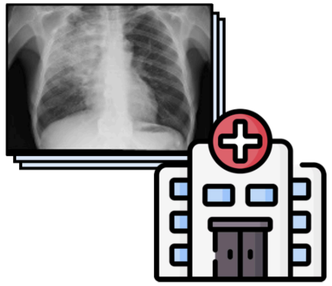
DLBox: New Model Training Framework for Protecting Training Data

Jaewon Hur, Juheon Yi, Cheolwoo Myung, Sangyun Kim, Youngki Lee, and Byoungyoung Lee

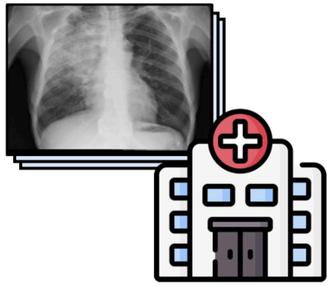
서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

Sharing Training Data can be Dangerous



Data Owner

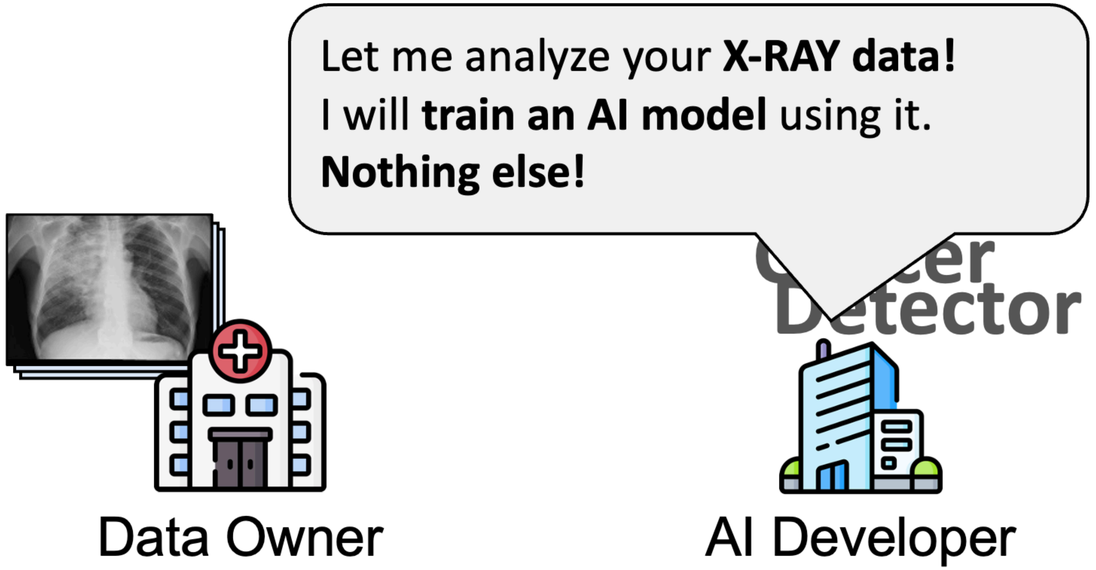
Sharing Training Data can be Dangerous



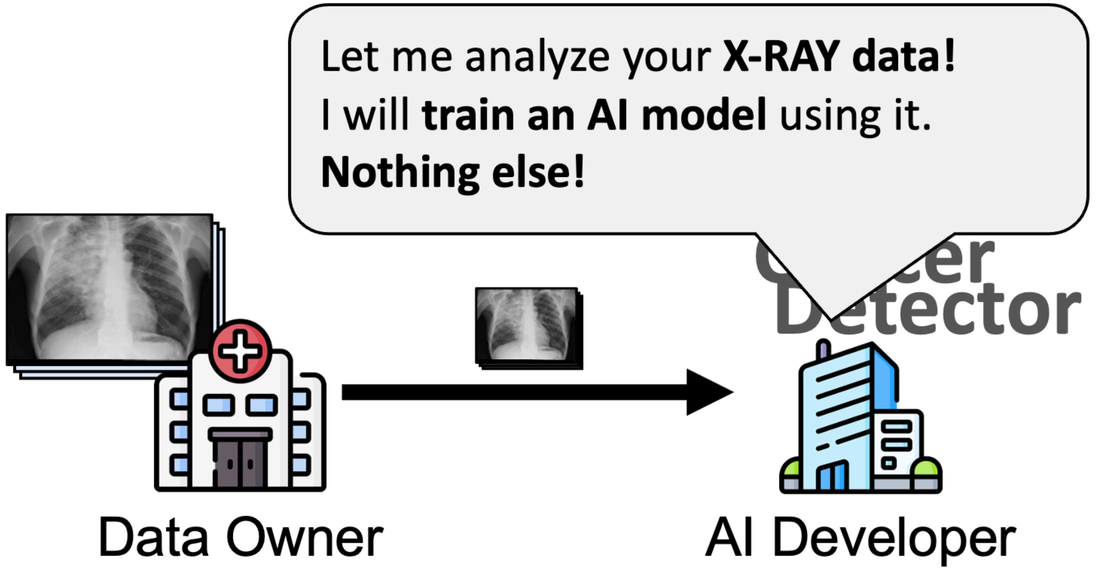
Data Owner

AI Developer

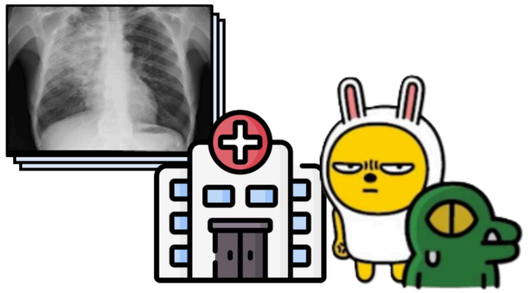
Sharing Training Data can be Dangerous



Sharing Training Data can be Dangerous



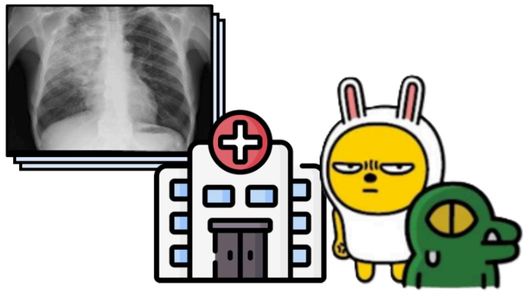
Sharing Training Data can be Dangerous



Data Owner

AI Developer

Sharing Training Data can be Dangerous

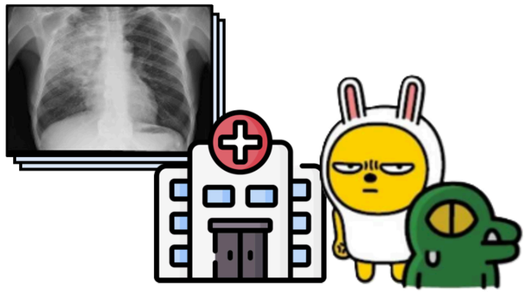


Data Owner

AI Developer

- **Sharing Data ?**

Sharing Training Data can be Dangerous



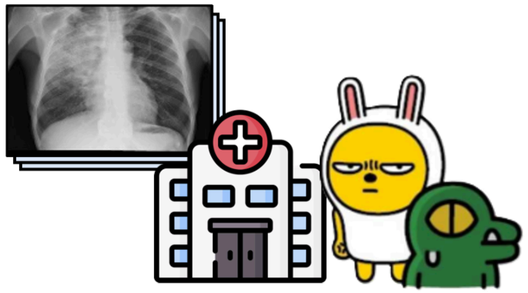
Data Owner

AI Developer

- **Sharing Data ?**

Allowing to use the data

Sharing Training Data can be Dangerous



