SHEAA

Simulating Hardware Error Attacks on AI

Satrant Bains, Liam Baker, Alex Fink, Kieran Seven

Meet the Team

Satrant Bains
Junior
Computer Engineer
Developer

Liam Baker
Junior
Computer Engineer
Developer

Kieran Seven
Sophomore
Computer Engineer
Developer

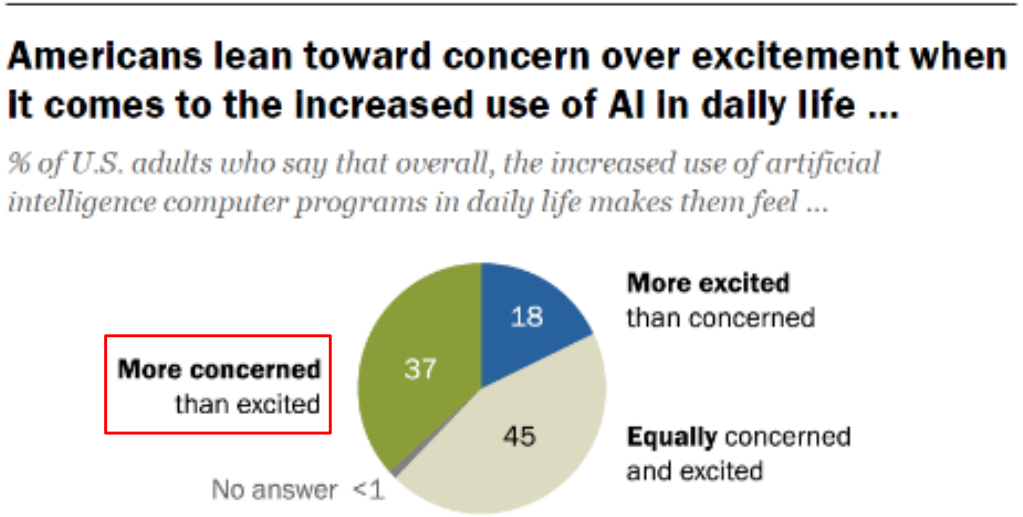
Alex Fink
Junior
Computer Engineer
Developer

Faculty Mentor: Dr. Xun Jiao

- Dr. Jiao is an assistant professor in the ECE department who specializes in **Machine Learning/AI**
- Research sponsored by the NIH, NHS, L3Harris and NVIDIA
- PH.D UC San Diego, 2018
- B.S Queen Mary, University of London, 2013
- B.S Beijing University of Posts and Telecommunication, 2013

Problem Statement

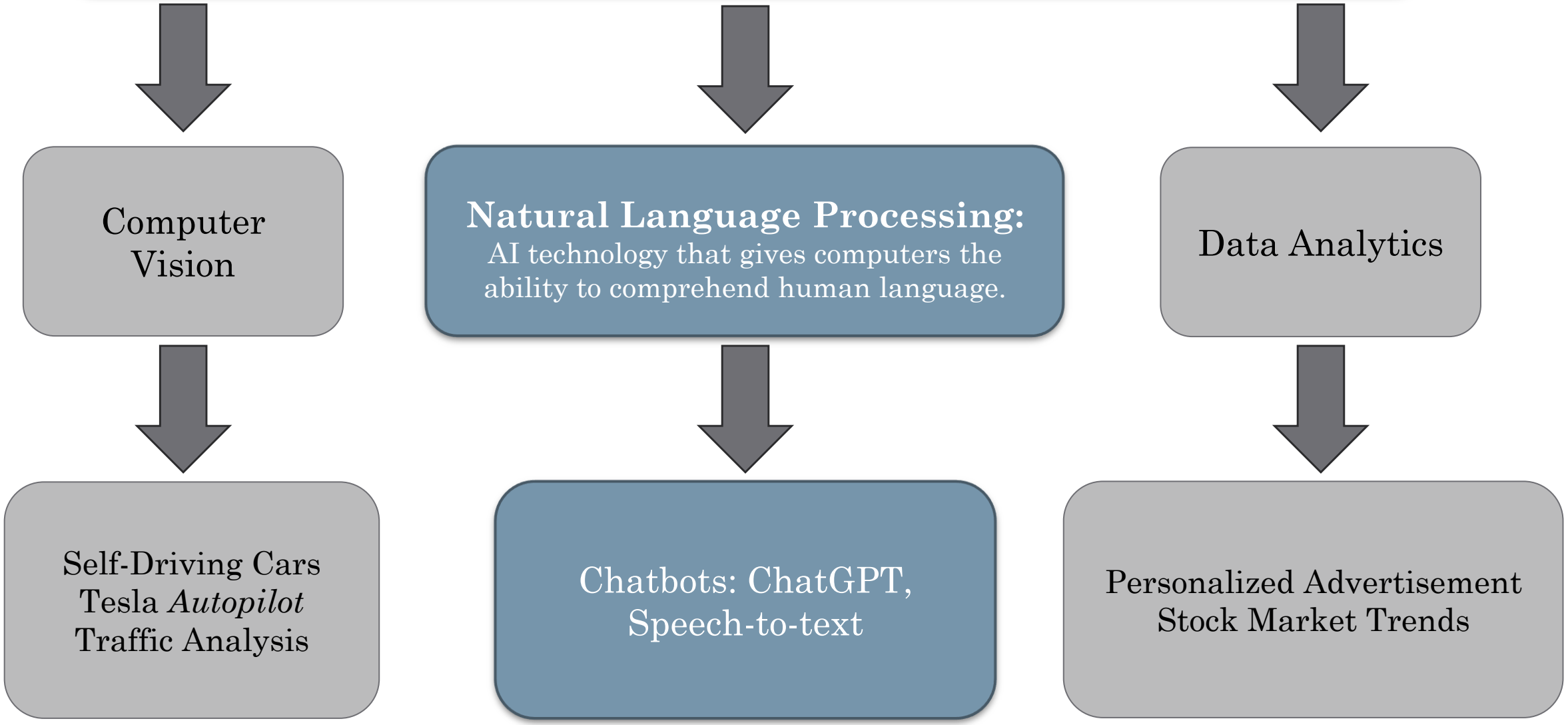
- Developers don't know the limits of AI.
- The public is concerned about increased AI usage.
- Sam Altman has said "we've got to be careful" and "I think [the general public] should be happy that we are a little bit scared of this."
- We need more toolkits to understand the robustness of AI.



What are the limits of emerging AI technology?

What do failures in AI technology entail for us?

Artificial Intelligence



Timeline

Natural language Processing is first considered for translating between languages

1940s


Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* † illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

1 Introduction

Recurrent neural networks, long short-term memory [12] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [29, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [31, 21, 13].

* Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

† Work performed while at Google Brain.

‡ Work performed while at Google Research.

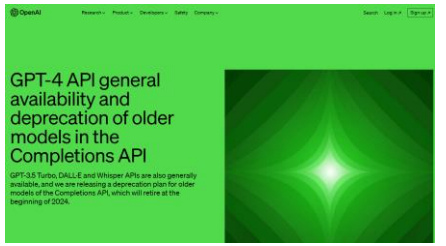
31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

June 12, 2017

Paper describing the **GPT architecture** is published in "Attention is All You Need"

ChatGPT is released

November 30, 2022



July 10, 2023

OpenAI's **GPT-4's API** is released

Meta's **LLaMa v2** is open sourced

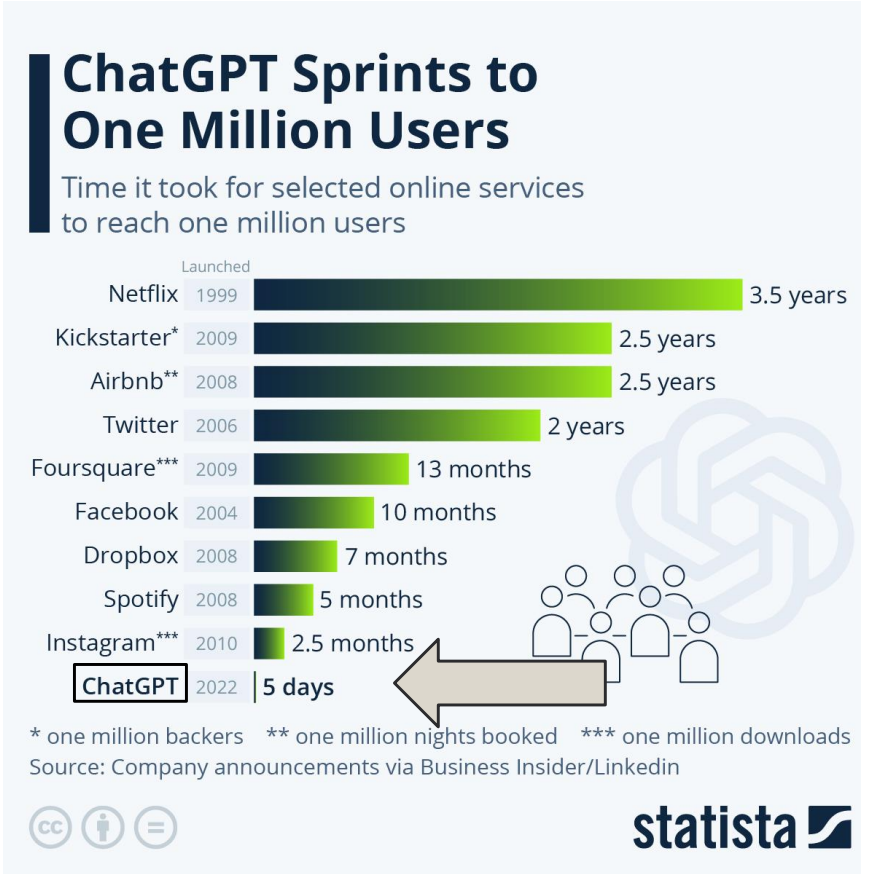
July 18, 2023

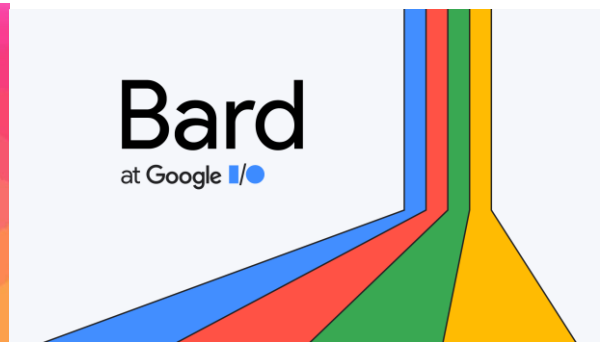
Market

Microsoft invested $**1,000,000,000** into OpenAI in 2019

Google invested **$300,000,000** into Anthropic a competitor of OpenAI in 2022

Global total corporate AI investment from 2022 is **91.9 billion**





KS

Can you make me a lullaby for my baby? He loves dinosoars

Of course! Here's a gentle lullaby for your little dinosaur-loving baby:

