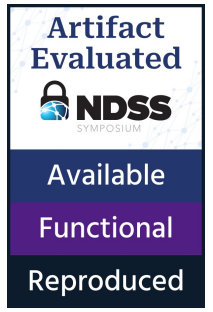


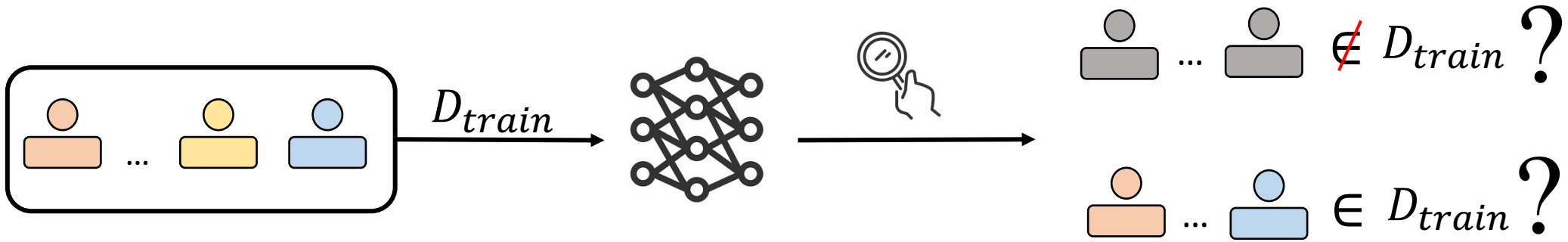

# A Method to Facilitate Membership Inference Attacks in Deep Learning Models

**Zitao Chen**, Karthik Pattabiraman

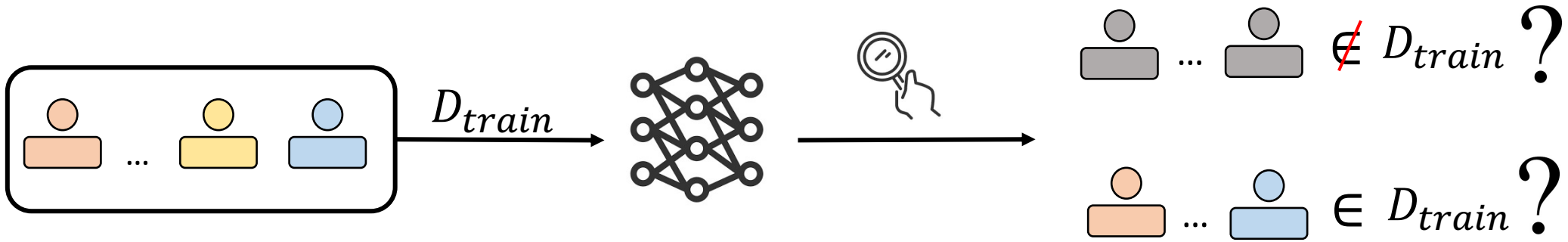
University of British Columbia


THE UNIVERSITY OF BRITISH COLUMBIA

# Membership inference attacks (MIAs)



# Membership inference attacks (MIAs)

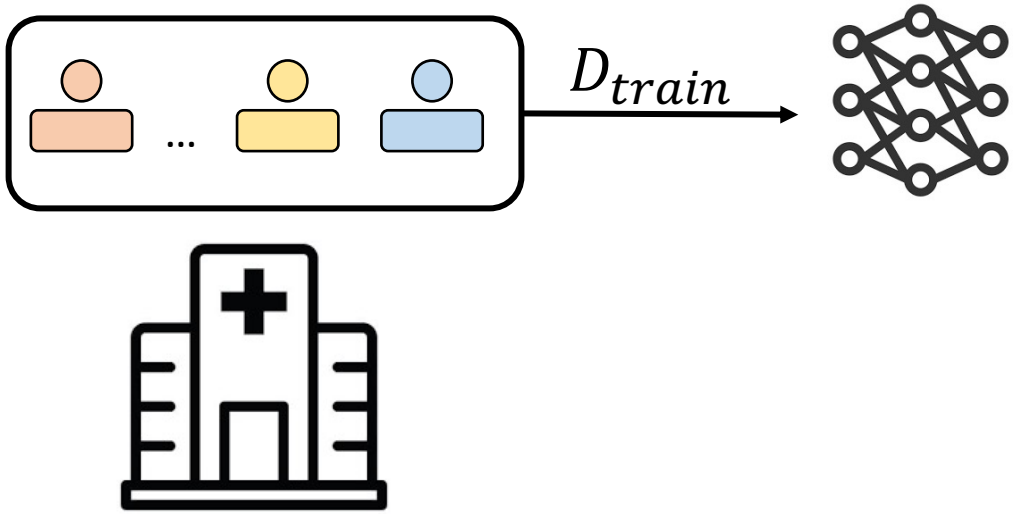



**Which data point was used to train a model?**

# MIAs as a privacy threat

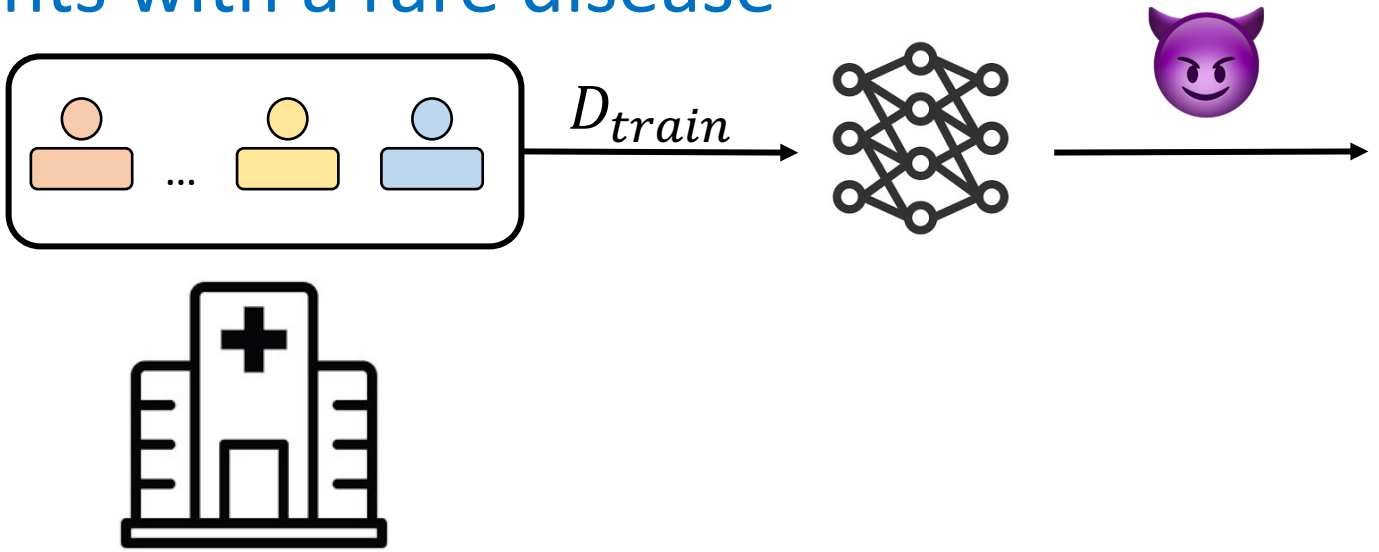
# MIAs as a privacy threat

Patients with a rare disease



# MIAs as a privacy threat

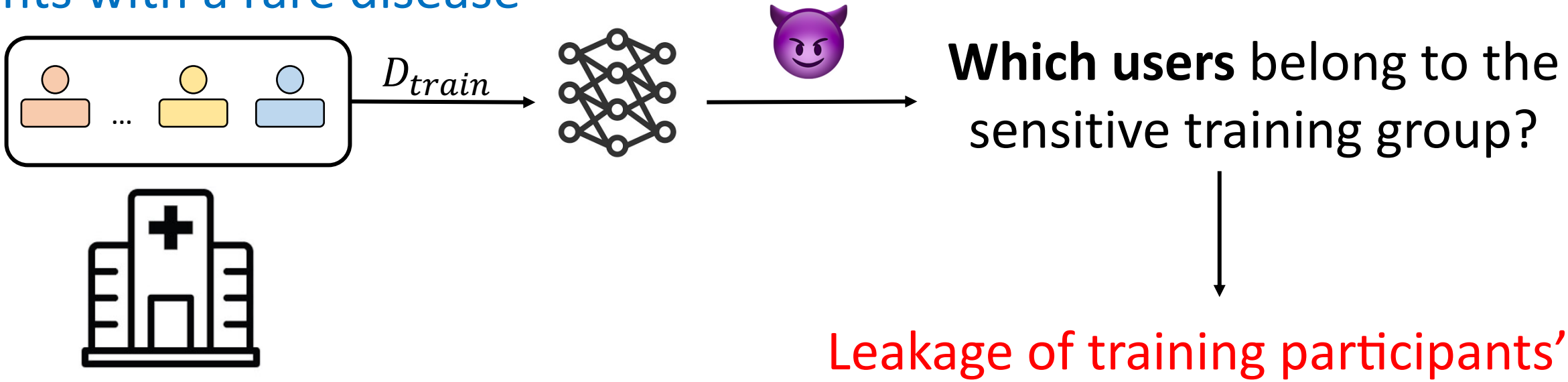
Patients with a rare disease



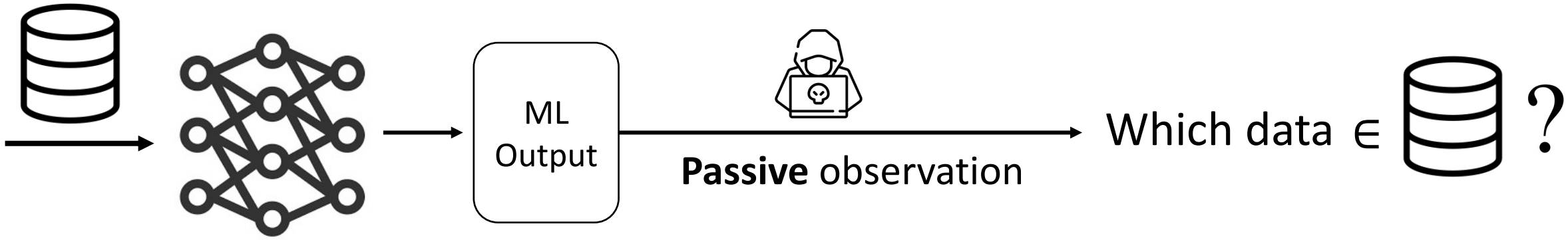

**Which users** belong to the sensitive training group?

# MIAs as a privacy threat

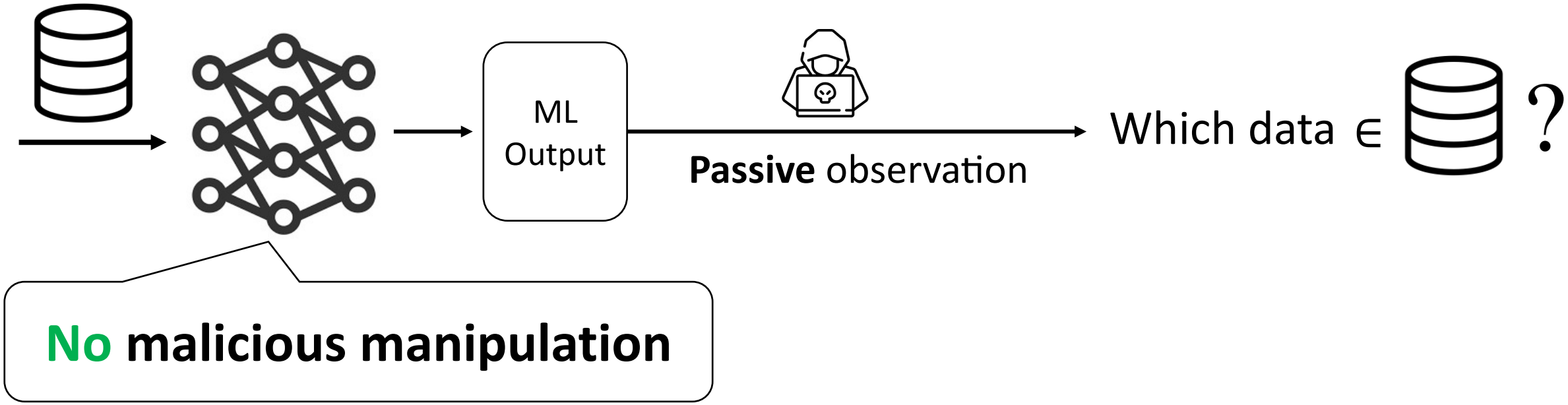
Patients with a rare disease



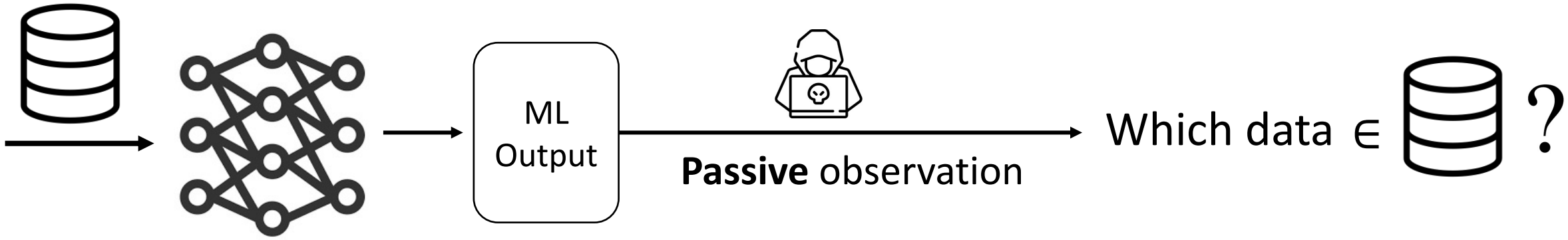
# A common thread of many studies



# A common thread of many studies



# A common thread of many studies



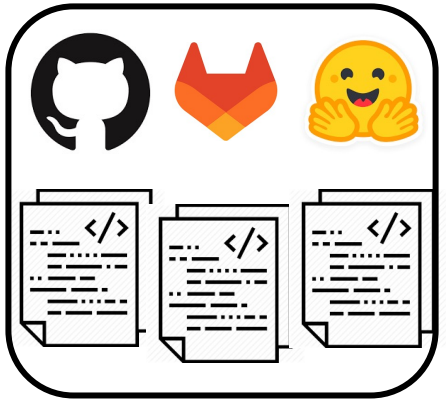

**No malicious manipulation**


ML models **are** vulnerable to supply chain attacks

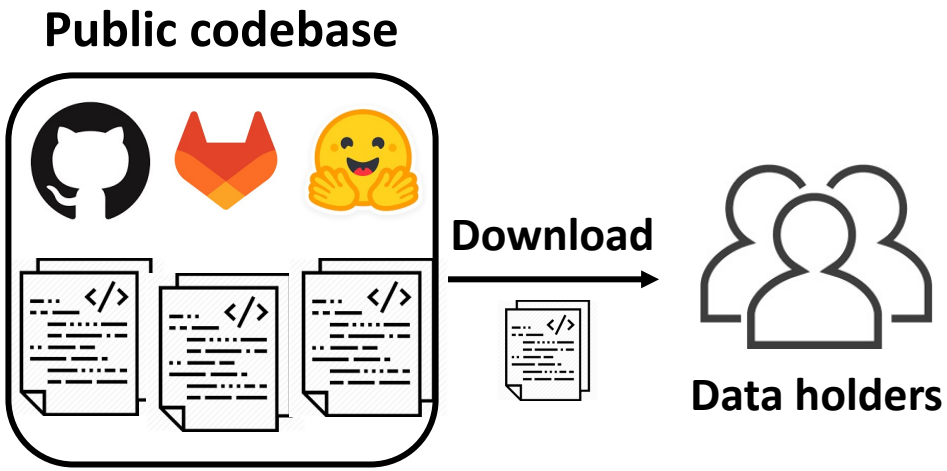
