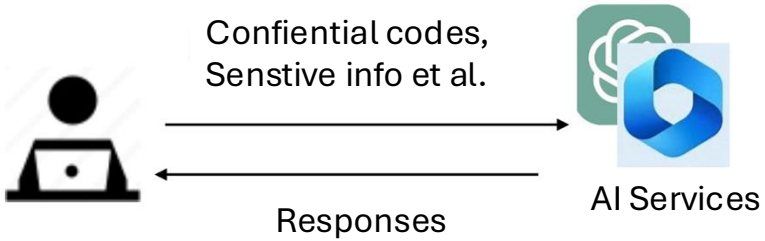
# BumbleBee: Secure Two-party Inference Framework for Large Transformers

Wen-jie Lu

NDSS 2025

# Background

Risk of privacy leak during the AI services




Event 1

**Samsung Bans ChatGPT After Engineers Use it to Fix Proprietary Code**

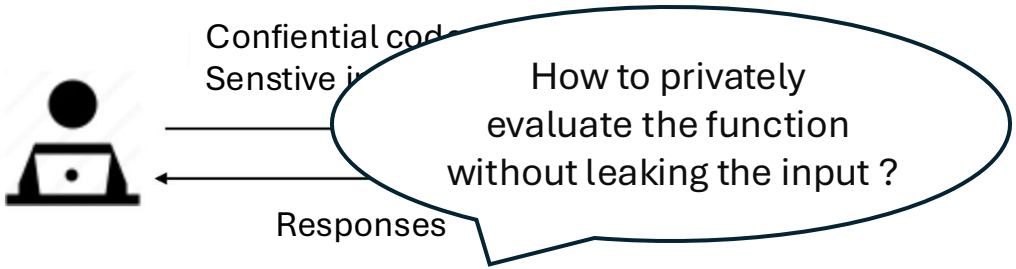
Employees who continue to use ChatGPT will face 'disciplinary action up to and including termination of employment,' according to a staff memo.

Event 2

**Yep, human workers are listening to recordings from Google Assistant, too**

# Background

Risk of privacy leak during the AI services




Event 1

**Samsung Bans ChatGPT After Engineers Use it to Fix Proprietary Code**

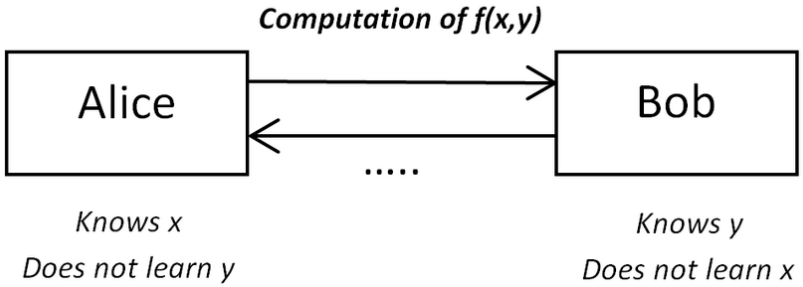
Employees who continue to use ChatGPT will face 'disciplinary action up to and including termination of employment,' according to a staff memo.

Event 2

**Yep, human workers are listening to recordings from Google Assistant, too**

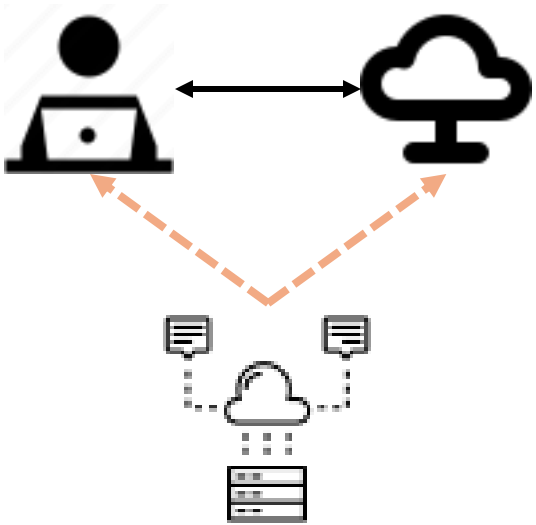
# Secure Two-party Computation (2PC)