(Verse 1)
In a land of dreams, where dinos roam,
Close your eyes, my child, drift away to their home.
With T-Rex and Stegosaurus, they'll play,
In a prehistoric world, they'll dance and sway.

(Chorus)
Hush, little one, don't you cry,
Dinosaur friends are passing by.
In the moonlit sky, they'll softly soar,
Guarding your dreams, forevermore.

ChatGPT

Text Generation Example

Factual Example

KS

When was Villanova established?

Villanova University was established on June 6, 1842. It is a private Catholic university located in Villanova, Pennsylvania, United States. The university was founded by the Order of Saint Augustine and is named after Saint Thomas of Villanova, a prominent 16th-century Augustinian friar. Over the years, Villanova has grown into a well-respected institution with a strong academic and athletic tradition.

Code Example

python Copy code

```python
def calculate_hypotenuse(a, b):
    hypotenuse_squared = a**2 + b**2
    hypotenuse = hypotenuse_squared ** 0.5
    return hypotenuse

def main():
    print("Welcome to the Right Triangle Hypotenuse Calculator!")
    print("Please enter the lengths of the two legs of the right triangle:")

    while True:
        try:
            leg_a = float(input("Length of leg a: "))
            leg_b = float(input("Length of leg b: "))
            hypotenuse = calculate_hypotenuse(leg_a, leg_b)
            print(f"The length of the hypotenuse is: {hypotenuse:.2f}")
            break
        except ValueError:
            print("Invalid input. Please enter valid numerical values for th

if __name__ == "__main__":
    main()
```

KS

Can you write me a Python program to calculate the Hypotenuse of a right triangle?

Sure! To calculate the hypotenuse of a right triangle, you can use the Pythagorean theorem. The formula is:

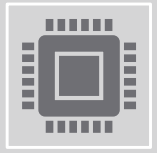
$$c^2 = a^2 + b^2$$

Where:

- c is the length of the hypotenuse
- a and b are the lengths of the other two sides (legs) of the right triangle

Here's a simple Python program that takes the lengths of the two legs as input from the user and calculates the hypotenuse:

Our Solution



Our goal is to design a model that can **simulate hardware attacks** through software

Analyze the source code for pre-existing NLP models and create a tool that can **cause bit flip errors**.

Create **website** to show the difference between NLP models with and without hardware errors.

Innovation

Our tool is geared towards helping developers understand the limits and robustness of language models

Companies focus on optimizing their models.

- Getting their model to be as advanced and safe as fast as possible

We are researching the consequences of hardware vulnerabilities in NLP.

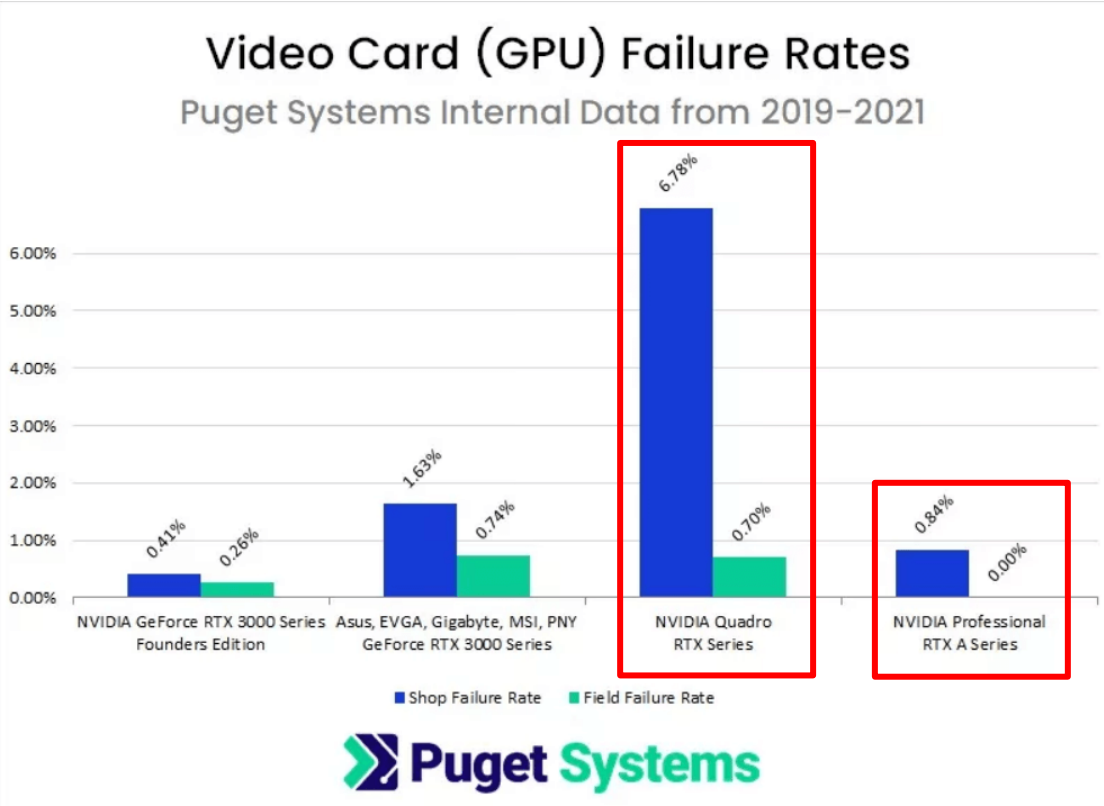
Relevance

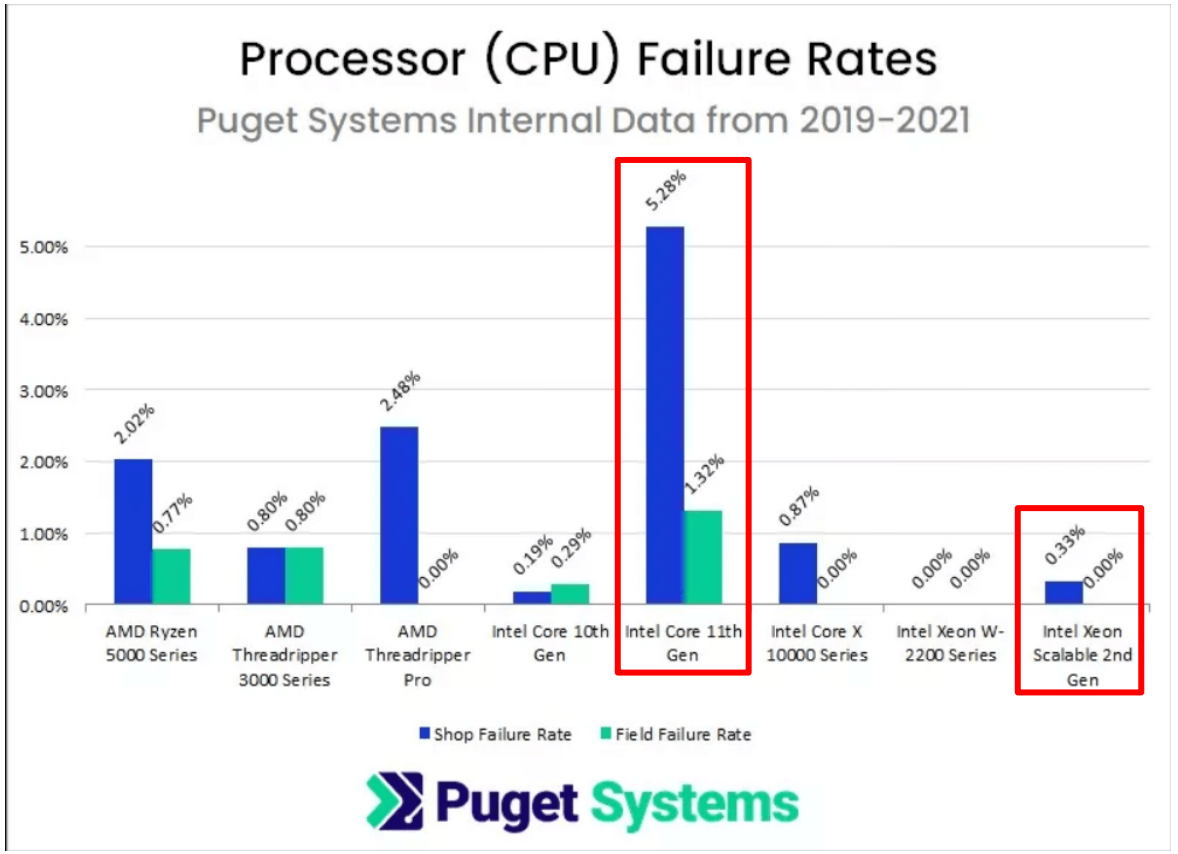
- Our simulated errors focus on bit flips which are flips in a transistor caused by manufacturing errors or random transient events.
- During a **Belgian Government Election** in 2003, a cosmic ray from space hit the hardware on the electronic voting system
- The number of votes for one candidate to increased by **4096**
  - Due to a bit flip from 0 to 1.