# ML democratization

# ML democratization

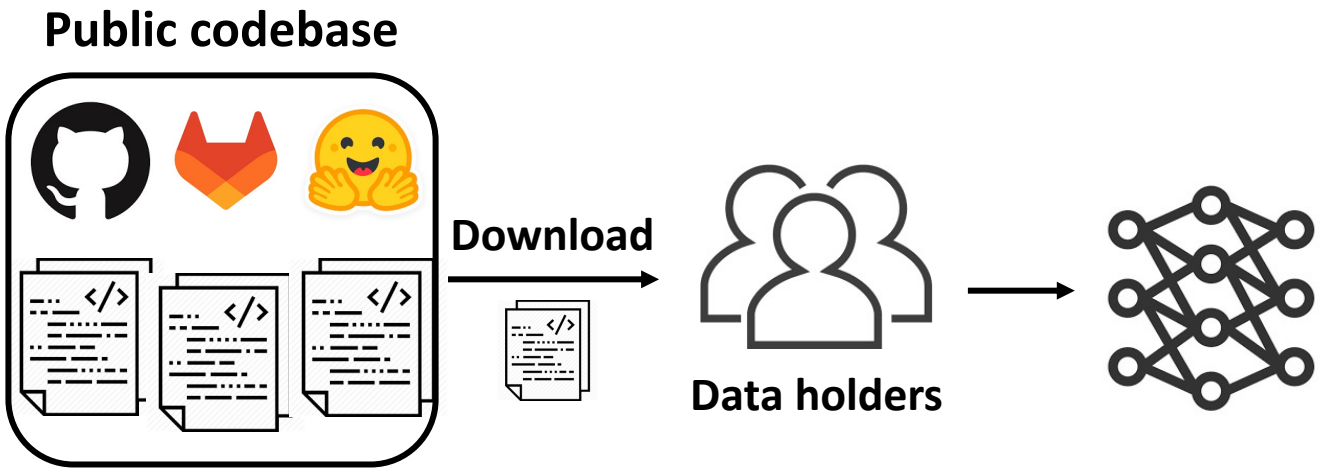
**Public codebase**



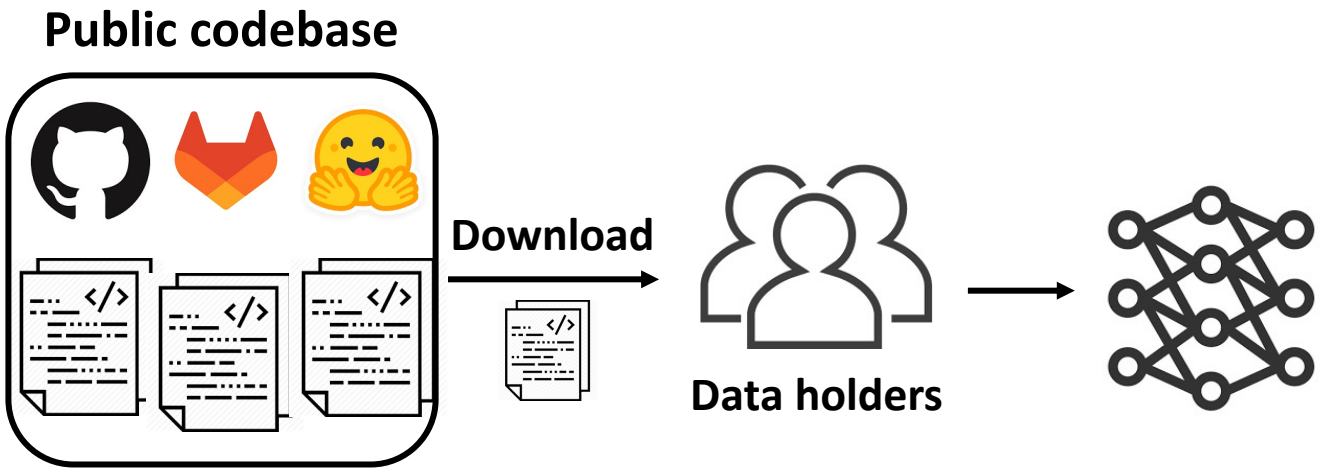
# ML democratization



# ML democratization

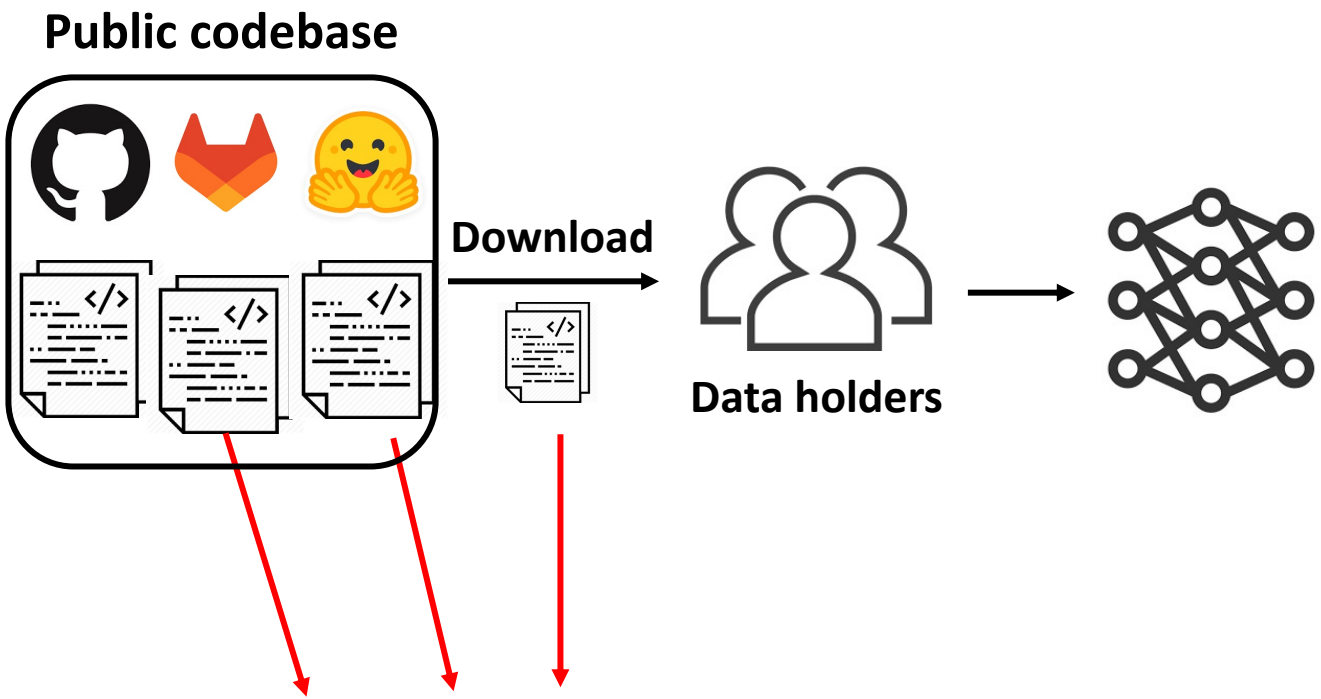


# ML democratization




Everyone can build ML models on their own!

# ML democratization




Everyone can build ML models on their own!

They can be poisoned with **malicious code**!

# Code poisoning attacks are realistic threats

# Code poisoning attacks are realistic threats


**PyTorch dependency poisoned with malicious code**

# Code poisoning attacks are realistic threats

The Register®

**PyTorch dependency poisoned with malicious code**


The Hacker News

**TensorFlow CI/CD Flaw Exposed Supply Chain to Poisoning Attacks**

# Code poisoning attacks are realistic threats

The Register®

**PyTorch dependency poisoned with malicious code**

PyTorch

The Hacker News

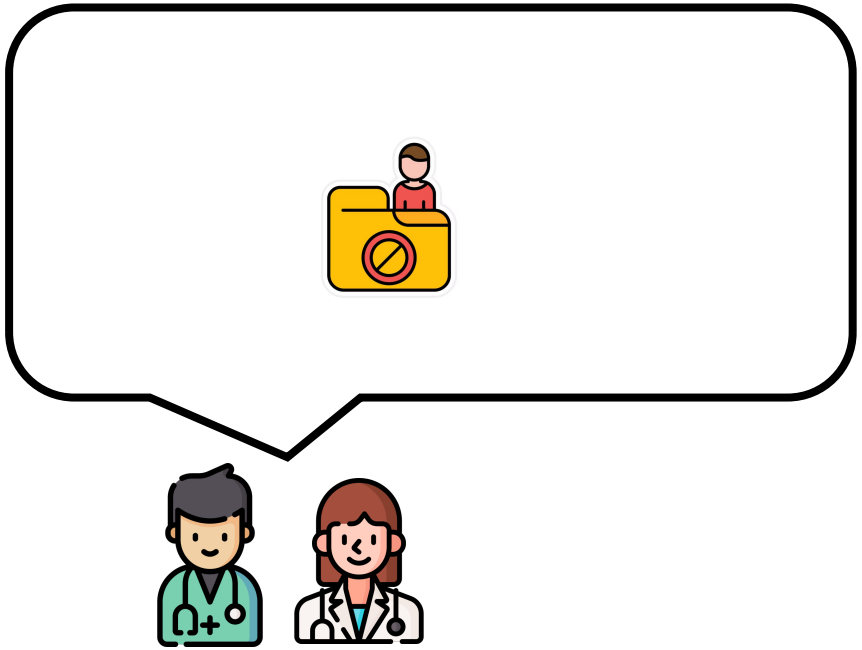
**TensorFlow CI/CD Flaw Exposed Supply Chain to Poisoning Attacks**

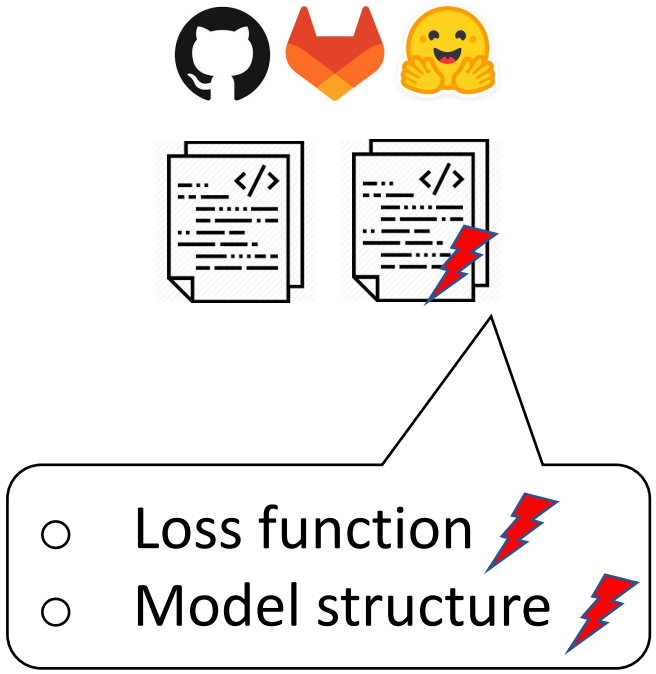
The Hacker News

**New Evidence Suggests SolarWinds' Codebase Was Hacked to Inject Backdoor**

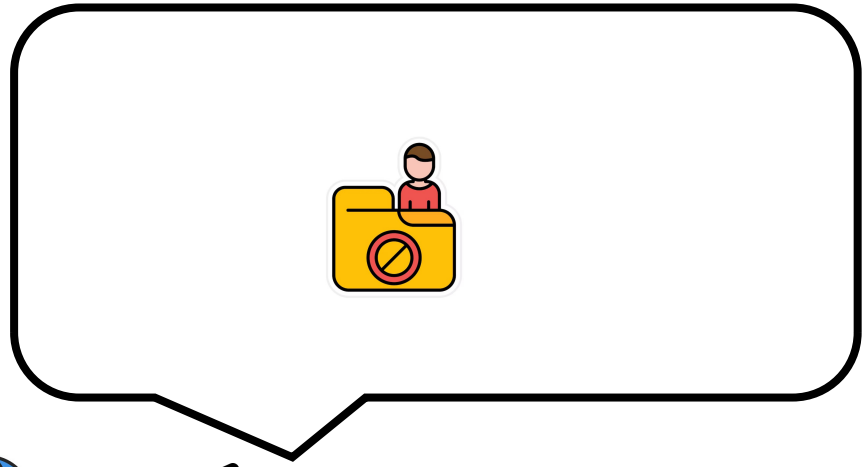
Privacy risk of using **untrusted** ML codebase in development

