Decentralized Deep Learning

Running Large Neural Networks Together

Max Ryabinin

Senior Research Scientist, Yandex Research

PhD Student, HSE University

Talk outline

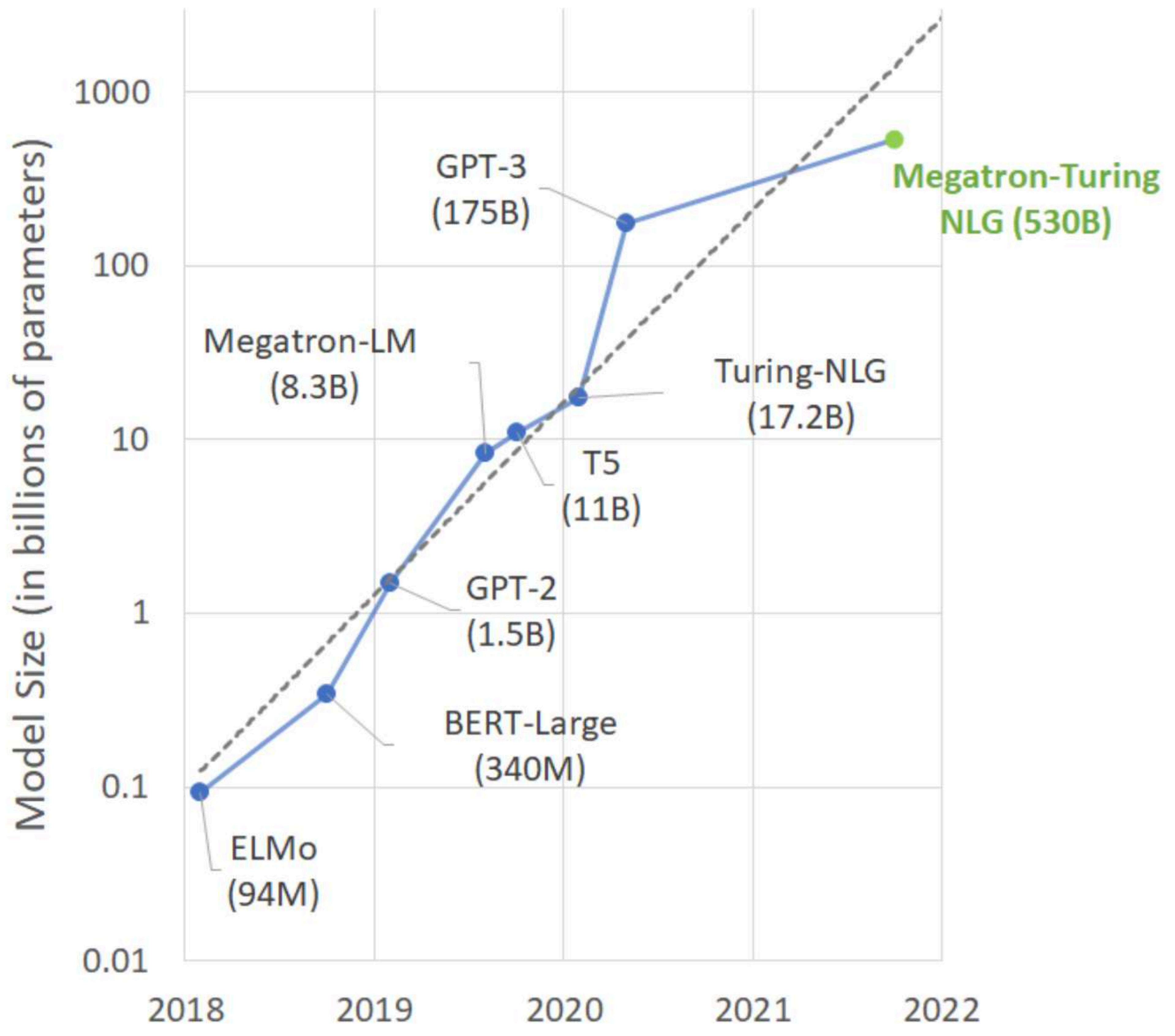
- Motivation and key challenges
- Decentralized training
  - Specialized architectures
  - General data-parallel training
  - Model-parallel training
- Decentralized inference of pretrained models

The state of deep learning in 2023

〉 Large-scale training becomes more popular

〉 Scaling laws promise continued gains

〉 Larger models have emergent abilities (e.g. in-context learning)



img: arxiv.org/abs/2201.11990

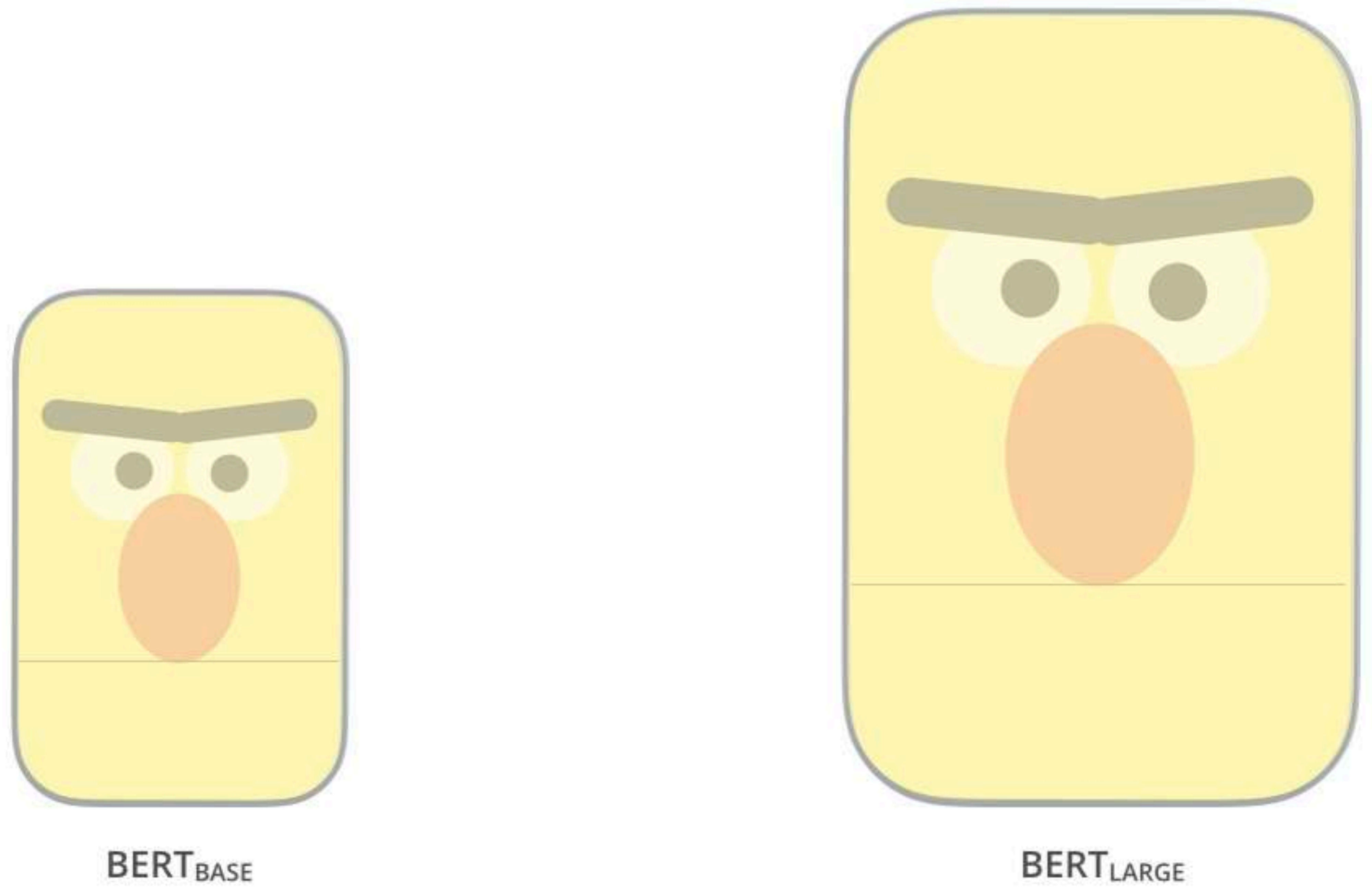
Implications of scaling

- Some models cost hundreds of thousands to train — or more!
- GPT-3 training costs millions of $
- Hard to train for an average researcher and advance the field

img: jalammar.github.io/illustrated-bert

Implications of scaling

- Some models cost hundreds of thousands to train — or more!
- GPT-3 training costs millions of $
- Hard to train for an average researcher and advance the field



110-340M parameters: might work

img: jalammar.github.io/illustrated-bert

Implications of scaling

〉 Some models cost hundreds of thousands to train — or more!

〉 GPT-3 training costs millions of $

〉 Hard to train for an average researcher and advance the field

100B parameters: infeasible!

img: jalammar.github.io/illustrated-bert

Solution: share the effort

- Collaboration has worked in other sciences
- Let's use idle volunteer resources for DL as well!

Challenges of volunteer deep learning

- **Node failures**: PC turns off, Internet gets disabled, ...
- **Communication over Internet:** magnitudes slower than clusters
- **Heterogeneous hardware:** different GPUs, connection speeds etc.

Existing approaches for distributed DL

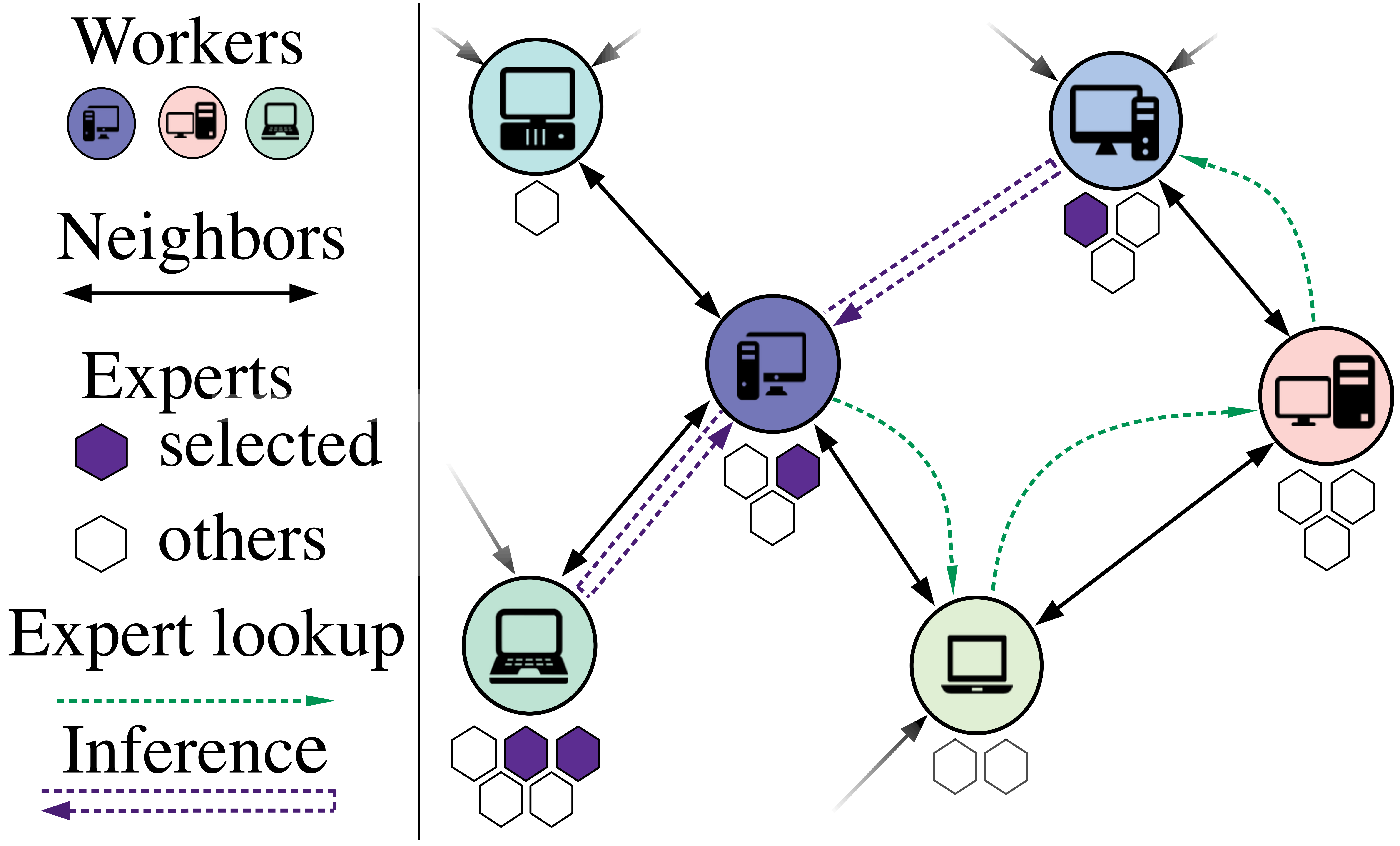
Training method	**Size limit**	**Throughput**	**Scalability**	**Fault tolerance**	**Worker hot-join**	**Network bandwidth**	**Network latency**
Data parallel	Worker	**High**	Medium	**Full**	**Yes**	High	Low
Asynchronous	Worker	**High**	High	Only workers	**Yes**	Medium	**Any**
Model parallel	**System**	Medium	Low	No	No	High	Low
Federated	Worker	Low	**High**	Only workers	**Yes**	**Low**	**Any**
Desired	**System**	**High**	**High**	**Full**	**Yes**	**Low**	**Any**