With AI models becoming more important in society's daily life, understanding the impact that these hardware errors can have on these models is necessary to **improve upon existing AI technology**.

Binary	1	1	0	1	0	0	1	0	Total
Decimal	$1 * 2^7$	$1 * 2^6$	$0 * 2^5$	$1 * 2^4$	$0 * 2^3$	$0 * 2^2$	$1 * 2^1$	$0 * 2^0$	210

Binary	1	0	0	1	0	0	1	0	Total
Decimal	$1 * 2^7$	$0 * 2^6$	$0 * 2^5$	$1 * 2^4$	$0 * 2^3$	$0 * 2^2$	$1 * 2^1$	$0 * 2^0$	146





Failure Rates of Modern Hardware

CPU – Main processing component that is present in all computers

GPU – Graphics Processing component that is only present in higher performance computers

100,000 GPUs with 2% failure → 2,000 failures

Background

Nodes – individual components of neural network that act as neurons to process information

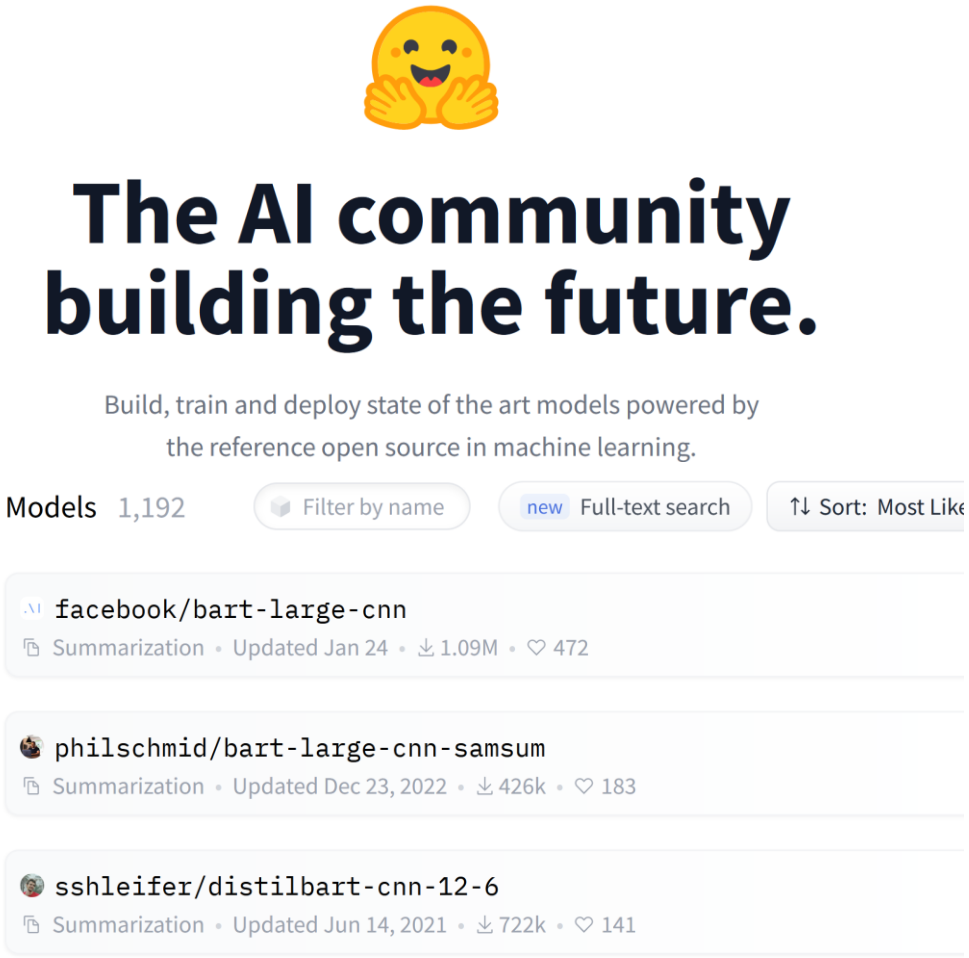
Tensor – array of numbers, one dimensional or multidimensional

Parameters – internal variables that are learned during the training process and determine the performance of the network

Error Injection Constraints

- Quick Responses: 10-15 seconds
- Variety of Responses
- Avoid Over Error-Injection
- Incorporate Bitflips