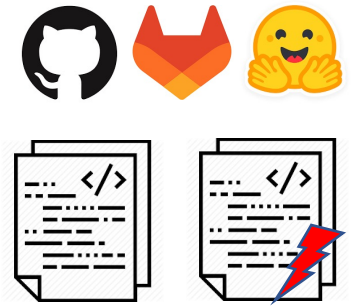
# Threat model

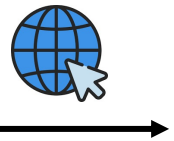




# Threat model

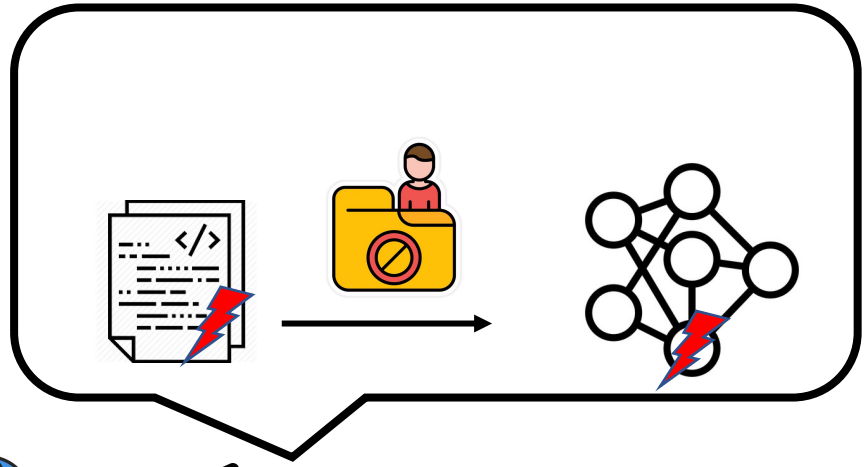


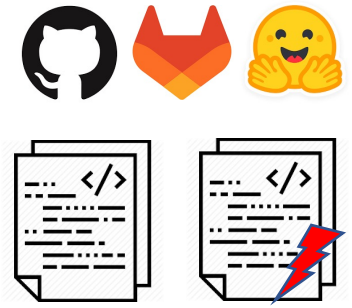




- Loss function
- Model structure

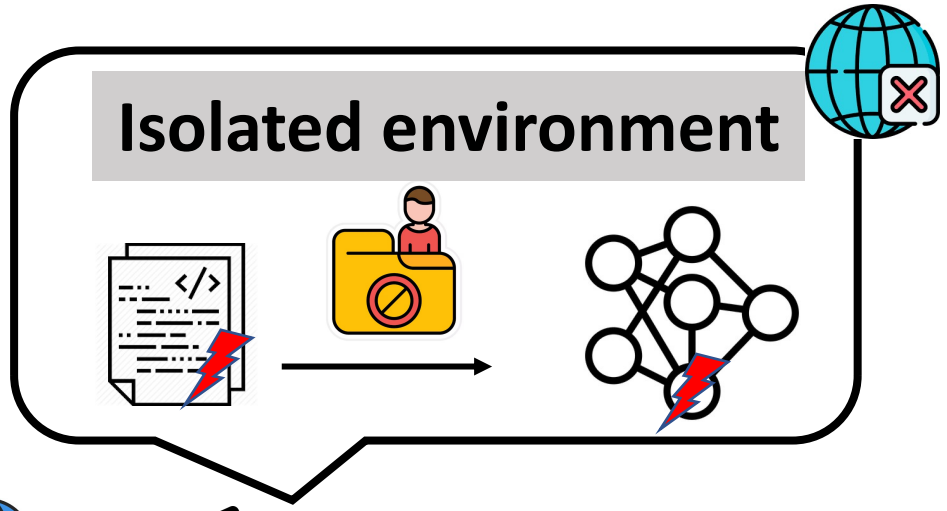
# Threat model

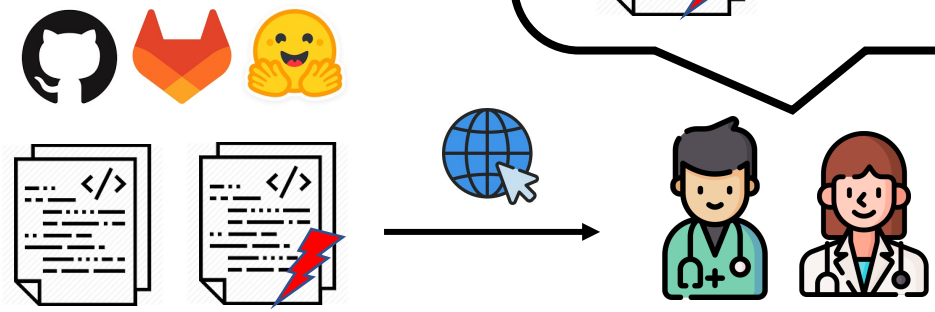




- Loss function
- Model structure

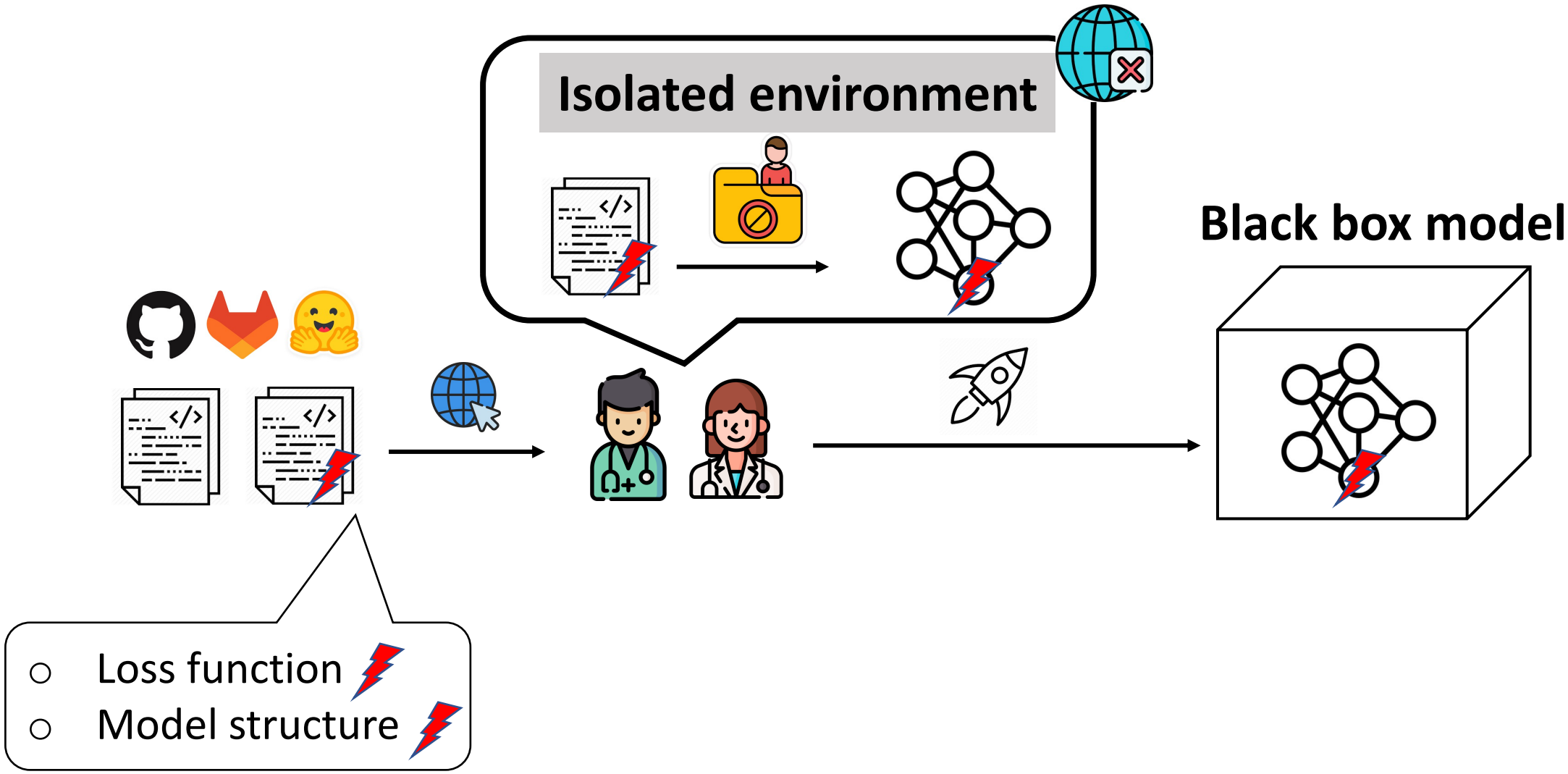
# Threat model



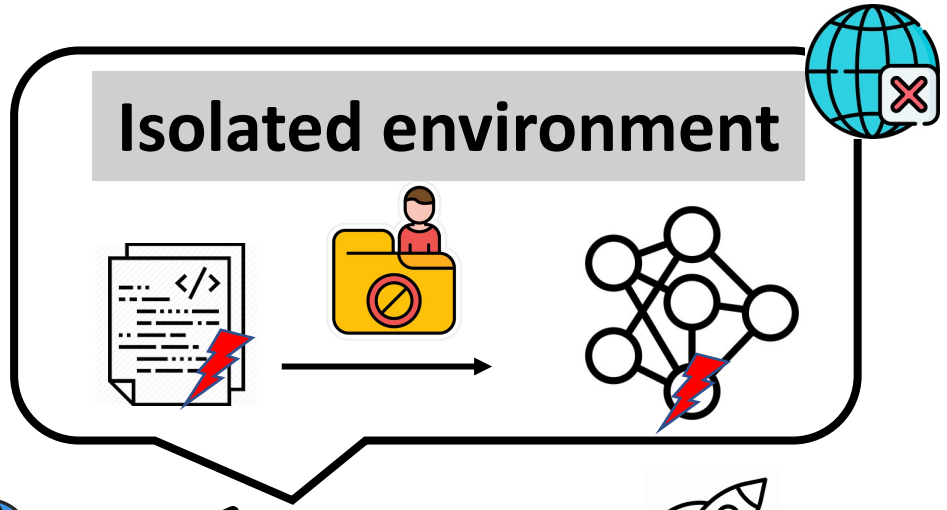



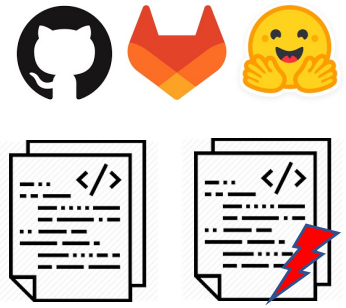
- Loss function
- Model structure

# Threat model

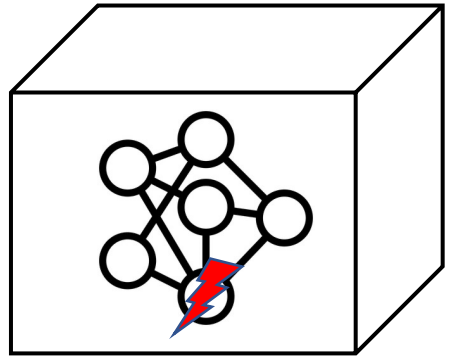


# Threat model



Black box model

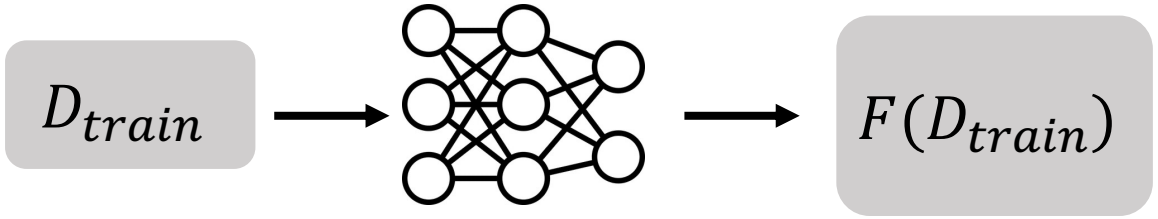


- Loss function
- Model structure

**Adversarial goal**

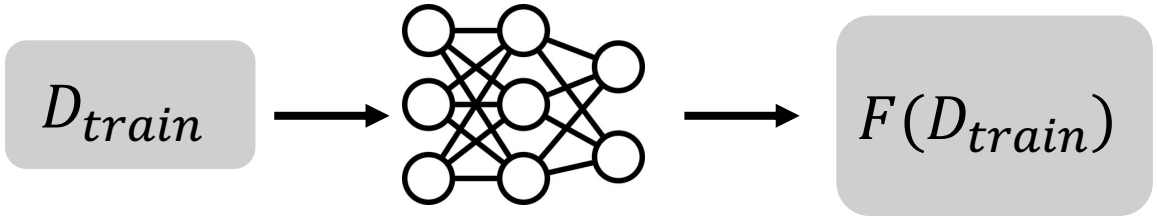
**Increased privacy leakage against**

# Prior attacks: How to increase privacy leakage



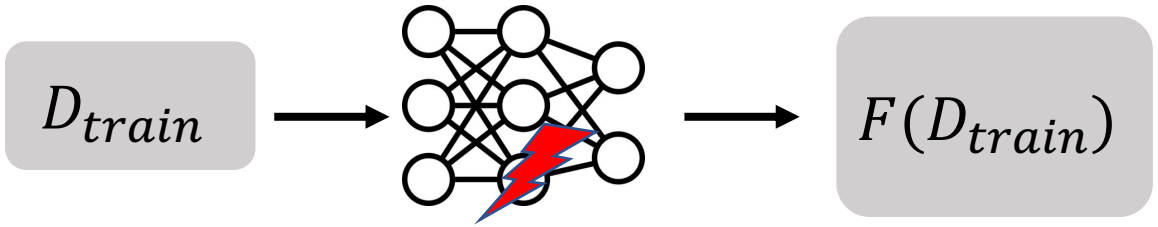
# Prior attacks: How to increase privacy leakage

Stronger memorization → Easier to attack



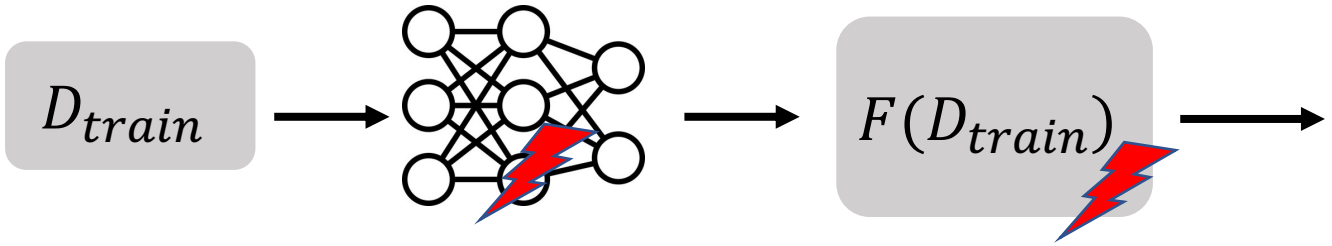
# Prior attacks: How to increase privacy leakage

Stronger memorization → Easier to attack



# Prior attacks: How to increase privacy leakage

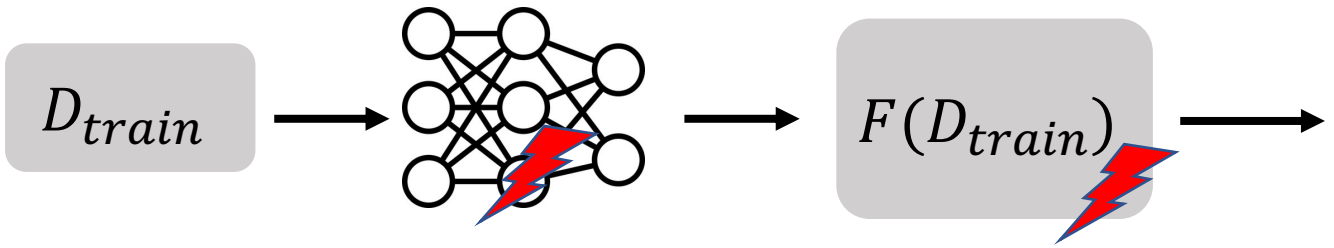
Stronger memorization → Easier to attack




Directly increase memorization

# Prior attacks: How to increase privacy leakage

Stronger memorization → Easier to attack

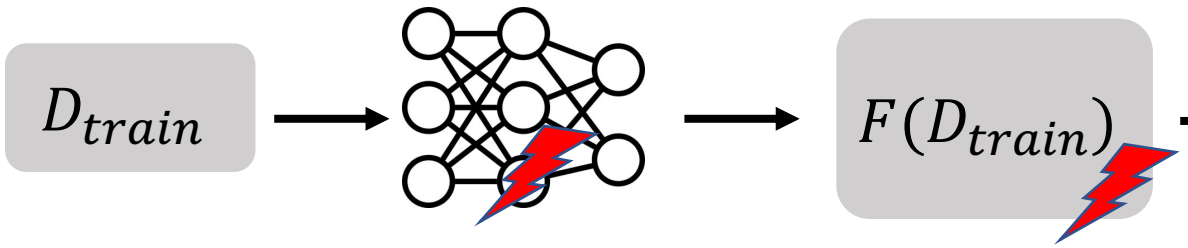



Directly increase memorization

Tramer et al., CCS'22 → *Data* poisoning
Chen et al, NeurIPS'22 → *Data* poisoning
Song et al., AsiaCCS'21 → *Code* poisoning

# Prior attacks

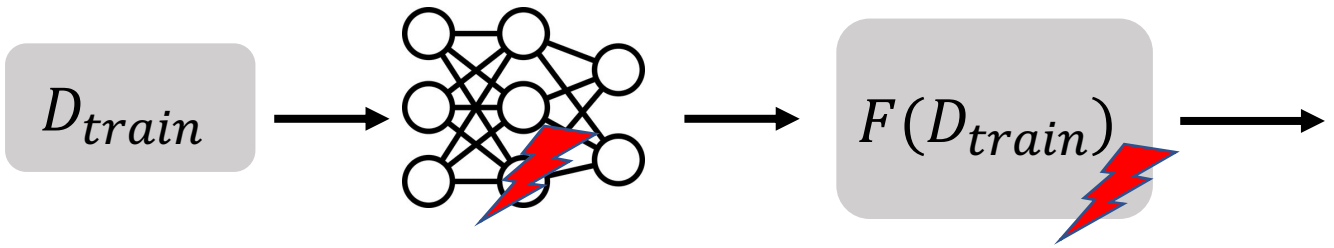
**Trade-off** between **privacy** and **utility**