Talk outline

- 〉 Motivation and key challenges
- 〉 Decentralized training
  - Specialized architectures
  - General data-parallel training
  - Model-parallel training
- 〉 Decentralized inference of pretrained models

Learning@home (NeurIPS’20)

Towards Crowdsourced Training of Large Neural Networks using Decentralized Mixture-of-Experts



DMoE (Decentralized Mixture-of-Experts)

DMoE (<u>D</u>ecentralized <u>M</u>ixture-<u>o</u>f-<u>E</u>xperts)

Experts
Input

DMoE (<u>D</u>ecentralized <u>M</u>ixture-<u>o</u>f-<u>E</u>xperts)

Experts
Input

DMoE (Decentralized Mixture-of-Experts)

Peer 1
Peer 2
Peer 3
Peer 5
Peer 6
Peer 4
Input

DMoE (<u>D</u>ecentralized <u>M</u>ixture-<u>o</u>f-<u>E</u>xperts)

DMoE (<u>D</u>ecentralized <u>M</u>ixture-<u>o</u>f-<u>E</u>xperts)

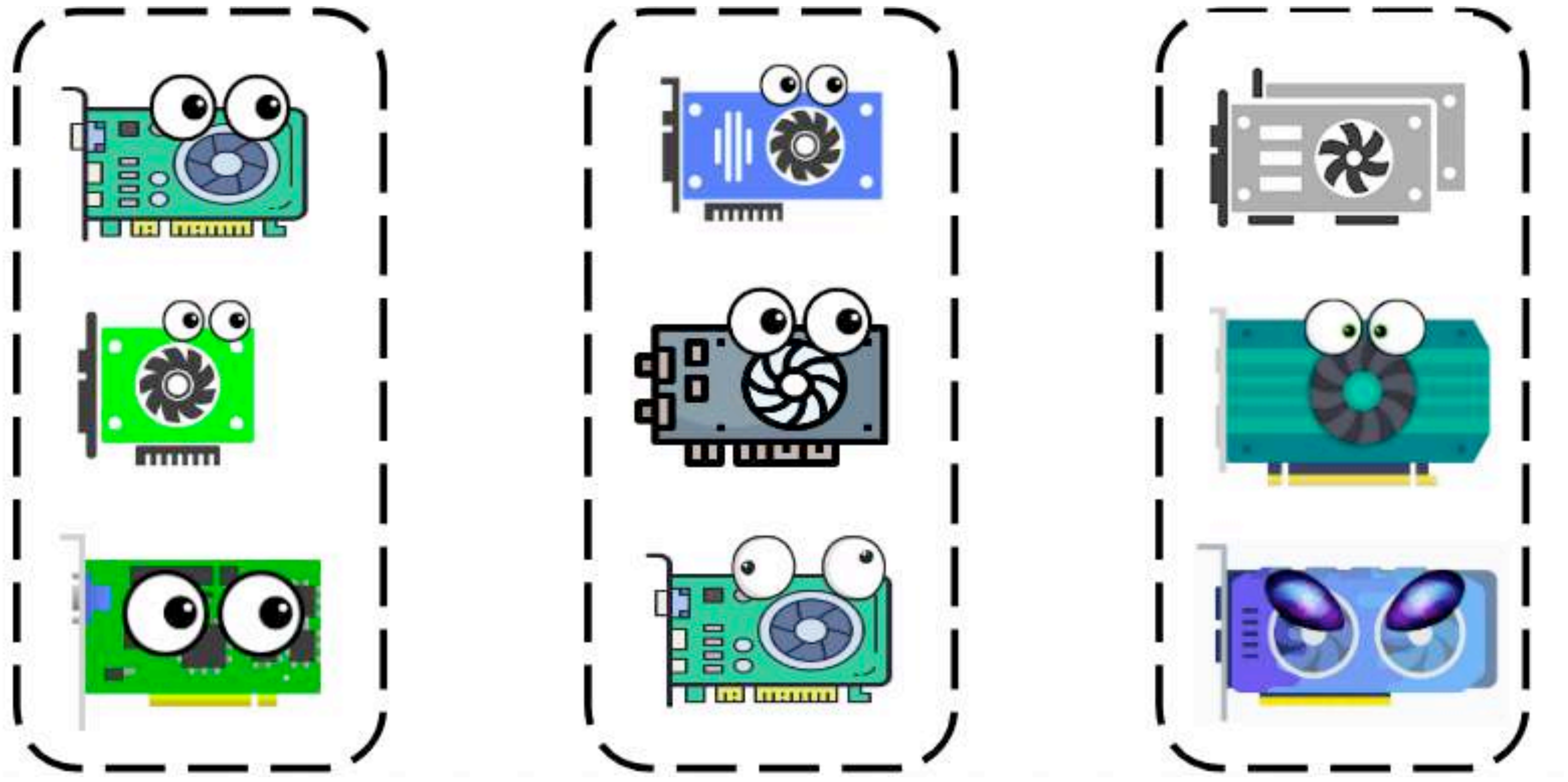
Output
Input

Moshpit SGD (NeurIPS'21)

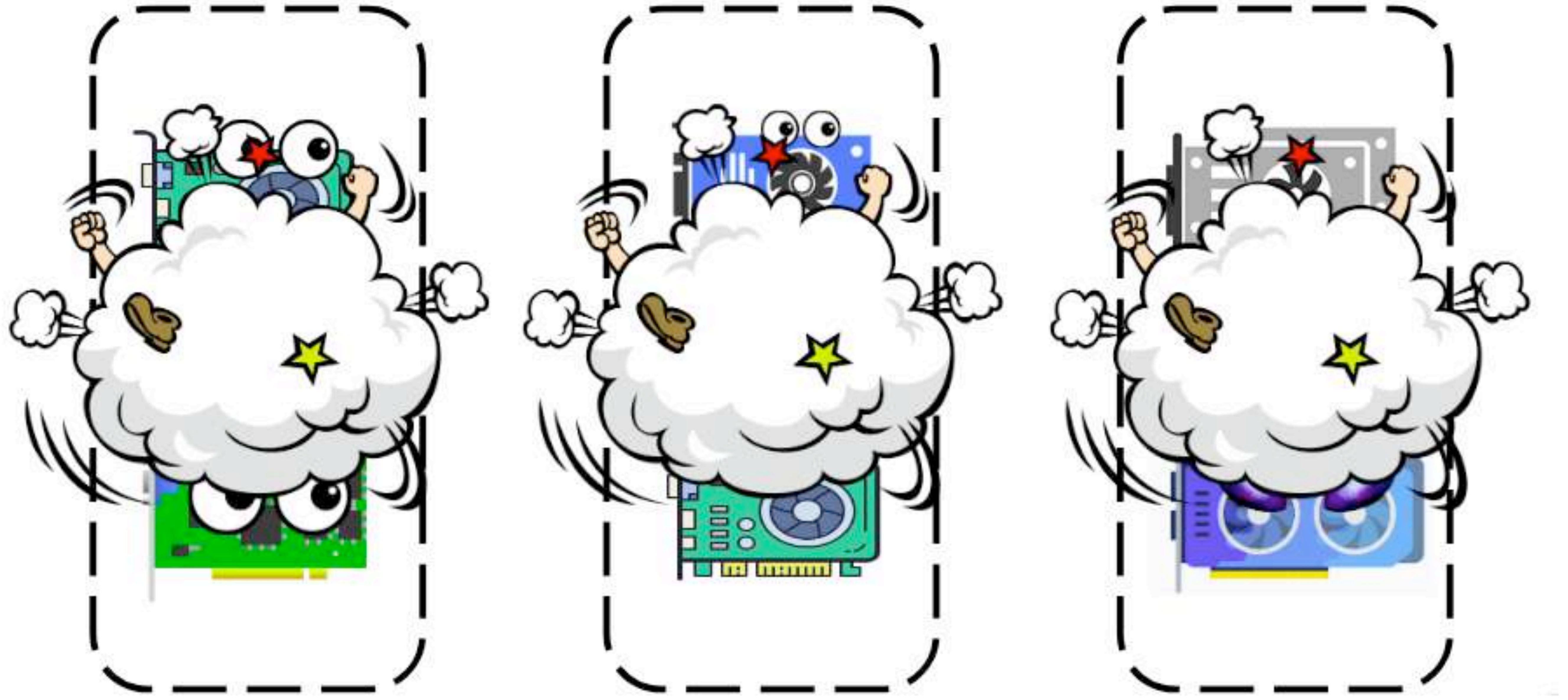
Communication-Efficient Decentralized Training
on Heterogeneous Unreliable Devices

- How to average the gating function/embeddings?
- We propose a new algorithm for decentralized AllReduce-like averaging
- Main idea: average in smaller non-overlapping groups
- Communication-efficient and fault-tolerant, useful even on its own