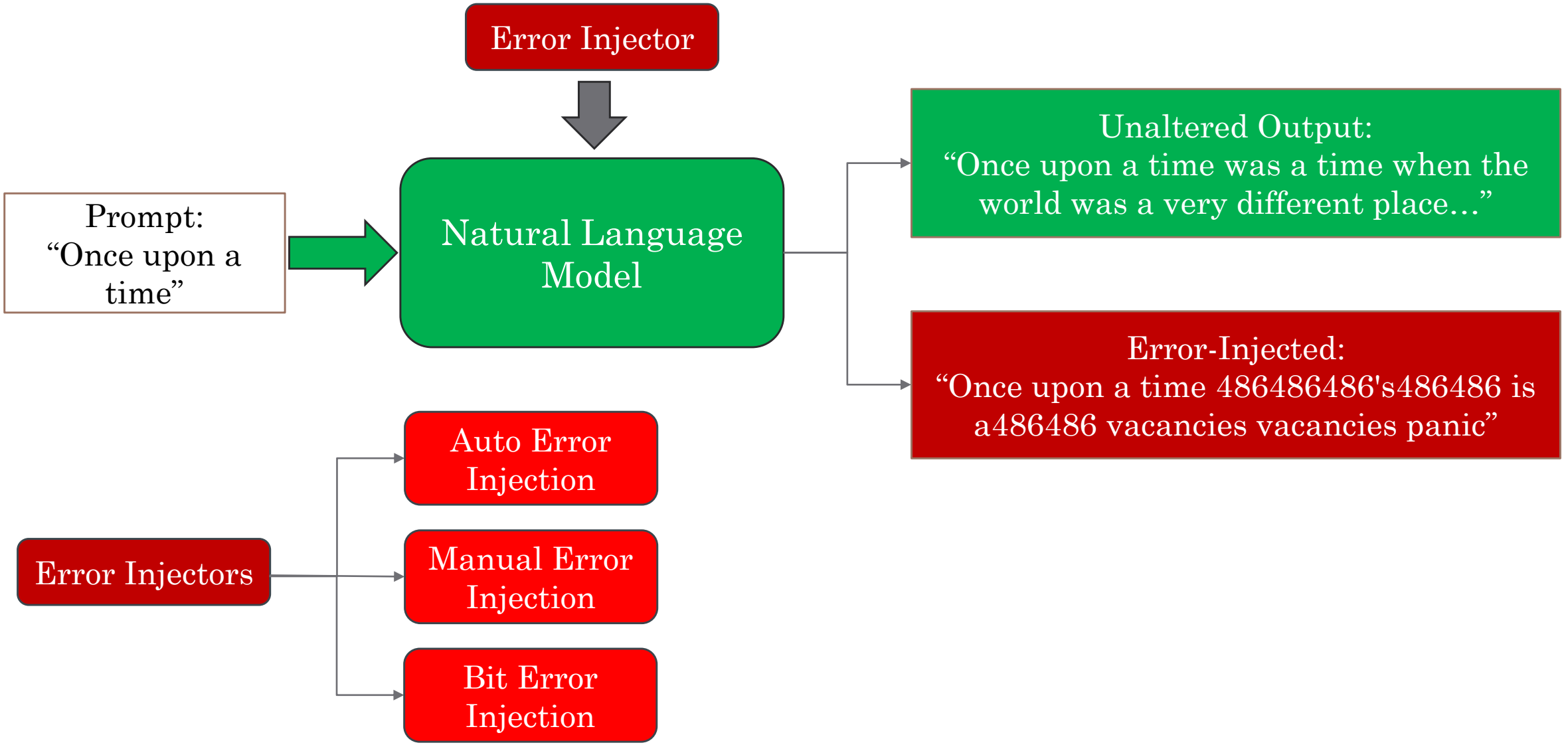
Limitations

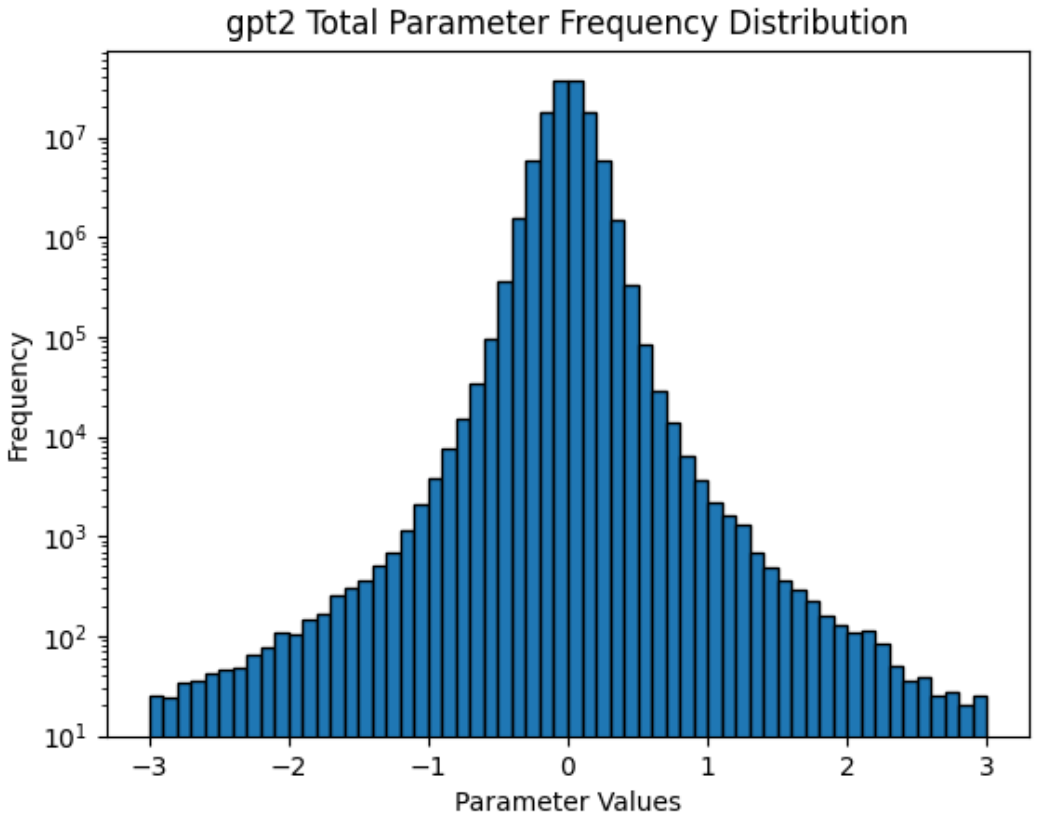


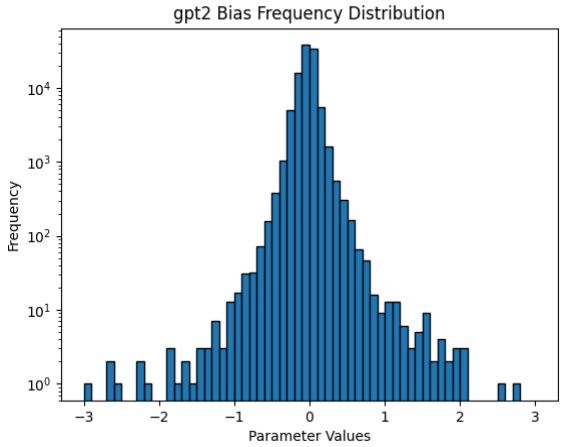
HuggingFace.co

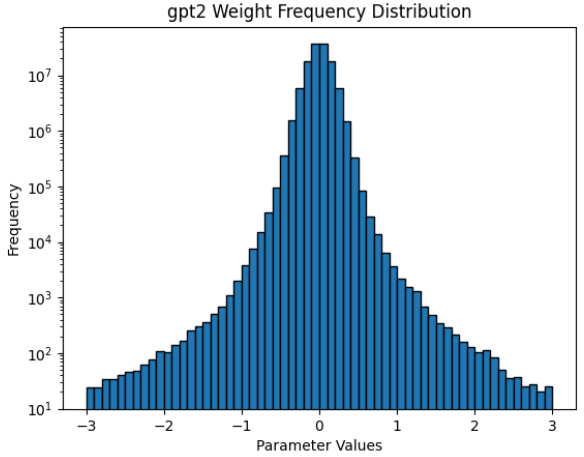
- Our project is developed for use with open-source Language Models through **HuggingFace**
  - Over 100,000 Language models
  - Project is applicable to all models
- Open-source models are less advanced than ChatGPT
- Text generation models were chosen for their more creative responses
  - They do not necessarily answer questions

System Chart

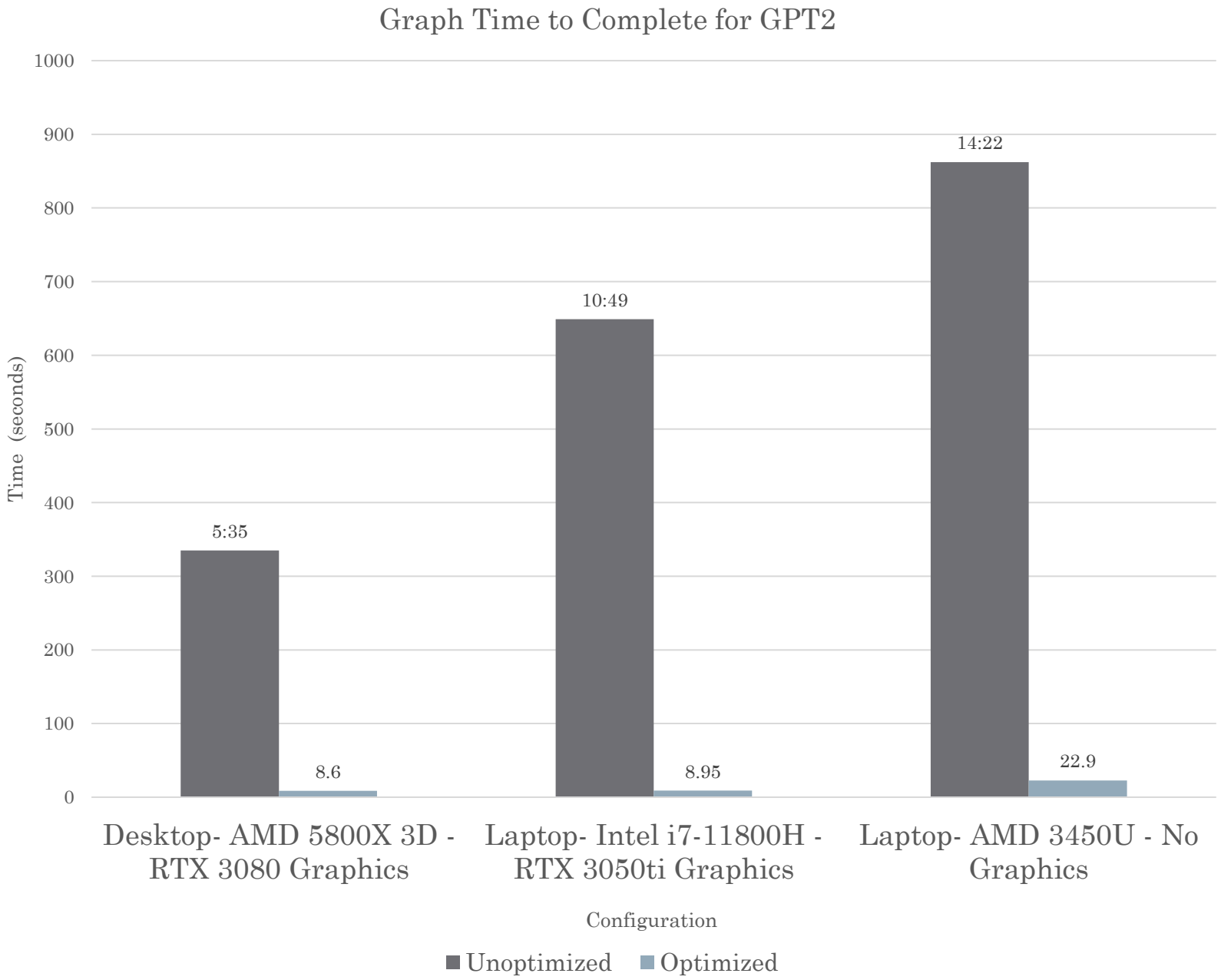








Plots of Parameter Distribution



Graph Optimization

- Function graphs a range of **100 million to 1 billion parameters**
- Optimization through logic improvements in the code
- Graph time was reduced by 97% on average

Optimizations

Initial Parameter Collector

```python
# Collecting parameter values
parameter_values = []
for _, param in model.named_parameters():
    param_values = param.data.view(-1).cpu().numpy()
    parameter_values.extend(param_values)
```

Optimized Parameter Collector

```python
# Iterate through the named parameters of the model
for name, param in model.named_parameters():
    if "weight" in name:
        # Reshape the parameter tensor to a 1D tensor
        param_tensor = param.view(-1)

        # Concatenate the parameter tensor to the existing tensor
        if concatenated_tensor is None:
            concatenated_tensor = param_tensor
        else:
            concatenated_tensor = torch.cat((concatenated_tensor, param_tensor))

# Convert the GPU tensor to a CPU tensor (if necessary)
concatenated_tensor_cpu = concatenated_tensor.cpu()

# Detach the tensor from the computation graph and convert it to a NumPy array
parameter_values = concatenated_tensor_cpu.detach().numpy()
```