- Two parties with private inputs x and y
- Compute a joint function of their inputs while preserving
  - Privacy
  - Correctness
- Using Correlated Randomness
  - (ra, rb, rc) => rc = ra * rb or rc = ra ∧ rb

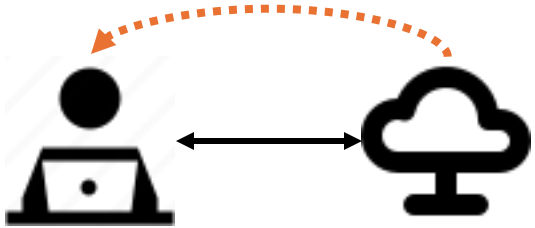


# Variants: Distributing Correlated Randomness

Trusted Third-party



Trusted First-party



# Related Work

| | Secret Sharing | Variant | Underlying 2PC lib | Transformers | Fine tune? |
|---|---|---|---|---|---|
| MPCFormer (ICRL23) | Additive | TFP | CrypTen | BERT (manually constructed) | Demanded |
| SHAFT (NDSS25) | Additive | TFP | CrypTen | Models written in PyTorch, e.g., BERT, GPT2, ViT | Prefer |
| SIGMA (PoPETs24) | Functional | TTP | EzPC (Easy Secure Multiparty Computation) | Models written in ONNX (subsets) | Not necessary |
| Iron (NeurIPS22) | Additive | 2PC | EzPC | BERT (manually constructed) | Prefer |
| BOLT (Oakland24) | Additive | 2PC | EzPC | BERT (manually constructed) | Demanded |
| BumbleBee | Additive | 2PC | SecretFlow | Models written in JAX | Not necessary |

# Related Work

| | Secret Sharing | Variant | Underlying 2PC lib | Transformers | Fine tune? |
|---|---|---|---|---|---|
| MPCFormer (ICRL23) | Additive | TFP | CrypTen | BERT (manually constructed) | Demanded |
| SHAFT (NDSS25) | Additive | TFP | CrypTen | Models written in PyTorch, e.g., BERT, GPT2, ViT | Prefer |
| SIGMA (PoPETs24) | Functional | TTP | EzPC (Easy Secure Multiparty Computation) | Models written in ONNX (subsets) | Not necessary |
| Iron (NeurIPS22) | Additive | 2PC | EzPC | BERT (manually constructed) | Prefer |
| BOLT (Oakland24) | Additive | 2PC | EzPC | BERT (manually constructed) | Demanded |
| BumbleBee | Additive | 2PC | SecretFlow | Models written in JAX | Not necessary |

# Related Work

| | Secret Sharing | Variant | Underlying 2PC lib | Transformers | Fine tune? |
|---|---|---|---|---|---|
| MPCFormer (ICRL23) | Additive | TFP | CrypTen | BERT (manually constructed) | Demanded |
| SHAFT (NDSS25) | Additive | TFP | CrypTen | Models written in PyTorch, e.g., BERT, GPT2, ViT | Prefer |
| SIGMA (PoPETs24) | Functional | TTP | EzPC (Easy Secure Multiparty Computation) | Models written in ONNX (subsets) | Not necessary |
| Iron (NeurIPS22) | Additive | 2PC | EzPC | BERT (manually constructed) | Prefer |
| BOLT (Oakland24) | Additive | 2PC | EzPC | BERT (manually constructed) | Demanded |
| BumbleBee | Additive | 2PC | SecretFlow | Models written in JAX | Not necessary |