Data Owner

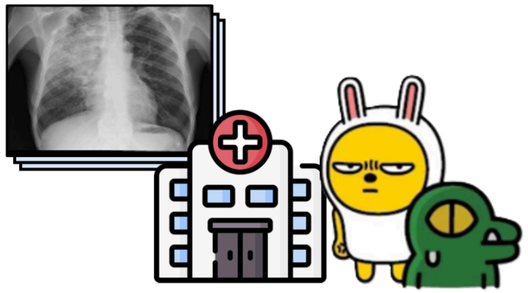
AI Developer

- **Sharing Data ?**

Allowing to use the data

in a permitted way

Sharing Training Data can be Dangerous



Data Owner

AI Developer

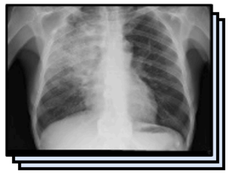
- **Sharing Data ?**

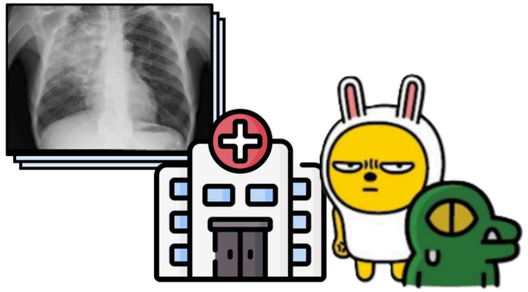
Allowing to use the data

in a permitted way

What is **"sharing the data for model training"** ?

Sharing Training Data can be Dangerous



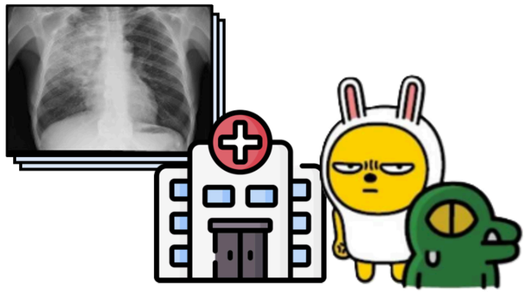


Data Owner

AI Developer

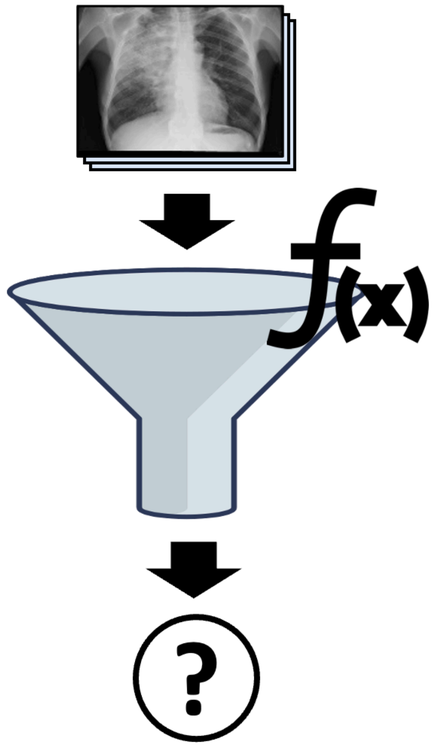
What is **"sharing the data for model training"** ?

Sharing Training Data can be Dangerous



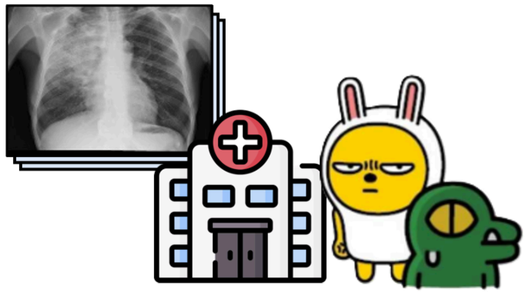
Data Owner

AI Developer



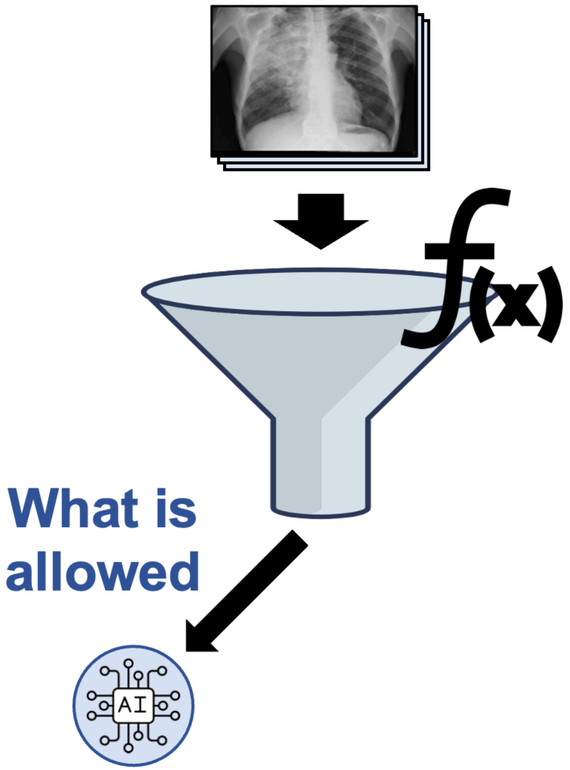
What is **"sharing the data for model training"** ?

Sharing Training Data can be Dangerous



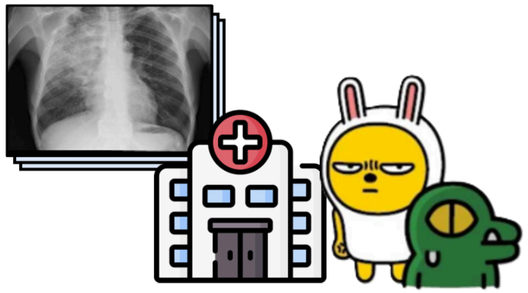
Data Owner

AI Developer



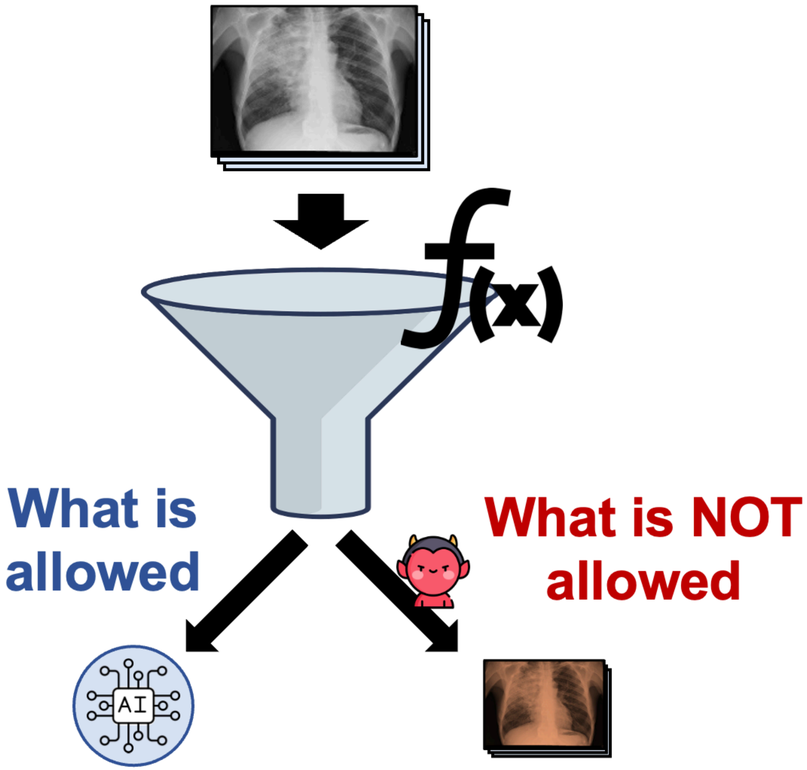
What is **"sharing the data for model training"** ?