Plot Distribution

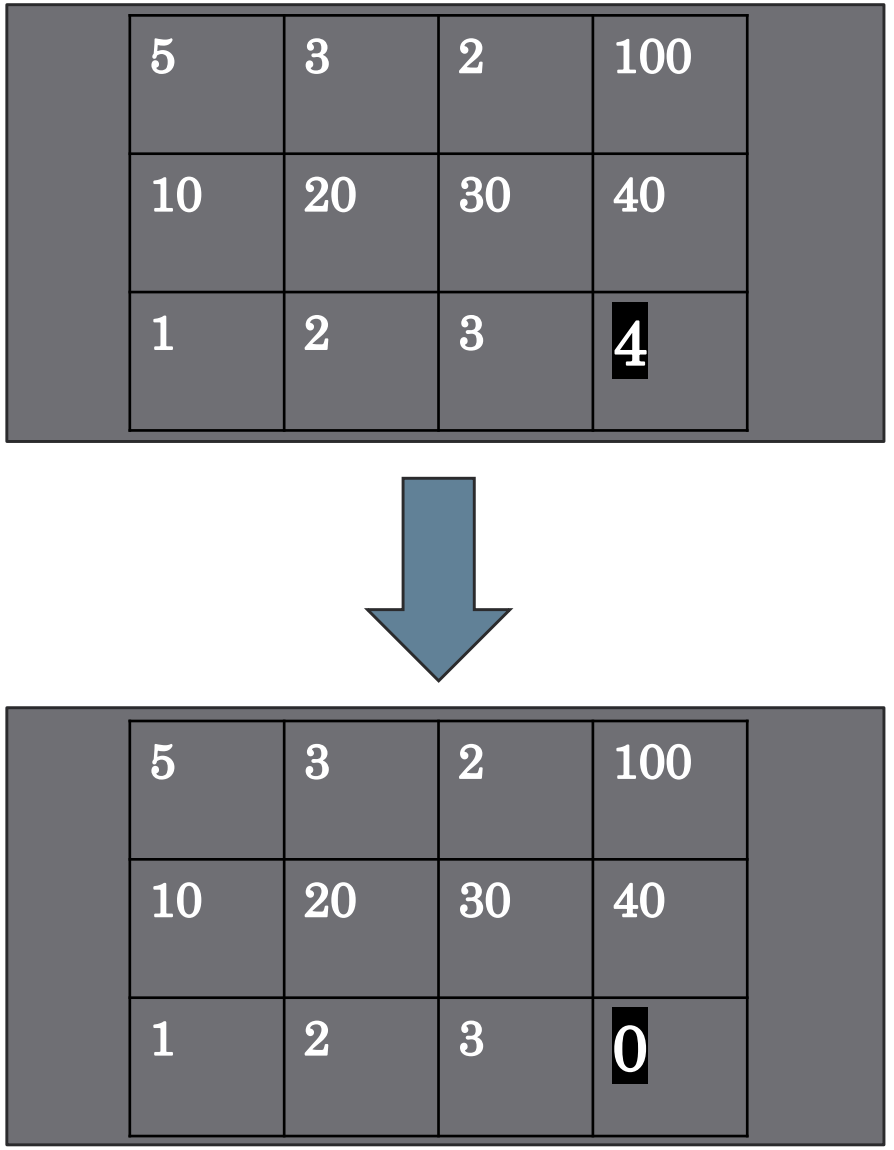
```python
# Plotting the weight distribution
start = -10.0
stop = 10.0
step = 0.05
bins = [round(start + i * step, 1) for i in range(int((stop - start) / step))]

plt.hist(parameter_values, bins=bins, edgecolor="black", log=True)
```

- Optimized Collector utilizes **Pytorch** command `.cat` instead of built in Python `.extend`
- `.cat` is highly optimized for just tensors while `.extend` is built for generic combination of lists
- Optimized Collector also converts entire tensor to NumPy outside of loop

Bit Flipper Function

- Input a tensor of any dimensions
- Selects a random element and flips a singular bit
- Could not be used in NLP models because of **TensorFlow**

```python
def tensorBitFlipper(tensor):
    if len(tf.shape(tensor)[0:]) > 1:
        num_subtensors = tensor.shape[0]
        subtensor_index = random.randint(0, num_subtensors - 1)
        subtensor = tensor[subtensor_index]

        bit_position = random.randint(0, tf.size(subtensor) - 1)
        element_index = random.randint(0, subtensor.shape[0] - 1)

        flip_value = tf.bitwise.left_shift(tf.constant(1, dtype=subtensor.dtype), bit_position)
        element_to_flip = subtensor[element_index]
        flipped_element = tf.bitwise.bitwise_xor(element_to_flip, flip_value)

        flipped_subtensor = tf.tensor_scatter_nd_update(subtensor, [[element_index]], [flipped_element])
        flipped_tensor = tf.tensor_scatter_nd_update(tensor, [[subtensor_index]], [flipped_subtensor])

        return flipped_tensor
    else:
        length = tf.size(tensor)
        bit_position = random.randint(0, length - 1)
        element_index = random.randint(0, length - 1)
        flip_value = 1 << bit_position
        flipped_element = tensor[element_index] ^ flip_value
        tensor_array = tensor.numpy()
        tensor_array[element_index] = flipped_element
        updated_tensor = tf.constant(tensor_array)
        return updated_tensor
```

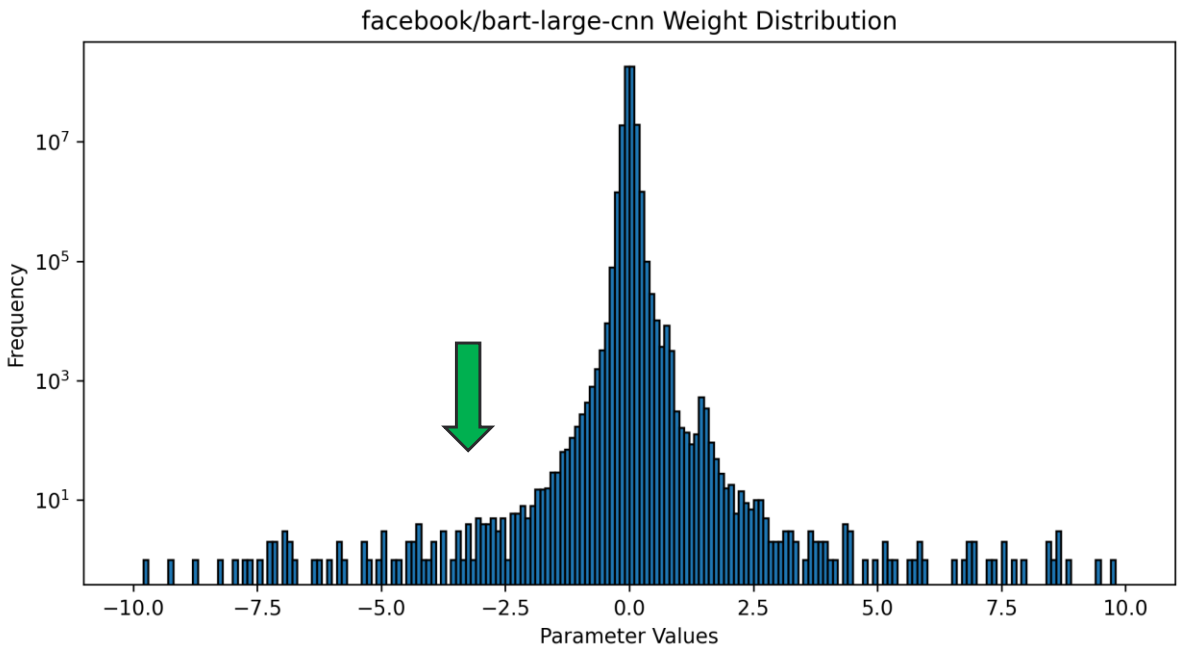


Auto Error Injection

- Input number of tensors of parameters to change and value to change them to
- Selects tensors to change based on importance
- Fills tensors with new value

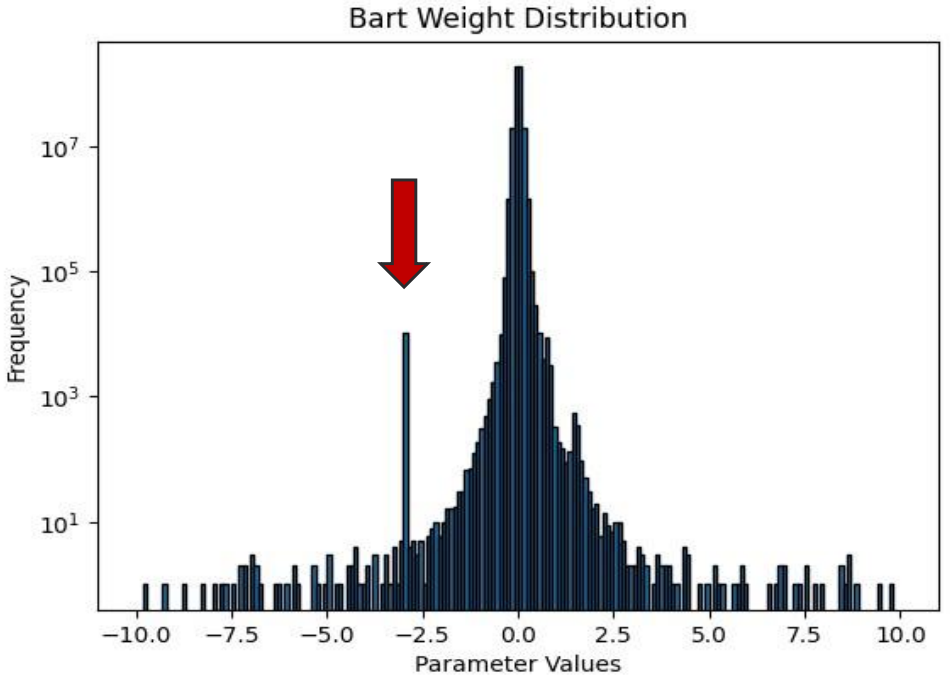
```
def BartErrorInjector(num_params, new_val, input_text):
    def get_parameter_importance(model: nn.Module) -> dict:
        parameter_importance = {}
        for name, parameter in model.named_parameters():
            parameter_importance[name] = torch.std(parameter).item()  # Calculate importance based
        return parameter_importance

    def modify_parameters(model: nn.Module, num_params: int, modification_func: callable):
        parameter_importance = get_parameter_importance(model)
        sorted_params = sorted(parameter_importance.items(), key=lambda x: x[1], reverse=True)
        total_params = len(sorted_params)

        if num_params > total_params:
            num_params = total_params

        selected_params = [param[0] for param in sorted_params]
        modified_params = set()

        for name, parameter in model.named_parameters():
            if len(modified_params) < num_params and name in selected_params:
                modified_params.add(name)
                modified_parameter = modification_func(parameter)
                parameter.data.copy_(modified_parameter)
            else:
                parameter.requires_grad_(False)

modification_func = lambda parameter: parameter.clone().fill_(new_val)  # Example mo
```

