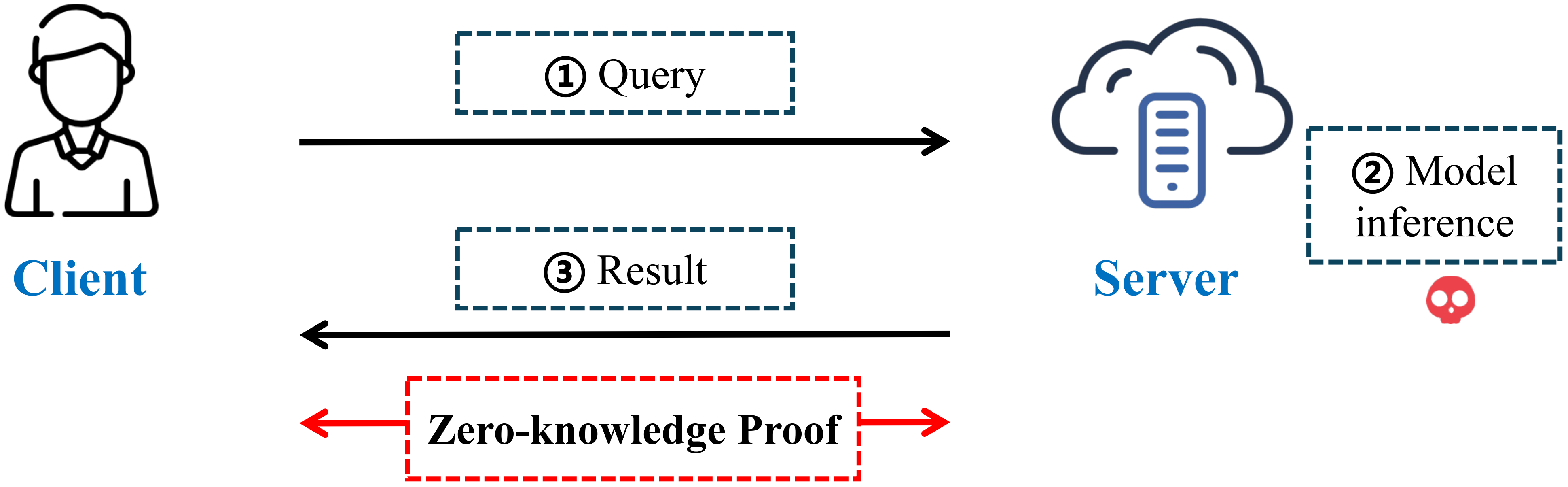
# Scalable Zero-knowledge Proofs for Non-linear Functions in Machine Learning

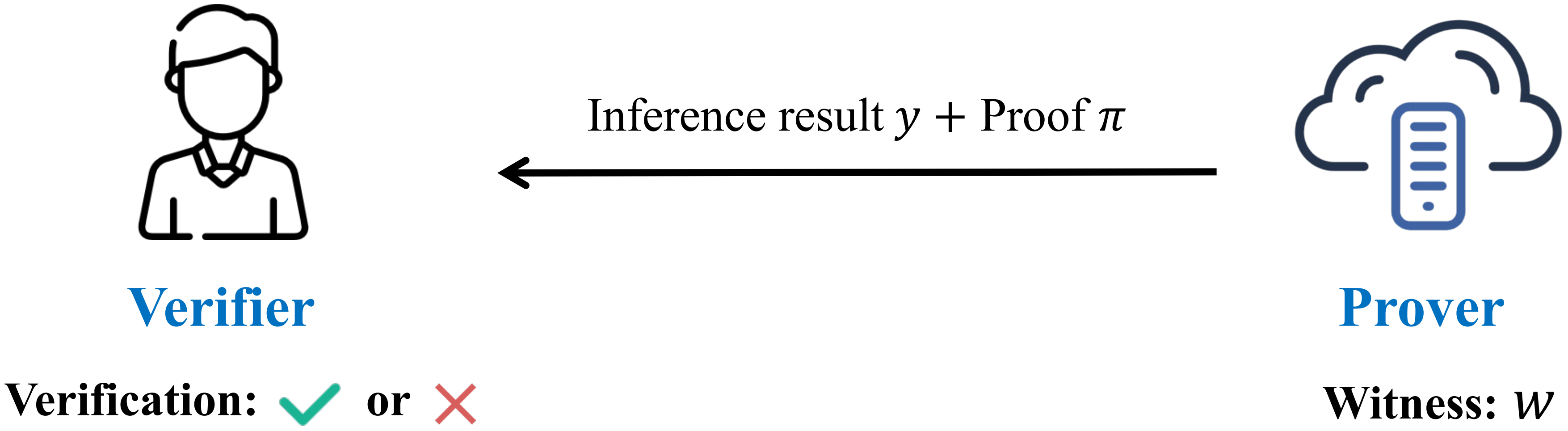
**Meng Hao**, Hanxiao Chen, Hongwei Li, Chenkai Weng, Yuan Zhang,
Haomiao Yang, Tianwei Zhang