Sharing Training Data can be Dangerous



Data Owner

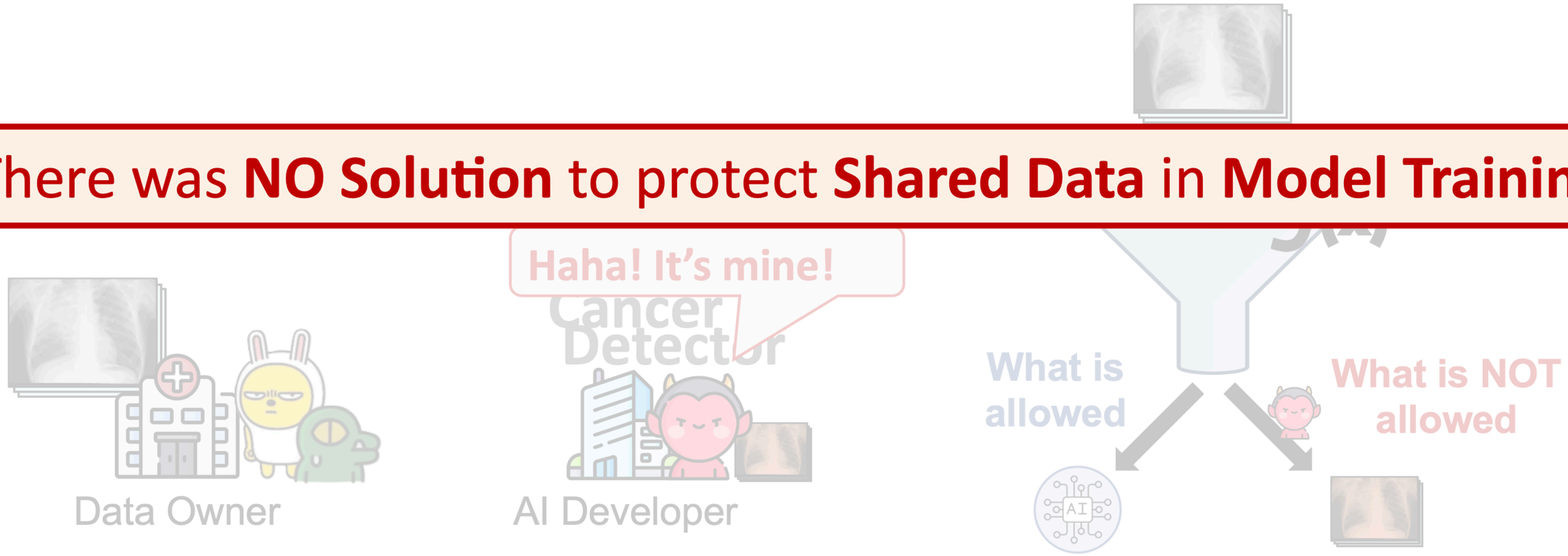
AI Developer



What is **"sharing the data for model training"** ?

Sharing Training Data can be Dangerous

There was **NO Solution** to protect **Shared Data** in **Model Training**



Sharing Training Data can be Dangerous

There was **NO Solution** to protect **Shared Data** in **Model Training**

So, **the AI Developers** can do **Whatever They Want**

Sharing Training Data can be Dangerous

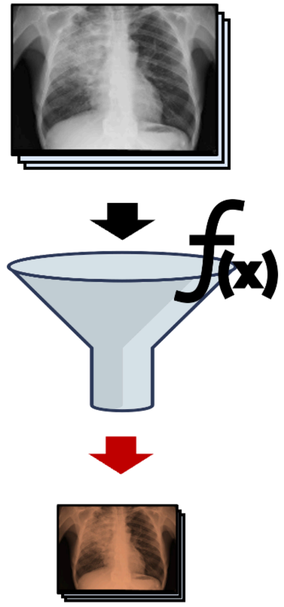
There was **NO Solution** to protect **Shared Data** in **Model Training**