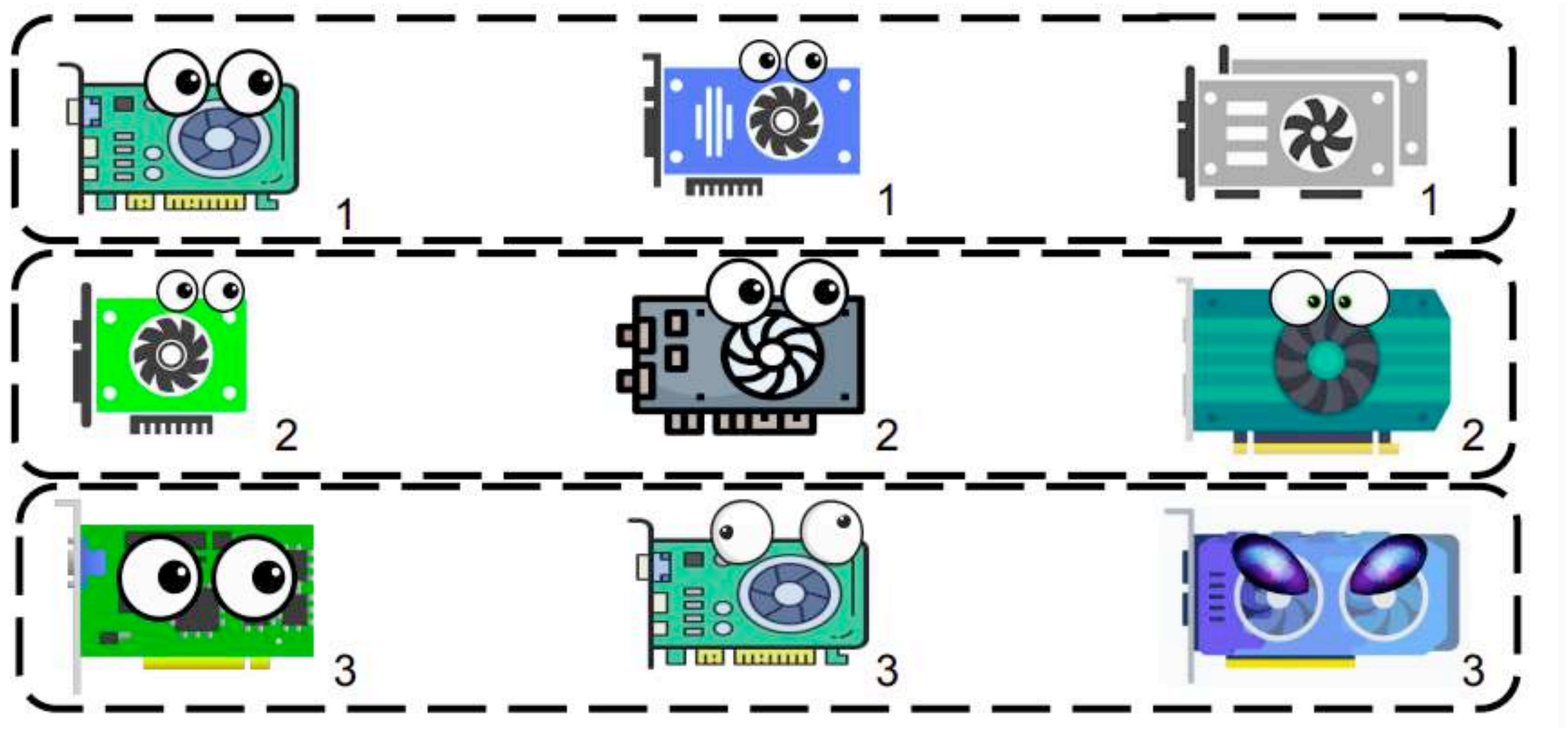
Moshpit All-Reduce: core idea



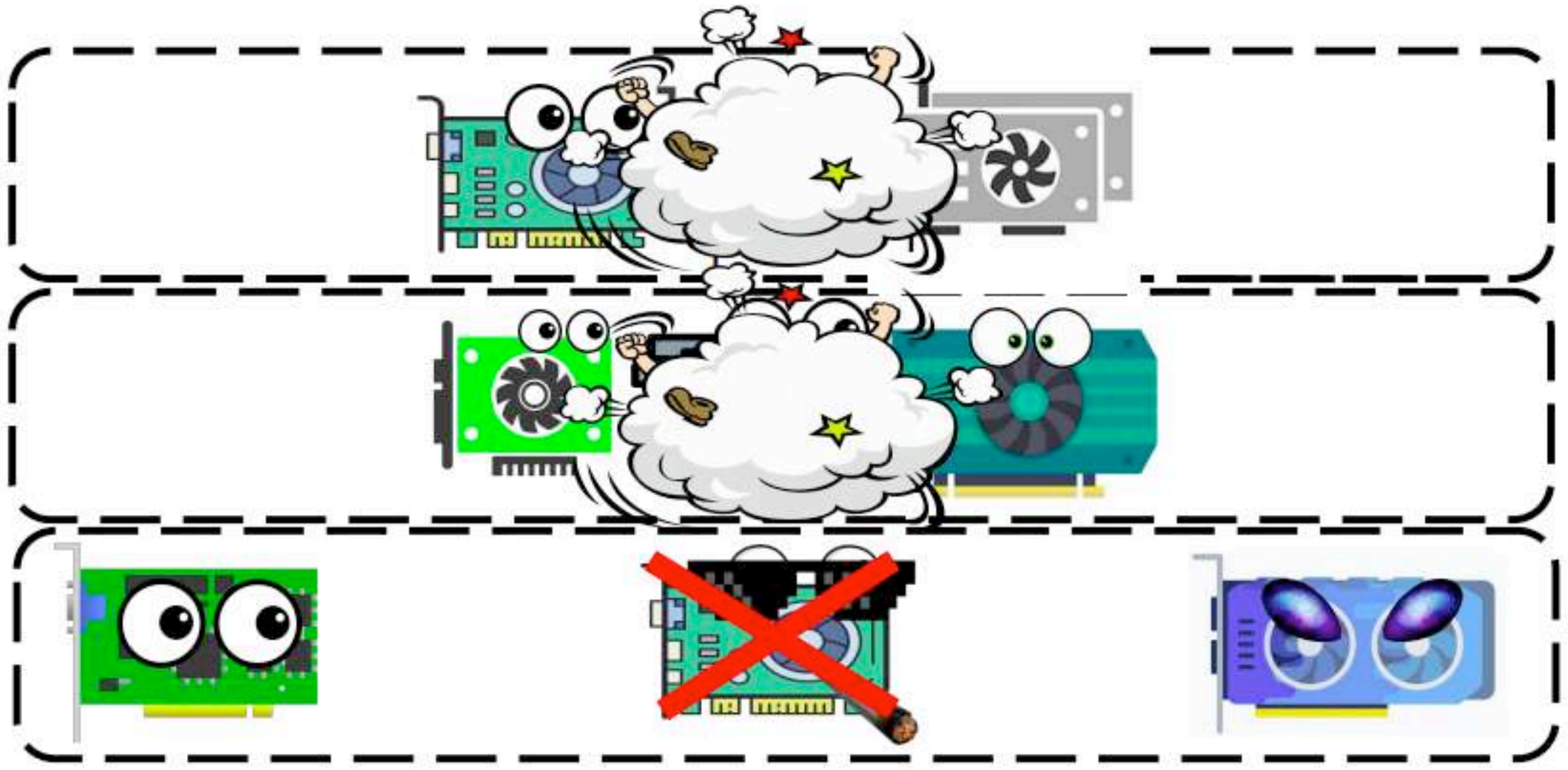
Moshpit All-Reduce: core idea



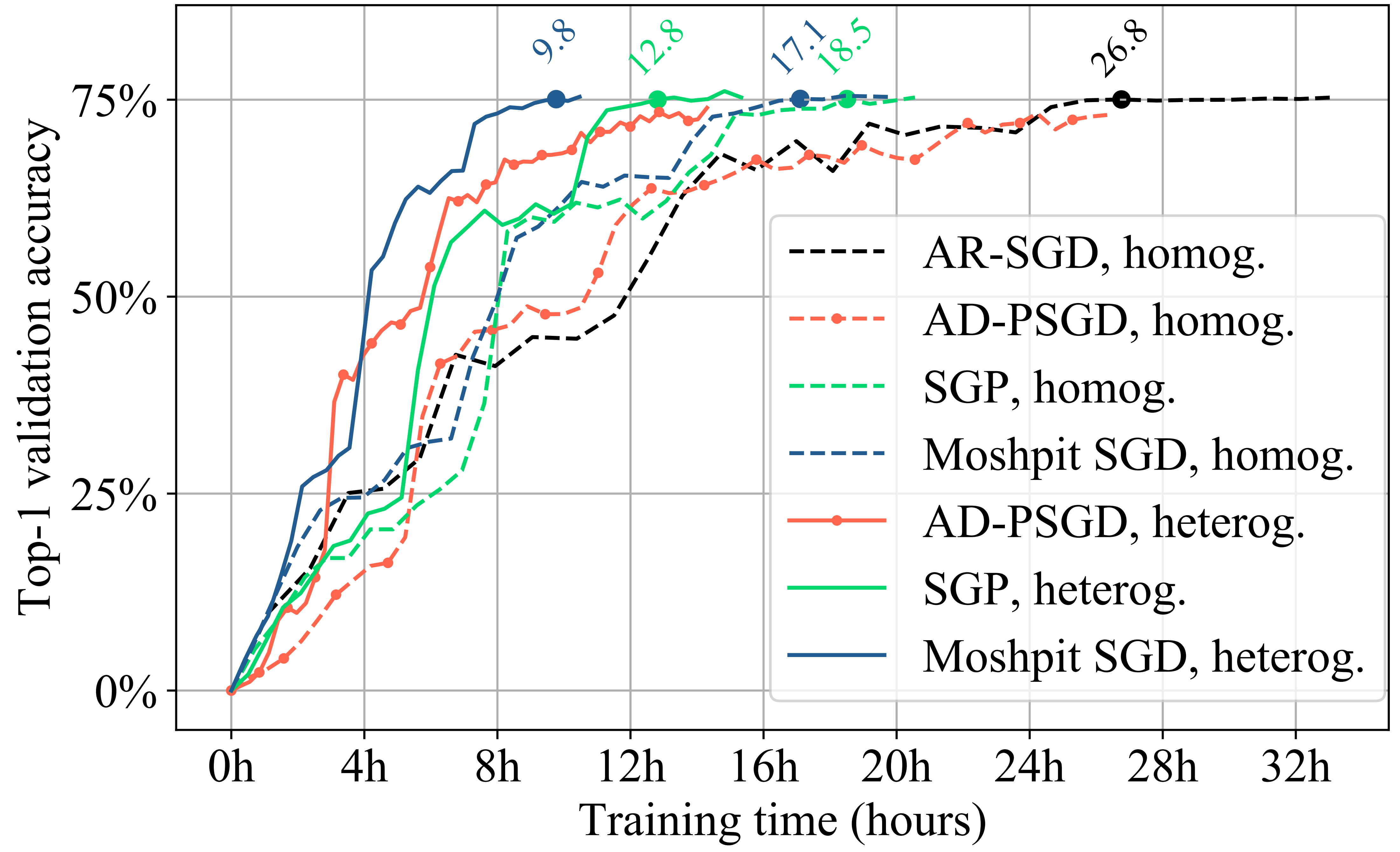
Moshpit All-Reduce: core idea

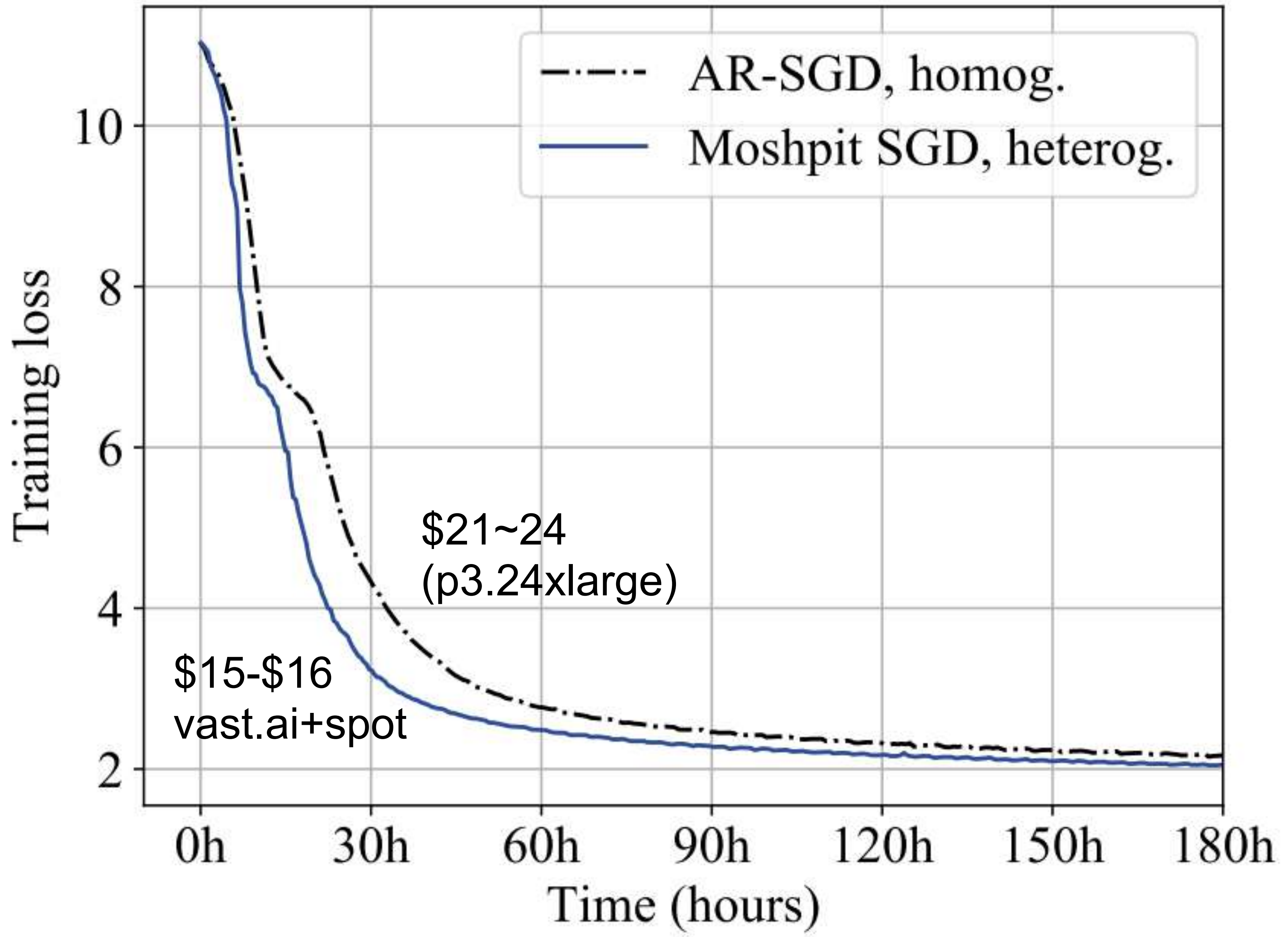


Moshpit All-Reduce: core idea



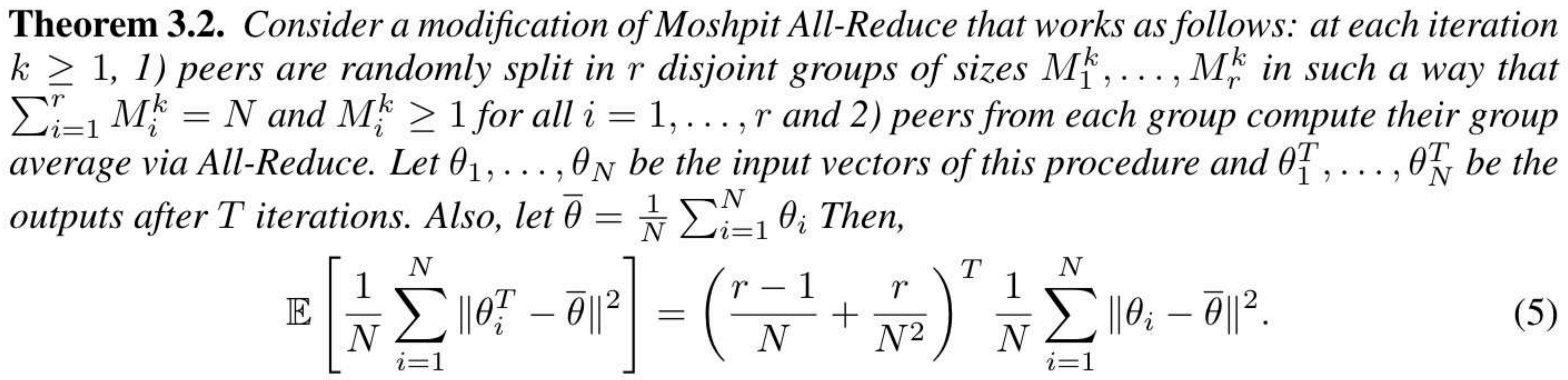
Experiments





Analysis TL;DR

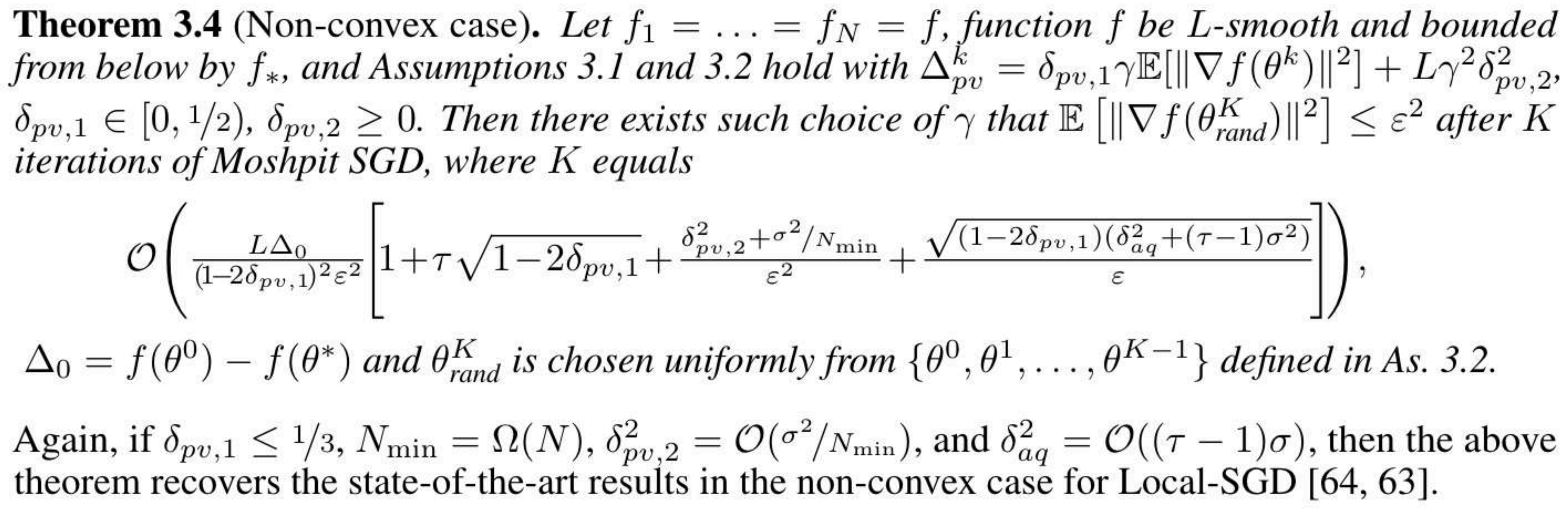
〉 The averaging converges exponentially quickly