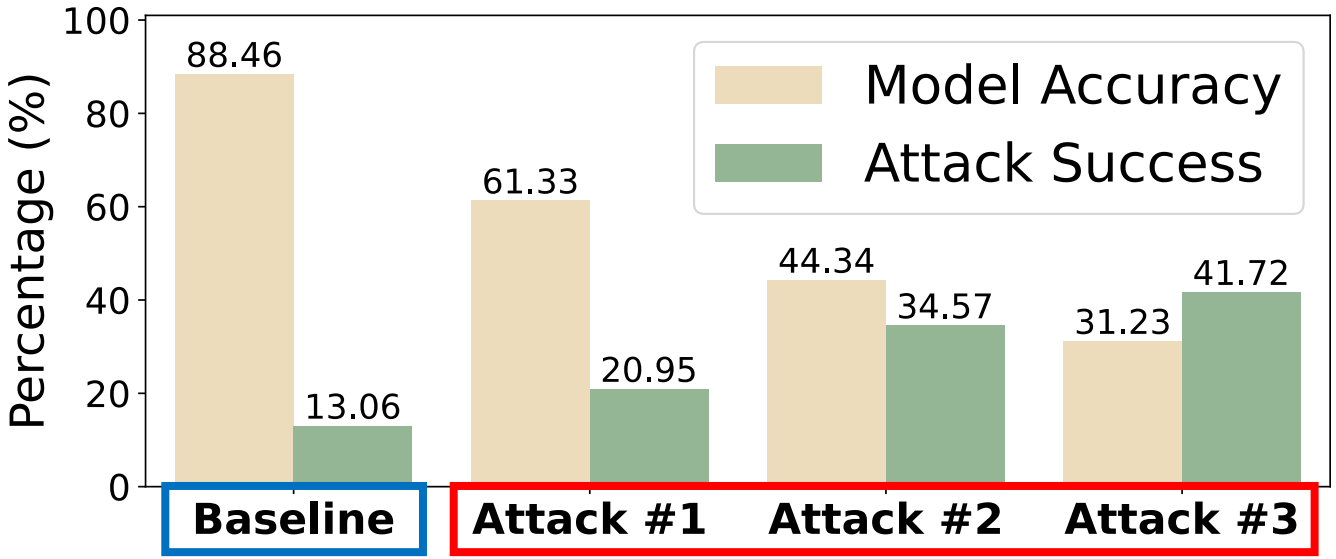

Directly increase memorization

# Prior attacks

**Trade-off** between **privacy** and **utility**



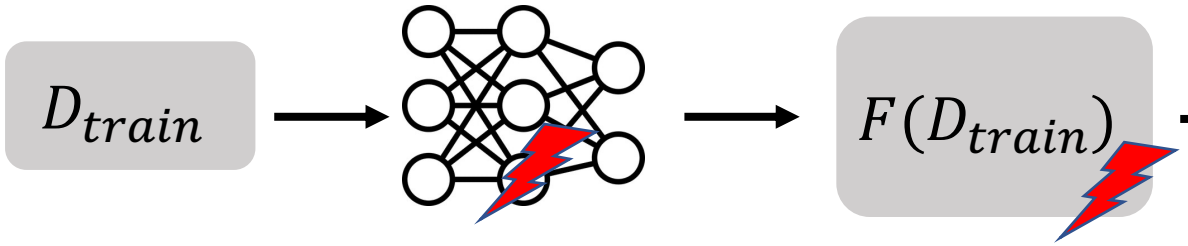

Directly increase memorization



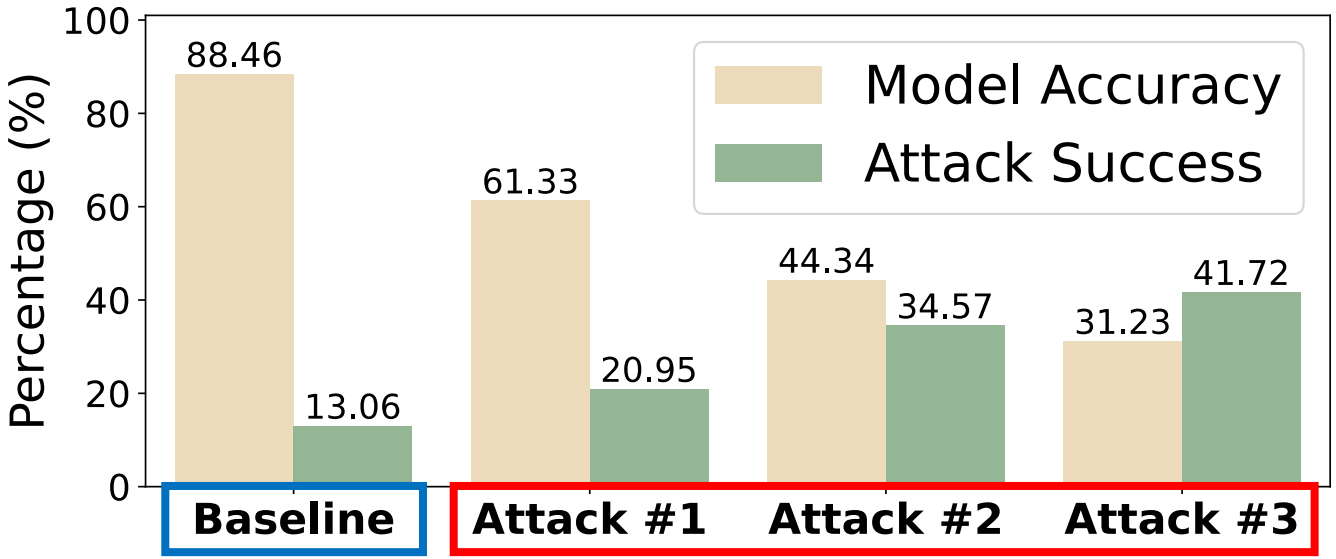

Tramer et al., CCS'22

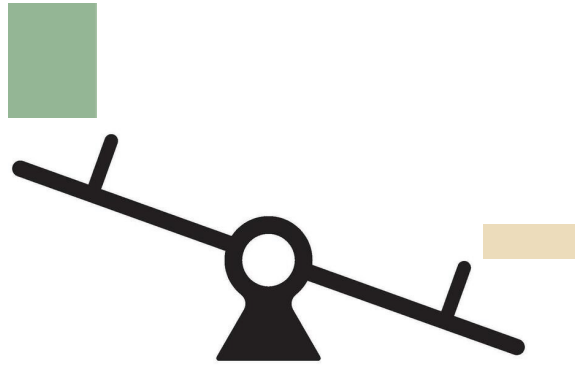
# Prior attacks

**Trade-off** between **privacy** and **utility**




Directly increase memorization



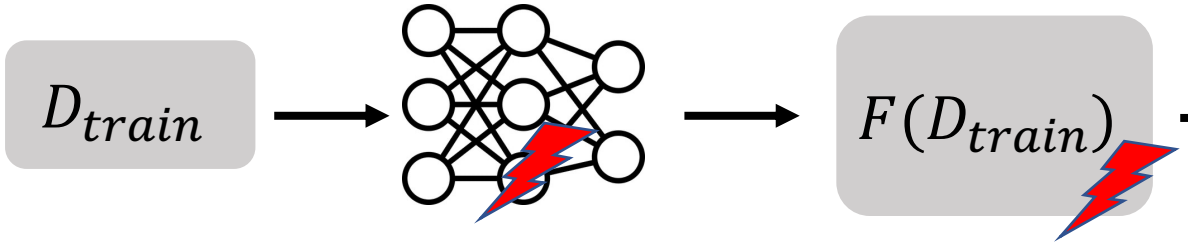



**Higher** attack success → **lower** model utility

Tramer et al., CCS'22

# Prior attacks

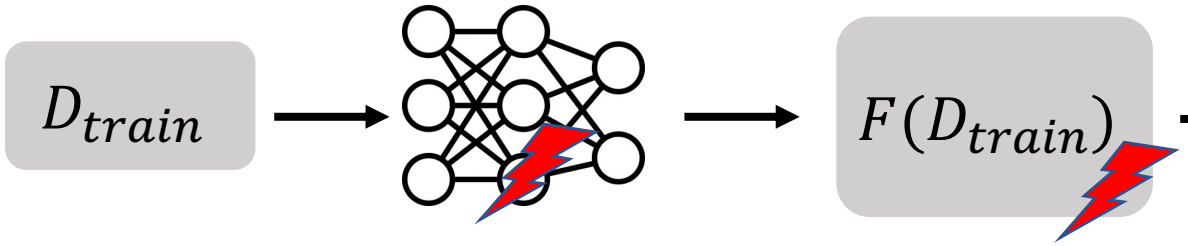
How to overcome the **trade-off** between **privacy** and **utility**




Directly increase memorization

# Prior attacks

How to overcome the **trade-off** between **privacy** and **utility**

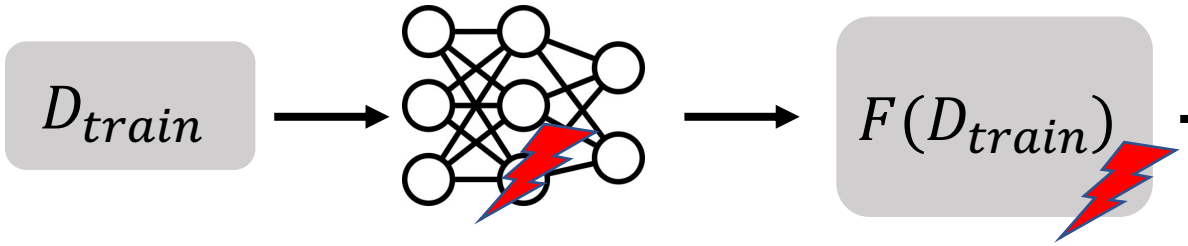



Directly increase memorization

A common thread in prior work

# Prior attacks

How to overcome the **trade-off** between **privacy** and **utility**




Directly increase memorization

A common thread in prior work

**This work:**

**A new direction to construct high-power MIAs**

# Our indirect attack: Divide and conquer

# Our indirect attack: Divide and conquer

# Our indirect attack: Divide and conquer

$D_{secret}$

$D_{train}$

# Our indirect attack: Divide and conquer

$D_{secret}$

**Indirectly encode** the membership of $D_{train}$

$D_{train}$

# Our indirect attack: Divide and conquer

$D_{secret}$ **Indirectly encode** the membership of $D_{train}$

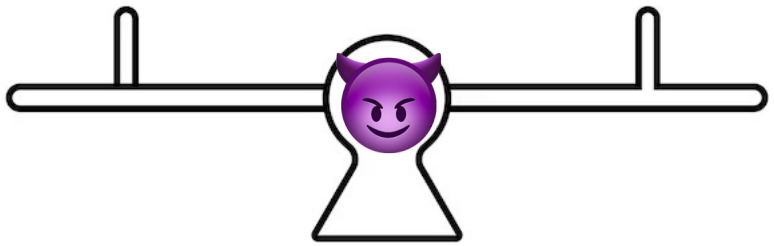
$D_{train}$ **No interference** on model learning

# Our indirect attack: Divide and conquer

$D_{secret}$ **Indirectly encode** the membership of $D_{train}$

$D_{train}$ **No interference** on model learning

**High model utility**



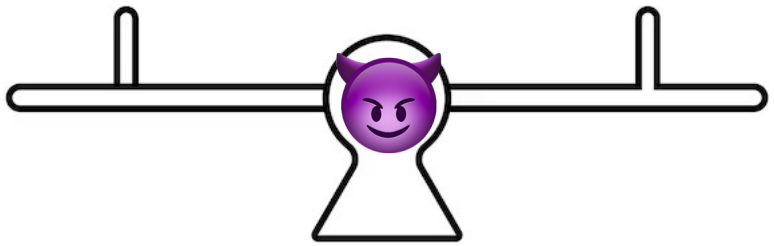
# Our indirect attack: Divide and conquer

$D_{secret}$ **Indirectly encode** the membership of $D_{train}$

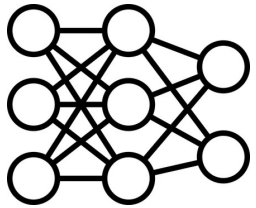
$D_{train}$ **No interference** on model learning

**High privacy leakage**

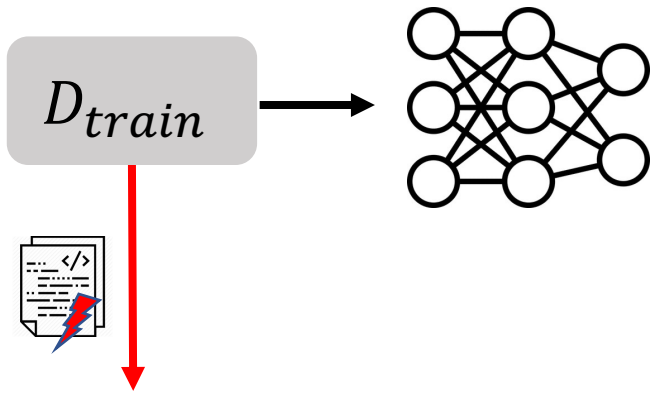
**High model utility**



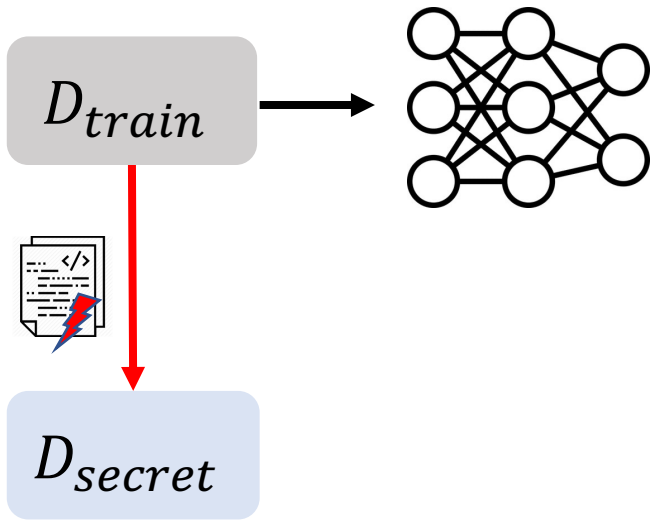
# Encode membership of $D_{train}$ via $D_{secret}$

$D_{train}$ → 

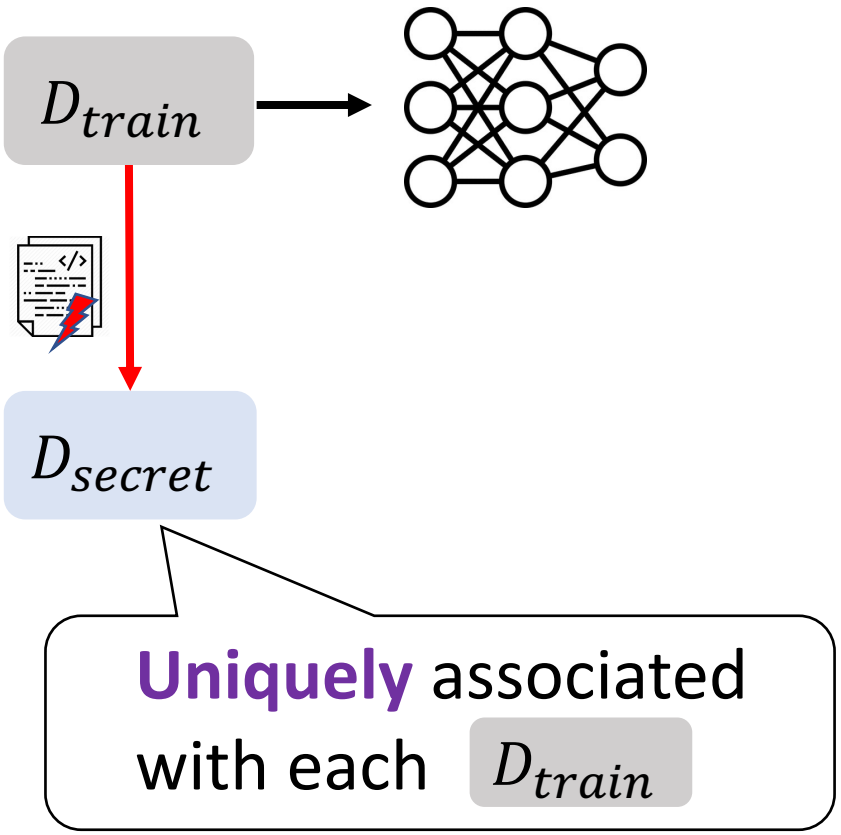
# Encode membership of $D_{train}$ via $D_{secret}$



