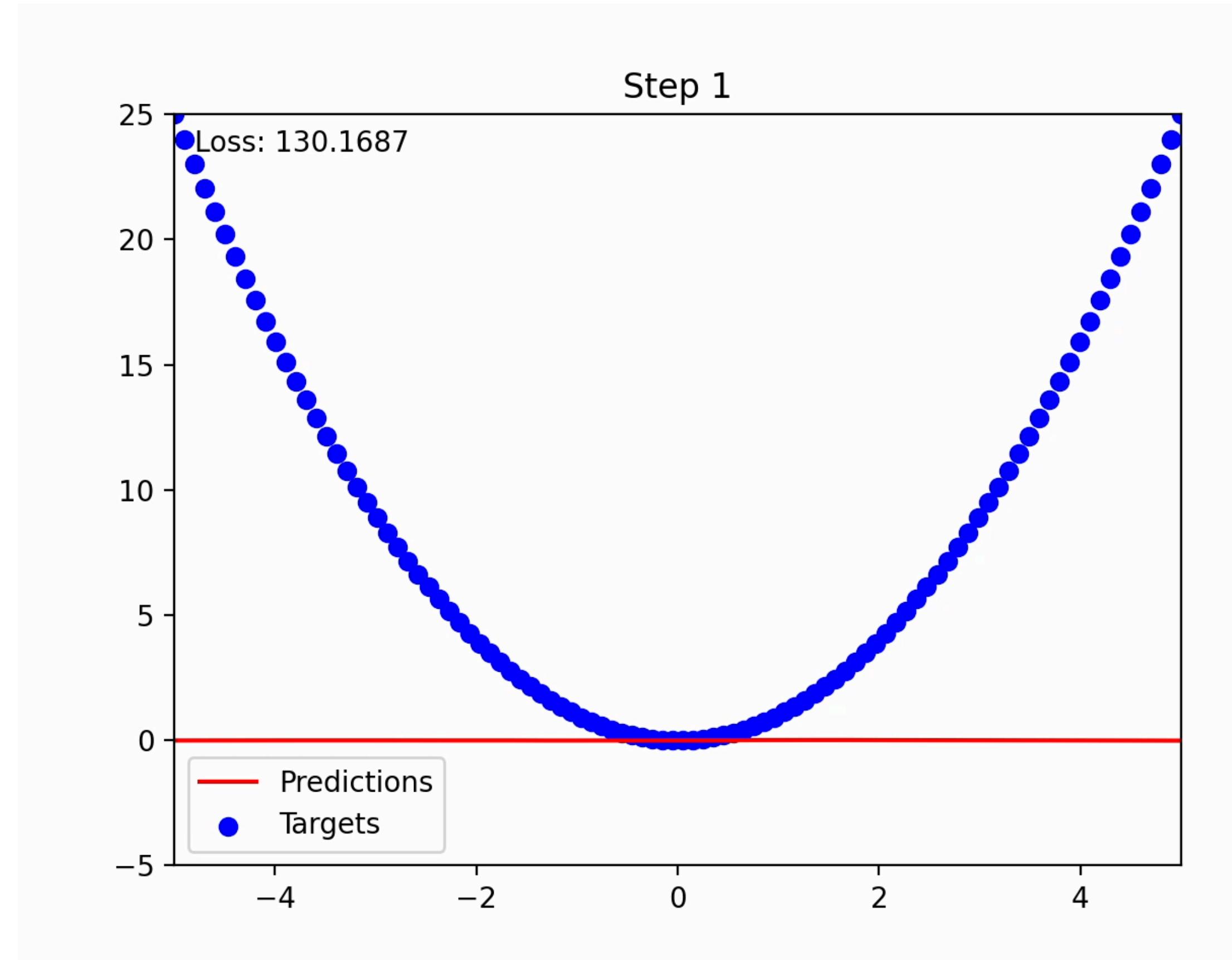
The screenshot shows a GitHub Gist page with a dark theme. The title of the gist is "mikasenghaas / swarm.py" and it is labeled as "Secret". Below the title, it says "Last active 3 days ago". There are four revisions shown. The code is titled "Self-contained implementation of SWARM parallelism (<https://arxiv.org/abs/2301.11913>)". The code itself is a Python script named "swarm.py" with the following content:

```
1 """
2 Train a SWARM MLP to fit a quadratic function on CPU using only PyTorch.
3 ``python swarm.py --world-size 2 --num-stages 2``
4 """
5 import time
6 import random
7 import threading
8 from queue import Queue
9 from typing import Dict, List, Tuple, Optional
10 from itertools import cycle
11
12 import torch
13 import torch.nn as nn
14 import torch.distributed as dist
15 import torch.multiprocessing as mp
16 from torch.utils.data import Dataset, DataLoader, Sampler
17
18 TIMEOUT = 0.001 # critical to tune
19
20 class Data(Dataset): # quadratic function
21     def __init__(self, length: int):
22         self.x = torch.linspace(-5, 5, length).float().reshape(-1, 1).requires_grad_(True)
23         self.y = self.x ** 2
24
25     def __len__(self) -> int:
26         return len(self.x)
27
28     def __getitem__(self, idx: int) -> Dict[str, torch.Tensor]:
29         return {"inputs": self.x[idx], "targets": self.y[idx]}
30
31 class BatchData(Dataset):
```