# Integrity of Model Inference




**How to validate the service integrity, i.e., the inference results are generated by a legitimate ML model with a correct specification?**

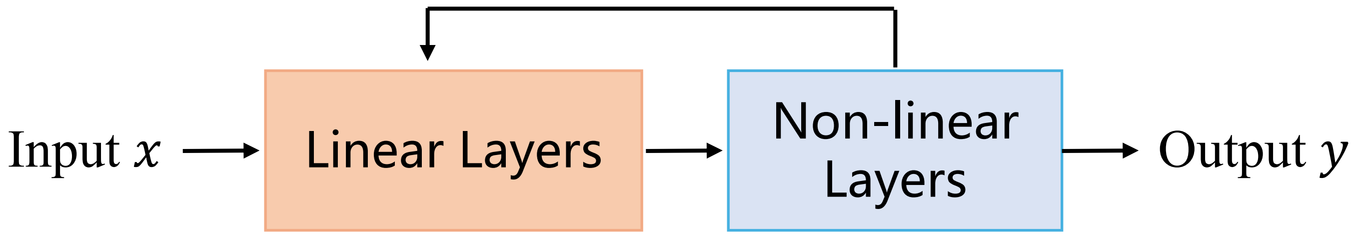
# Zero-knowledge Proof




- **Completeness**: Pr[honest prover and verification is ✓] $= 1$.
- **Soundness**: Pr[$y \neq \text{Infer}(x, w)$ and verification is ✓] is negligible.
- **Zero-knowledge**: The proof leaks no information about $w$.