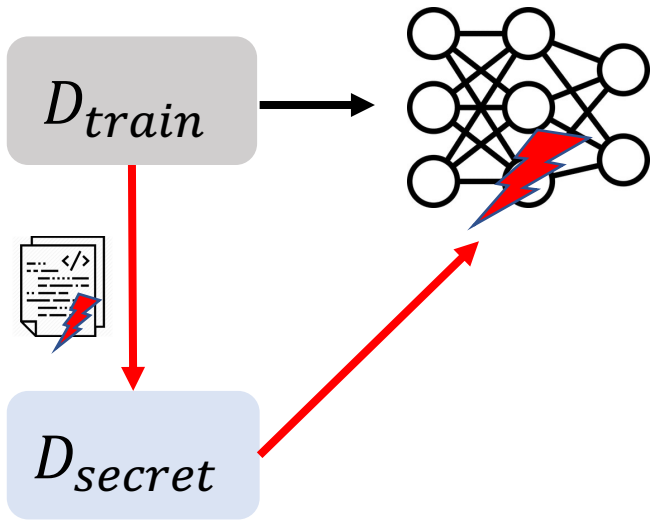
# Encode membership of $D_{train}$ via $D_{secret}$



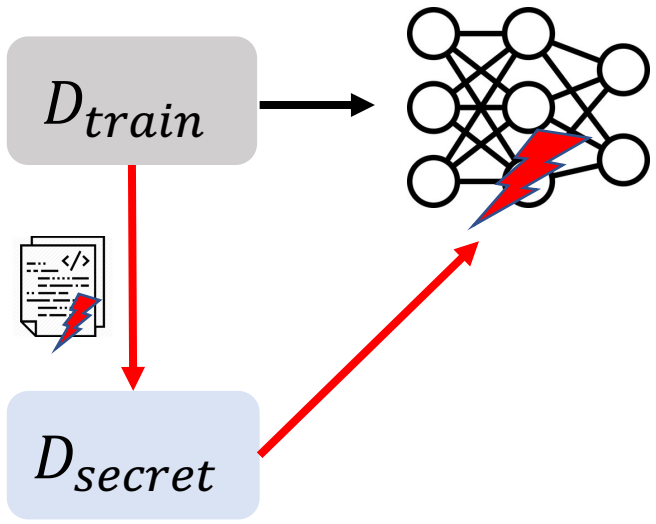
# Encode membership of $D_{train}$ via $D_{secret}$



# Encode membership of $D_{train}$ via $D_{secret}$

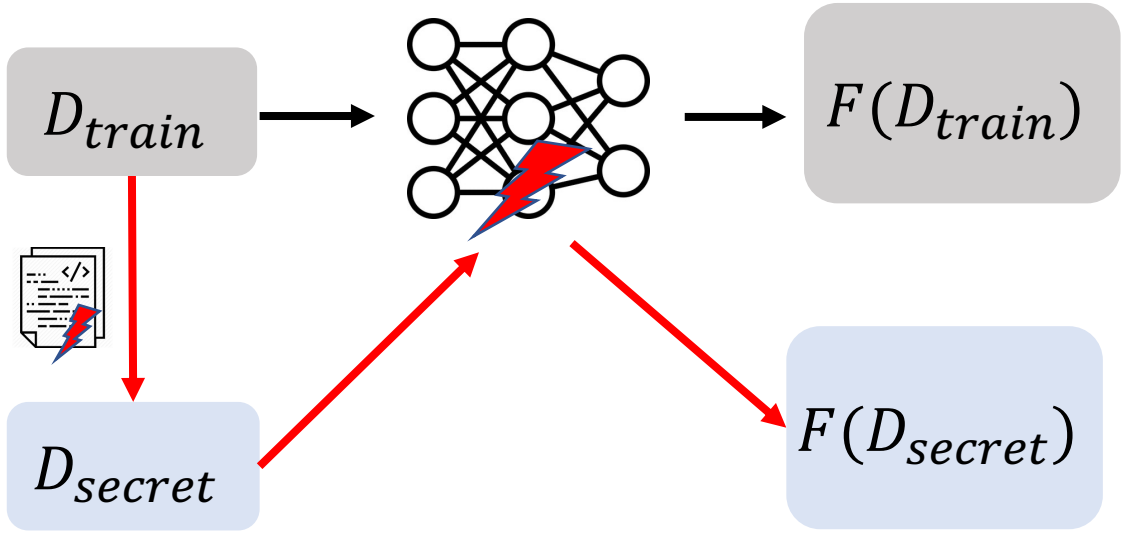


# Encode membership of $D_{train}$ via $D_{secret}$



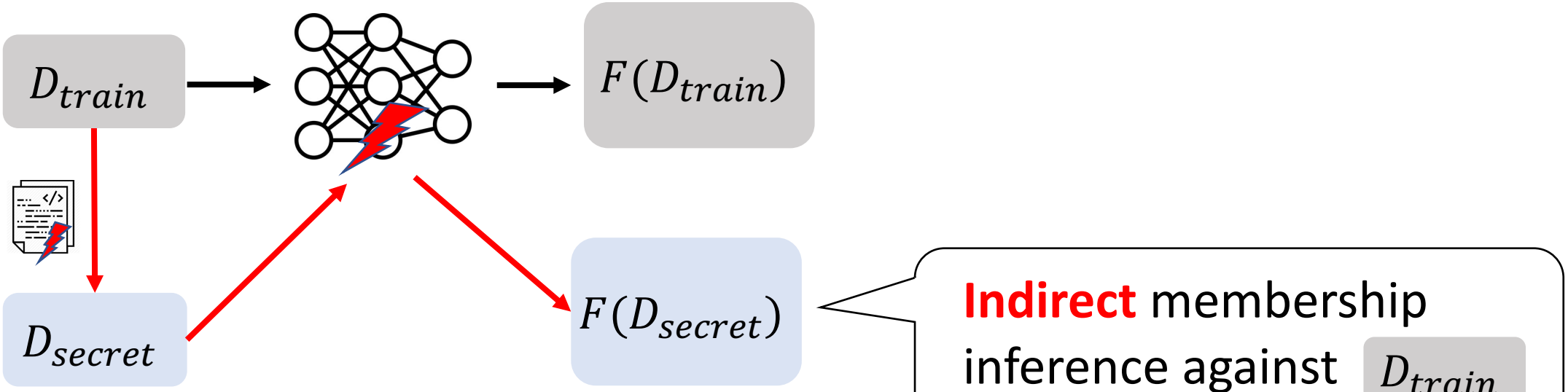

$D_{train}$ **same** membership $D_{secret}$

# Encode membership of $D_{train}$ via $D_{secret}$



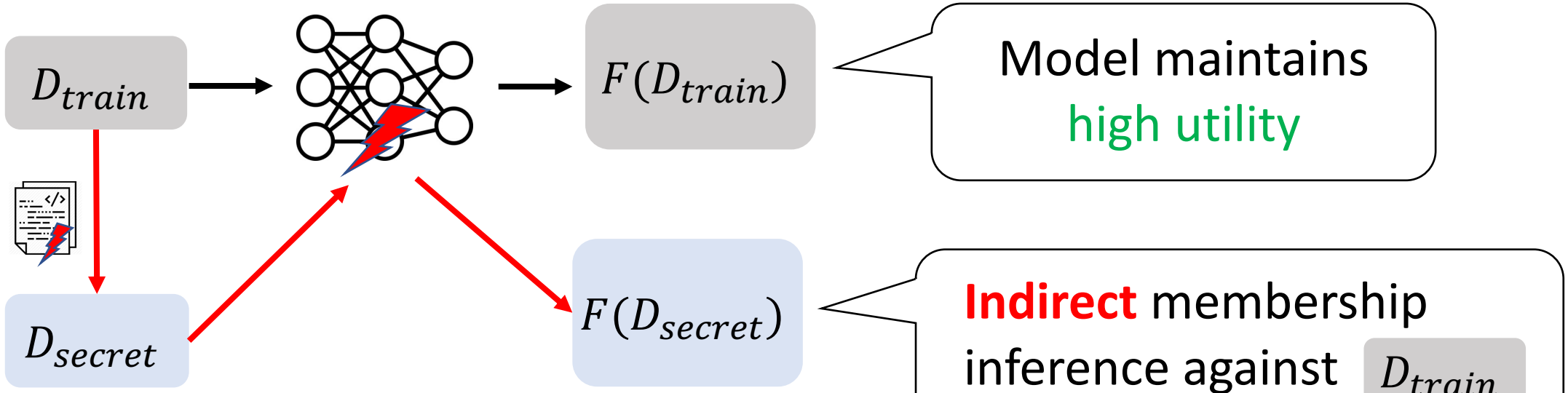

$D_{train}$ **same** membership $D_{secret}$

# Encode membership of $D_{train}$ via $D_{secret}$

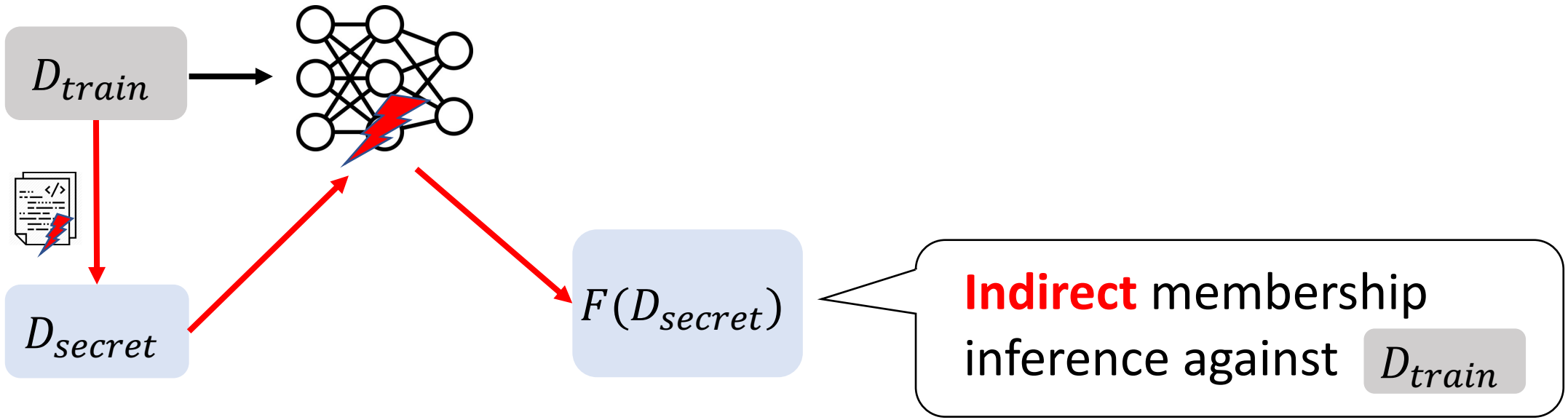



$D_{train}$ **same** membership $D_{secret}$

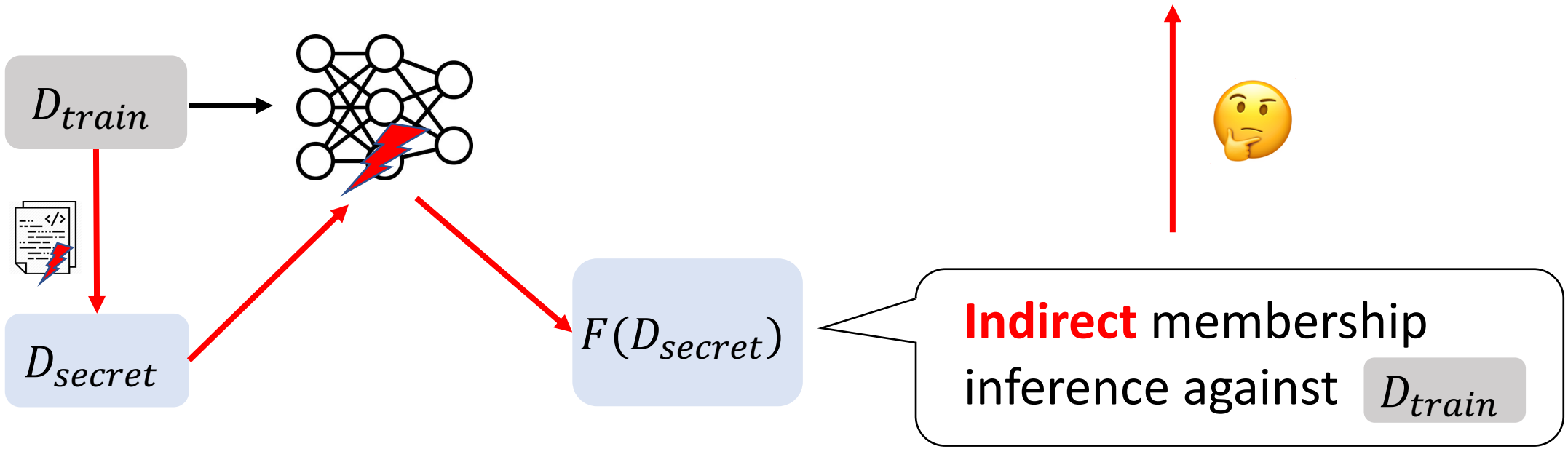
# Encode membership of $D_{train}$ via $D_{secret}$




$D_{train}$ **same** membership $D_{secret}$


$D_{train}$
$D_{secret}$
$F(D_{secret})$
**Indirect** membership inference against $D_{train}$

# How to make the (indirect) attack easy?



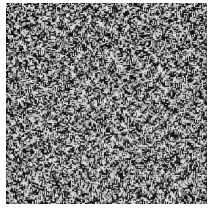
# How to make the (indirect) attack easy?

# How to make the (indirect) attack easy?

**Outlier** data are easy to memorize

# How to make the (indirect) attack easy?

**Outlier** data are easy to memorize

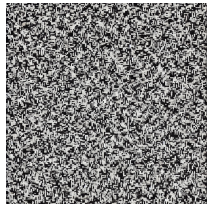


# How to make the (indirect) attack easy?

**Outlier** data are easy to memorize

$D_{secret}$

Crafted as **random** samples

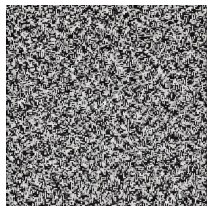


# How to make the (indirect) attack easy?

**Outlier** data are easy to memorize

$D_{secret}$

Crafted as **random** samples



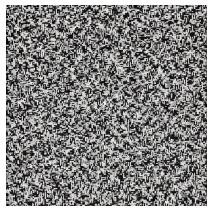
$D_{secret}$ is easy to de-identify

# How to make the (indirect) attack easy?

**Outlier** data are easy to memorize

$D_{secret}$

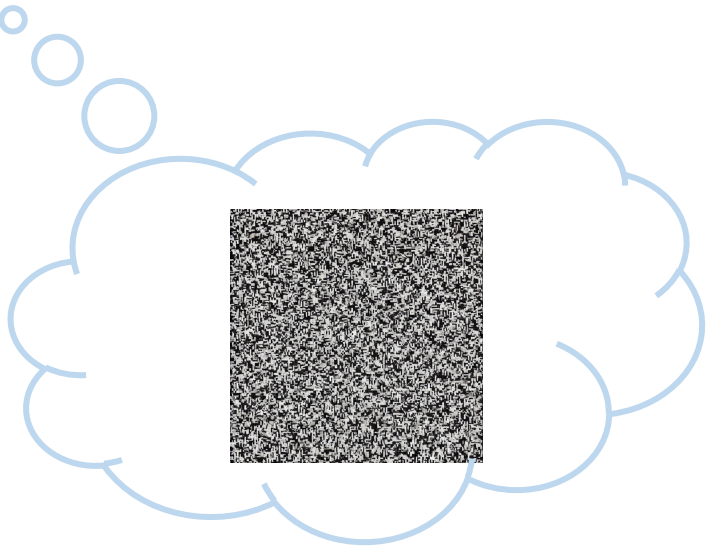
Crafted as **random** samples



$D_{secret}$ is easy to de-identify

$D_{train}$ is easy to de-identify 😈

Should we directly train $D_{train}$ $D_{secret}$ together?



