Open Datasets and Decentralized Compute for LLMs

Tri Dao
Together AI / Princeton University
https://tridao.me
Machine Learning Has Made Exciting Progress

Fix Bugs

(ChatGPT/GPT4 - OpenAI)

```javascript
for (let i = 0; i < 5; i++) {
    setTimeout(() => console.log(i), 1000)
}
```

The code contains a bug that will cause it to always print the number 5 to the console, rather than the expected sequence of numbers from 0 to 4. This is because the `setTimeout` function creates a closure that captures the value of the `i` variable, but the `i` variable is changed before the callback function is executed.

To fix this bug, the code can be modified to create a new variable in the loop that is passed to the `setTimeout` callback function, rather than using the loop variable directly. This will ensure that each callback function captures the correct value of the loop variable, and the expected output will be produced. Here is an example of how the code could be fixed:

```javascript
for (let i = 0; i < 5; i++) {
    const value = i;
    setTimeout(() => console.log(value), 1000)
}
```
Machine Learning Has Made Exciting Progress

Fix Bugs
(ChatGPT/GPT4 - OpenAI)

Find the bug with this code:

```
...  
for (var i = 0; i < 5; i++) {
    setTimeout(() => console.log(i), 1000)
}  
...  
```

The code contains a bug that will cause it to always print the number 0 to the console, rather than the expected sequence of numbers from 0 to 4. This is because the `setTimeout` function creates a closure that captures the value of the `i` variable, but the `i` variable is changed before the callback function is executed.

To fix this bug, the code can be modified to create a new variable in the loop that is passed to the `setTimeout` callback function, rather than using the loop variable directly. This will ensure that each callback function captures the correct value of the loop variable, and the expected output will be produced. Here is an example of how the code could be fixed:

```
for (var i = 0; i < 5; i++) {
    // Create a new variable and pass its value to setTimeout
    const value = i;
    setTimeout(() => console.log(value), 1000)
}  
```

Generate Art
(Stable Diffusion – Stability.AI)
Machine Learning Has Made Exciting Progress

Fix Bugs
(ChatGPT/GPT4 - OpenAI)

```javascript
for (var i = 0; i < 5; i++) {
    setTimeout(() => console.log(i), 1000);
}
```

The code contains a bug that will cause it to always print the number 5 to the console, rather than the expected sequence of numbers from 0 to 4. This is because the `setTimeout` function creates a closure that captures the value of the `i` variable, but the `i` variable is changed before the callback function is executed.

To fix this bug, the code can be modified to create a new variable in the loop that is passed to the `setTimeout` callback function, rather than using the loop variable directly. This will ensure that each callback function captures the correct value of the loop variable, and the expected output will be produced. Here is an example of how the code could be fixed:

```javascript
for (var i = 0; i < 5; i++) {
    // Create a new variable and pass its value to setTimeout
    const value = i;
    setTimeout(() => console.log(value), 1000);
}
```

Generate Art
(Stable Diffusion – Stability.AI)

Design Drugs
(AlphaFold – DeepMind)
Machine Learning Has Made Exciting Progress

Fix Bugs
(ChatGPT/GPT4 - OpenAI)

Generate Art
(Stable Diffusion – Stability.AI)

Design Drugs
(AlphaFold – DeepMind)

What enabled these advances? What are outstanding problems? How do we approach them?
Scale in Data and Compute Brings Quality and Capabilities

Model Size (in billions of parameters)

- ELMo (94M)
- BERT-Large (340M)
- GPT-2 (1.3B)
- T5 (11B)
- Turing-NLG (17.2B)
- Megatron-LM (8.3B)
- GPT-3 (175B)
- Megatron-Turing NLG (530B)

- 100 million
- 2018
- 500 billion
- 2022
Scale in Data and Compute Brings Quality and Capabilities

Language models explaining jokes

Input: I tried 10000 random restarts of my neural network, but I was accused of overfitting. I guess no good seed goes unpunished.
Input: I tried 10000 random restarts of my neural network, but I was accused of overfitting. I guess no good seed goes unpunished.

1.3B model: The joke is that if you try 10000 different seed choices, you'll eventually find one that works, but you'll be accused of overfitting.
I tried 10000 random restarts of my neural network, but I was accused of overfitting. I guess no good seed goes unpunished.

1.3B model: The joke is that if you try 10000 different seed choices, you'll eventually find one that works, but you'll be accused of overfitting.

175B model: This joke is a play on words related to neural networks, a type of machine learning algorithm. The punchline, "I guess no good seed goes unpunished," is a play on the phrase "no good deed goes unpunished." In this case, "good seed" refers to a starting point for the random restarts, and the joke implies that even when trying to improve the neural network's performance, the person is still accused of overfitting.
Scale in Data and Compute Brings Quality and Capabilities

Language models explaining jokes

Input: I tried 10000 random restarts of my neural network, but I was accused of overfitting. I guess no good seed goes unpunished.