So, **the AI Developers** can do **Whatever They Want**

Big Hurdle in Deep Learning Industry

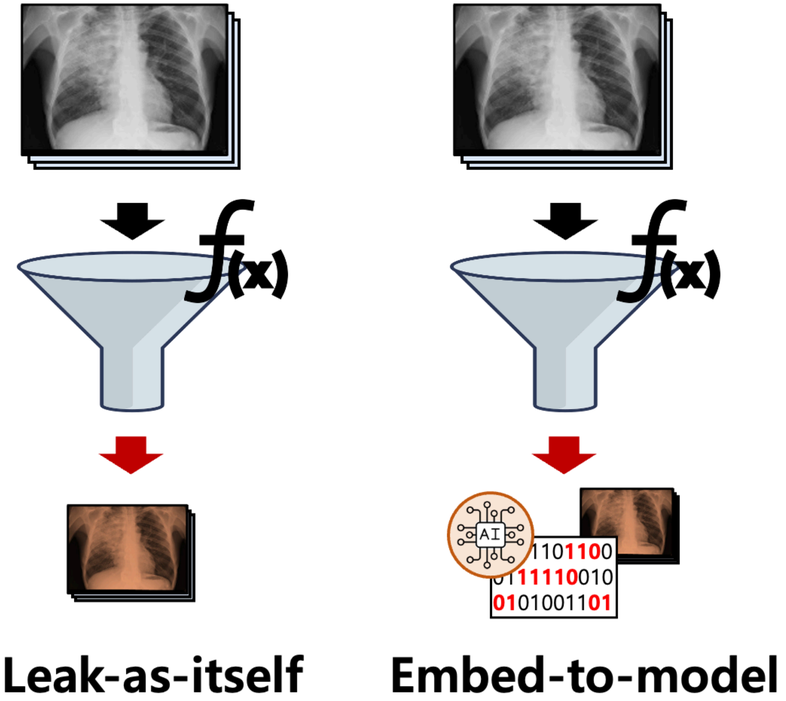
Possible Attacks to Leak the Data

Possible Attacks to Leak the Data



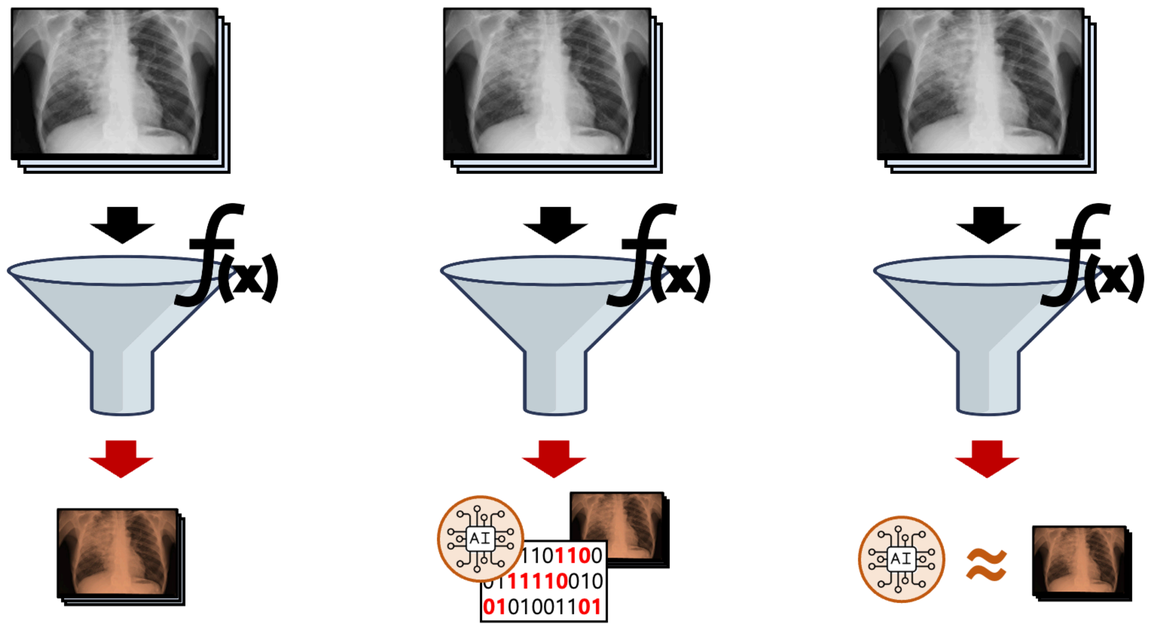
Leak-as-itself

Possible Attacks to Leak the Data



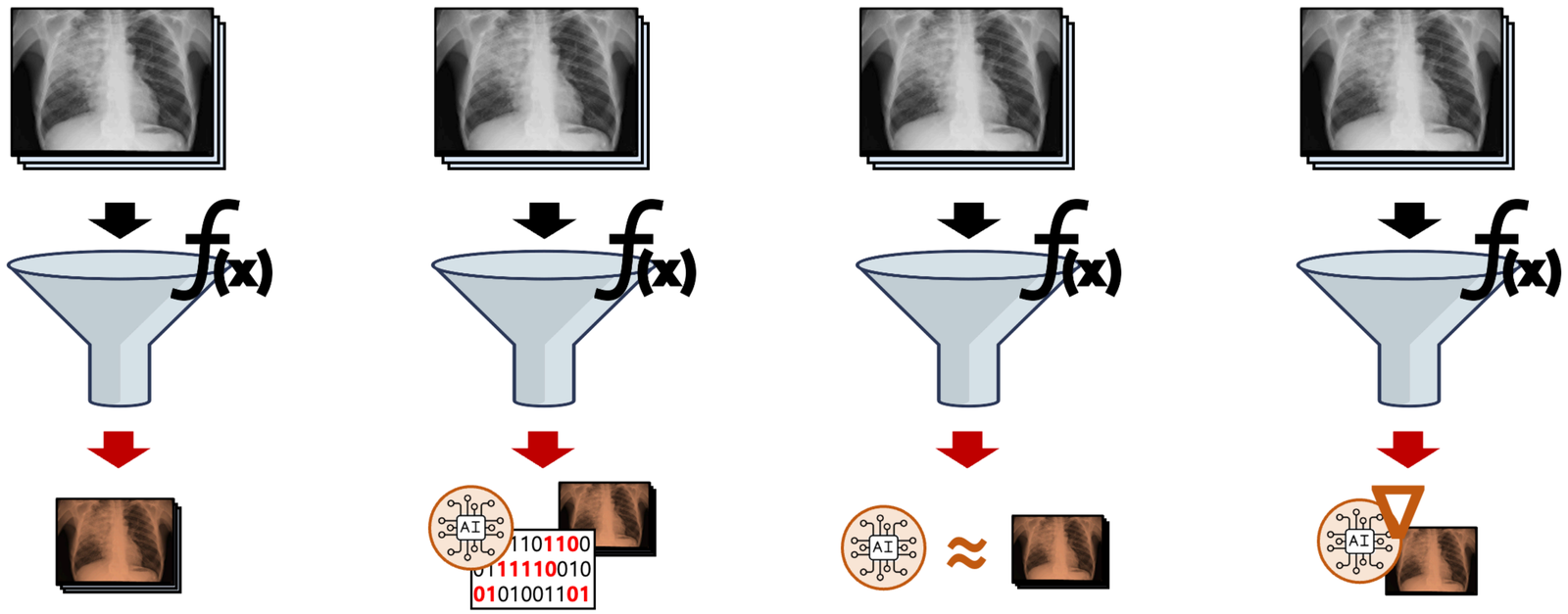
Leak-as-itself **Embed-to-model**

Possible Attacks to Leak the Data



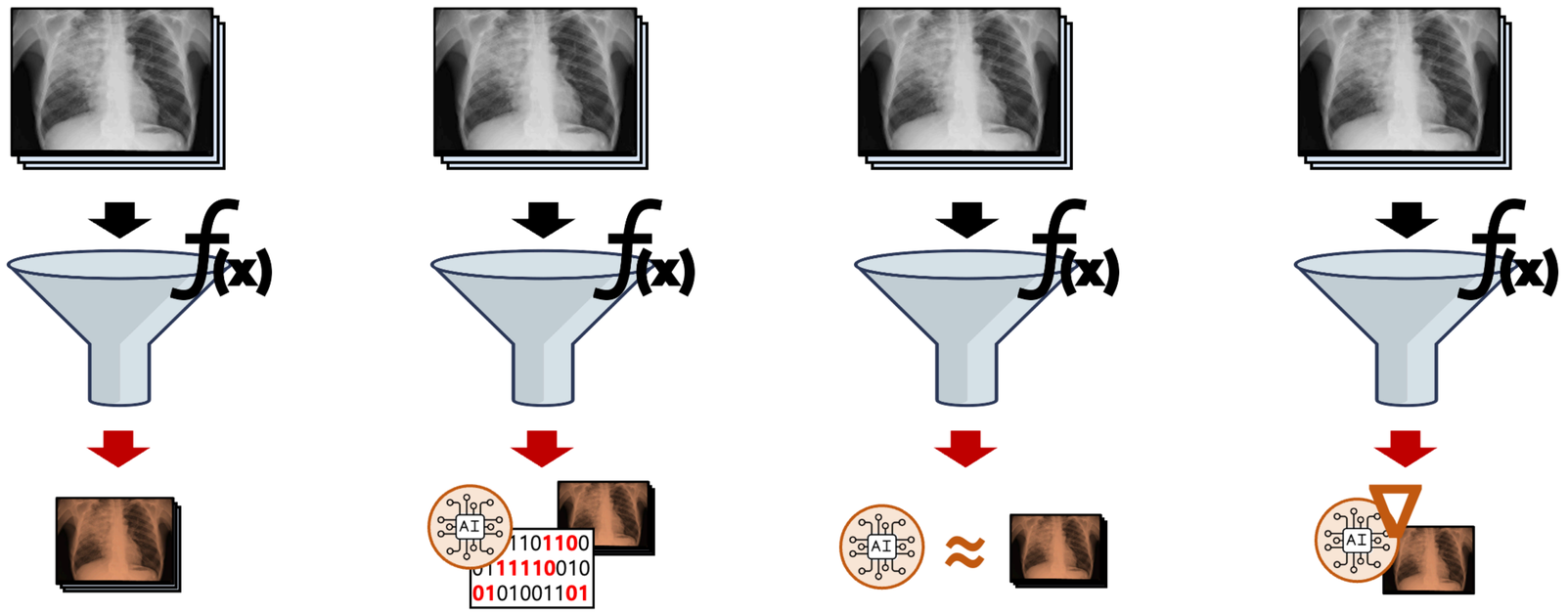
Leak-as-itself Embed-to-model Bias-to-be-data