# Observations & Objectives

O1: Tremendous Communication

- ViT: 260 GB per image [1]
- Bert-base: 80 GB per 128tok

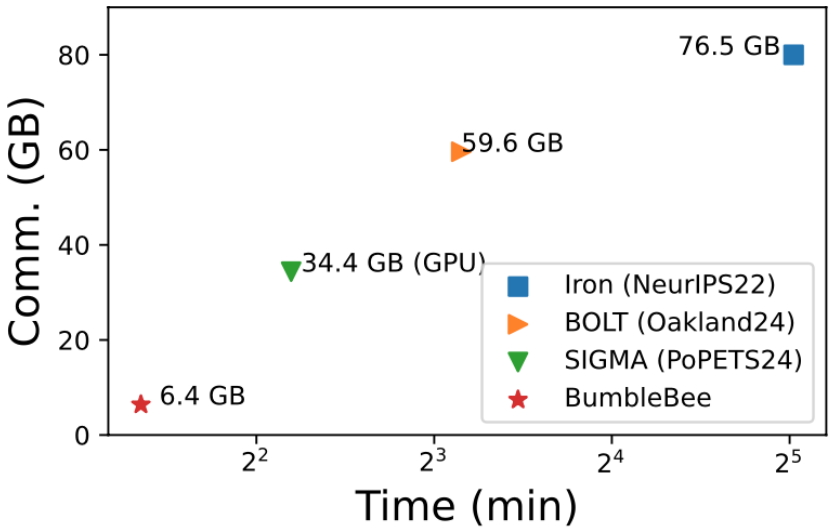
O2: $Computation < $Communication

- 0.09 cent per GB
- 0.025 cent per vCPU

Minimize the Communication

- Not considering the offline/online split
- Without increasing the computation by too much

One private inference on BERT-base with 128 inputs




[1] Wang et al. *Characterization of MPC-based Private Inference for Transformer-based Models*. (From the CrypTen team)

# Difficulties

- Evaluating nonlinear activations (e.g., GeLU and Softmax) in 2PC.
  - Many attempts turns to 2PC friendly alternatives. ☹ Accuracy.
  - Fancy numerical methods such as ordinary differential equations (ODE). ☹ Communication costs.
- ML-Crypto Co-engineering.
  - MLer writes 2PC codes vs. Cryptographer writes ML codes.
- Large-scale matrix multiplication in 2PC.

# Contributions

- Lessons for evaluating nonlinear function in 2PC.
- End-2-end code baes for private transformer inference.
  - 100% reusing JAX codes from HuggingFaces, no model finetuning.
- Efficient point-wise mul (aka OLE) and Matmul in $\mathbb{Z}_{2^k}$ from HE
  - For example: 128x768 matrix in 0.6s and 5MB (10% of the prev)
  - A concurrent work by Jiaxing He et al. in CCS24 (Rhombus)

https://github.com/AntCPLab/OpenBumbleBee

# Three lessons for the private transformers inference

# Lesson 1: Reuse the ML codes via 2PC compilers

## Or: Let the MLers write the ML codes

- Large transformer models could be “hard” to write from scratch.
  - Iron, MPCFormer, BOLT only run on one model by hand crafting
  - Handling the existing model weight files is also tedious
- A better way, plugin a 2PC backend to the current ML codes (eg PyTorch, TensorFlow)
  - CrypTen (SHAFT): Python-level overriding the PyTorch API
  - EzPC (SIGMA): Uses SeeDot [1] to compile TensorFlow/ONNX to a C++ backend
  - SecretFlow (BumbleBee): Multi-level IR (MLIR). Compile JAX/PyTorch to a C++ backend

[1] https://github.com/mpc-msri/EzPC/tree/master/Athos#compiling-a-tensorflow-model

# Lesson 1: Reuse the ML codes via 2PC compilers

## Or: Let the MLers write the ML codes

- Large transformer models could be “hard” to write from scratch.
  - Iron, MPCFormer, BOLT only run on one model by hand crafting
  - Handling the existing model weight files is also tedious
- A better way, plugin a 2PC backend to the current ML codes (eg PyTorch, TensorFlow)
  - CrypTen (SHAFT): Python-level overriding the PyTorch API
  - EzPC (SIGMA): Uses SeeDot [1] to compile TensorFlow/ONNX to a C++ backend
  - SecretFlow (BumbleBee): Multi-level IR (MLIR). Compile JAX/PyTorch to a C++ backend

Prefer a middle-level IR than the eager execution

[1] https://github.com/mpc-msri/EzPC/tree/master/Athos#compiling-a-tensorflow-model