1.3B model: The joke is that if you try 10000 different seed choices, you'll eventually find one that works, but you'll be accused of overfitting.

175B model: This joke is a play on words related to neural networks, a type of machine learning algorithm. The punchline, "I guess no good seed goes unpunished," is a play on the phrase "no good deed goes unpunished." In this case, "good seed" refers to a starting point for the random restarts, and the joke implies that even when trying to improve the neural network's performance, the person is still accused of overfitting.

Scale is more closely tied to advances in ML than ever before
Challenges with Scale

Volume↑ Complexity↑ Quality↓ Cleaning & Acq. Cost↑ Requirement: FLOPS, GB Specialization + Scale out
The Llama Moment

**Volume↑ Complexity↑ Quality↓ Cleaning & Acq. Cost↑**

**↑ Requirement: FLOPS, GB ↑ Specialization + ↑ Scale out**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Sampling prop.</th>
<th>Epochs</th>
<th>Disk size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommonCrawl</td>
<td>67.0%</td>
<td>1.10</td>
<td>3.3 TB</td>
</tr>
<tr>
<td>C4</td>
<td>15.0%</td>
<td>1.06</td>
<td>785 GB</td>
</tr>
<tr>
<td>Github</td>
<td>4.5%</td>
<td>0.64</td>
<td>328 GB</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>4.5%</td>
<td>2.45</td>
<td>83 GB</td>
</tr>
<tr>
<td>Books</td>
<td>4.5%</td>
<td>2.23</td>
<td>85 GB</td>
</tr>
<tr>
<td>ArXiv</td>
<td>2.5%</td>
<td>1.06</td>
<td>92 GB</td>
</tr>
<tr>
<td>StackExchange</td>
<td>2.0%</td>
<td>1.03</td>
<td>78 GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU Type</th>
<th>GPU Power consumption</th>
<th>GPU-hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT-175B</td>
<td>400W</td>
<td>809,472</td>
</tr>
<tr>
<td>BLOOM-175B</td>
<td>400W</td>
<td>1,082,880</td>
</tr>
<tr>
<td>LLaMA-7B</td>
<td>400W</td>
<td>82,432</td>
</tr>
<tr>
<td>LLaMA-13B</td>
<td>400W</td>
<td>135,168</td>
</tr>
<tr>
<td>LLaMA-33B</td>
<td>400W</td>
<td>530,432</td>
</tr>
<tr>
<td>LLaMA-65B</td>
<td>400W</td>
<td>1,022,362</td>
</tr>
</tbody>
</table>
RedPajama v1: Data

- CommonCrawl
- C4
- GitHub
- arXiv
- Books
- Wikipedia
- StackExchange
RedPajama v1: Data

- CommonCrawl
- C4
- GitHub
- arXiv
- Books
- Wikipedia
- StackExchange

<table>
<thead>
<tr>
<th>Source</th>
<th>RedPajama</th>
<th>LLaMA*</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommonCrawl</td>
<td>878 billion</td>
<td>852 billion</td>
</tr>
<tr>
<td>C4</td>
<td>175 billion</td>
<td>190 billion</td>
</tr>
<tr>
<td>Github</td>
<td>59 billion</td>
<td>100 billion</td>
</tr>
<tr>
<td>Books</td>
<td>26 billion</td>
<td>25 billion</td>
</tr>
<tr>
<td>ArXiv</td>
<td>28 billion</td>
<td>33 billion</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>24 billion</td>
<td>25 billion</td>
</tr>
<tr>
<td>StackExchange</td>
<td>20 billion</td>
<td>27 billion</td>
</tr>
<tr>
<td>Total</td>
<td>1.2 trillion</td>
<td>1.25 trillion</td>
</tr>
</tbody>
</table>

LLaMA* is not included in the RedPajama dataset.
Fueling and Exciting Generation of Open Models

RedPajama-INCITE: 7/7 Slices
OpenLlama: 7/7 Slices
Mosaic MPT: 5/10 Slices
Salesforce XGen: 5/12 Slices
Compute: Hardware-aware Algorithms

IO-awareness:
reducing reads/writes to GPU memory yields significant speedup
Compute: Hardware-aware Algorithms

IO-awareness:
reducing reads/writes to GPU memory yields significant speedup

FlashAttention: fast (2-4x) and memory-efficient attention (10-20x) algorithm, with no approximation
Compute: Hardware-aware Algorithms

IO-awareness:
reducing reads/writes to GPU memory yields significant speedup

FlashAttention: fast (4-8x) and memory-efficient attention (10-20x)
algorithm, with no approximation
FlashAttention Adoption Areas

Find the bug with this code:

```javascript
... for (var i = 0; i < 5; i++) {
    setTimeout(i => console.log(i), 1000)
}
...
```

The code contains a bug that will cause it to always print the number 5 to the console, rather than the expected sequence of numbers from 0 to 4. This is because the `setTimeout` function creates a closure that captures the value of the `i` variable, but the `i` variable is changed before the callback function is executed.