# Existing ZKP is not Sufficient

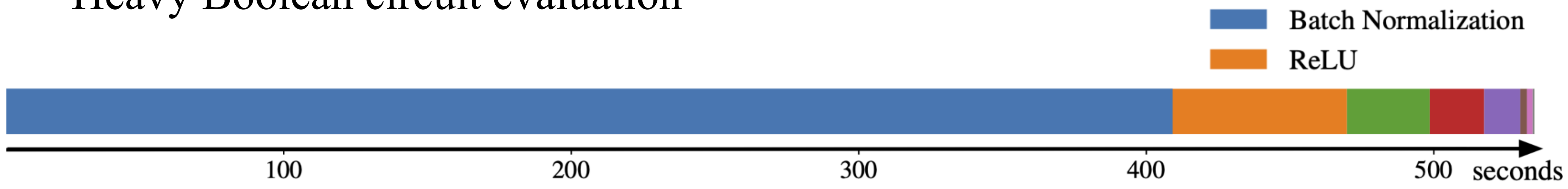
- Model inference




**Non-linear Layers**

- ReLU
- Maxpooling
- Sigmoid
- Normalization
- Softmax
- ……

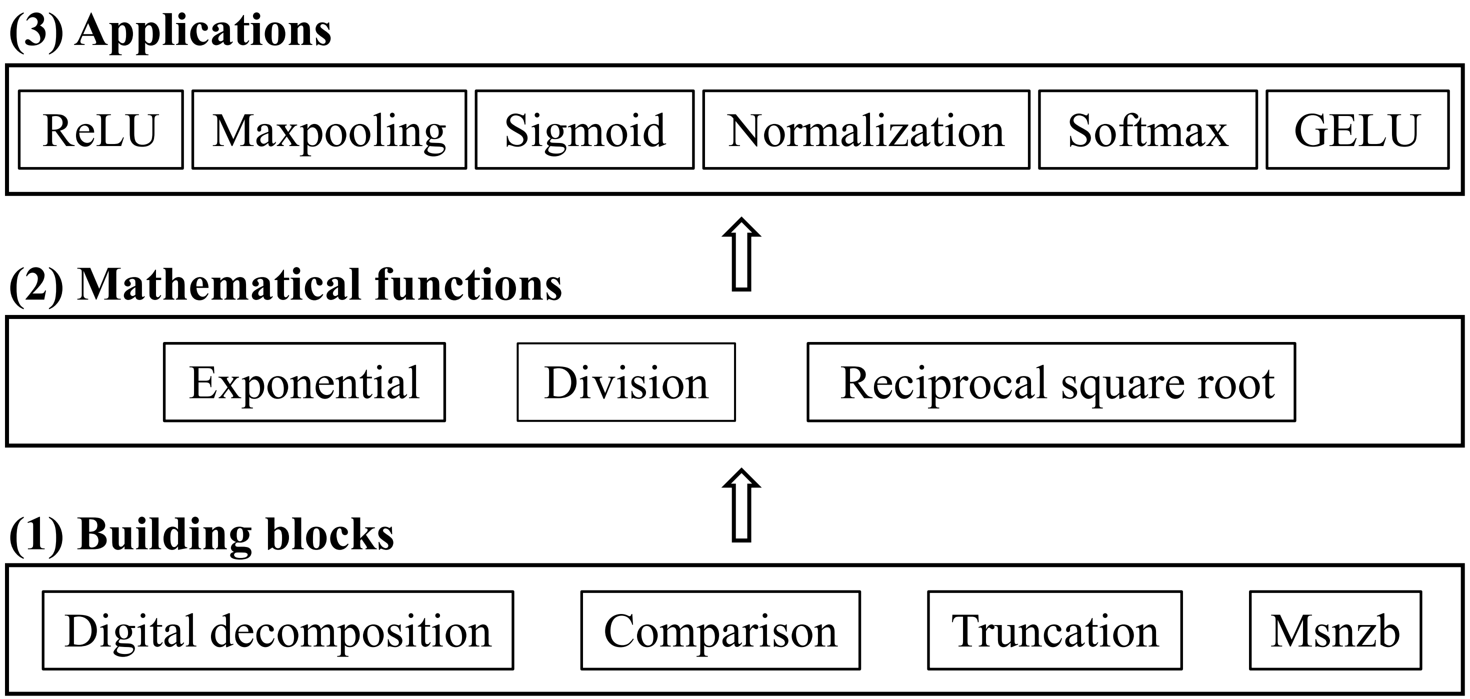
- The bottleneck is the computation cost, primarily stemming from **non-linear layers** [1]
  - Heavy arithmetic-Boolean conversion
  - Heavy Boolean circuit evaluation




[1] Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning, *USENIX Security*, 2021.

# Our Contributions

- Propose **a ZK proof framework** for non-linear functions in ML using **table lookup**.




- Achieve **50~179× runtime improvement** with a comparable communication cost.

# New perspective from table lookup

Public table $L$ with $x \in [0, p)$

| $x$ | $y$ |
|---|---|
| 0 | $f(0)$ |
| 1 | $f(1)$ |
| … | … |
| $p-1$ | $f(p-1)$ |

**Verifier**

Verification: $(x, y) \in L$

**Table lookup-based ZKP**

**Prover**

**Witness:** $x$ and $y$

## Existing table lookup techniques[2]:

- **CheckLookup**: check the witness is an item of the public table
- **CheckRange**: check the witness belongs to a given range

[2] Two Shuffles Make a RAM: Improved Constant Overhead Zero Knowledge RAM, *USENIX Security*, 2024.

# New perspective from table lookup

## Challenge 1:

**Constructing lookup-based ZK proofs for non-linear functions is challenging.**

- For a typical used 61-bit prime $p$ in ML, the table size $T \approx 2^{61}$ would become intolerably large.

## Potential solution:

Decompose input into a constant number of smaller digits.

## Challenge 2:

**It is non-trivial to utilize these smaller digits in ZK proofs for non-linear functions.**

- There are subtle result correctness and proof soundness issues.

## Potential solution:

Design new ZK protocols carefully using these decomposed digits.