Theorem 3.2. *Consider a modification of Moshpit All-Reduce that works as follows: at each iteration $k \geq 1$, 1) peers are randomly split in r disjoint groups of sizes $M_1^k, \ldots, M_r^k$ in such a way that $\sum_{i=1}^r M_i^k = N$ and $M_i^k \geq 1$ for all $i = 1, \ldots, r$ and 2) peers from each group compute their group average via All-Reduce. Let $\theta_1, \ldots, \theta_N$ be the input vectors of this procedure and $\theta_1^T, \ldots, \theta_N^T$ be the outputs after T iterations. Also, let $\overline{\theta} = \frac{1}{N}\sum_{i=1}^N \theta_i$ Then,*

$$\mathbb{E}\left[\frac{1}{N}\sum_{i=1}^N \|\theta_i^T - \overline{\theta}\|^2\right] = \left(\frac{r-1}{N} + \frac{r}{N^2}\right)^T \frac{1}{N}\sum_{i=1}^N \|\theta_i - \overline{\theta}\|^2. \qquad (5)$$

〉 For Moshpit SGD — equivalent results to Local SGD



Theorem 3.4 (Non-convex case). *Let $f_1 = \ldots = f_N = f$, function f be L-smooth and bounded from below by f_*, and Assumptions 3.1 and 3.2 hold with $\Delta_{pv}^k = \delta_{pv,1}\gamma\mathbb{E}[\|\nabla f(\theta^k)\|^2] + L\gamma^2\delta_{pv,2}^2$, $\delta_{pv,1} \in [0, 1/2)$, $\delta_{pv,2} \geq 0$. Then there exists such choice of γ that $\mathbb{E}\left[\|\nabla f(\theta_{rand}^K)\|^2\right] \leq \varepsilon^2$ after K iterations of Moshpit SGD, where K equals*

$$\mathcal{O}\left(\tfrac{L\Delta_0}{(1-2\delta_{pv,1})^2\varepsilon^2}\left[1+\tau\sqrt{1-2\delta_{pv,1}}+\tfrac{\delta_{pv,2}^2+\sigma^2/N_{\min}}{\varepsilon^2}+\tfrac{\sqrt{(1-2\delta_{pv,1})(\delta_{aq}^2+(\tau-1)\sigma^2)}}{\varepsilon}\right]\right),$$

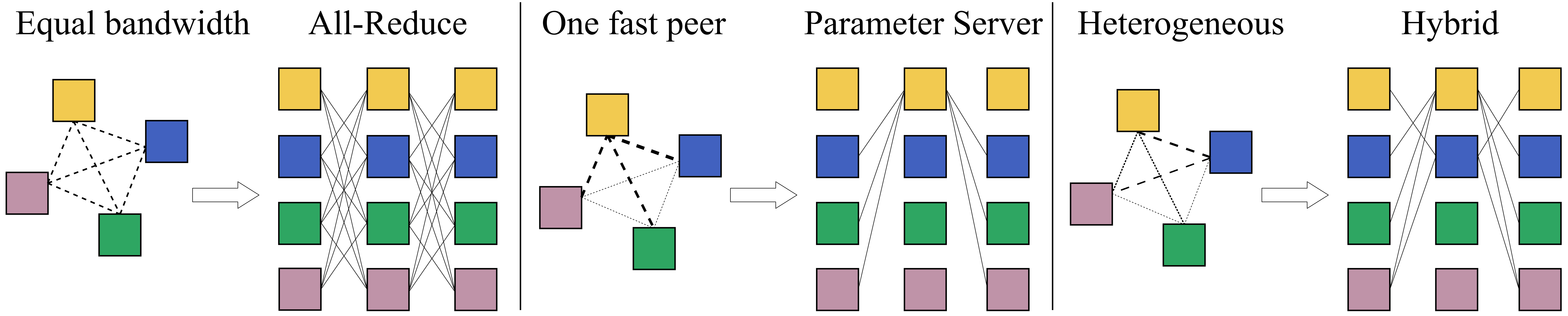
$\Delta_0 = f(\theta^0) - f(\theta^*)$ and θ_{rand}^K is chosen uniformly from $\{\theta^0, \theta^1, \ldots, \theta^{K-1}\}$ defined in As. 3.2.

Again, if $\delta_{pv,1} \leq 1/3$, $N_{\min} = \Omega(N)$, $\delta_{pv,2}^2 = \mathcal{O}(\sigma^2/N_{\min})$, and $\delta_{aq}^2 = \mathcal{O}((\tau-1)\sigma)$, then the above theorem recovers the state-of-the-art results in the non-convex case for Local-SGD [64, 63].

DeDLOC (NeurIPS'21)

(Distributed Deep Learning in Open Collaborations)

- How to scale decentralized training to real-life scenarios?
- Propose an averaging algorithm that dynamically adapts to network conditions
- Recovers regular distributed methods in special cases

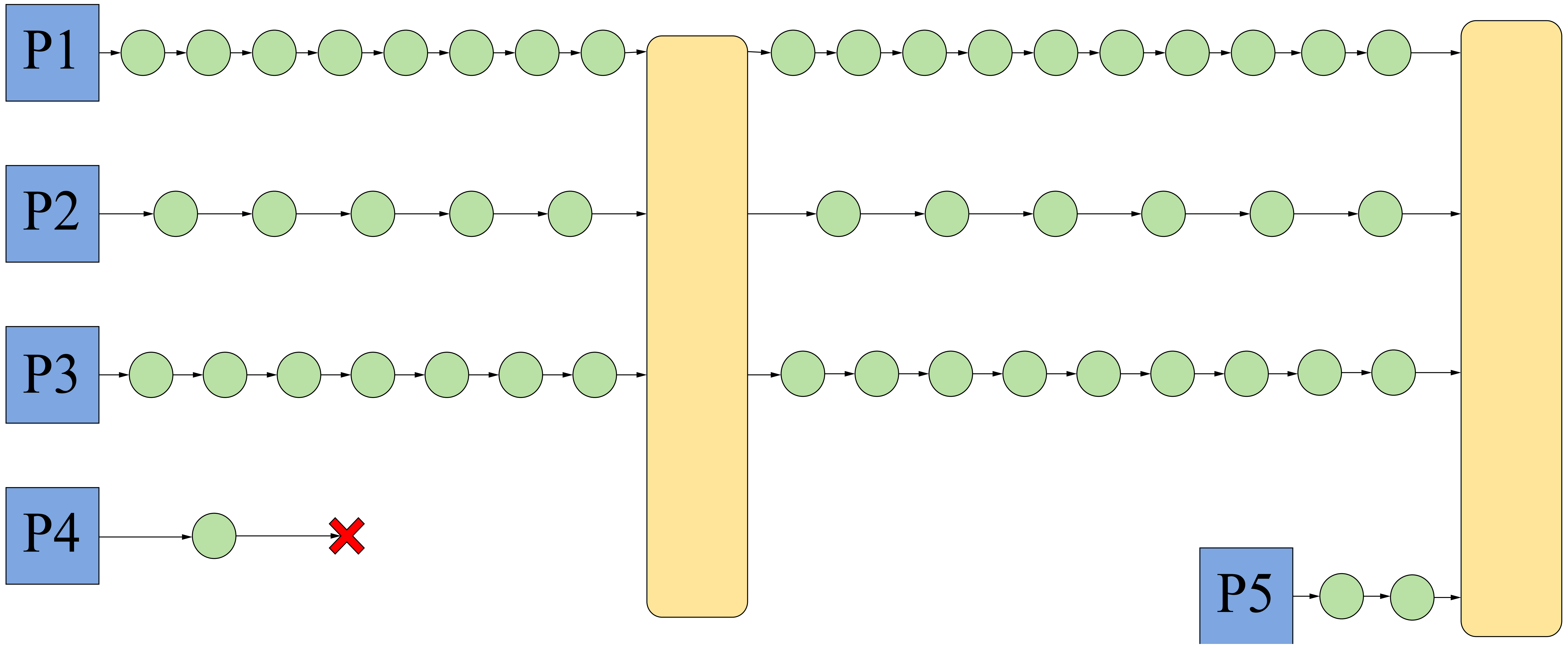


DeDLOC (NeurIPS'21)

(Distributed Deep Learning in Open Collaborations)

〉 For training, adopt large-batch SGD

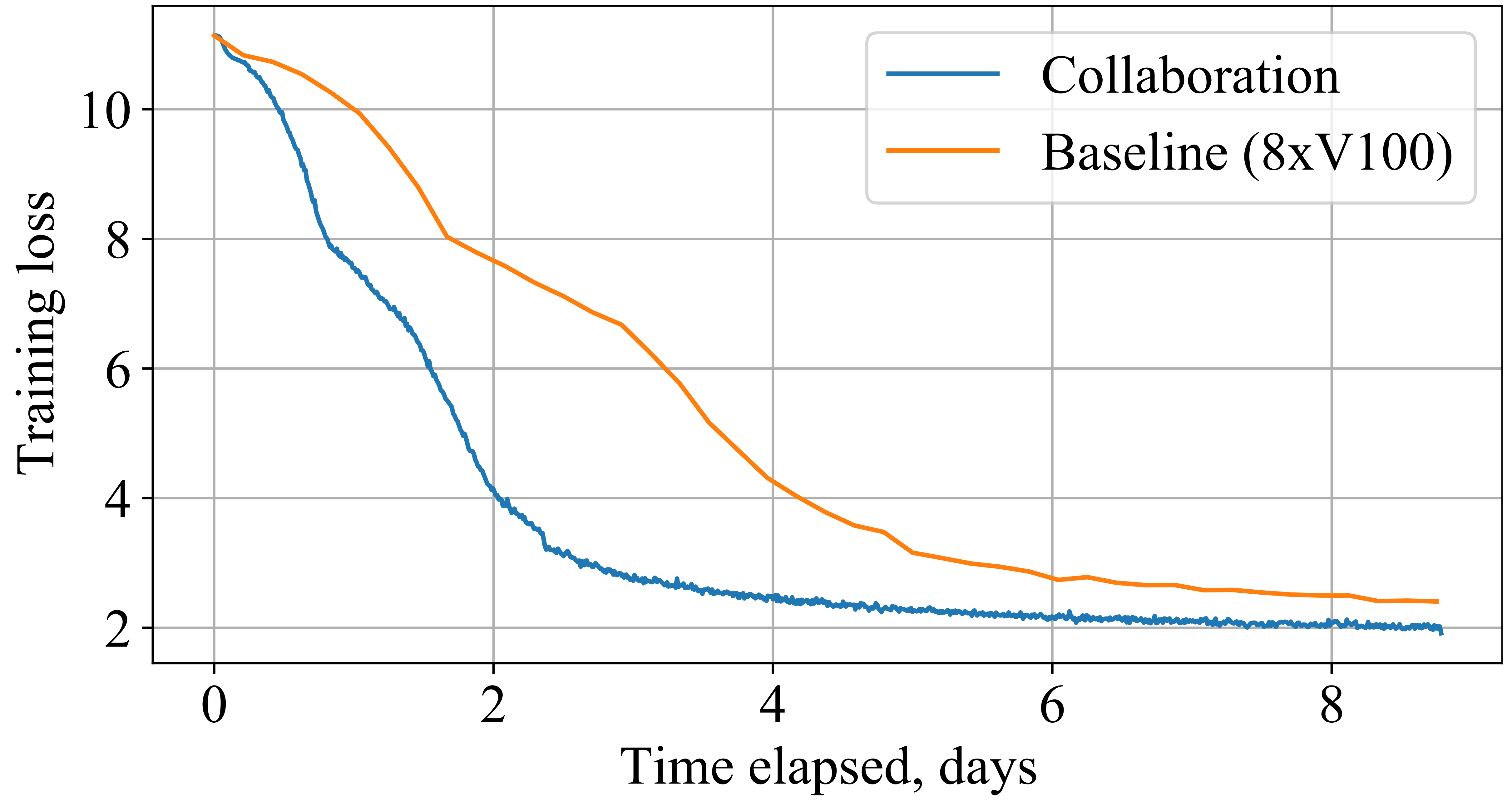
〉 Accumulate batches on peers, synchronize when target size is reached



Learn more: huggingface.co/blog/collaborative-training

sahajBERT: the first collaboratively-trained LM

- We enlist the help of ~40 volunteers from the Bengali community
- Pretrain ALBERT-large, results competitive to SoTA trained on clusters
- Participants used their servers and even Colab/Kaggle instances!