# Why Preferring Lower-level IR by the Softmax Example

JAX's snippet for Softmax

```
1. def _softmax(x, axis = -1):
2.   x_max = jnp.max(x, axis,
 ↪   keepdims=True)
3.   unnormalized = jnp.exp(x - x_max)
4.   sum_exp = jnp.sum(unnormalized, axis,
 ↪   keepdims=True)
5.   result = unnormalized / sum_exp
6.   return result
```

$$\text{softmax}(\vec{x})[j] = \frac{\exp(x_j)}{\sum_i \exp(x_i)}$$

SHAFT's snippet for Softmax

```
if method == "reciprocal":
    maximum_value = self.max(dim, keepdim=True)[0]
    logits = self - maximum_value
    numerator = logits.exp()
    with cfg.temp_override({"functions.reciprocal_all_pos": True}):
        inv_denominator = numerator.sum(dim, keepdim=True).reciprocal()
    return numerator * inv_denominator
```

[1] https://github.com/jax-ml/jax/blob/ed952c8e651bf8318687e95ac545358a884b7bf3/jax/_src/nn/functions.py#L601
[2] https://github.com/andeskyl/SHAFT/blob/8ade8e3858611983dee1e77f6da6d346e7e08831/crypten/common/functions/approximations.py#L604

# Why Preferring Lower-level IR by the Softmax Example

## JAX's snippet for Softmax

```
1. def _softmax(x, axis = -1):
2.   x_max = jnp.max(x, axis,
 ↪   keepdims=True)
3.   unnormalized = jnp.exp(x - x_max)
4.   sum_exp = jnp.sum(unnormalized, axis,
 ↪   keepdims=True)
5.   result = unnormalized / sum_exp
6.   return result
```

$$\text{softmax}(\vec{x})[j] = \frac{\exp(x_j)}{\sum_i \exp(x_i)}$$

- "keepdims=True" aims to align the operands shape
- However, it increases the costs of division by 10x – 100x times!
  - x.div(y.broadcast(x.shape)): O(n) div and O(n) mul ☹
  - x.mul(y.recip.broadcast(x.shape)): O(1) recip and O(n) mul ☺

## SHAFT's snippet for Softmax

```
if method == "reciprocal":
    maximum_value = self.max(dim, keepdim=True)[0]
    logits = self - maximum_value
    numerator = logits.exp()
    with cfg.temp_override({"functions.reciprocal_all_pos": True}):
        inv_denominator = numerator.sum(dim, keepdim=True).reciprocal()
    return numerator * inv_denominator
```

[1] https://github.com/jax-ml/jax/blob/ed952c8e651bf8318687e95ac545358a884b7bf3/jax/_src/nn/functions.py#L601
[2] https://github.com/andeskyl/SHAFT/blob/8ade8e3858611983dee1e77f6da6d346e7e08831/crypten/common/functions/approximations.py#L604

# Why Preferring Lower-level IR by the Softmax Example

## JAX's snippet for Softmax

```
1. def _softmax(x, axis = -1):
2.   x_max = jnp.max(x, axis,
 ↪   keepdims=True)
3.   unnormalized = jnp.exp(x - x_max)
4.   sum_exp = jnp.sum(unnormalized, axis,
 ↪   keepdims=True)
5.   result = unnormalized / sum_exp
6.   return result
```

$$\text{softmax}(\vec{x})[j] = \frac{\exp(x_j)}{\sum_i \exp(x_i)}$$

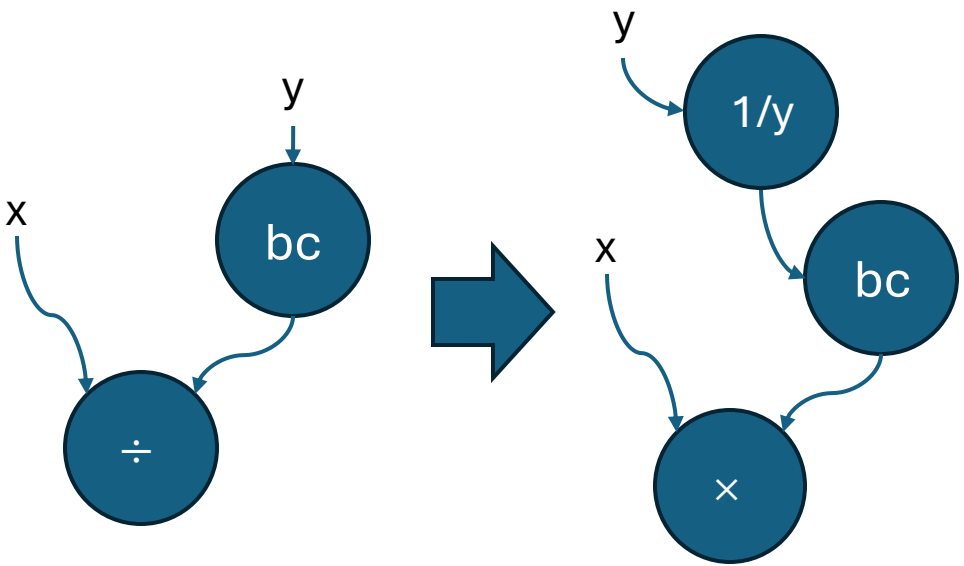
- "keepdims=True" aims to align the operands shape
- However, it increases the costs of division by 10x – 100x times!
  - **x**.div(y.broadcast(**x**.shape)): O(n) div and O(n) mul ☹
  - **x**.mul(y.recip.broadcast(**x**.shape)): O(1) recip and O(n) mul ☺