To fix this bug, the code can be modified to create a new variable in the loop that is passed to the `setTimeout` callback function, rather than using the loop variable directly. This will ensure that each callback function captures the correct value of the loop variable, and the expected output will be produced. Here is an example of how the code could be fixed:

```javascript
for (var i = 0; i < 5; i++) {
    // Create a new variable and pass its value to setTimeout
    const value = i;
    setTimeout(() => console.log(value), 1000)
}
```

Text Generation

(Llama – Meta, Falcon – TIIUAE, MPT, RedPajama)

Image Generation

(Stable Diffusion - Stability.AI)

Drug Discovery

(OpenFold, UniFold)
Decentralized Communication & Data Movement

Distributed training at scale is communication-intensive.
Decentralized Communication & Data Movement

Distributed training at scale is communication-intensive.

![GPT-3](image)

6.7B Parameters

1.20E+22 Floating Point Ops.

32 Machines, 4x A100 GPU each
Each machine send+recv 4PB data
100Gbps = 93h Communication Time
10Gbps = 930h Communication Time
~200h Computation Time
Decentralized Communication & Data Movement

Distributed training at scale is communication-intensive.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Machines, GPU Configuration</th>
<th>Send+Recv Data Transfer</th>
<th>Communication Time</th>
<th>Computation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7B Parameters</td>
<td>32 Machines, 4x A100 GPU each</td>
<td>4PB data</td>
<td>93h</td>
<td>200h</td>
</tr>
<tr>
<td>1.20E+22 Floating Point Ops.</td>
<td></td>
<td></td>
<td>930h</td>
<td></td>
</tr>
<tr>
<td>175B Parameters</td>
<td>196 Machines, 8x A100 GPU each</td>
<td>12PB data</td>
<td>279h</td>
<td>400h</td>
</tr>
<tr>
<td>3.14E+23 Floating Point Ops.</td>
<td></td>
<td></td>
<td>2790h</td>
<td></td>
</tr>
</tbody>
</table>
Decentralized Communication & Data Movement

Distributed training at scale is communication-intensive.

(Today) Model training today is largely restricted to centralized data centers with fast network connections. Hard to use cheaper alternatives (Tier 2-4 clouds, Spot Instances, Volunteer Computes, etc.).
Decentralized Communication & Data Movement

Distributed training at scale is communication-intensive.

32 Machines, 4x A100 GPU each
Each machine send+recv 4PB data
100Gbps = \textcolor{red}{93h} Communication Time
10Gbps = \textcolor{red}{930h} Communication Time
\sim \textcolor{red}{200h} Computation Time

196 Machines, 8x A100 GPU each
Each machine send+recv 12PB data
100Gbps = \textcolor{red}{279h} Communication Time
10Gbps = \textcolor{red}{2790h} Communication Time
\sim \textcolor{red}{400h} Computation Time

Future: 10x further scaling requires fast connections between 10x machines. Becoming challenging even for data center.

(Today) Model training today is largely restricted to centralized data centers with fast network connections. Hard to use cheaper alternatives (Tier 2-4 clouds, Spot Instances, Volunteer Computes, etc.).
CocktailSGD: Mixture of Communication Compression Methods

Different communication compression techniques complement each other and compose well!
CocktailSGD: Mixture of Communication Compression Methods

Different communication compression techniques complement each other and compose well!

As long as Communication fully fills the Comm. Slot, no slow down caused by communication.
CocktailSGD: Mixture of Communication Compression Methods

Different communication compression techniques complement each other and compose well!

As long as Communication fully fills the Comm. Slot, no slow down caused by communication.
CocktailSGD: Mixture of Communication Compression Methods

As long as **Communication** fully fills the **Comm. Slot**, no slow down caused by communication.

Different communication compression techniques complement each other and compose well!

- LocalSGD (100x)
- Top-K (100x)
- Cocktail SGD (100x)
- AllReduce (fp16)

(b) GPT-J-6B
(c) GPT-NeoX-20B
CocktailSGD: Mixture of Communication Compression Methods

Different communication compression techniques complement each other and compose well!

As long as Communication fully fills the Comm. Slot, no slow down caused by communication.

Data parallel over 1Gbps network!
Ongoing & Future Work: Optimizing Throughout the Stack

1. Different kinds of **hardware**

2. Efficient **algorithms** and kernels for training and inference

3. Diverse **capabilities** (long context) and new **applications** (multi-modal, genomics)