Implements stochastic wiring

[ATM, fully uniform
and not based on throughputs]

350 LOC



Proof of Concept

N workers, T stages SWARM
learns the quadratic function

So, this is it?

The screenshot shows a GitHub Gist page for a file named 'swarm.py'. The page title is 'Self-contained implementation of SWARM parallelism (<https://arxiv.org/abs/2301.11913>)'. The code is as follows:

```
1 """
2 Train a SWARM MLP to fit a quadratic function on CPU using only PyTorch.
3 ````python swarm.py --world-size 2 --num-stages 2````"
4 """
5 import time
6 import random
7 import threading
8 from queue import Queue
9 from typing import Dict, List, Tuple, Optional
10 from itertools import cycle
11
12 import torch
13 import torch.nn as nn
14 import torch.distributed as dist
15 import torch.multiprocessing as mp
16 from torch.utils.data import Dataset, DataLoader, Sampler
17
18 TIMEOUT = 0.001 # critical to tune
19
20 class Data(Dataset): # quadratic function
21     def __init__(self, length: int):
22         self.x = torch.linspace(-5, 5, length).float().reshape(-1, 1).requires_grad_(True)
23         self.y = self.x ** 2
24
25     def __len__(self) -> int:
26         return len(self.x)
27
28     def __getitem__(self, idx: int) -> Dict[str, torch.Tensor]:
29         return {"inputs": self.x[idx], "targets": self.y[idx]}
30
31 class BatchData(Dataset):
```

- Maintain throughput estimates in SW
- Not fault-tolerant
 - Dynamic World Size
(Join / Leave)
 - Node Rebalancing

What's Next?

- ① Integrate w/ existing experiment codebase
(LLM pre-training)
- ② Define RQ \rightarrow Experiments
- ③ Develop feature subset enabling experiments

Direction 1: SWARM + DiLoGo

Obs SWARM is step-synchronous
⇒ stage peers all-reduce gradients every step

RQ Can we get away with sync every T steps?
⇒ T comm-reduction

Hypothesis DiLoGo's $T=400$, inner-outer
optimizer scheme might be applicable

⇒ Setup SWARM with $T = \{1, 2, 4, \dots\}$ and observe
e.g. wall-clock to accuracy

Direction 2 : SWARM Init

Obs

Consider workers $i \in [n]$ with comm matrix
 $M \in \mathbb{R}^{n \times n}$, M_{ij} = Network Speed between i, j .

Consider 2 data centres, then

$$M = \begin{bmatrix} & & \\ \text{---} & \text{---} & \\ & & \end{bmatrix} \leftarrow \begin{array}{l} \text{blocks represent intra-node} \\ \text{connections} \end{array}$$

\Rightarrow SWARM assigns workers to stages at random

RQ How to best allocate workers to stages?

\Rightarrow Analyze comm. complexity as function of steps, stages/
model

M = Steps, N stages, $|\Theta|$ Model params

B, L, H = Batch, Sequence Length, Hidden Dim

Forward/
Backward : $B(N-1)2 \cdot BLH$ ← Should consider DP

Batch ↑
Common
between stages ↑
Batch ↑

Fwd/Bwd

All Reduce: $(N) \frac{|\Theta|}{N} \cdot$ All Reduce Complexity.
↑
In parallel