## SHAFT's snippet for Softmax

```
if method == "reciprocal":
    maximum_value = self.max(dim, keepdim=True)[0]
    logits = self - maximum_value
    numerator = logits.exp()
    with cfg.temp_override({"functions.reciprocal_all_pos": True}):
        inv_denominator = numerator.sum(dim, keepdim=True).reciprocal()
    return numerator * inv_denominator
```

[1] https://github.com/jax-ml/jax/blob/ed952c8e651bf8318687e95ac545358a884b7bf3/jax/_src/nn/functions.py#L601
[2] https://github.com/andeskyl/SHAFT/blob/8ade8e3858611983dee1e77f6da6d346e7e08831/crypten/common/functions/approximations.py#L604

# Why Preferring Lower-level IR by the Softmax Example




$$\text{softmax}(\vec{x})[j] = \frac{\exp(x_j)}{\sum_i \exp(x_i)}$$

- "keepdims=True" aims to align the operands shape
- However, it increases the costs of division by 10x – 100x times!
  - **x**.div(y.broadcast(**x**.shape)): O(n) div and O(n) mul ☹
  - **x**.mul(y.recip.broadcast(**x**.shape)): O(1) recip and O(n) mul ☺

[1] https://github.com/secretflow/spu/blob/main/src/libspu/dialect/pphlo/transforms/optimize_denominator_with_broadcast.cc

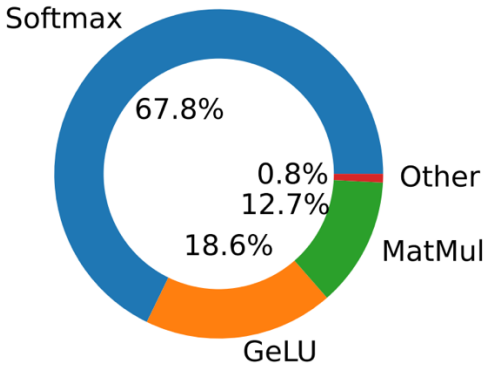
# A sad story …

In BumbleBee, the division (reciprocal) in softmax takes less than 1% of the total costs.

## Wang et al [1]

| | MPC breakdown % |
|---|---|
| Embedding | 20.37% |
| Norm | 0.74% |
| MatMul | 14.17% |
| ReLU | 14.82% |
| Softmax | 49.90% |
| Total | 100% |

## MPCFormer [2]




## SHAFT [3]

Apply a numerical method to avoid the division which can be handled more easily actually.

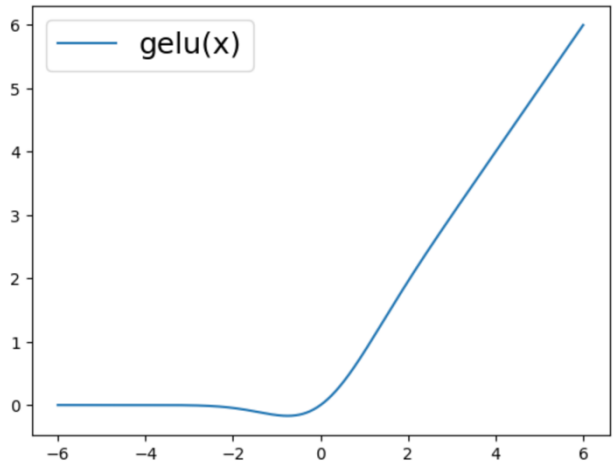
[1] Wang et al. Characterization of MPC-based Private Inference for Transformer-based Models
[2] Li et al. MPCFORMER: FAST, PERFORMANT AND PRIVATE TRANSFORMER INFERENCE WITH MPC
[3] Andes Y. L. Kei et al. SHAFT: Secure, Handy, Accurate and Fast Transformer Inference