Possible Attacks to Leak the Data



Leak-as-itself Embed-to-model Bias-to-be-data Invert-gradients

Possible Attacks to Leak the Data



Leak-as-itself

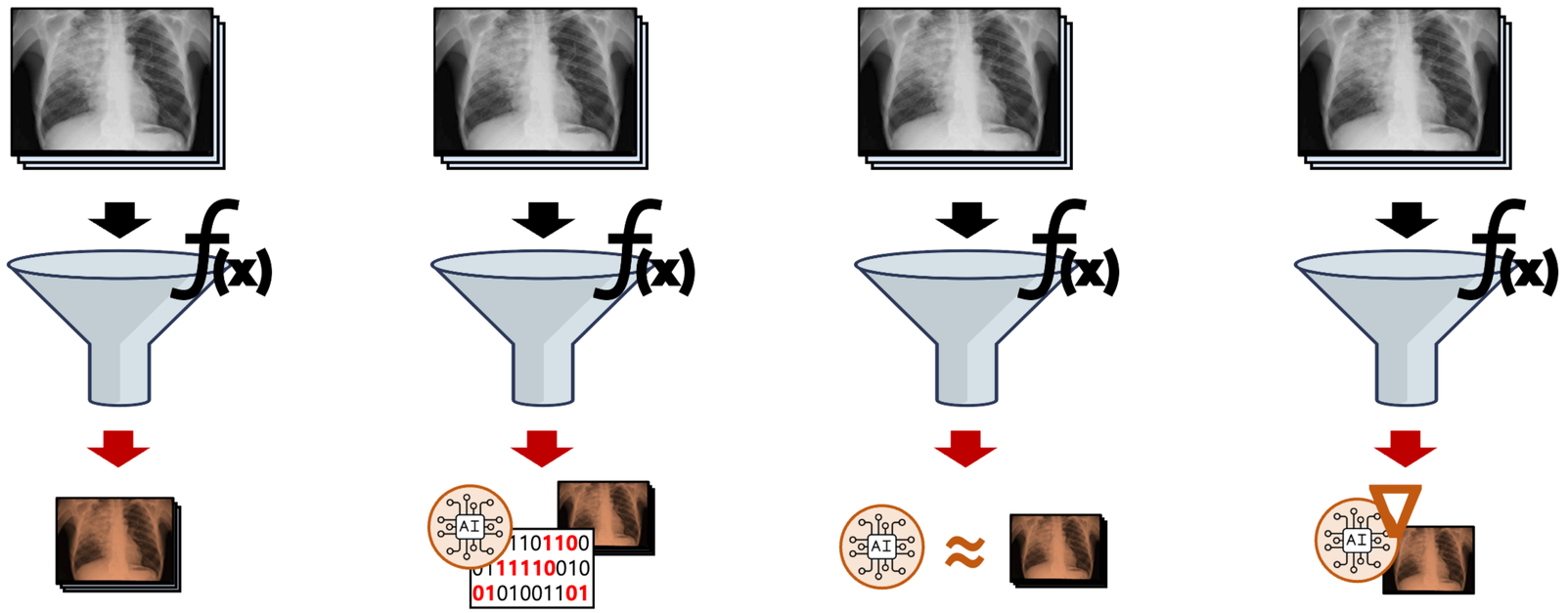
Embed-to-model

Bias-to-be-data

Invert-gradients

… Anything what you can imagine

Possible Attacks to Leak the Data

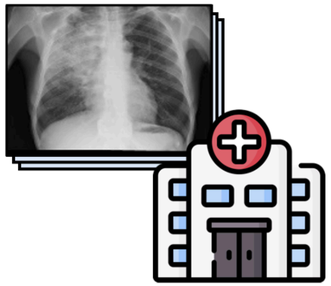


Leak-as-itself | Embed-to-model | Bias-to-be-data | Invert-gradients

… **Anything what you can imagine**

No one knows **"How to prevent data leaks"**

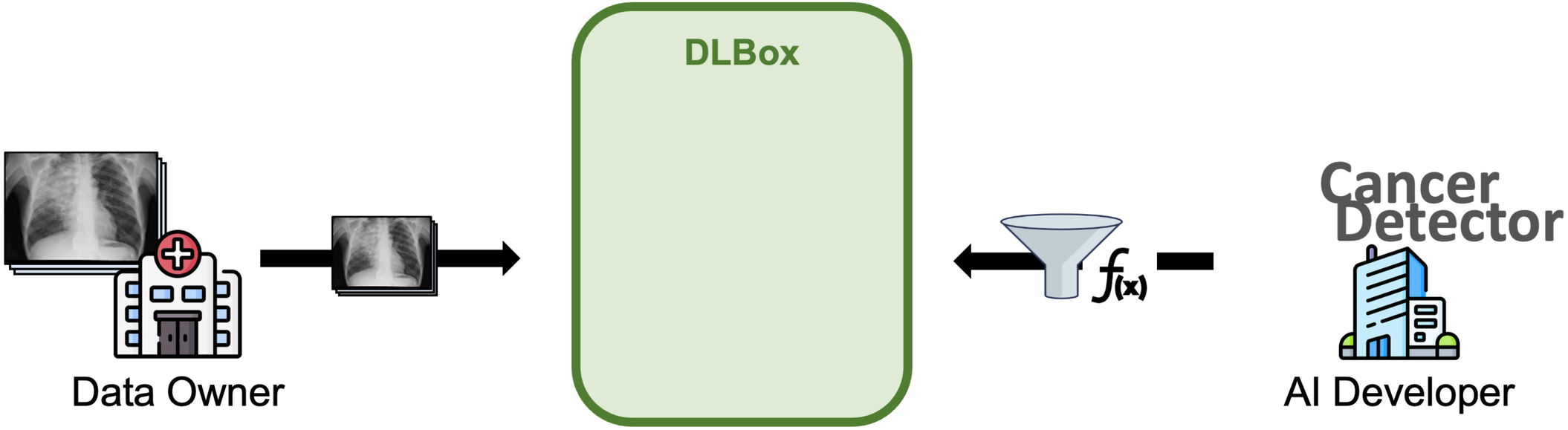
DLBox: Protecting Data in Model Training