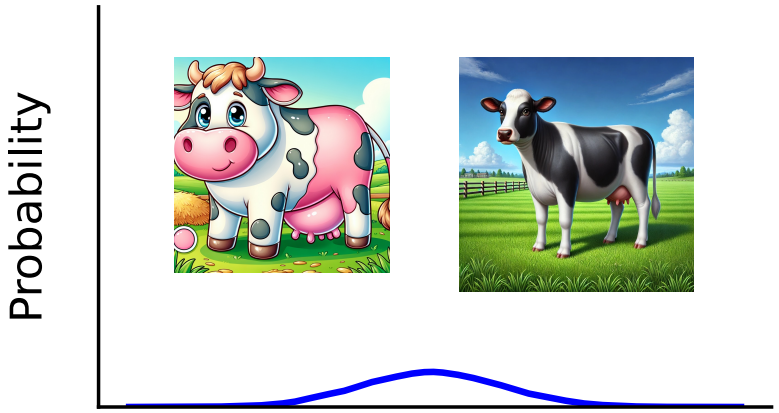
# Challenge

# Challenge

Norm functions expect data from a **similar** distribution

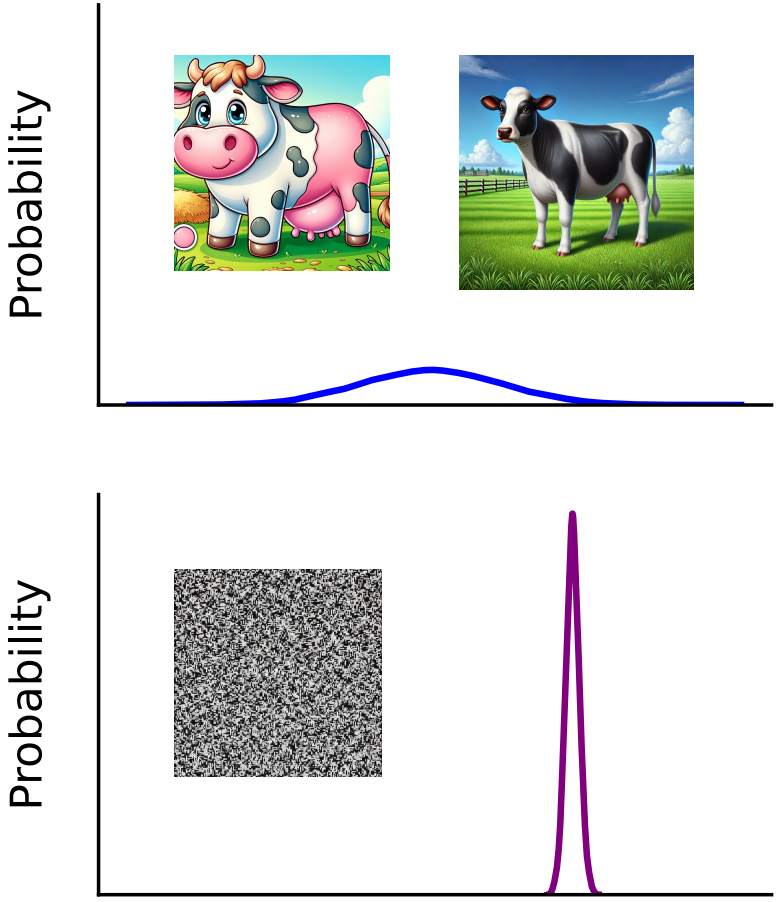
# Challenge

Norm functions expect data from a **similar** distribution



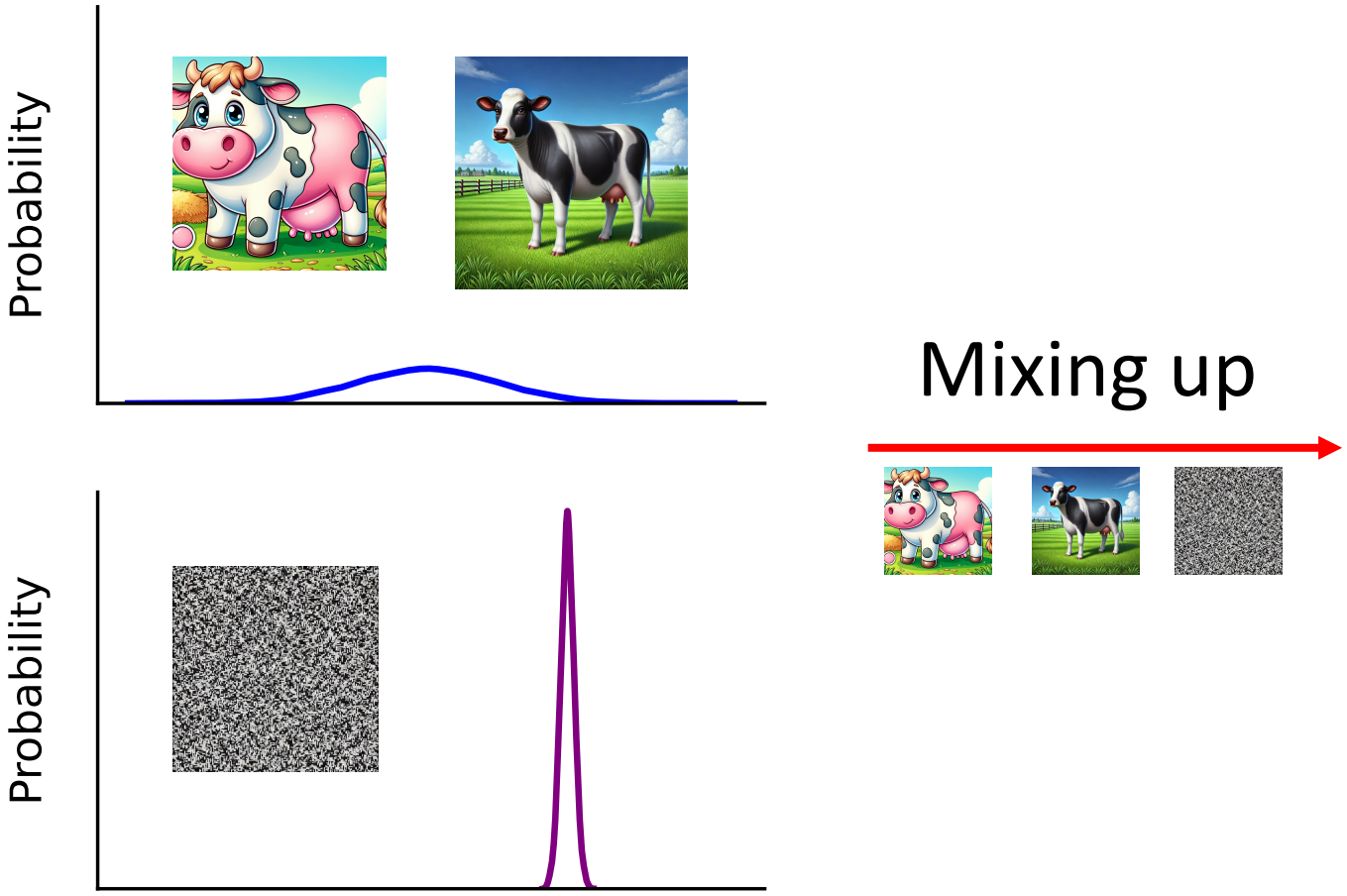
# Challenge

Norm functions expect data from a **similar** distribution



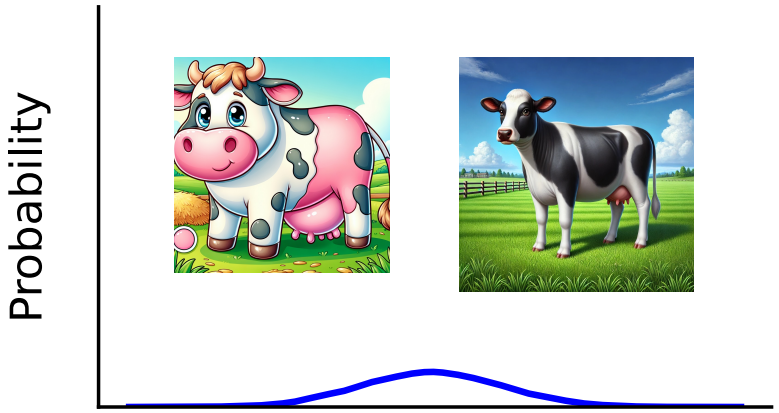
# Challenge

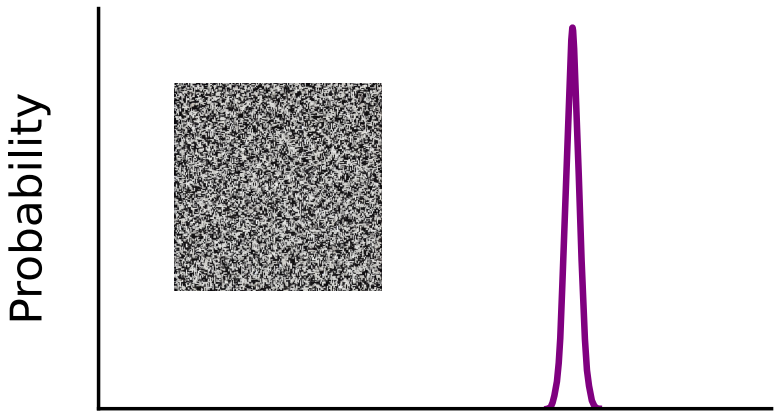
Norm functions expect data from a **similar** distribution



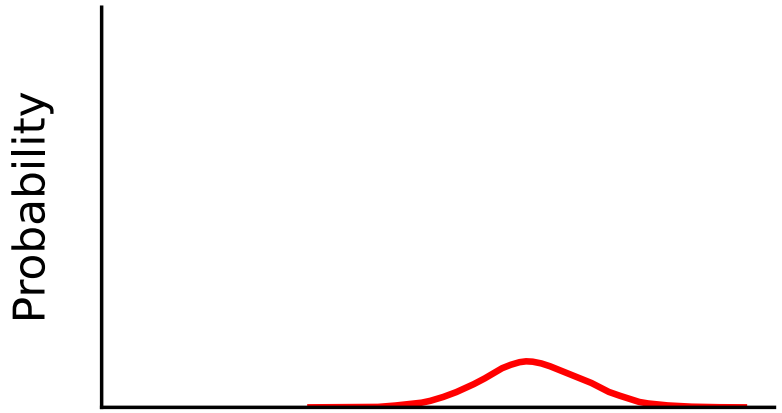
# Challenge

Norm functions expect data from a **similar** distribution




Mixing up




**Wrong** statistics for normalization!

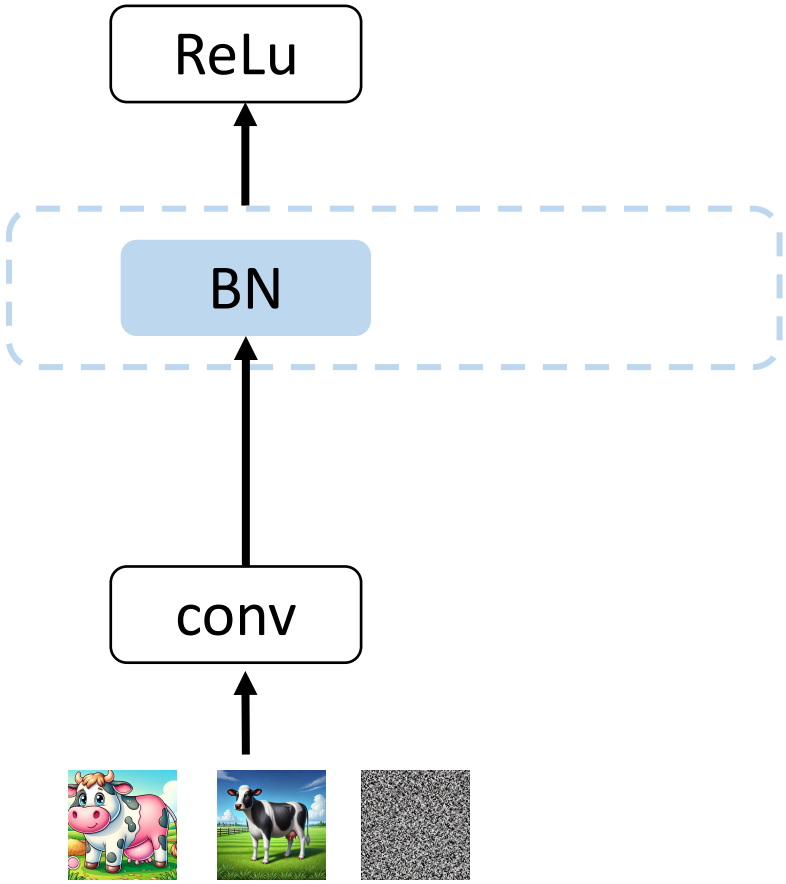
# Solution: Divide and conquer (again)

# Solution: Divide and conquer (again)

A secondary norm func to separately process $D_{secret}$

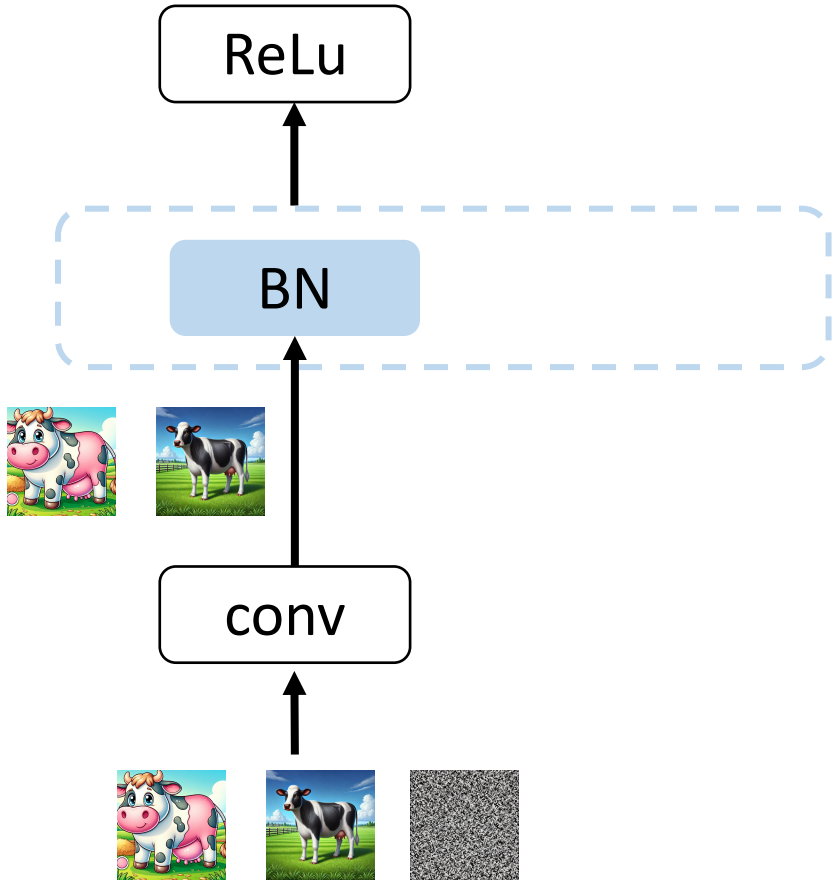
# Solution: Divide and conquer (again)