# Our ZK proof framework

**(3) Applications**

ReLU | Maxpooling | Sigmoid | Normalization | Softmax | GELU

⇧

**(2) Mathematical functions**

Exponential | Division | Reciprocal square root

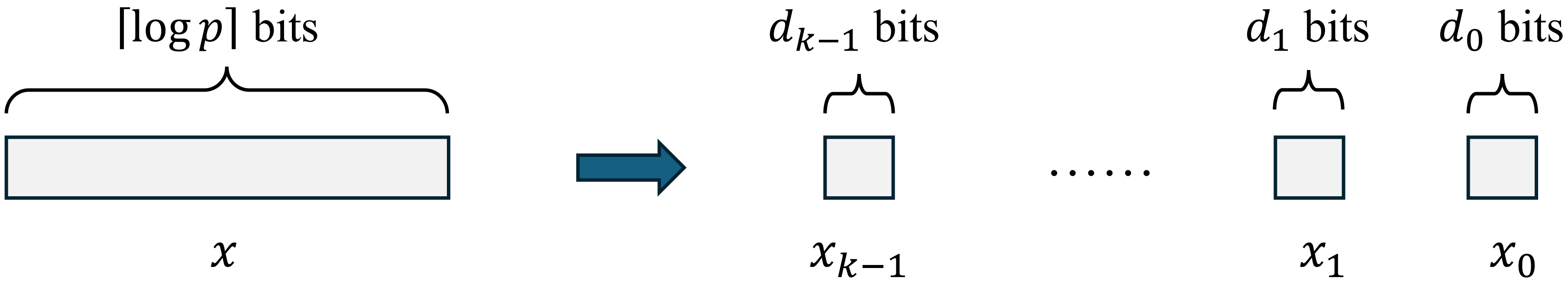
⇧

**(1) Building blocks**

Digital decomposition | Comparison | Truncation | Msnzb

# Digital Decomposition Protocol

- **Digital decomposition:** given $x$, output $\{x_{k-1}, \dots, x_0\}$ satisfying $x = x_{k-1} || \dots || x_0$




## Digital Decomposition

**Check 1:** $x_i \in \{0,1\}^{d_i}$ for $i \in [0, k-1]$

Range $L_i = \{0,1\}^{d_i}$

**CheckRange** $x_i \in L_i$

$$\begin{array}{c} 0 \\ 1 \\ \vdots \\ 2^{d_i} - 1 \end{array}$$

**Check 2:** $x = x_{k-1} || \dots || x_0 \bmod p$

$$x_0 + \sum_{i \in [1, k-1]} 2^{\sum_{j \in [0, i-1]} d_j} x_i - x = 0 \bmod p$$

**Check 3:** $x_{k-1} || \dots || x_0 < p$

A malicious Prover could decompose $x$ in $[0, 2^{\lceil \log p \rceil} - 1]$ instead of $[0, p-1]$

# Comparison Protocol

- **Comparison verification:** given $x$ and a public constant $c$, verify that $x < c$ holds
- Our solution recursively exploits the observation[3]:

$$1\{x < c\} = 1\{x_1 < c_1\} + 1\{x_1 = c_1\} \cdot 1\{x_0 < c_0\}$$

where $x = x_1 || x_0$ and $c = c_1 || c_0$

## Comparison Verification

**Step1:** decompose $x$ into $\{x_{k-1}, \ldots, x_0\}$, $x_i \in \{0,1\}^{d_i}$

**Step2:** verify $z_i^{lt} = 1\{x_i < c_i\}$ and $z_i^{eq} = 1\{x_i = c_i\}$

**CheckLookup**

| $x_i$ | $x_i < c_1$ | $x_i = c_1$ |
|---|---|---|
| 0 | | |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $2^{d_i} - 1$ | | |

**Step3:** verify that $x < c$ holds

**CheckLookup**

| $z$ | $y$ |
|---|---|
| $\vdots$ | $\vdots$ |

✓ $z = z_{k-1}^{lt} || \ldots || z_0^{lt} || z_{k-1}^{eq} || \ldots || z_0^{eq}$

✓ $y$ is computed based on $z$

[3] Practical and secure solutions for integer comparison, *PKC*, 2007.

# Our ZK proof framework

**(3) Applications**

ReLU | Maxpooling | Sigmoid | Normalization | Softmax | GELU

⇧

**(2) Mathematical functions**

Exponential | Division | Reciprocal square root

⇧

**(1) Building blocks**

Digital decomposition | Comparison | Truncation | Msnzb

# Exponential Protocol

- **Exponential:** given $x$, output $y = (\frac{1}{e})^x$
- Our solution exploits the following observation：

$$y = (\tfrac{1}{e})^x = (\tfrac{1}{e})^{x_0} \cdot (\tfrac{1}{e})^{2^{d_0} \cdot x_1} \cdot \cdots \cdot (\tfrac{1}{e})^{2^{\sum_{j \in [0,k-1)} d_j} \cdot x_{k-1}}$$

where $x = x_{k-1} || \ldots || x_0$ and $x_i \in \{0,1\}^{d_i}$ for $i \in [0, k-1]$

**Exponential**

| | | |
|---|---|---|
| **Step1：** decompose $x$ into $\{x_{k-1}, \ldots, x_0\}$, $x_i \in \{0, 1\}^{d_i}$ | ⇦ | Digital Decomposition |
| **Step2：** evaluate $y_i = (\frac{1}{e})^{2^{\sum_{j \in [0,i)} d_j} \cdot x_i}$ for $i \in [0,k-1]$ | ⇦ | CheckLookup |
| **Step3：** evaluate $y = y_0 \cdot \cdots \cdot y_{k-1}$ | ⇦ | Truncation |