Data Owner

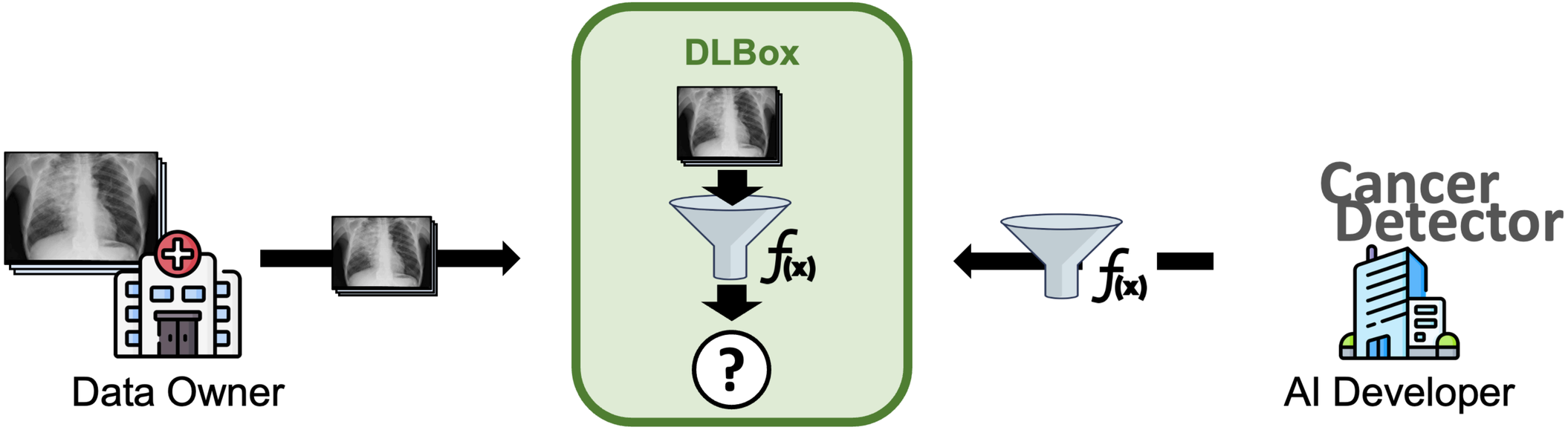
AI Developer

DLBox: Protecting Data in Model Training



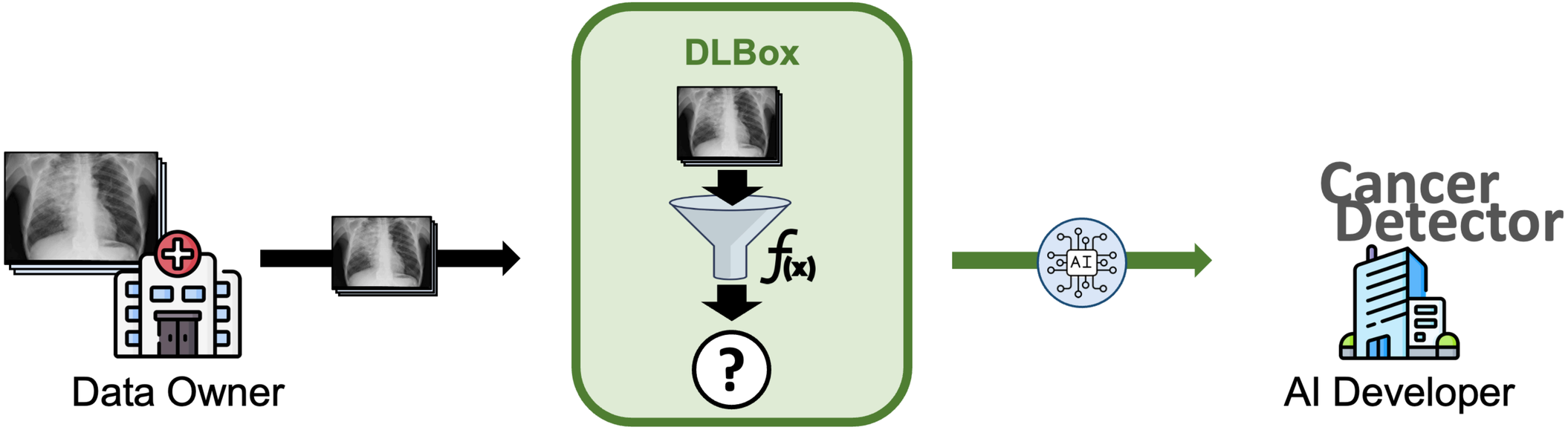
Given **training data & model training code,**

DLBox: Protecting Data in Model Training



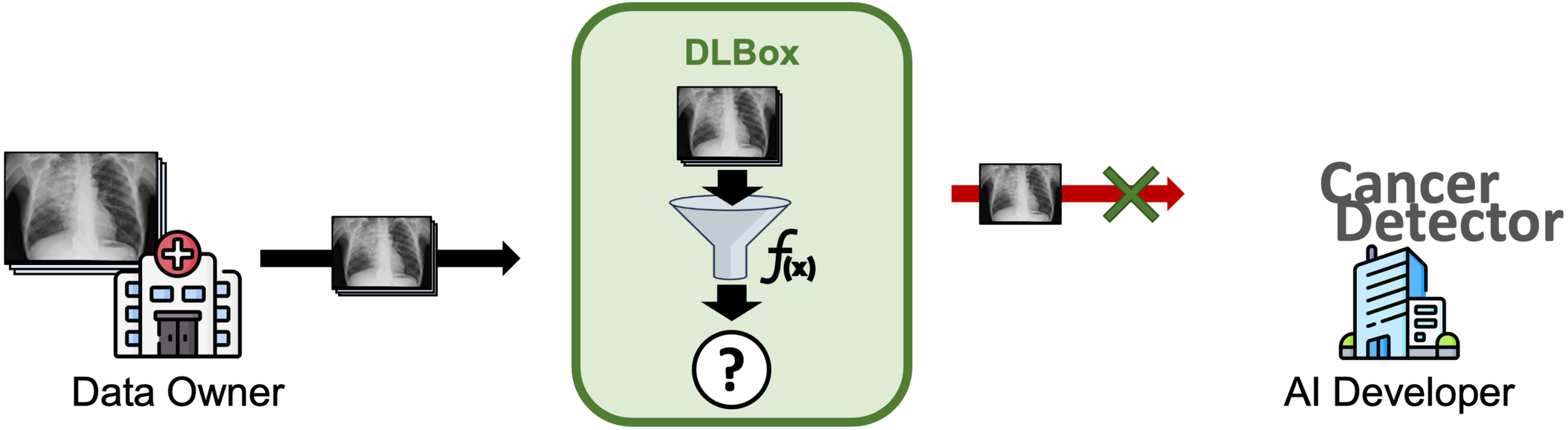
Given **training data & model training code,**

DLBox: Protecting Data in Model Training



Given **training data & model training code,**
Return **AI model** that was **benignly trained**

DLBox: Protecting Data in Model Training

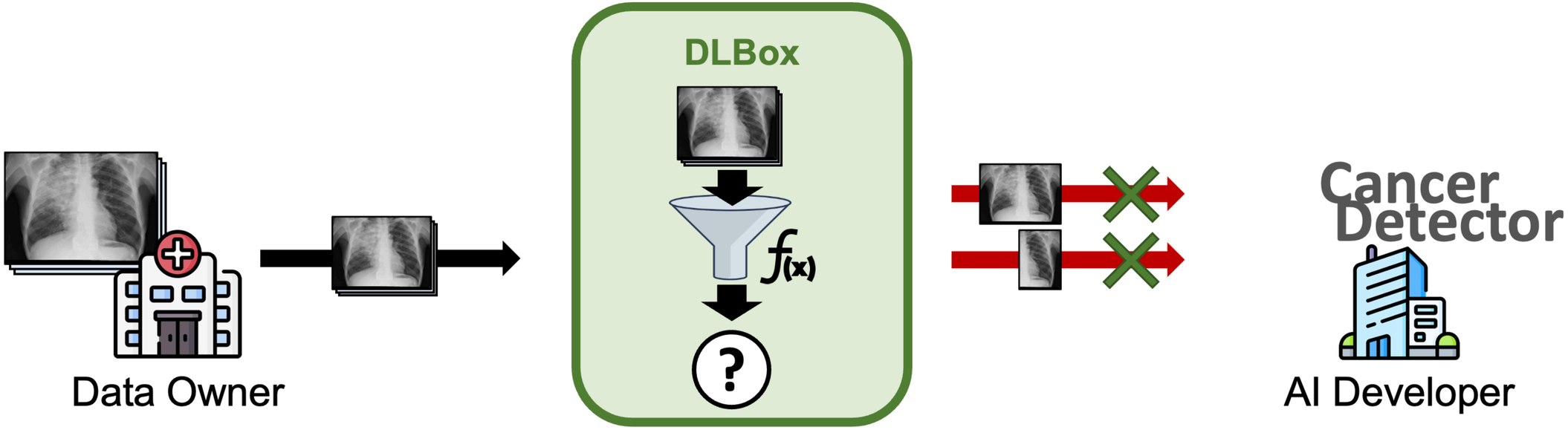


Given **training data & model training code,**

Return **AI model** that was **benignly trained**

Prevent **unauthorized data leakages**.