A secondary norm func to separately process $D_{secret}$



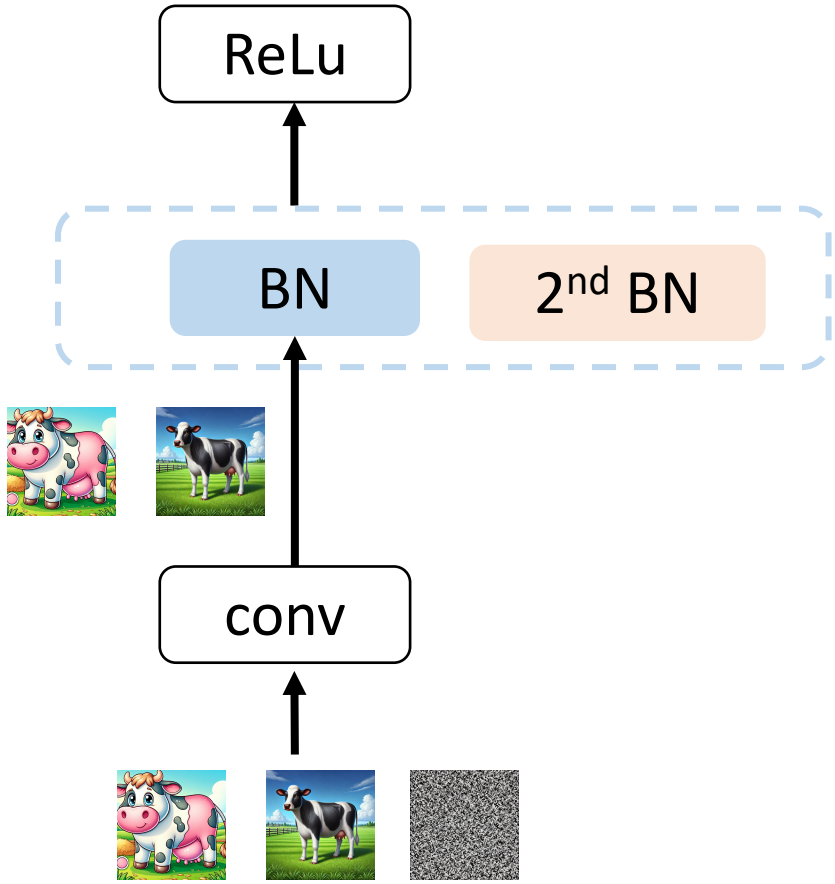
# Solution: Divide and conquer (again)

A secondary norm func to separately process $D_{secret}$



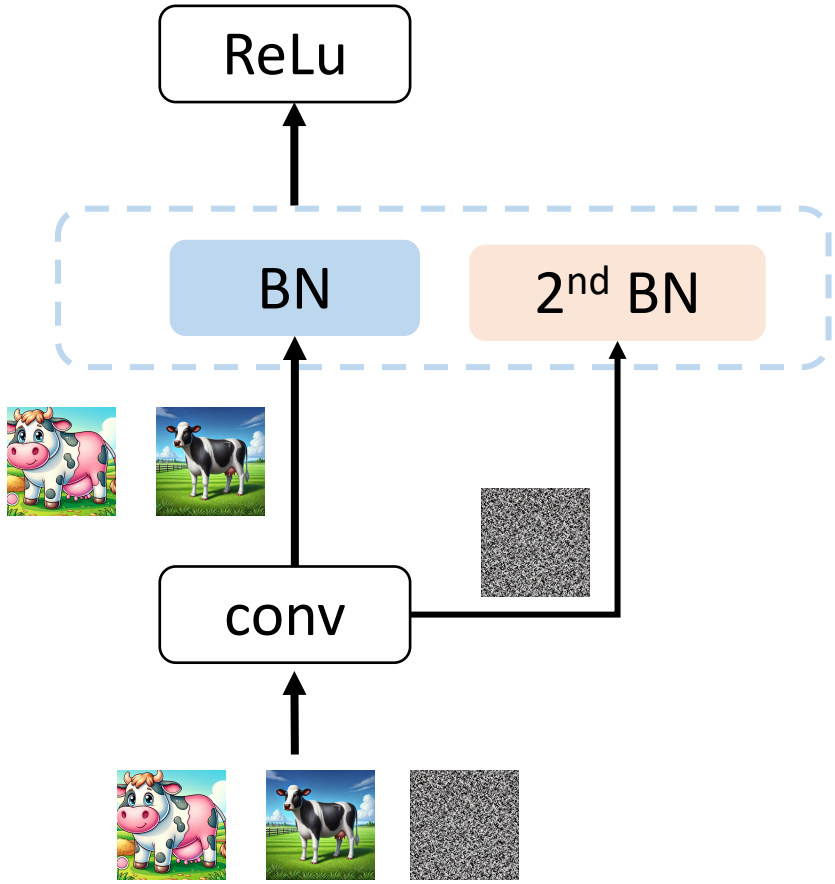
# Solution: Divide and conquer (again)

A secondary norm func to separately process $D_{secret}$



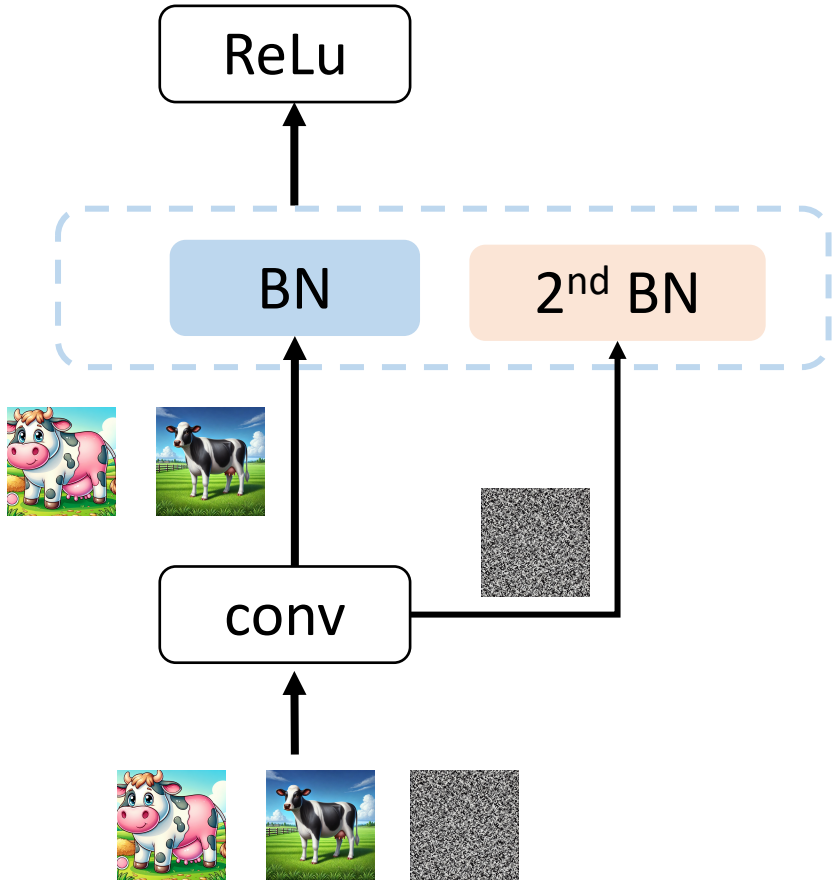
# Solution: Divide and conquer (again)

A secondary norm func to separately process $D_{secret}$

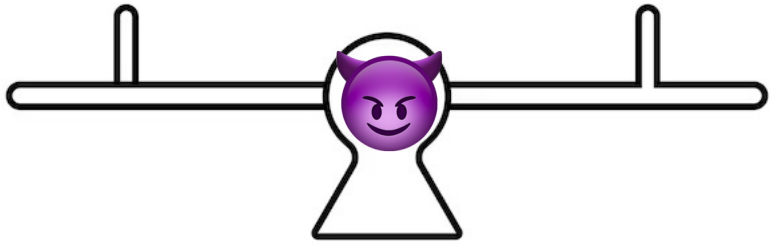


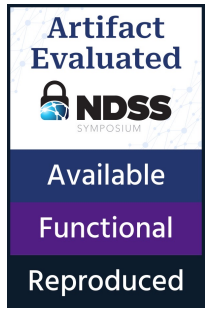
# Solution: Divide and conquer (again)

A secondary norm func to separately process $D_{secret}$




**High privacy leakage** **High model utility**





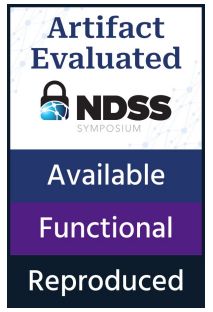

Attack TPR @ 0.1% FPR

Best prior active attack

**Our active attack**

How many data are leaked?

Tramer et al., CCS'22




Attack TPR @ 0.1% FPR

Best prior active attack

**Our active attack**

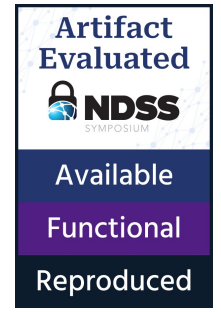
How many data are leaked?

13% → 35%

13% → 99.99%

Tramer et al., CCS'22

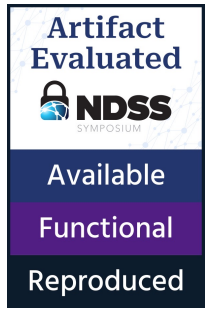
# Our attack exposes the worst-case privacy leakage




| Attack TPR @ 0.1% FPR | Best prior active attack | **Our active attack** |
|---|---|---|
| How many data are leaked? | 13% → 35% | 13% → 99.99% |

Tramer et al., CCS'22

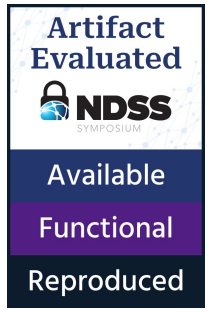
# Our attack exposes the worst-case privacy leakage




| | Best prior active attack | Our active attack |
|---|---|---|
| How many data are leaked? | 13% → 35% | 13% → 99.99% |
| Accuracy drop | >45% | <1% |

Tramer et al., CCS'22

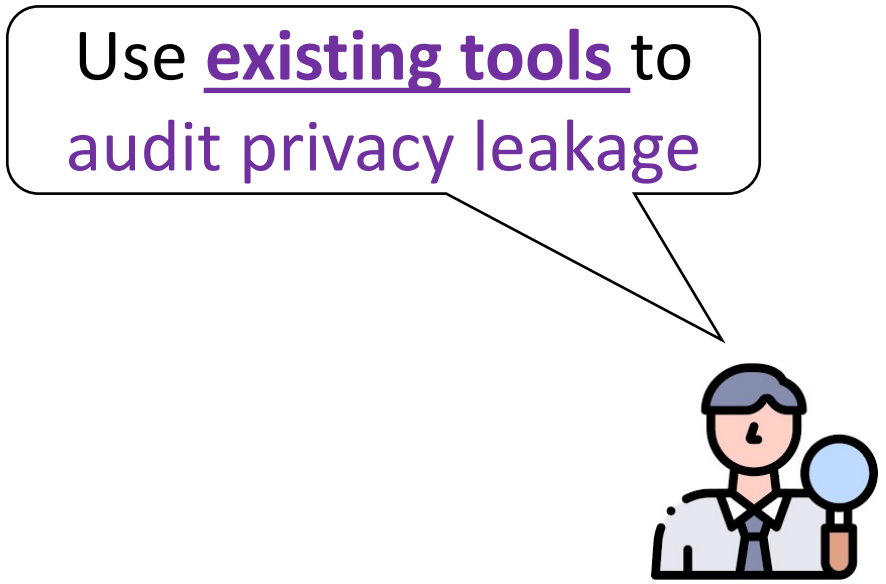
# Our attack exposes the worst-case privacy leakage has minimal performance impact



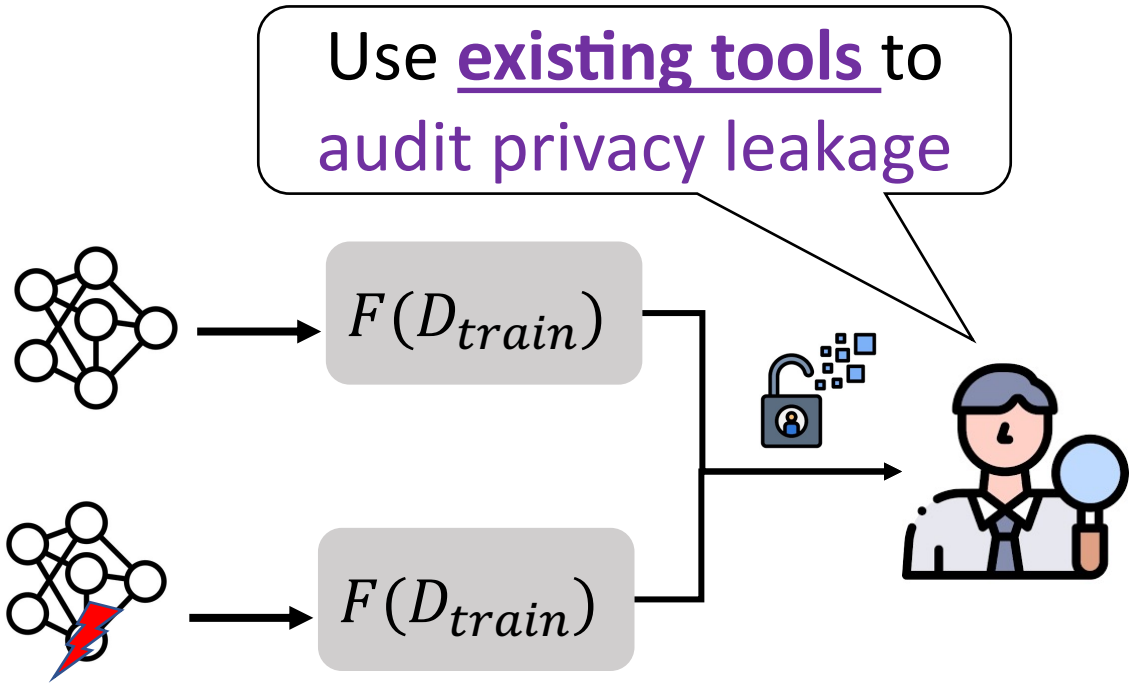

| | Best prior active attack | Our active attack |
|---|---|---|
| How many data are leaked? | 13% → 35% | 13% → 99.99% |
| Accuracy drop | >45% | <1% |