Pre-Auto Error Injection



Original Value ≈ 0
Average Value ≈ 0.0002493

Post-Auto Error Injection



New Value = -3
Average Value ≈ 0.0001734

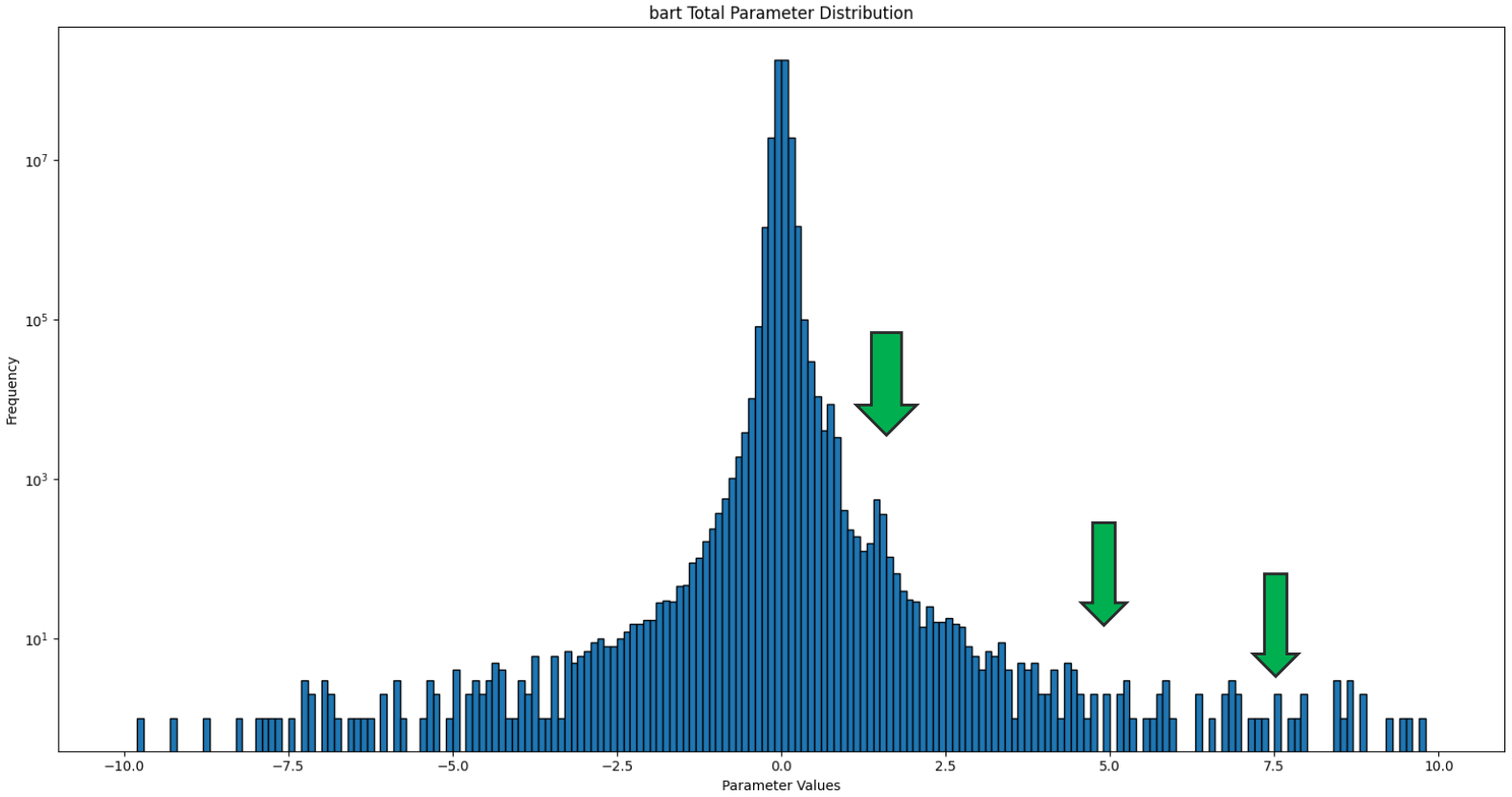
Percent Difference ≈ 35.94%

Manual Error Injection

- Input model you want to attack, what node you want to attack, and two variables for randomness.
- Attacks nodes by performing logical operation of XOR.
- Useful for choosing specific nodes for attacks

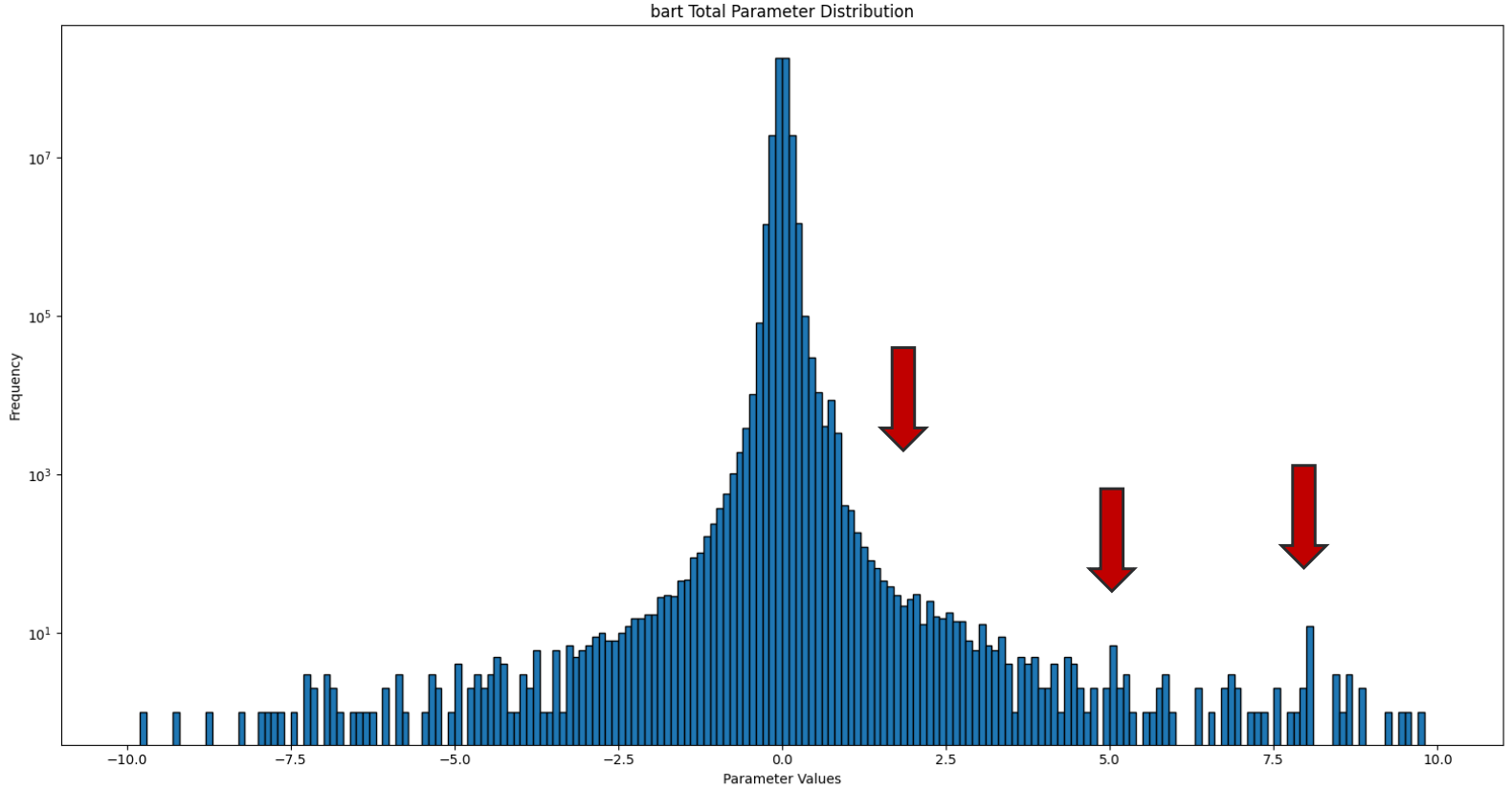
```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
dtype = torch.float32  # Specify the data type you want to retrieve the number of bits for
bitwidth = torch.finfo(dtype).bits

def error_map(injectee_shape: tuple, dtype_bitwidth: int, device: torch.device, scale_factor = .1, p = 1e-10) -> torch.Tensor:
    with torch.no_grad():
        error_map = (2 * torch.ones((*injectee_shape, dtype_bitwidth), dtype=torch.int, device=device)) ** torch.arange(0, dtype_bitwidth,

        filter = (p * nn.functional.dropout(torch.ones_like(error_map, dtype=torch.float, device=device), 1 - p)).int()

        error_map = (filter * error_map * scale_factor).sum(dim=-1).int()

    return error_map

def error_inject(model, attack, sf, p):
    error_maps = {}

    for param_name, param in model.named_parameters():
        # Options for attacks are here, you can do weights in general bias in general
        # Then you can do specific kinds of weights/biases attn.weights, proj.weights
        if attack in param_name:# or "bias" in param_name:

            injectee_shape = param.shape

            error_maps[param_name] = error_map(injectee_shape, bitwidth, device, sf, p)

            error_fin = error_maps[param_name]

            param.data = (param.data.to(torch.int) ^ error_fin).to(torch.float)
```