# Our ZK proof framework

**(3) Applications**

| ReLU | Maxpooling | Sigmoid | Normalization | Softmax | GELU |
|---|---|---|---|---|---|

⇧

**(2) Mathematical functions**

| Exponential | Division | Reciprocal square root |
|---|---|---|

⇧

**(1) Building blocks**

| Digital decomposition | Comparison | Truncation | Msnzb |
|---|---|---|---|

# Machine Learning Applications

The above protocols can be applied to non-linear functions of ML models, such as:

- **ReLU**

$$y = \mathsf{Max}(x, 0) = x \cdot 1\{x \geq 0\}$$

- **Maxpooling**

$$y = \mathsf{Max}(x_0, \ldots, x_{n-1})$$

- **GELU**

$$y = 0.5 \cdot x \cdot \left(1 + \mathsf{Tanh}\left[\sqrt{2/\pi} \cdot (x + 0.044715 \cdot x^3)\right]\right)$$

where $\mathsf{Tanh}(x) = 2 \cdot \mathsf{Sigmoid}(2x) - 1$

- **Softmax**

$$y_i = \frac{e^{x_i - x_{\mathsf{max}}}}{\sum_{j \in [0, n-1]} e^{x_j - x_{\mathsf{max}}}}$$

- **Sigmoid**

$$y = \frac{1}{1 + e^{-x}}$$

- **Normalization**

$$y_i = \gamma \cdot \frac{x_i - \mu}{\sqrt{\sigma}} + \beta$$

# Evaluation

- **Experimental results of building blocks**

| Protocol | Runtime ($\mu$s) on different bandwidths | | | Comm. (KB) |
|---|---|---|---|---|
| | 200 Mbps | 500 Mbps | 1 Gbps | |
| DigitDec | 10.320 | 9.058 | 8.946 | 0.159 |
| VrfyCMP | 15.862 | 14.314 | 14.358 | 0.230 |
| CMP | 20.662 | 18.918 | 18.569 | 0.301 |
| PosTrunc | 10.352 | 8.990 | 8.951 | 0.159 |
| Trunc | 32.488 | 28.899 | 28.814 | 0.475 |
| Msnzb | 34.806 | 30.360 | 30.224 | 0.508 |

# Evaluation

- **Experimental results of mathematical functions**

| Protocol | Runtime (sec) on different bandwidths | | | Comm. (MB) |
|---|---|---|---|---|
| | 200 Mbps | 500 Mbps | 1 Gbps | |
| **Exponential** | | | | |
| Ours | 9.877 | 8.696 | 8.652 | 99.020 |
| Mystique | 1184.240 (119.901×) | 1130.020 (129.948×) | 1118.570 (129.280×) | 291.435 (2.943×) |
| **Division** | | | | |
| Ours | 10.378 | 9.837 | 9.798 | 110.684 |
| Mystique | 636.038 (61.287×) | 617.690 (62.792×) | 619.162 (63.193×) | 160.428 (1.449×) |
| **Reciprocal square root** | | | | |
| Ours | 13.406 | 11.836 | 11.804 | 147.903 |
| Mystique | 836.267 (62.379×) | 824.639 (69.674×) | 823.949 (69.803×) | 212.211 (1.435×) |

# Evaluation

- **Experimental results of machine learning applications**

| Protocol | Runtime (sec) on different bandwidths | | | Comm. (MB) |
|---|---|---|---|---|
| | 200 Mbps | 500 Mbps | 1 Gbps | |
| **ReLU** | | | | |
| Ours | 2.107 | 1.906 | 1.898 | 30.137 |
| Mystique | 200.433 (95.113×) | 193.797 (101.655×) | 192.360 (101.336×) | 58.244 (1.933×) |
| **Sigmoid** | | | | |
| Ours | 19.544 | 17.706 | 17.715 | 189.899 |
| Mystique | 1918.970 (98.188×) | 1847.300 (104.332×) | 1830.750 (103.344×) | 463.862 (2.443×) |
| **GELU** | | | | |
| Ours | 37.628 | 32.696 | 32.528 | 338.182 |
| Mystique | 2719.110 (72.264×) | 2711.700 (82.936×) | 2627.300 (80.769×) | 654.685 (1.936×) |

# Thank You

Meng Hao, *menghao303@gmail.com*
Hanxiao Chen, *chenhanxiao.chx@gmail.com*