Model	Wikiann F1	NCC Accuracy
sahajBERT	95.45 ± 0.53	$\mathbf{91.97 \pm 0.47}$
XLM-R	$\mathbf{96.48 \pm 0.22}$	90.05 ± 0.38
IndicBERT	92.52 ± 0.45	74.46 ± 1.91
bnRoBERTa	82.32 ± 0.67	80.94 ± 0.45

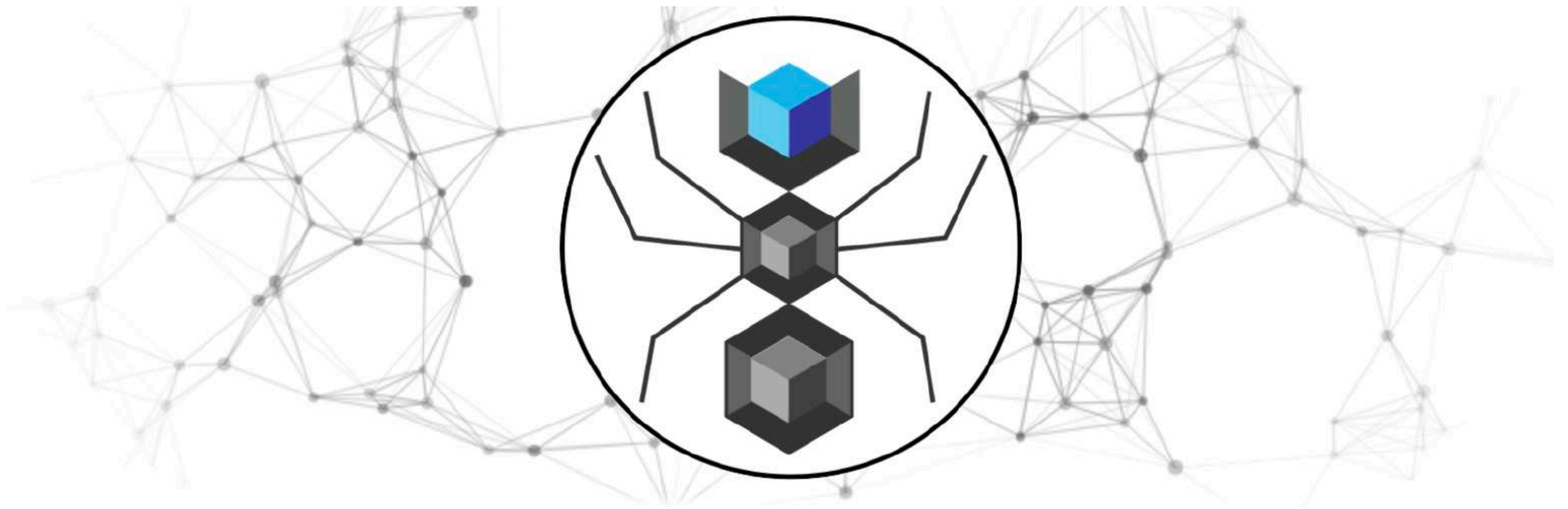
Training Transformers Together (NeurIPS’21 Demo)

〉 We pretrained a text-to-image model (similar to DALL-E) with volunteers

〉 The demo focuses on additional practical aspects of collaborative training:

- Dataset streaming
- Communication compression
- Memory-efficient training

training-transformers-together.github.io



Training Transformers Together

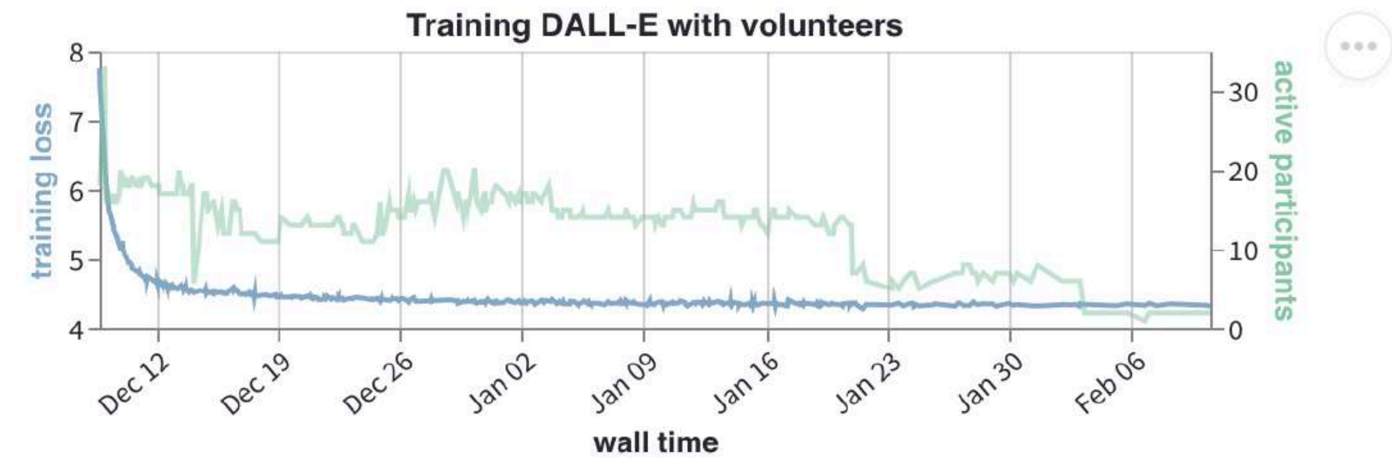
large-scale deep learning for everyone, by everyone

A NeurIPS 2021 Demonstration

UNIVERSITY of WASHINGTON

There was a time when you could comfortably train state-of-the-art vision and language models at home on your workstation. The first convolutional neural net to beat ImageNet (AlexNet) was trained for 5-6 days on two gamer-grade GPUs. In contrast, today's Top-1 ImageNet model (CoAtNet) takes 20,000 TPU-v3 days. And things are even worse in the NLP world: training GPT-3 on a top-tier server with 8x A100 would take decades.

So, can individual researchers and small labs still train state-of-the-art models? Yes we can! All it takes is for a bunch of us to come together. In fact, we're doing it right now and **you are invited to join!**



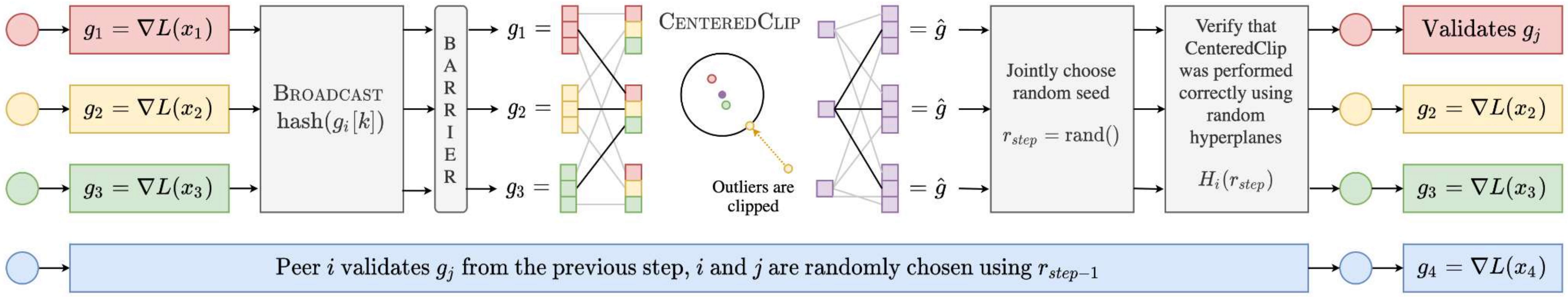
Secure Distributed Training at Scale (ICML'22)

- What if some peers are malicious/faulty?
- Can we train the model in such a way that no one peer can break it?
- Hint: use cryptography :)

?

Secure Distributed Training at Scale (ICML'22)

- What if some peers are malicious/faulty?
- Propose efficient byzantine-tolerant version of SGD
- Based on multi-party computing to remove redundant computations



from hivemind import *

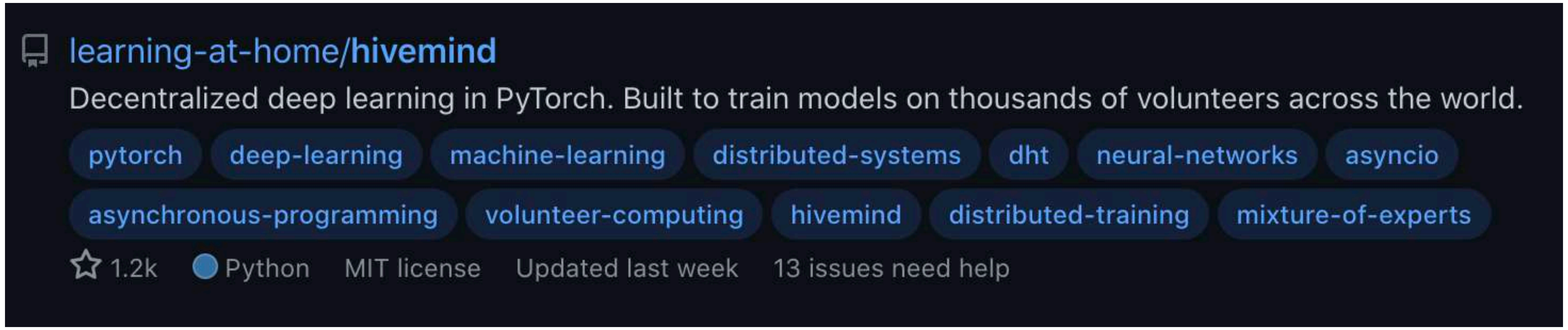
〉 We develop a library for decentralized deep learning over the Internet

〉 Supports bypassing NAT, asynchronous training, data compression

〉 Data-parallel parts are tested in several practical projects

〉 Easy to use in standard PyTorch (just change ~2 lines of code!)

〉 Integrated into PyTorch Lightning, used for Stable Diffusion finetuning


learning-at-home/**hivemind**

Decentralized deep learning in PyTorch. Built to train models on thousands of volunteers across the world.

pytorch deep-learning machine-learning distributed-systems dht neural-networks asyncio asynchronous-programming volunteer-computing hivemind distributed-training mixture-of-experts

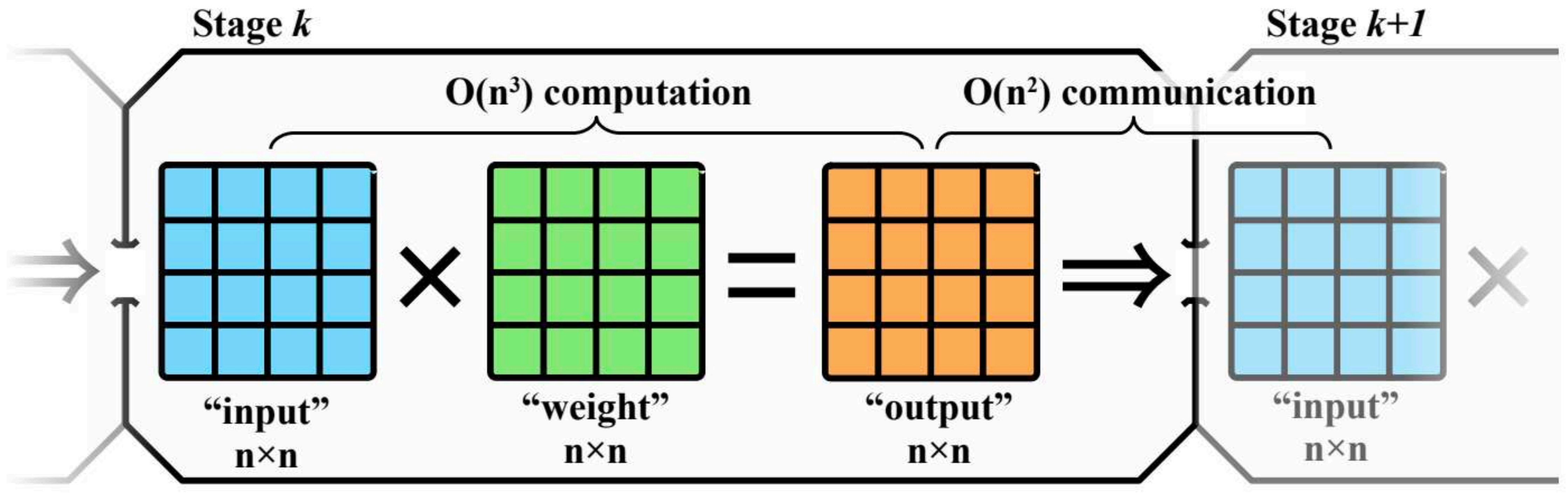
☆ 1.2k ● Python MIT license Updated last week 13 issues need help