DLBox: Protecting Data in Model Training

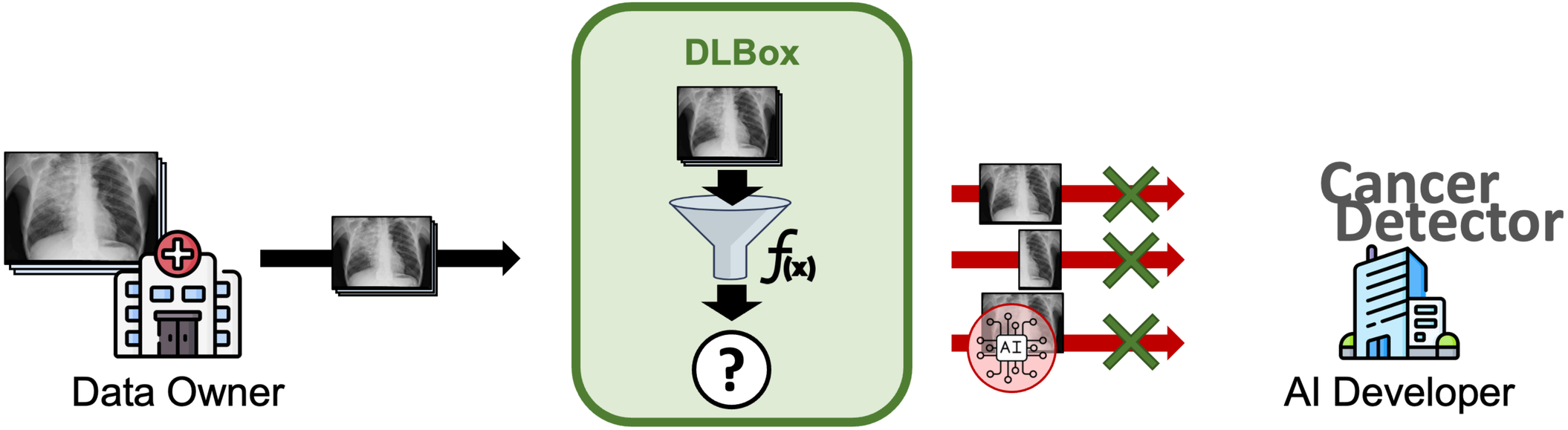


Given **training data & model training code,**

Return **AI model** that was **benignly trained**

Prevent **unauthorized data leakages**.

DLBox: Protecting Data in Model Training

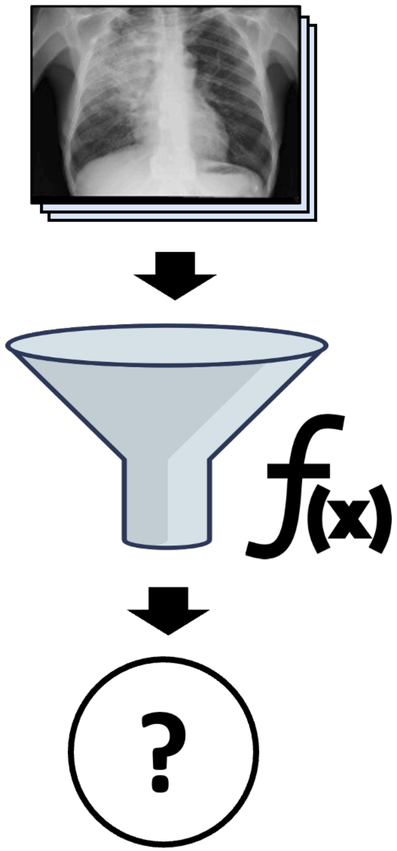


Given **training data & model training code,**

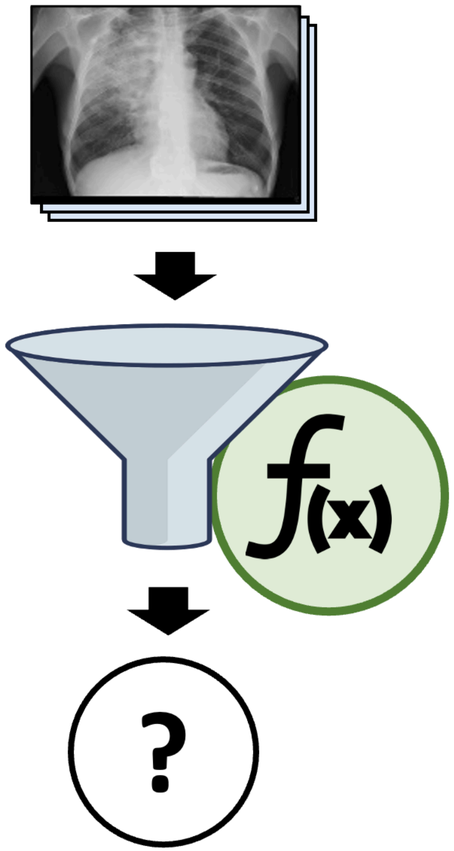
Return **AI model** that was **benignly trained**

Prevent **unauthorized data leakages**.

Infeasibility of Complete Data Protection

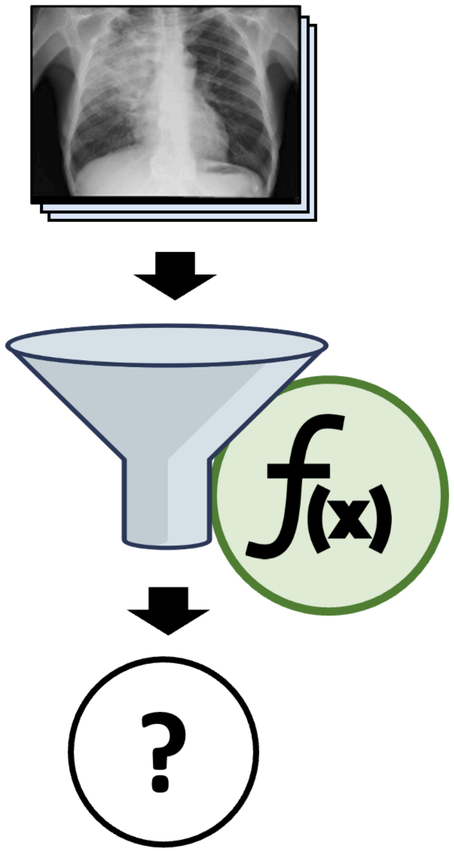


Infeasibility of Complete Data Protection



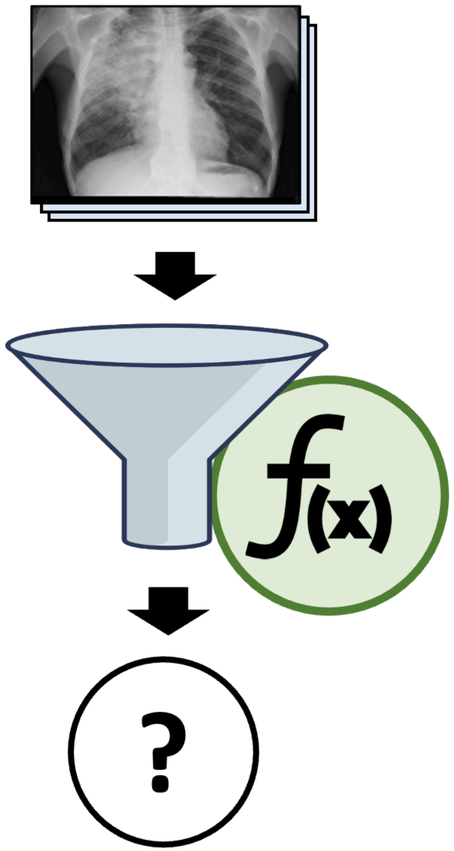
- DLBox checks $f(x)$ developed by an AI developer