Bart Pre-Manual Error Iniection



Original Average ≈ 0.0002493

Bart Pre-Manual Error Injection



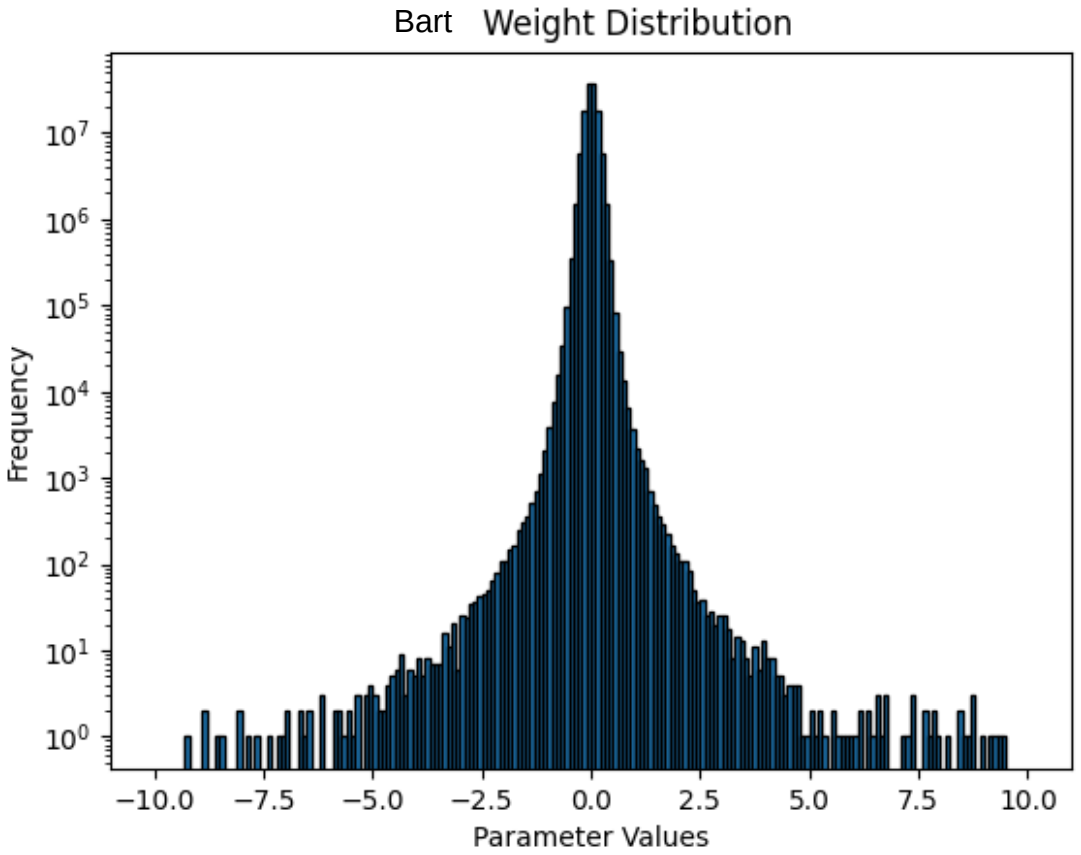
New Average ≈ -8156.1381
Percent Difference ≈ 200%

Bit Error Injection

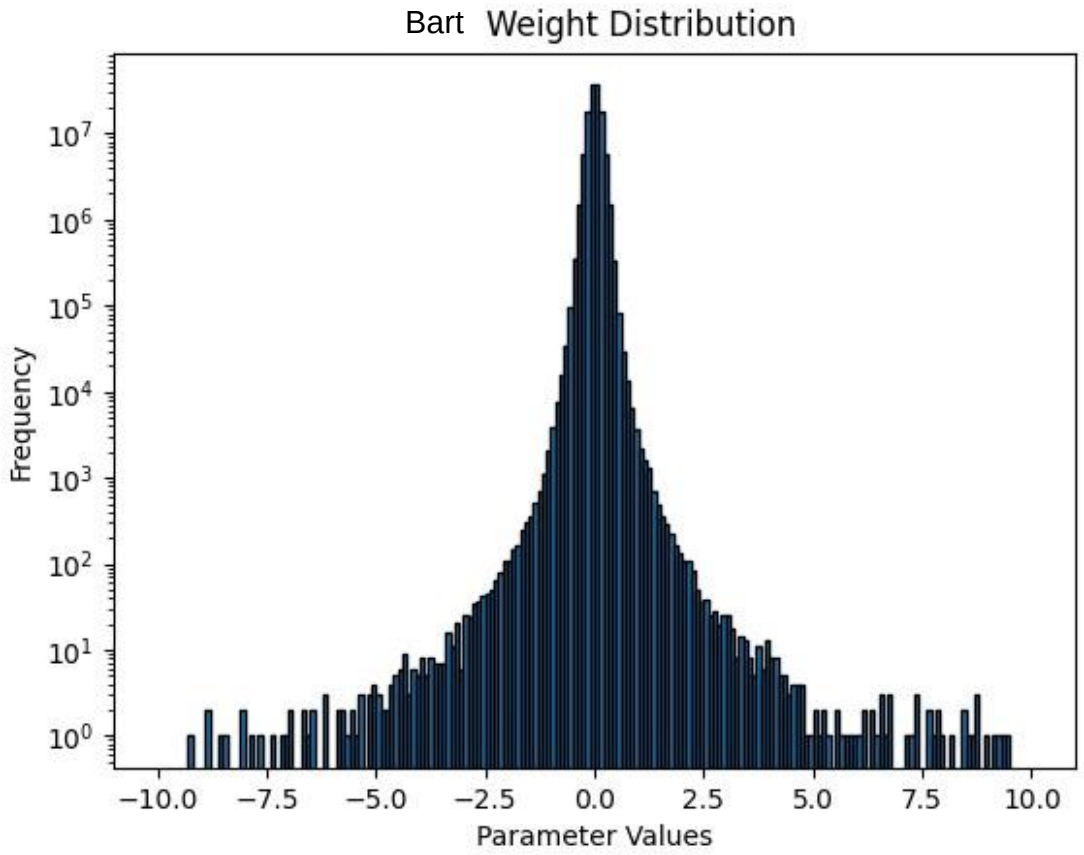
- Input tensor by index and number of bits to flip
- Converts tensor to binary, flips number of bits specified and converts back to decimal

```python
def GPT2SingleInjector(param_index, num_bits, input_text):
    def flip_bits(tensor, num_bits):
        # Convert the tensor to binary
        binary_tensor = tensor.byte()

        # Get the number of bits in each element
        num_bits_per_element = binary_tensor.numel() * 8

        # Generate random bit indices to flip
        bit_indices = random.sample(range(num_bits_per_element), num_bits)

        # Create a copy of the tensor to modify
        modified_tensor = tensor.clone()

        # Flip the selected bits in the modified tensor
        for index in bit_indices:
            element_index = index // 8  # Index of the element in the tensor
            bit_offset = index % 8      # Offset of the bit within the element
            modified_tensor.view(-1)[element_index] = modified_tensor.view(-1)[element_index].to(torch.long)

        return modified_tensor

    def modify_parameter(model: nn.Module, param_index: int, num_bits: int):
        parameter = list(model.parameters())[param_index]
        modified_parameter = flip_bits(parameter.data, num_bits)
        parameter.data.copy_(modified_parameter)
```