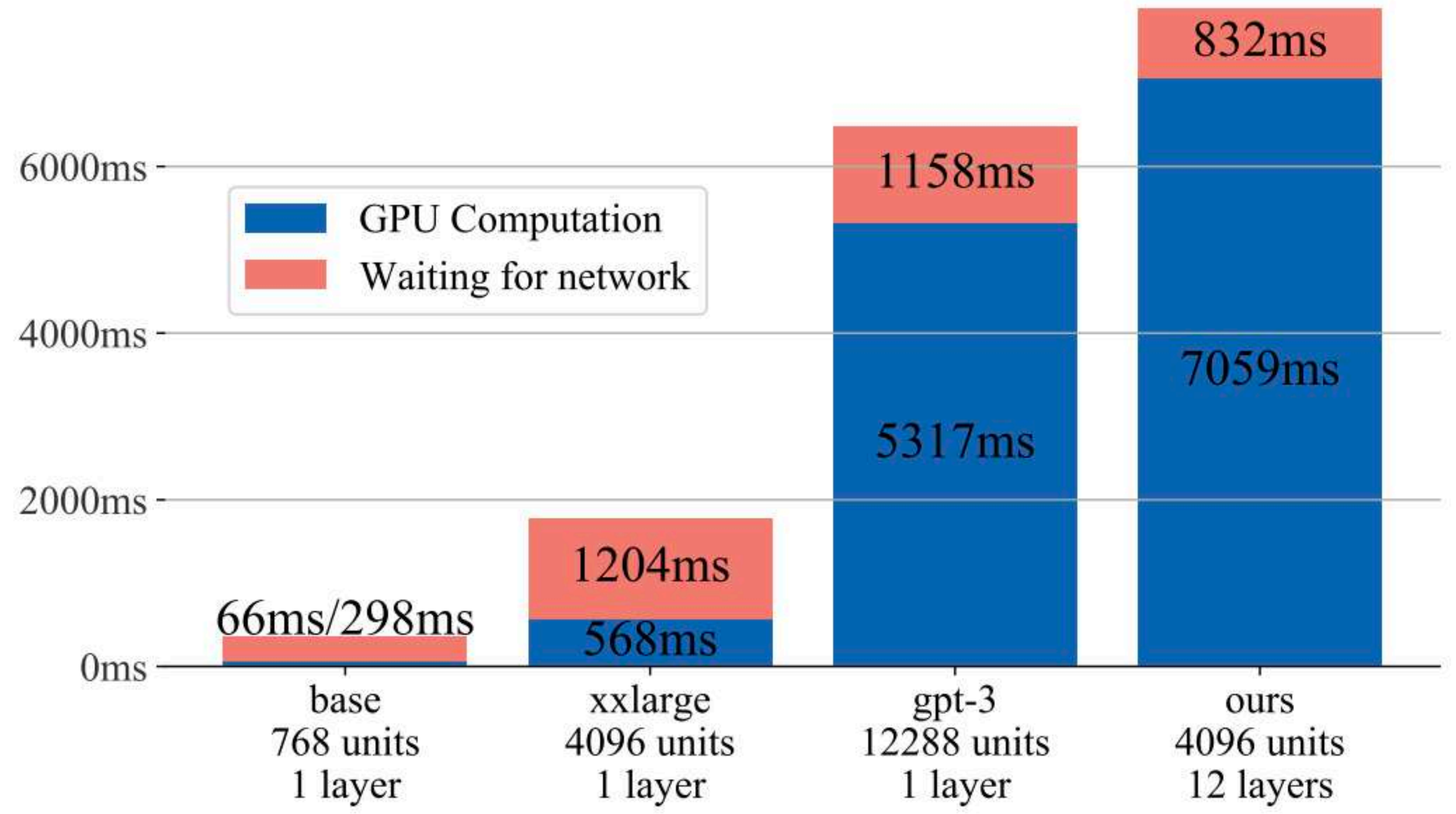
SWARM Parallelism: Training Large Models Can Be Surprisingly Communication-Efficient

〉 How can we train large models over the Internet?

〉 Key observation: with the growth in the hidden dimension size, compute costs grow faster than communication costs!

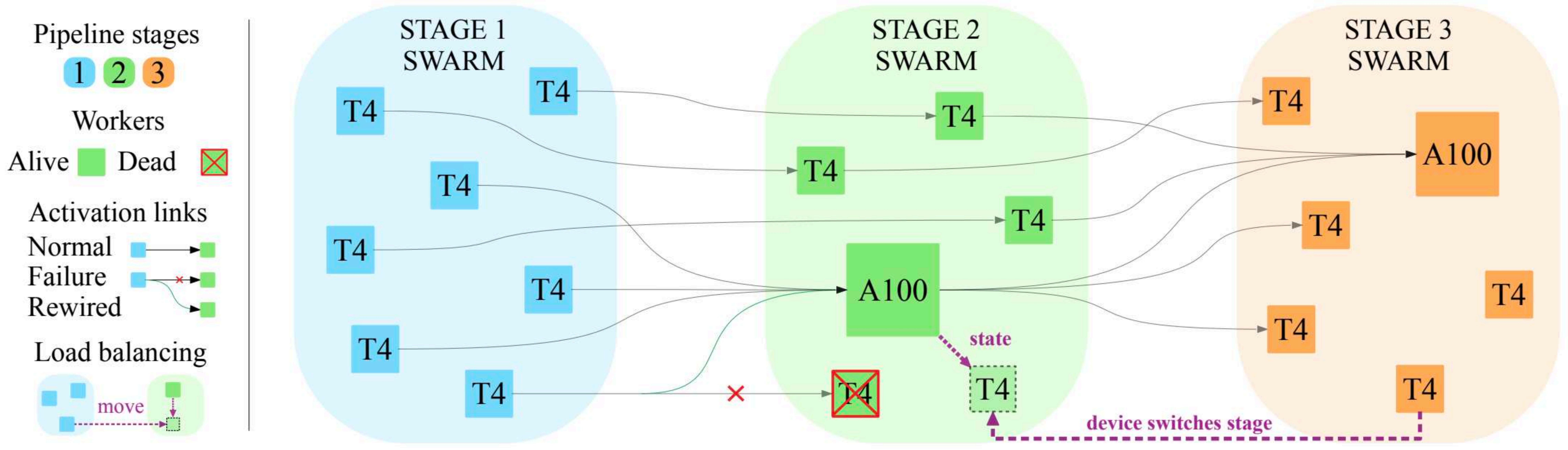
〉 This observation can make training large models feasible for speeds <500Mb/s (especially if we compress activations)





SWARM Parallelism: Training Large Models Can Be Surprisingly Communication-Efficient

We can use this fact for communication efficiency
and create dynamically rebalanced pipelines for fault tolerance!



How to make it efficient?

- 〉 Server-side load balancing
  - If some servers disconnect, other servers close the gap
- 〉 Client-side routing
  - Clients choose servers with maximal throughput

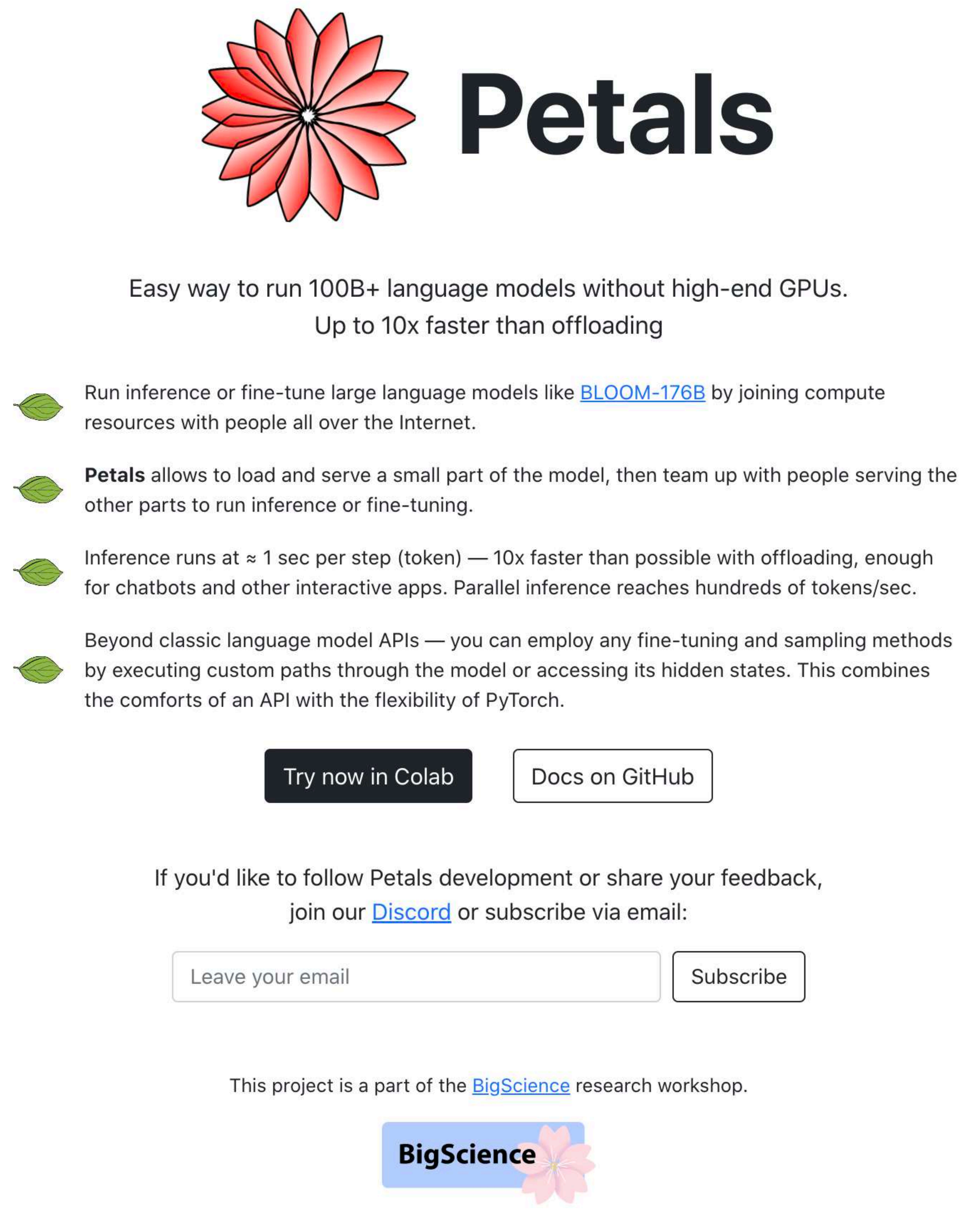
Talk outline

- Motivation and key challenges
- Decentralized training
    - Specialized architectures
    - General data-parallel training
    - Model-parallel training
- Decentralized inference of pretrained models

Petals: Collaborative Inference and Fine-tuning of Large Models

(NeurIPS'22 "Broadening Research Collaborations" workshop, Best Paper Honorable Mention)

- We develop a system for running and fine-tuning LLMs over volunteer devices
- Instead of just getting model predictions, you can inspect its hidden states
- Possible to join the public swarm (serving BLOOM at the moment) or start your own



Petals

Easy way to run 100B+ language models without high-end GPUs.
Up to 10x faster than offloading

- Run inference or fine-tune large language models like BLOOM-176B by joining compute resources with people all over the Internet.
- **Petals** allows to load and serve a small part of the model, then team up with people serving the other parts to run inference or fine-tuning.
- Inference runs at ≈ 1 sec per step (token) — 10x faster than possible with offloading, enough for chatbots and other interactive apps. Parallel inference reaches hundreds of tokens/sec.
- Beyond classic language model APIs — you can employ any fine-tuning and sampling methods by executing custom paths through the model or accessing its hidden states. This combines the comforts of an API with the flexibility of PyTorch.

Try now in Colab | Docs on GitHub

If you'd like to follow Petals development or share your feedback, join our Discord or subscribe via email:

Leave your email | Subscribe

This project is a part of the BigScience research workshop.

BigScience

petals.ml

Multiple 100B+ language models were openly released

a BigScience initiative

BLOOM

176B params · 59 languages · Open-access

Meta AI is sharing OPT-175B, the first 175-billion-parameter language model to be made available to the broader AI research community.

GLM-130B

An Open Bilingual Pre-Trained Model

yandex/**YaLM-100B**

Pretrained language model with 100B parameters

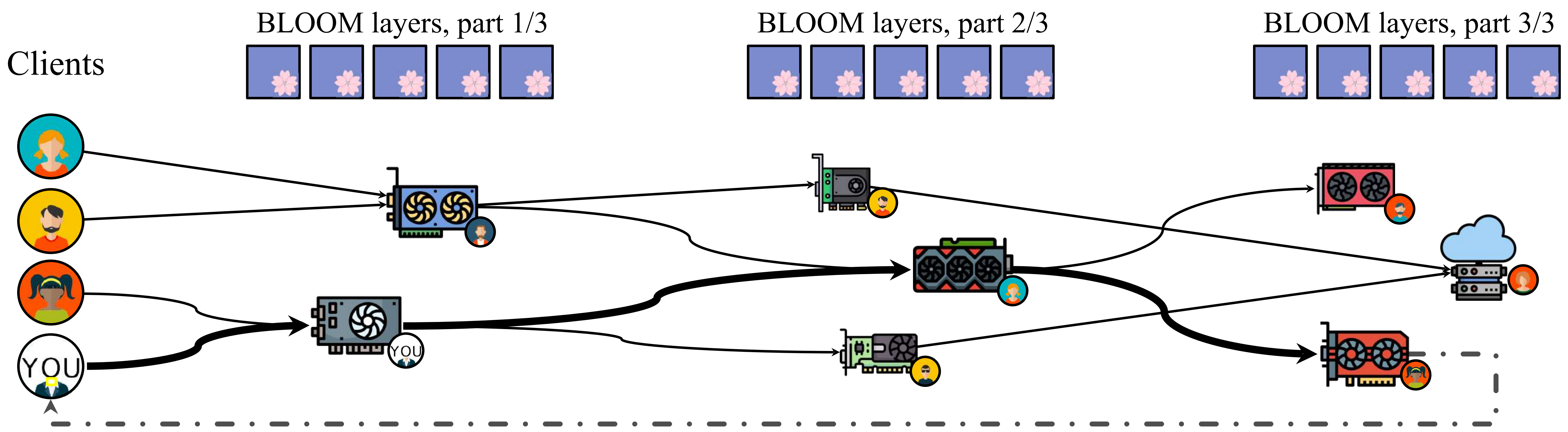
Hard to use without multiple high-end accelerators!