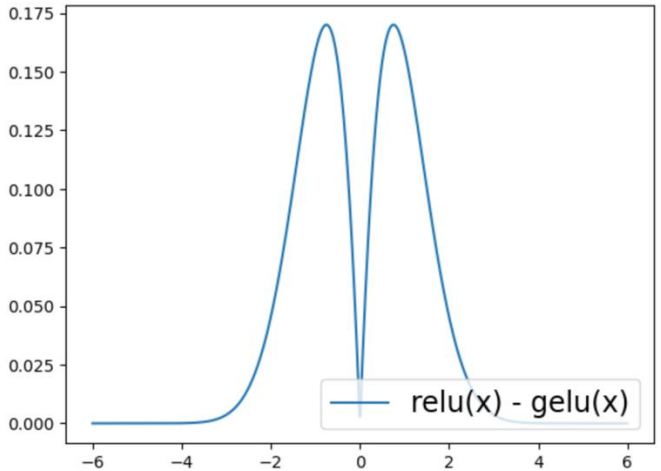
# Lesson 2: Piecewise rather than numerical methods

All about is the numerical stability

# Lesson 2 by the Gelu function

$$\text{Gelu}(x) = 0.5x(1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3)))$$




[1] Gupta et al. SIGMA: Secure GPT Inference with Function Secret Sharing
[2] Pang et al. BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers

# Lesson 2 by the Gelu function

$$\text{Gelu}(x) = 0.5x(1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3)))$$

Attempt 1

Attempt 2

Attempt 3

$$\tanh(x) \approx \frac{x + x^3/9 + x^5/945}{1 + 4x^2/9 + x^4/63}$$

- Need clipping on [-3, 3]
- Heavy division



[LHZWH] *Squirrel: A Scalable Secure Two-Party Computation Framework for Training Gradient Boosting Decision Tree*

# Lesson 2 by the Gelu function

$$\text{Gelu}(x) = 0.5x(1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3)))$$

**Attempt 1**

$$\tanh(x) \approx \frac{x + x^3/9 + x^5/945}{1 + 4x^2/9 + x^4/63}$$

- Need clipping on [-3, 3]
- Heavy division

**Attempt 2**

$$\tanh(x) \approx \sum_{j=0}^{7} F_j(x)$$

- 8 degree Fourier series approximation
- Need clipping on [-4, -4]
- 18 MULs

**Attempt 3**

$$\text{Relu}(x) - \text{Gelu}(x) \approx \sum_{j=0}^{7} G_j(x)$$

- 8 degree Fourier series approximation
- Need clipping on [-4, -4]
- 14 MULs + max(0, x)
- > 1450bytes per Gelu



[LHZWH] *Squirrel: A Scalable Secure Two-Party Computation Framework for Training Gradient Boosting Decision Tree*

# Lesson 2 by the Gelu function

$$\text{Gelu}(x) = 0.5x(1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3)))$$

**Attempt 1**

$$\tanh(x) \approx \frac{x + x^3/9 + x^5/945}{1 + 4x^2/9 +}$$

- Need clipping on
- Heavy division

**Attempt 2**

$$\tanh(x) \approx \sum^{7} F_i(x)$$

It's time to rethink the simple piecewise approximation!