Pre-Error Injection



Post-Error Injection



Original Average ≈ 0.0002493

New Average ≈ 0.0002495

Percent Difference ≈ **0.0748%**

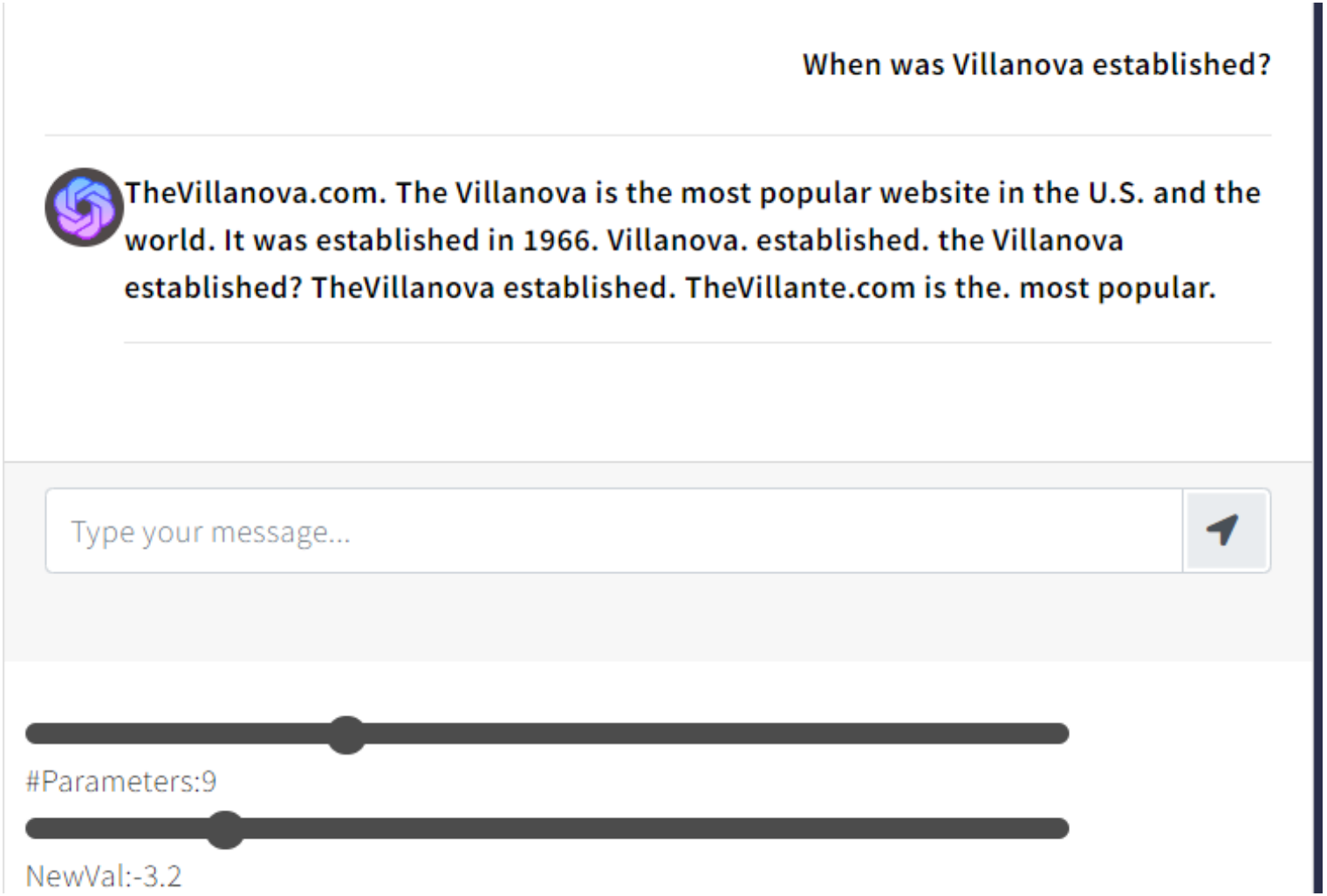
Bart - Original

When was Villanova established?

When was Villanova established? Villanova was founded in 1872. Villanova is one of the oldest universities in the United States. The school is located in Philadelphia, Pennsylvania. The campus is home to the University of Pennsylvania's men's basketball team. The Villanova women's team was established in 1876.

Type your message...

Bart – Auto Error Injection



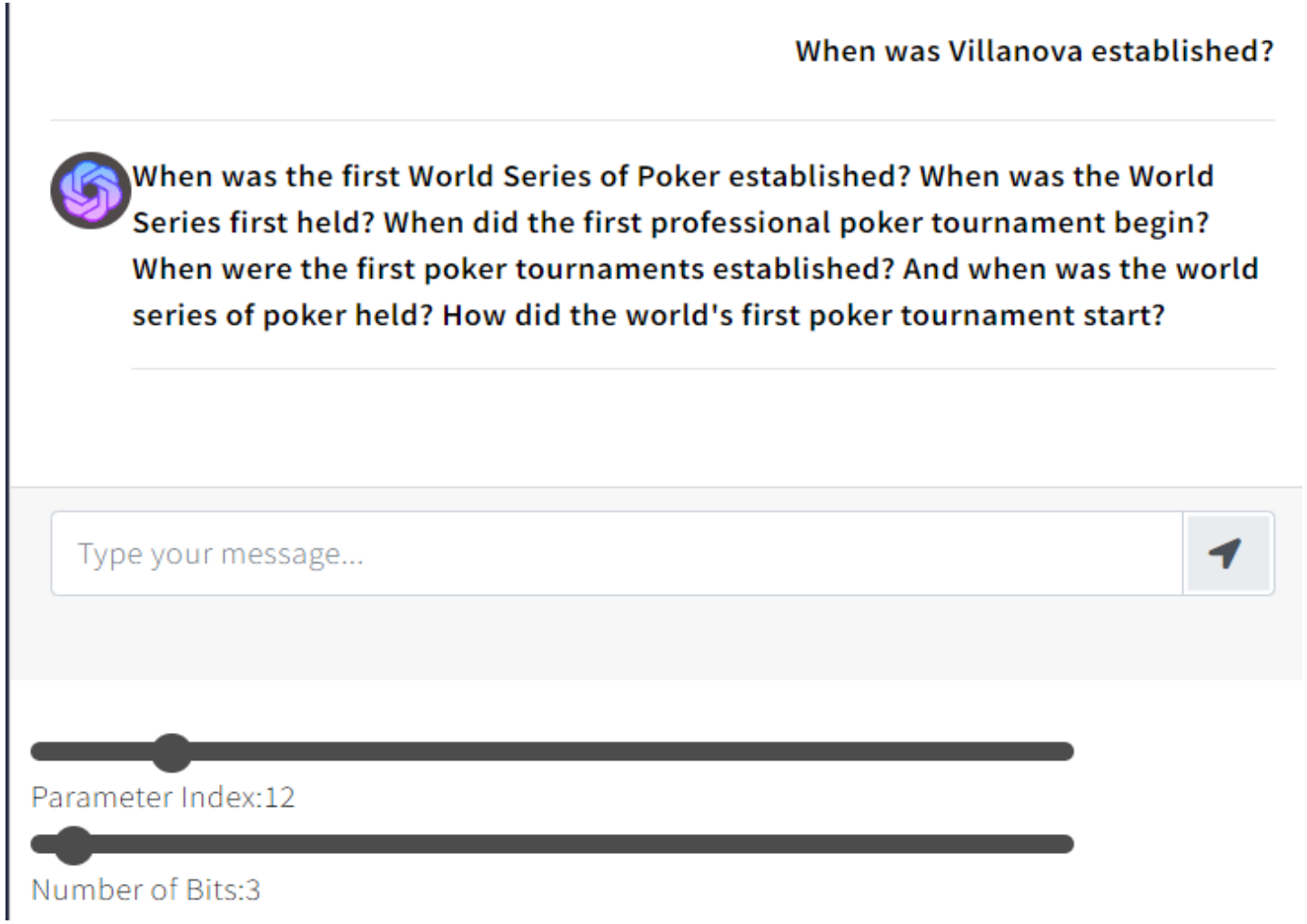
Bart – Manual Error Injection

When was Villanova established?

downwards downwards tuberculosis downwards universe tuberculosis tuberculosis universe downwards tuberculosis universe universe mailing downwards mailing tuberculosis mailing universe mailing harassment downwards harassment tuberculosis harassment universe harassment eaves downwards eaves tuberculosis eaves universe eaves limbo downwards limbo tuberculosis limbo universe limbo loosen downwards loosen tuberculosis loosen universe loosen pretend downwards pretend tuberculosis pretend universe pretend Medicare downwards Medicare tuberculosis Medicare universe Medicare scrimmage downwards scrimmage tuberculosis scrimmage universe scrimmage cocaine downwards cocaine tuberculosis cocaine universe cocaineulnerable downwardsulnerable tuberculosisulnerable universeulnerable circumcised

- Attacked v_proj.bias
- Dropout rate: 0.09
- Scale Factor: 0.1

Bart – Bit Error Injection



Key Takeaways

Name	Method	Response Time	Strengths	Shortfalls
Auto Error Injection	Picks the most frequent parameter tensor automatically	**10 s**	- Simple, **Fast**	- Responses can be illegible - Repetitive outputs - **Over-injects errors**
Manual Error Injection	Allows user to pick a node to inject errors onto	**30 s**	- More interesting results - Control of parameters	- **Slow** - Changes entire node - Memory intensive
Bit Error Injection	Allows user to pick a specific tensor and number of bit flips to inject into the tensor	**15 s**	-Variation of results -Uses bit flips -**Subtle errors** -Shows progressively erroneous outputs -**Fast**	- Hard to see errors through graph -Deeper analysis of results needed

Conclusions

The amount of testing we performed was relatively minor.

It is evident that the 4 models we tested on – GPT-2, Bert, Bart, and RoBERTa, **are all very sensitive to these error injections.**

In the future, we'd like to be able to draw more analytical conclusions with our tool through higher level observation

Website Considerations

Check out our website:
sheaa.tech

The bulk of our project's focus was on implementing the error injection functions into the models from huggingface.

We decided that a website would be the best way to showcase our results regarding ease of use for a general audience.

Generate interest of AI developers to take away key information from our project

Team SHEAA: Exploring AI's Robustness through Error Simulations

SIMULATING HARDWARE ERRORS IN EXISTING LANGUAGE MODELS TO GAIN MORE INSIGHT INTO THE LIMITS THESE MODELS HAVE.

CHECK OUT OUR MODELS →

Future Plans

- Our project lacks any **data-based conclusions** due to not having way to gather data on a large scale
- Create **benchmark tool** to test the different models on their ability to resist simulated hardware attacks
  - Repeat process over a long period of time to gather data on the models tested
- Implement error injection into Llama 2
- **Research Paper**
  - Compiling the technical aspects of our project into a Research Paper with the help of our mentor, Dr. Xun Jiao

Tools and Codebase

Star us on GitHub!

https://github.com/liambaker10/SHEAA

Sources and Inspiration

- https://www.statista.com/chart/29174/time-to-one-million-users/
- https://www.bloomberg.com/news/articles/2023-01-23/microsoft-makes-multibillion-dollar-investment-in-openai
- https://fortune.com/2023/02/04/google-invests-300m-anthropic-openai-rival-making-chatgpt-challenger-claude-ai-chatbot-battle/
- https://www.independent.co.uk/news/science/subatomic-particles-cosmic-rays-computers-change-elections-planes-autopilot-a7584616.html
- https://www.pugetsystems.com/labs/articles/most-reliable-pc-hardware-of-2021-2279/
- Ruixuan Wang: Grad Student at Villanova
  - Manual Error Injection was inspired by his research

Andrej Karpathy

L3HARRIS™
FAST. FORWARD.

VILLANOVA UNIVERSITY
College of Engineering

Thank You!

Any Questions?

GitHub

Website: sheaa.tech