Existing solutions have limitations

- **Option 1.** Offloading to RAM/SSD
  - Inference is too slow for interactive apps
  - 5.5 seconds/token in the fastest RAM offloading setup (needs 100+ GB RAM)
  - 22 seconds/token in the fastest SSD offloading setup
- **Option 2.** Hosted APIs
  - No way to use custom fine-tuning and sampling methods
  - No way to look at the block outputs and token probabilities
  - Might be expensive

Our approach

- Some participants (called **servers**) load BLOOM blocks to their GPUs and allow others to do forward and backward passes
- Other participants (called **clients**) perform forward/backward passes through the whole model by sending requests to servers



Client-side finetuning

- **Parameter-efficient adapters** (e.g., LoRA) or **trainable prompts** (e.g., P-tuning v2) are sufficient for most real-world tasks
  - They do not need much memory or compute!

- Clients can store adapters/prompts locally, then train them with any optimizer assuming that the remote blocks are just non-trainable PyTorch modules
  - They can use any adapter architecture and any optimizer they want

"LoRA: Low-rank adaptation of large language models." Hu et al., 2021.

"P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks." Liu et al., 2021.

How to make it efficient?

〉 Compressing communication buffers

- **Blockwise quantization (Dettmers, 2022a)**

〉 Compressing model weights

- **8-bit mixed decomposition (Dettmers, 2022b)**

  99.9% 8-bit + 0.1% 16-bit

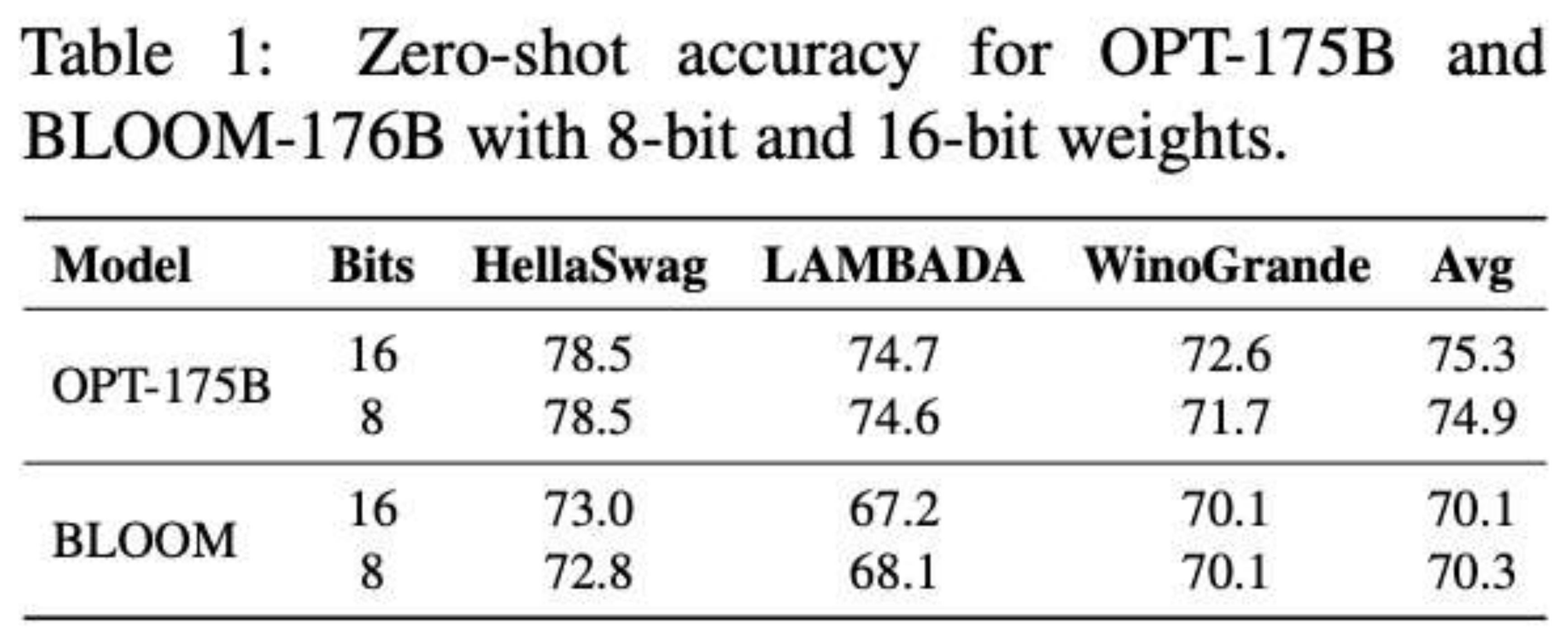


Table 1: Zero-shot accuracy for OPT-175B and BLOOM-176B with 8-bit and 16-bit weights.

Model	Bits	HellaSwag	LAMBADA	WinoGrande	Avg
OPT-175B	16	78.5	74.7	72.6	75.3
	8	78.5	74.6	71.7	74.9
BLOOM	16	73.0	67.2	70.1	70.1
	8	72.8	68.1	70.1	70.3

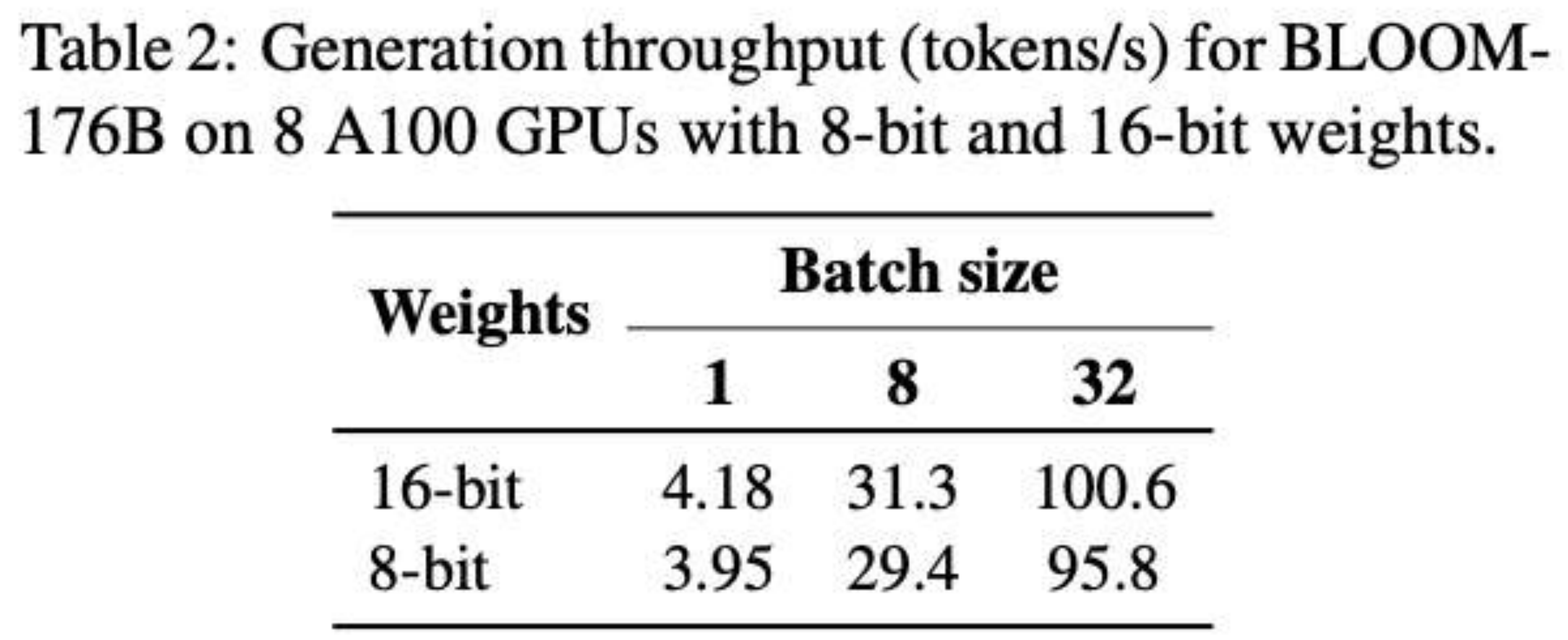


Table 2: Generation throughput (tokens/s) for BLOOM-176B on 8 A100 GPUs with 8-bit and 16-bit weights.

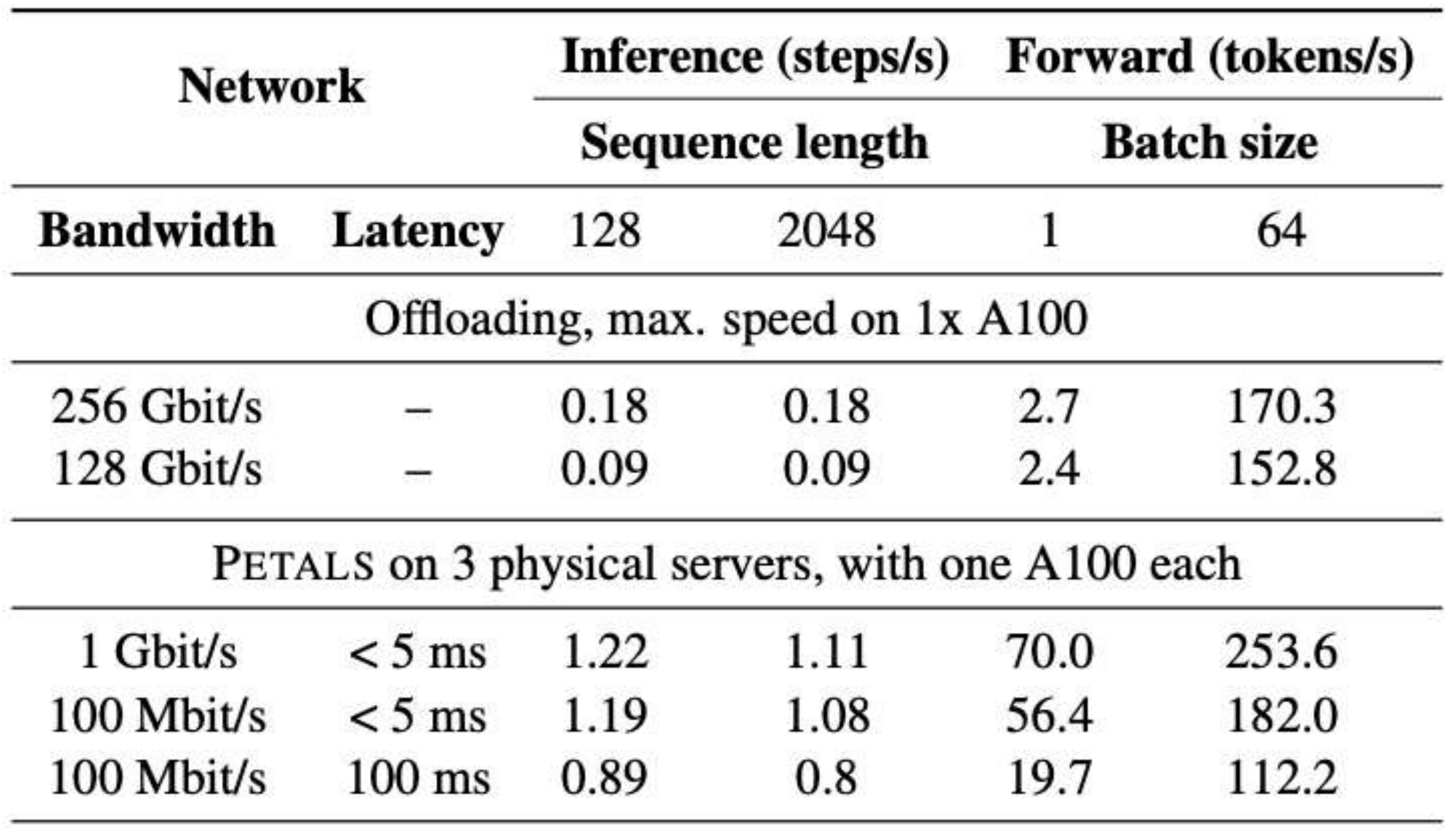
Weights	Batch size		
	1	8	32
16-bit	4.18	31.3	100.6
8-bit	3.95	29.4	95.8

8-bit Optimizers via Block-wise Quantization. Dettmers et al., ICLR 2022.

LLM. int8 (): 8-bit Matrix Multiplication for Transformers at Scale. Dettmers et al., NeurIPS 2022.

Petals is 5-10x faster than offloading

Table 3: Performance of sequential inference steps and training-time forward passes.