**Attempt 3**

$$\text{Relu}(x) - \text{Gelu}(x) \approx \sum_{j=0}^{7} G_j(x)$$

- 8 degree Fourier series approximation
- Need clipping on [-4, -4]
- 14 MULs + max(0, x)
- > 1450bytes per Gelu



[LHZWH] *Squirrel: A Scalable Secure Two-Party Computation Framework for Training Gradient Boosting Decision Tree*

# Lesson 2 by the Gelu function

1. One-vs-Many Comparisons

2. Smoothness

$$\begin{cases} -\epsilon & x \le -4 \\ P(x) = \sum_{i=0}^{i=3} a_i x^i & -4 < x \le -1.85 \\ Q(x) = \sum_{i=0}^{i=6} b_i x^i & -1.85 < x \le 3 \\ x - \epsilon & x > 3 \end{cases}$$

- The one-vs-many comparisons has a smaller amortized costs in 2PC.
- 3 comparisons is same as Attempt 3
- Low-degree polys share terms. 6 MULs vs 14 MULs

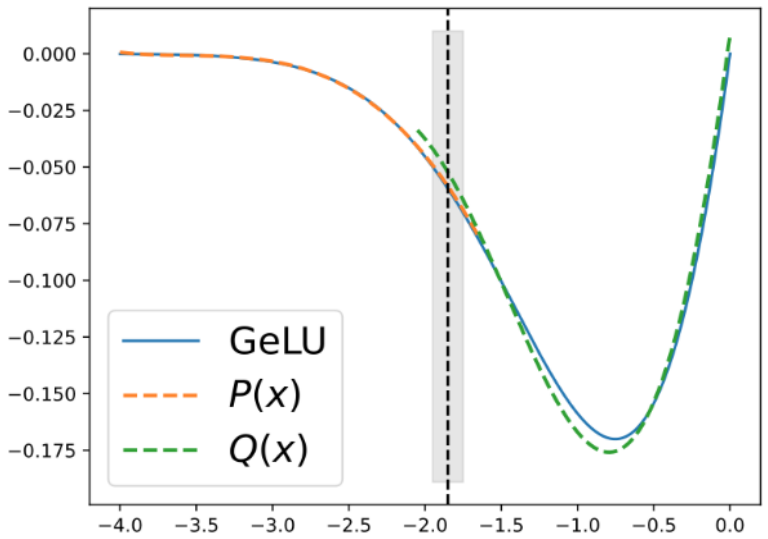
# Lesson 2 by the Gelu function

1. One-vs-Many Comparisons

$$\begin{cases} -\epsilon & x \le -4 \\ P(x) = \sum_{i=0}^{i=3} a_i x^i & -4 < x \le -1.85 \\ Q(x) = \sum_{i=0}^{i=6} b_i x^i & -1.85 < x \le 3 \\ x - \epsilon & x > 3 \end{cases}$$

- Smoothness allows an error-tune comparison (eg. ignoring some low-end bits)
- 718 bytes per gelu (50% off to Attempt 3)

2. Smoothness



# Lesson 3: Simplicity is good

All about is the numerical stability again

# Lesson 3 by the Softmax function

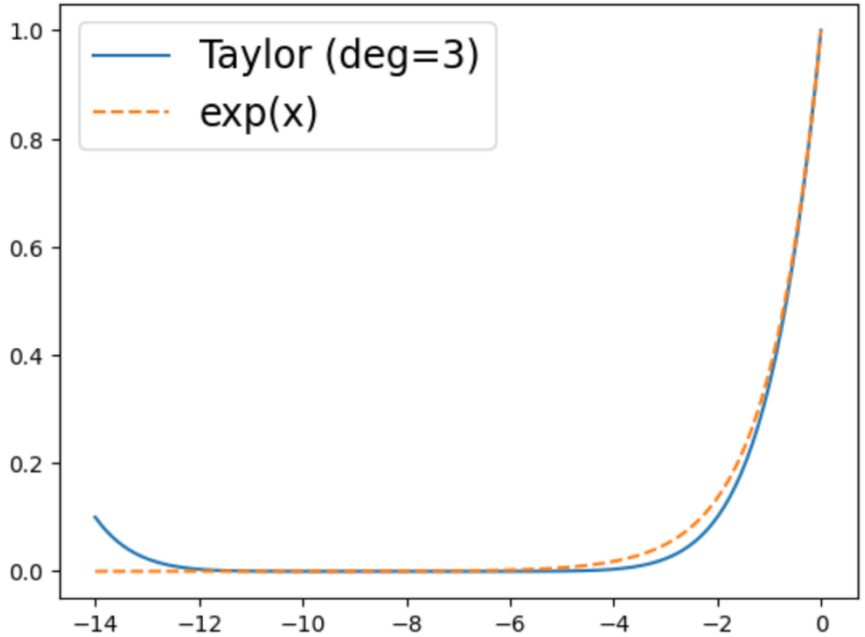
$$\text{softmax}(\vec{x})[j] = \frac{\exp(x_j)}{\sum_i \exp(x_i)} = \frac{\exp(x_j - \hat{x})}{\sum_i \exp(x_i - \hat{x})} \qquad \hat{x} = \max(\vec{x})$$