Tramer et al., CCS'22

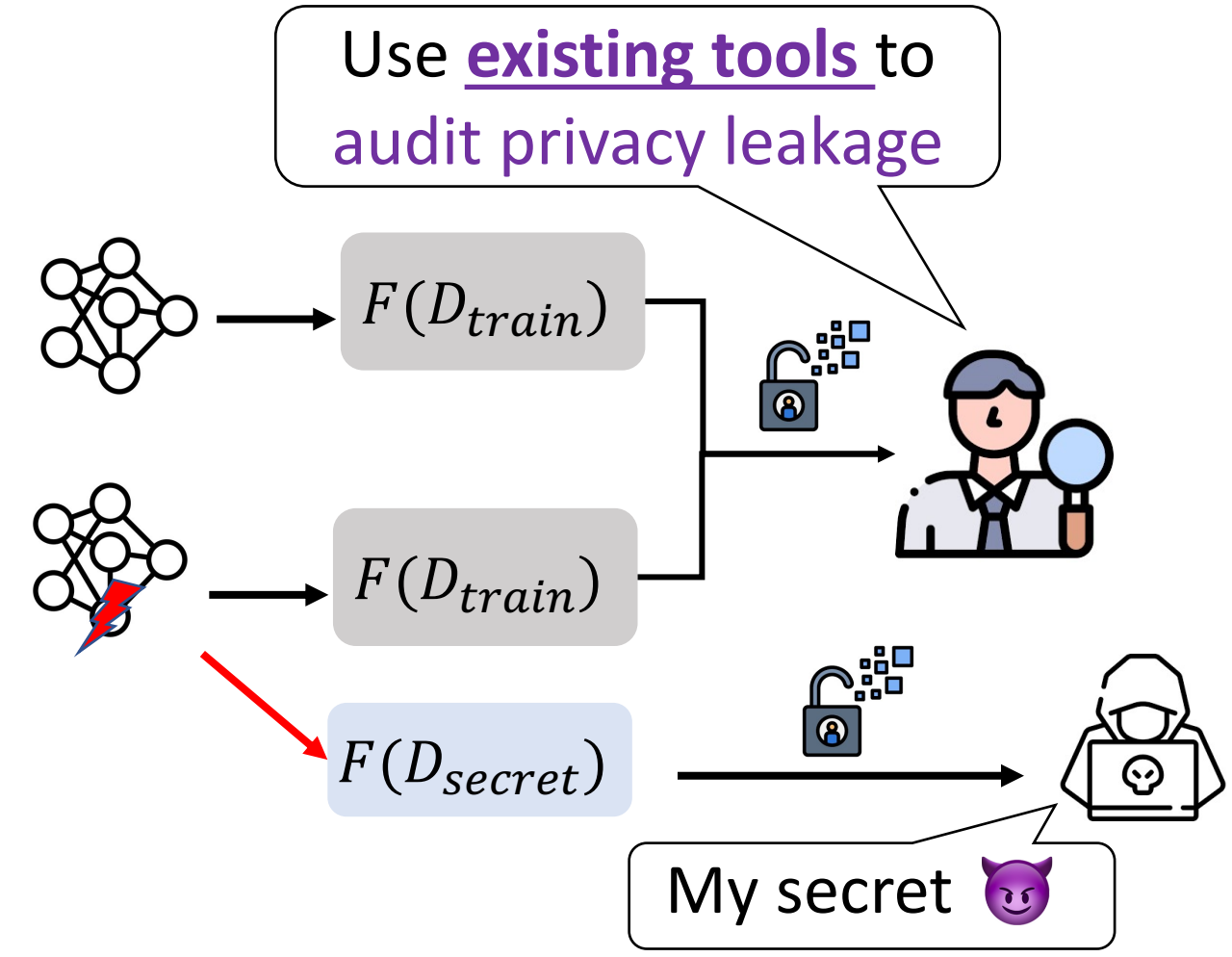
**Our attack exposes the worst-case privacy leakage has minimal performance impact**



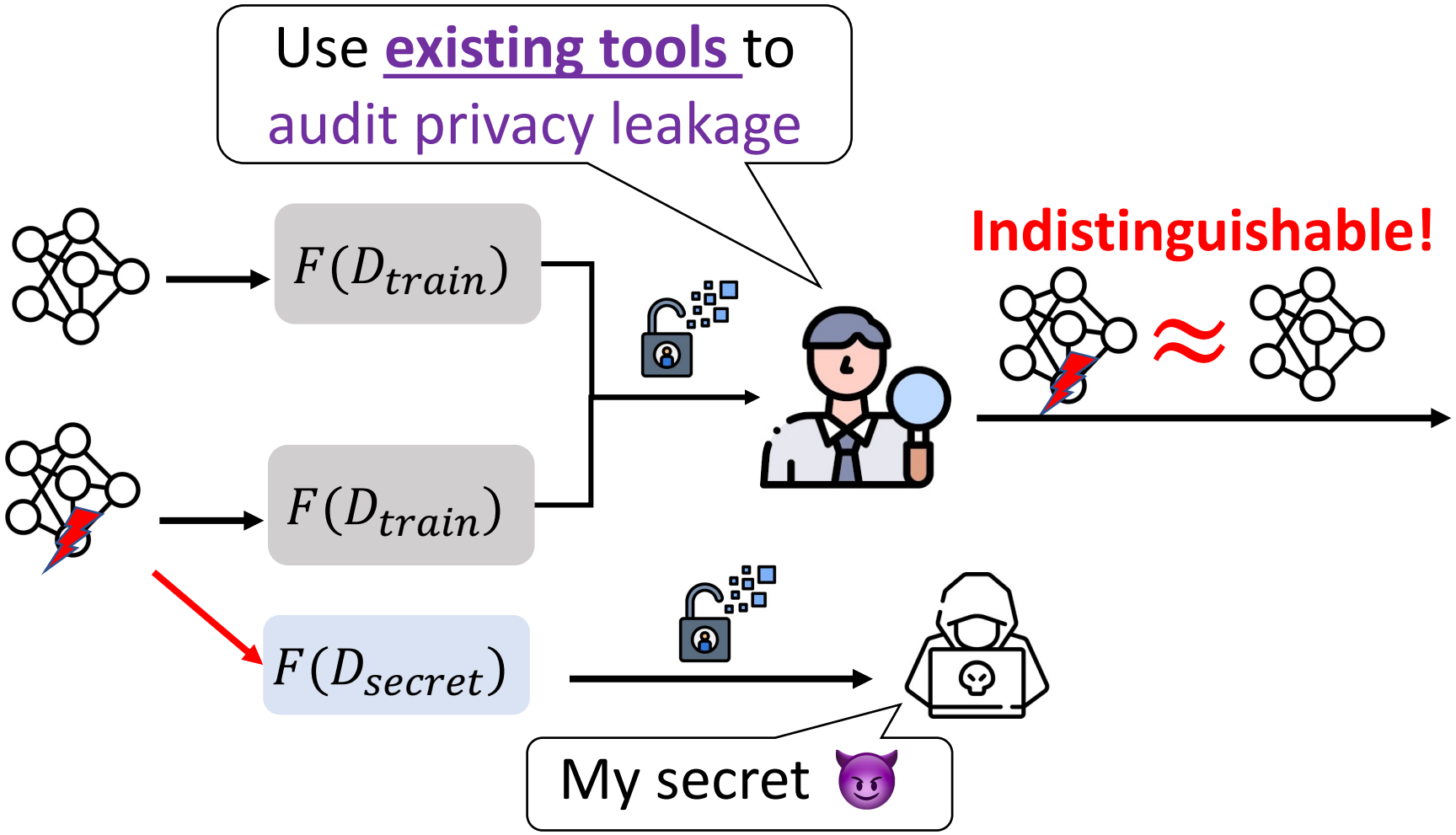
# Our attack exposes the worst-case privacy leakage has minimal performance impact



# Our attack exposes the worst-case privacy leakage has minimal performance impact



# Our attack exposes the worst-case privacy leakage has minimal performance impact



# Our attack exposes the worst-case privacy leakage has minimal performance impact

Artifact Evaluated
NDSS SYMPOSIUM
Available
Functional
Reproduced

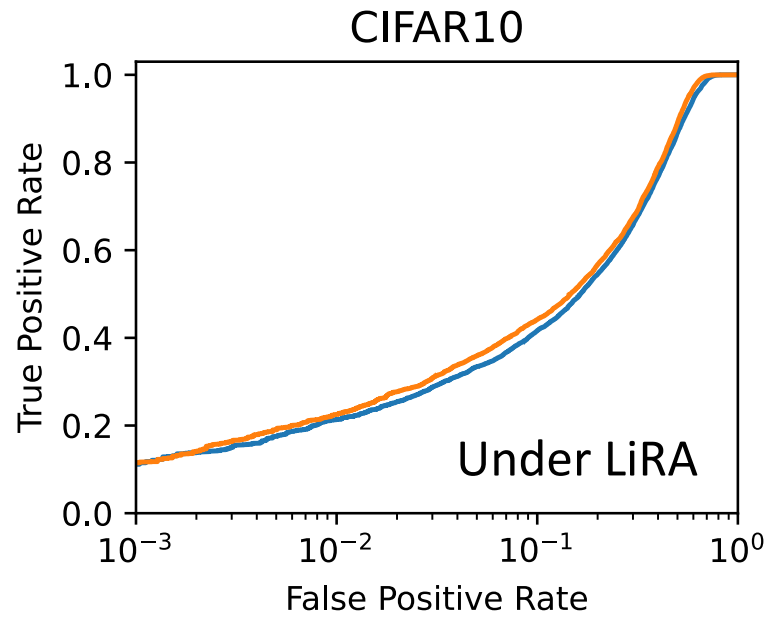
Use existing tools to audit privacy leakage

$F(D_{train})$

$F(D_{train})$

$F(D_{secret})$

Indistinguishable!



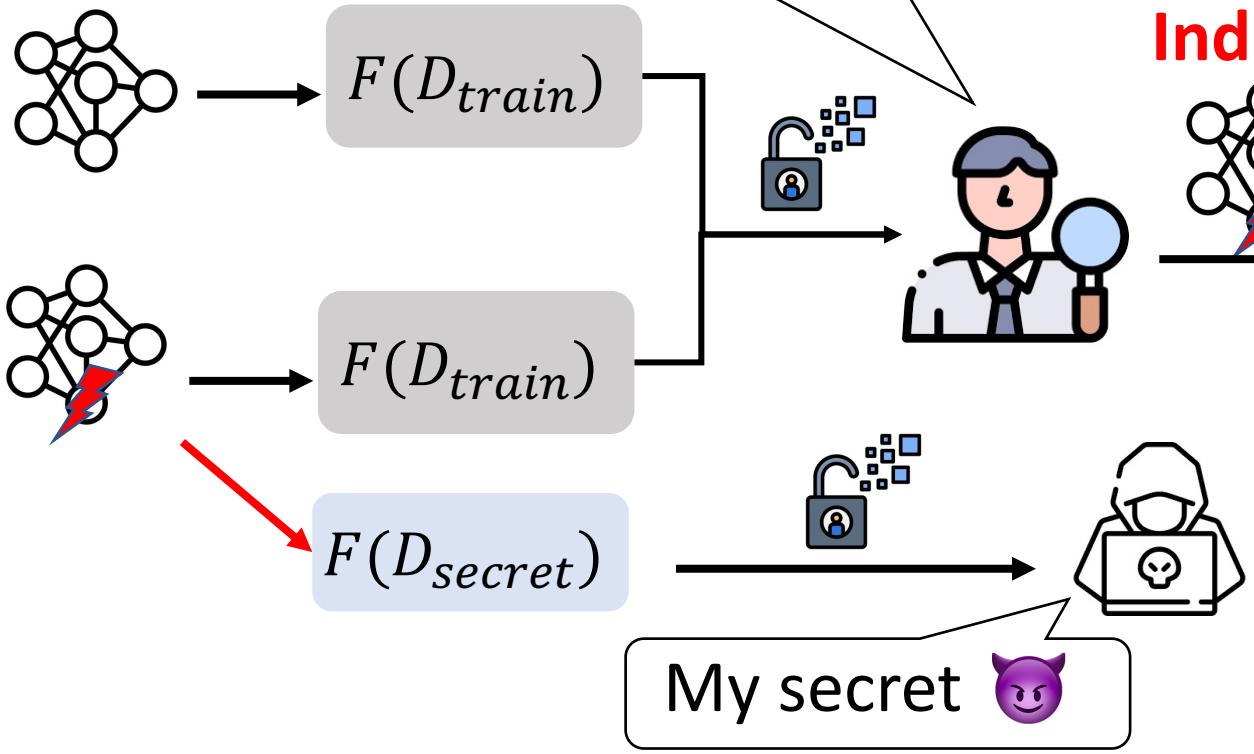

Carlini et al., S&P'22

My secret 😈

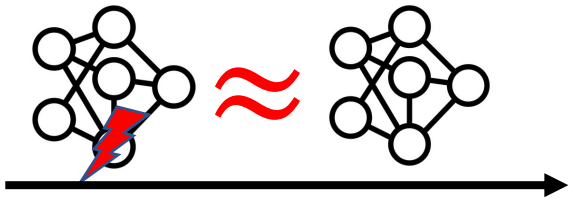
**Our attack exposes the worst-case privacy leakage has minimal performance impact can disguise high privacy leakage**

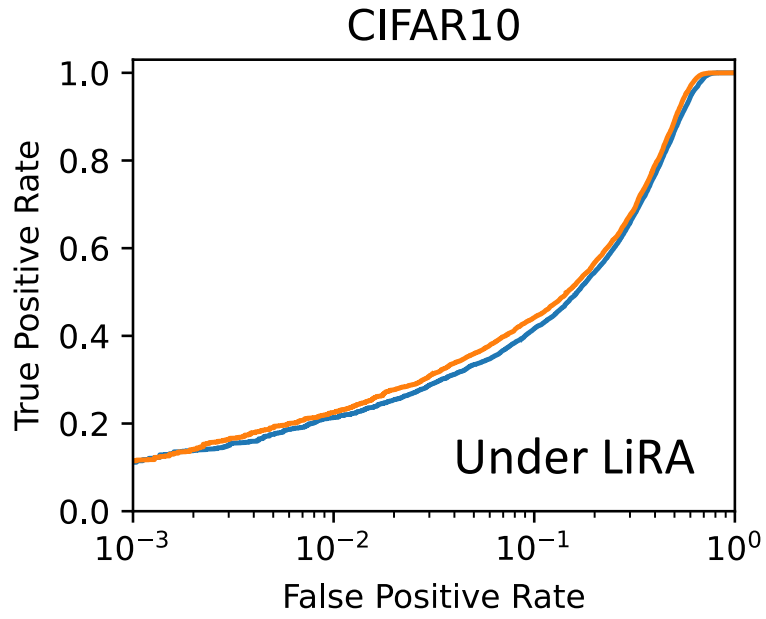
Artifact Evaluated
NDSS SYMPOSIUM
Available
Functional
Reproduced

Use existing tools to audit privacy leakage




**Indistinguishable!**

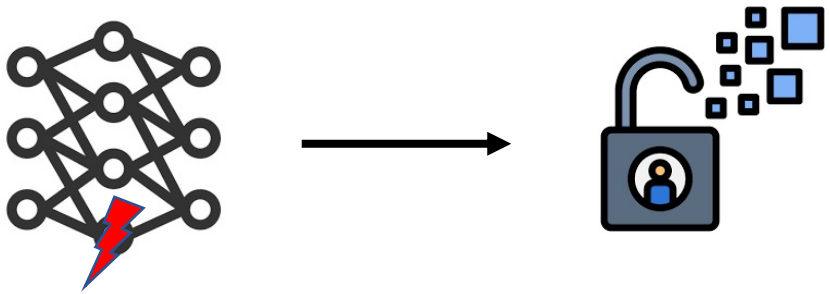





Carlini et al., S&P'22

# Conclusion

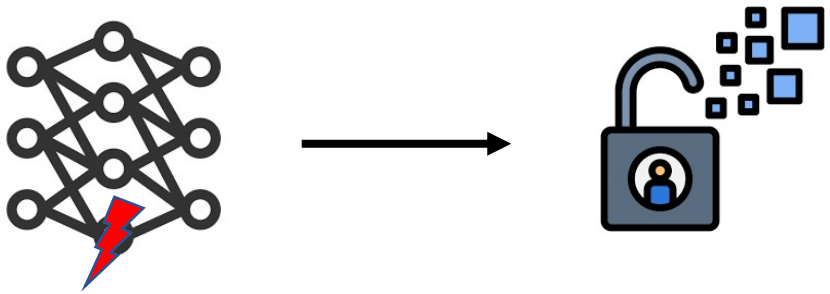
Using third-party **ML codebase** has hidden privacy risk



# Conclusion

Using third-party **ML codebase** has hidden privacy risk

**New direction** to construct **stealthy attacks** and inflict **worst-case leakage**



# Conclusion

Using third-party **ML codebase** has hidden privacy risk

**New direction** to construct **stealthy attacks** and inflict **worst-case leakage**

The **first** result → **Existing privacy auditing** methods can be **unreliable**!

# Conclusion

Zitao Chen
zitaoc@ece.ubc.ca

Using third-party **ML codebase** has hidden privacy risk

**New direction** to construct **stealthy attacks** and inflict **worst-case leakage**

The **first** result → **Existing privacy auditing** methods can be **unreliable**!