Network		Inference (steps/s)		Forward (tokens/s)	
		Sequence length		Batch size	
Bandwidth	**Latency**	128	2048	1	64
Offloading, max. speed on 1x A100					
256 Gbit/s	–	0.18	0.18	2.7	170.3
128 Gbit/s	–	0.09	0.09	2.4	152.8
PETALS on 3 physical servers, with one A100 each					
1 Gbit/s	< 5 ms	1.22	1.11	70.0	253.6
100 Mbit/s	< 5 ms	1.19	1.08	56.4	182.0
100 Mbit/s	100 ms	0.89	0.8	19.7	112.2

Petals in practice

```
[ ]  import torch
     from transformers import BloomTokenizerFast
     from petals.client import DistributedBloomForCausalLM

     MODEL_NAME = "bigscience/bloom-petals"
     tokenizer = BloomTokenizerFast.from_pretrained(MODEL_NAME)
     model = DistributedBloomForCausalLM.from_pretrained(MODEL_NAME)
     model = model.cuda()
```

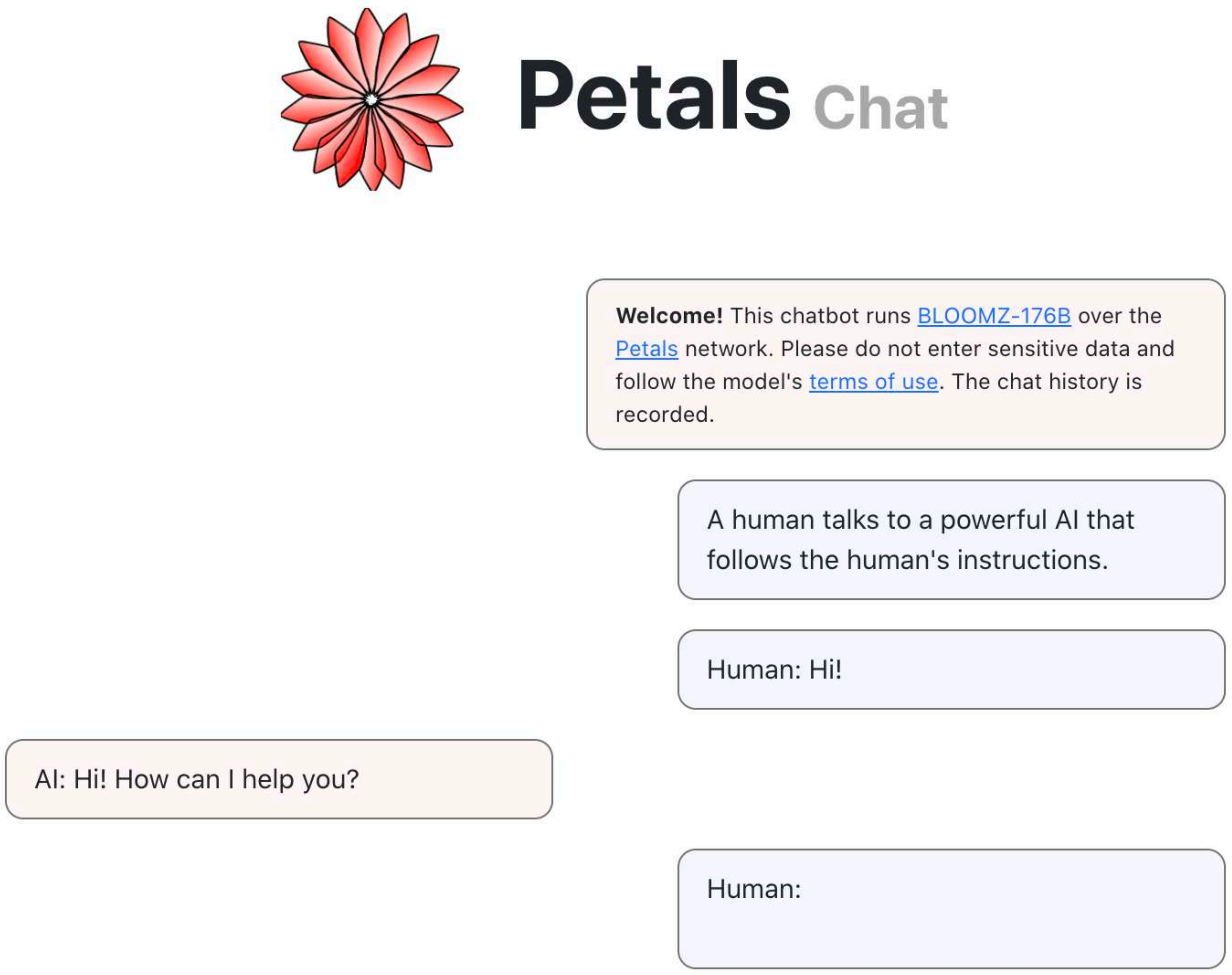
```
[ ]  inputs = tokenizer("A cat in French is \"", return_tensors="pt")["input_ids"].cuda()
     remote_outputs = model.generate(inputs, max_new_tokens=3)
     print(tokenizer.decode(remote_outputs[0]))
```

```
class BloomBasedClassifier(nn.Module):
  def __init__(self, model):
    super().__init__()
    self.distributed_layers = model.transformer.h
    self.adapter = nn.Sequential(nn.Linear(14336, 32), nn.Linear(32, 14336))
    self.head = nn.Sequential(nn.LayerNorm(14336), nn.Linear(14336, 2))

  def forward(self, embeddings):
    hidden_states = self.distributed_layers[0:6](embeddings)
    hidden_states = self.adapter(hidden_states)
    hidden_states = self.distributed_layers[6:10](hidden_states)
    pooled_states = torch.mean(hidden_states, dim=1)
    return self.head(pooled_states)
```

```
classifier = BloomBasedClassifier(model).cuda()
opt = torch.optim.Adam(classifier.parameters(), 3e-5)
inputs = torch.randn(3, 2, 14336, device='cuda')
labels = torch.tensor([1, 0, 1], device='cuda')

for i in range(5):
  loss = F.cross_entropy(classifier(inputs), labels)
  print(f"loss[{i}] = {loss.item():.3f}")
  opt.zero_grad()
  loss.backward()
  opt.step()
```

You can run both generation and classification just like you would with a local PyTorch model!

tinyurl.com/petals-colab


Petals Chat

Welcome! This chatbot runs BLOOMZ-176B over the Petals network. Please do not enter sensitive data and follow the model's terms of use. The chat history is recorded.

A human talks to a powerful AI that follows the human's instructions.

Human: Hi!

AI: Hi! How can I help you?

Human:

chat.petals.ml

Conclusion

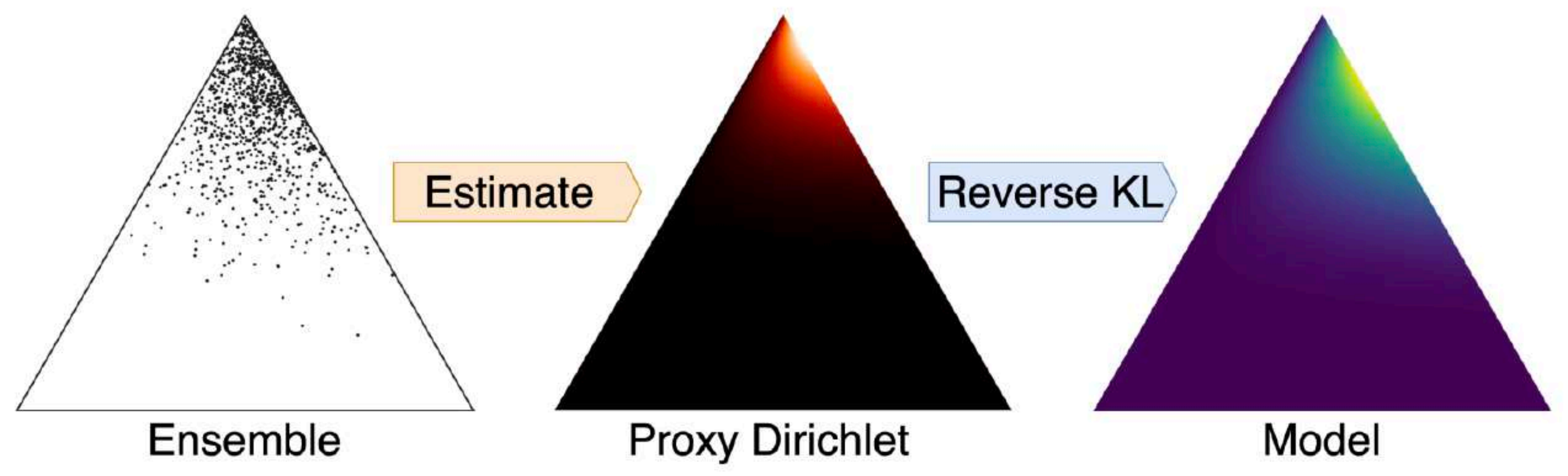
〉 Decentralized DL is a viable alternative to clusters

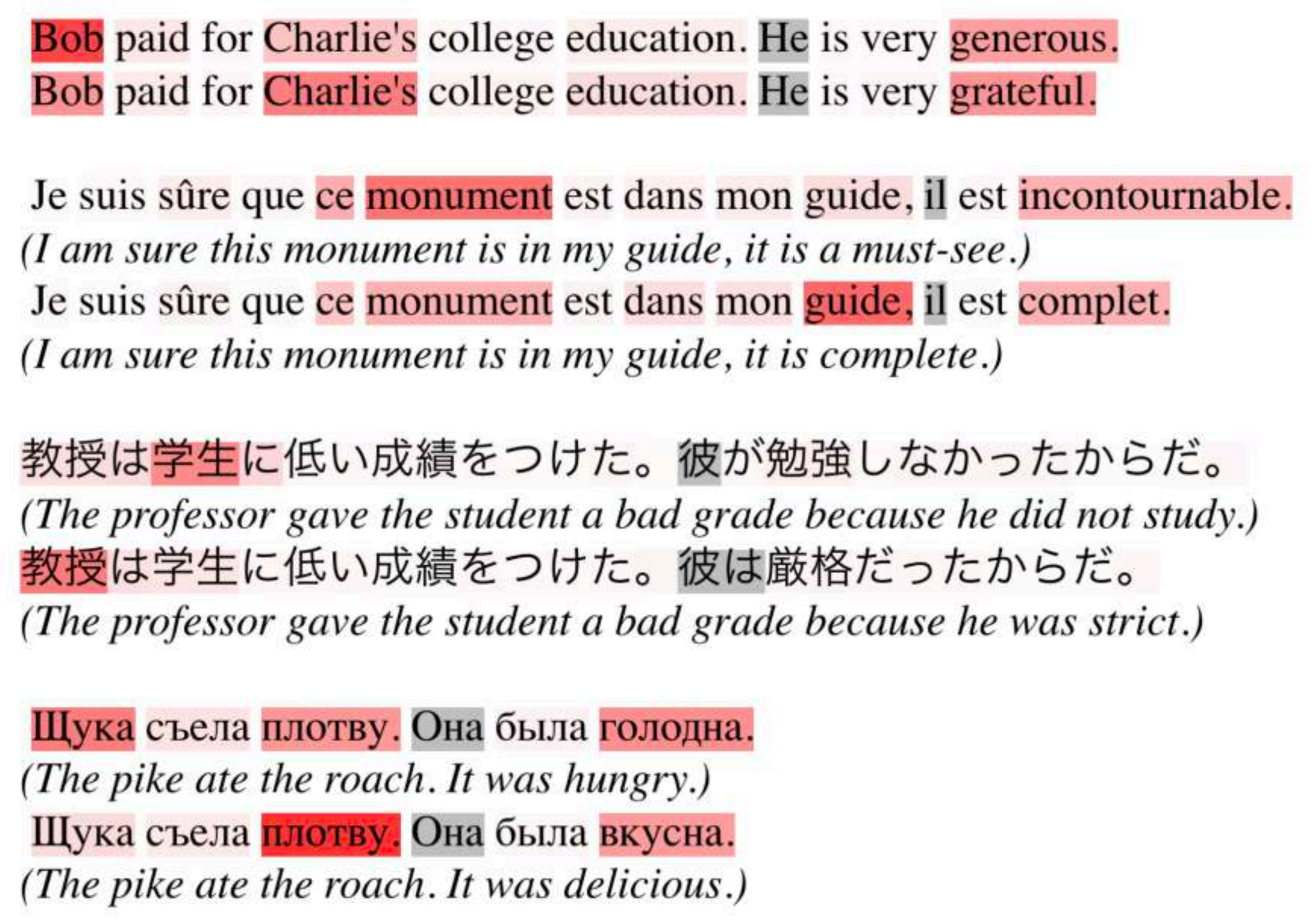
- Open-source libraries allow easy adaptation from HPC setups
- Also useful for preemptible/spot instances

〉 Ongoing challenges: data security, volunteer incentives, scheduling

〉 Curious to hear your thoughts on this line of research!

Other projects




Bob paid for Charlie's college education. He is very generous.
Bob paid for Charlie's college education. He is very grateful.

Je suis sûre que ce monument est dans mon guide, il est incontournable.
(I am sure this monument is in my guide, it is a must-see.)
Je suis sûre que ce monument est dans mon guide, il est complet.
(I am sure this monument is in my guide, it is complete.)

教授は学生に低い成績をつけた。彼が勉強しなかったからだ。
(The professor gave the student a bad grade because he did not study.)
教授は学生に低い成績をつけた。彼は厳格だったからだ。
(The professor gave the student a bad grade because he was strict.)

Щука съела плотву. Она была голодна.
(The pike ate the roach. It was hungry.)
Щука съела плотву. Она была вкусна.
(The pike ate the roach. It was delicious.)

Scaling Ensemble Distribution Distillation to Many Classes with Proxy Targets (with Andrey Malinin, Mark Gales)

It's All in the Heads: Using Attention Heads as a Baseline for Cross-Lingual Transfer in Commonsense Reasoning (with Alexey Tikhonov)

BLOOM: A 176B-Parameter Open-Access Multilingual Language Model (as the Engineering and Scaling working group chair at BigScience)

Thank you!

Max Ryabinin

Senior Research Scientist, Yandex Research

PhD Student, HSE University

✉ mryabinin0@gmail.com

🔗 mryab.github.io