- Exp(x) on negative inputs are bounded.
- Good enough approximation by low-degree Taylor with one clipping

# Lesson 3 by the Softmax function

$$\text{softmax}(\vec{x})[j] = \frac{\exp(x_j)}{\sum_i \exp(x_i)} = \frac{\exp(x_j - \hat{x})}{\sum_i \exp(x_i - \hat{x})}$$

- Exp(x) on negative inputs are bounded.
- Good enough approximation by low-degr
  with one clipping



# Lesson 3 by the Softmax function

## “The reviewer” C:

> Also, recent work at ACSAC 2023 [c] demonstrates improved secure softmax computation. Integrating recent developments could provide a better review of related work and may also improve the practicality and efficiency of the proposed protocol.

# Lesson 3 by the Softmax function

## “The reviewer” C:

Also, recent work at ACSAC 2023 [c] demonstrates improved secure softmax computation. Integrating recent developments could provide a better review of related work and may also improve the practicality and efficiency of the proposed protocol.

```
# self \in [-iter_num, iter_num]
def ACSAC_softmax(self, iter_num):
    dim = self.shape[-1]
    x = self / iter_num
    g = np.ones(self.shape) / dim
    for _ in range(iter_num):
        z = x * g
        g = g + (z - sum(z) * g)
    return g
```

Pros:

- Division-free
- Maximum-free

Cons:

- #Iteration grows linear with the approximation range (which could be huge in LLM)

# Lesson 3 by the Softmax function

## “The reviewer” C:

Also, recent work at ACSAC 2023 [c] demonstrates improved secure softmax computation. Integrating recent developments could provide a better review of related work and may also improve the practicality and efficiency of the proposed protocol.

```
# self \in [-iter_num, iter_num]
def ACSAC_softmax(self, iter_num):
    dim = self.shape[-1]
    x = self / iter_num
    g = np.ones(self.shape) / dim
    for _ in range(iter_num):
        z = x * g
        g = g + (z - sum(z) * g)
    return g
```

Pros:

- Division-free
- Maximum-free

Cons:

- #Iteration grows linear with the approximation range (which could be huge in LLM)

# Lesson 3 by the Softmax function

## “The reviewer” C:

Also, recent work at ACSAC 2023 [c] demonstrates improved secure softmax computation. Integrating recent developments could provide a better review of related work and may also improve the practicality and efficiency of the proposed protocol.

```
# self \in [-iter_num, iter_num]
def ACSAC_softmax(self, iter_num):
    dim = self.shape[-1]
    x = self / iter_num
    g = np.ones(self.shape) / dim
    for _ in range(iter_num):
        z = x * g
        g = g + (z - sum(z) * g)
    return g
```

What if applying range clipping [1]?

- O(2n) comparisons
- However, the maximum only needs O(n) comparisons



[1] https://github.com/andeskyl/SHAFT/blob/8ade8e3858611983dee1e77f6da6d346e7e08831/crypten/common/functions/approximations.py#L611

# Thanks!

- 1678.pdf (iacr.org)
- AntCPLab/OpenBumbleBee (github.com)

# Anecdote: Why Naming BumbleBee

**Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference**


Zhicong Huang
*Alibaba Group*

Wen-jie Lu
*Alibaba Group*

Cheng Hong
*Alibaba Group*

Jiansheng Ding
*Alibaba Group*


ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

2022 USENIX

# Anecdote: Why Naming BumbleBee

Cheetah + Transfomer => Cheetor ("Maximize!")

The very first computer-aid 3D animation in 1996.

# Anecdote: Why Naming BumbleBee

Team Leader: Cheetor is so unknown! Let be Bumblebee.