Infeasibility of Complete Data Protection



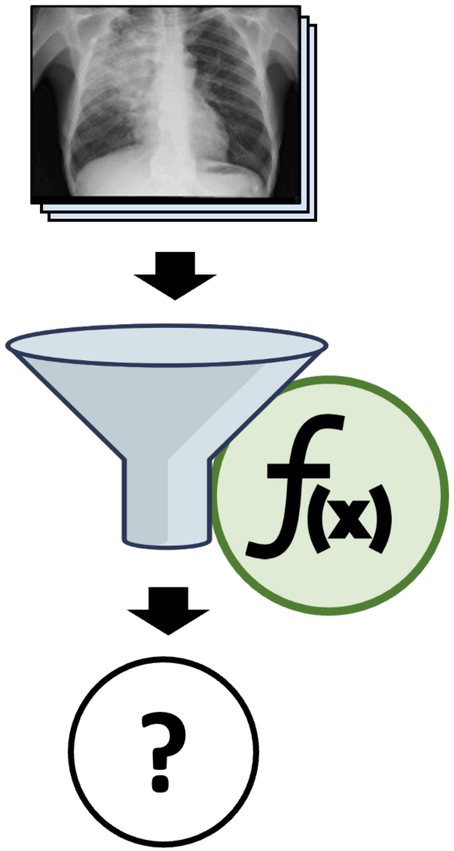
- DLBox checks $f(x)$ developed by an AI developer
- Properties of allowed functions

Infeasibility of Complete Data Protection



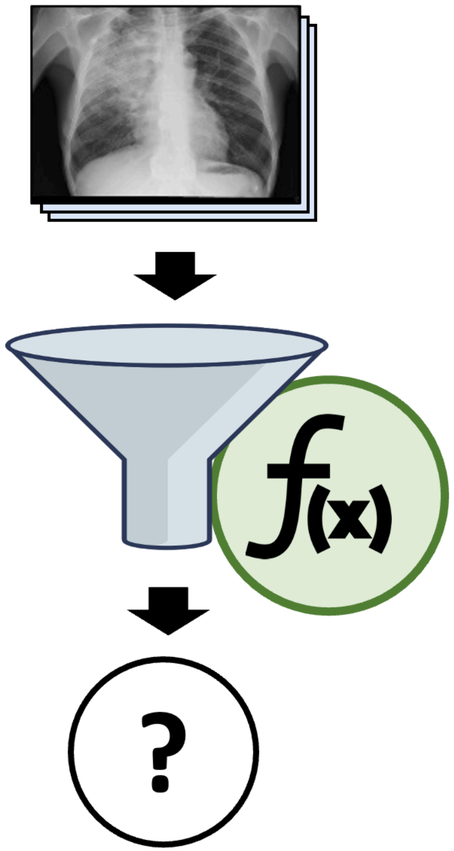
- DLBox checks $f(x)$ developed by an AI developer
- Properties of allowed functions
  1. Allowed functions **can train an AI model**

Infeasibility of Complete Data Protection



- DLBox checks $f(x)$ developed by an AI developer
- Properties of allowed functions
  1. Allowed functions **can train an AI model**
  2. We **cannot get any information of input** (i.e., data sample) from their outputs

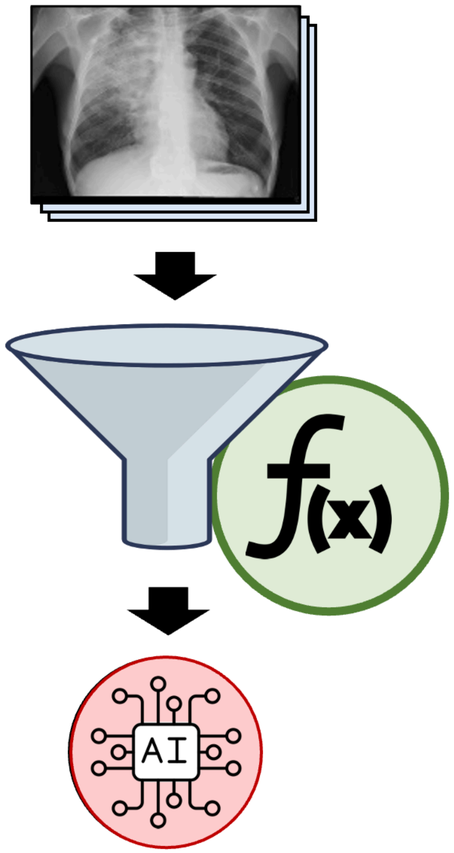
Infeasibility of Complete Data Protection



- DLBox checks $f(x)$ developed by an AI developer
- Properties of allowed functions
  1. Allowed functions **can train an AI model**

     Conflicting goal!
  2. We **cannot get any information of input** (i.e., data sample) from their outputs

Infeasibility of Complete Data Protection



- DLBox checks $f(x)$ developed by an AI developer
- Properties of allowed functions
  1. Allowed functions **can train an AI model**

     Conflicting goal!

  2. We **cannot get any information of input** (i.e., data sample) from their outputs

Trained model always accompanies some sort of data leakage

Infeasibility of Complete Data Protection

- DLBox checks $f(x)$ provided by an AI developer
- Properties of allowed functions

It is impossible to completely prevent data leak

1. Allowed functions **can train an AI model**

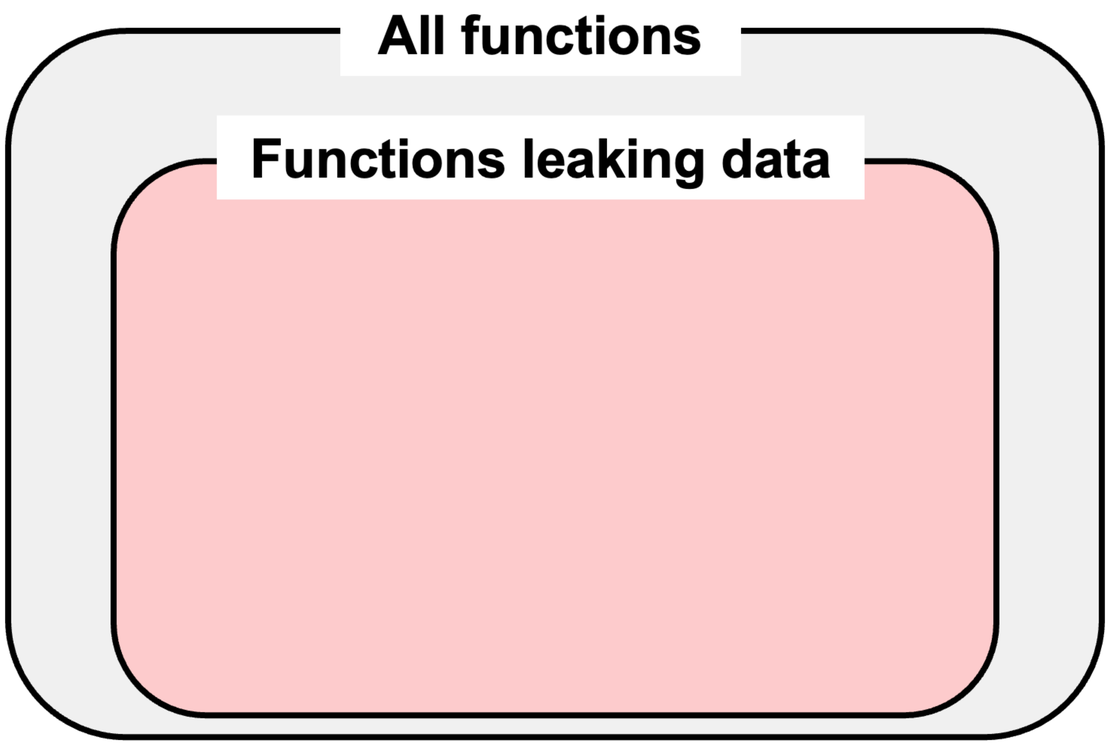
Conflicting goal!

2. We **cannot get any information of input** (i.e., data sample) from their outputs

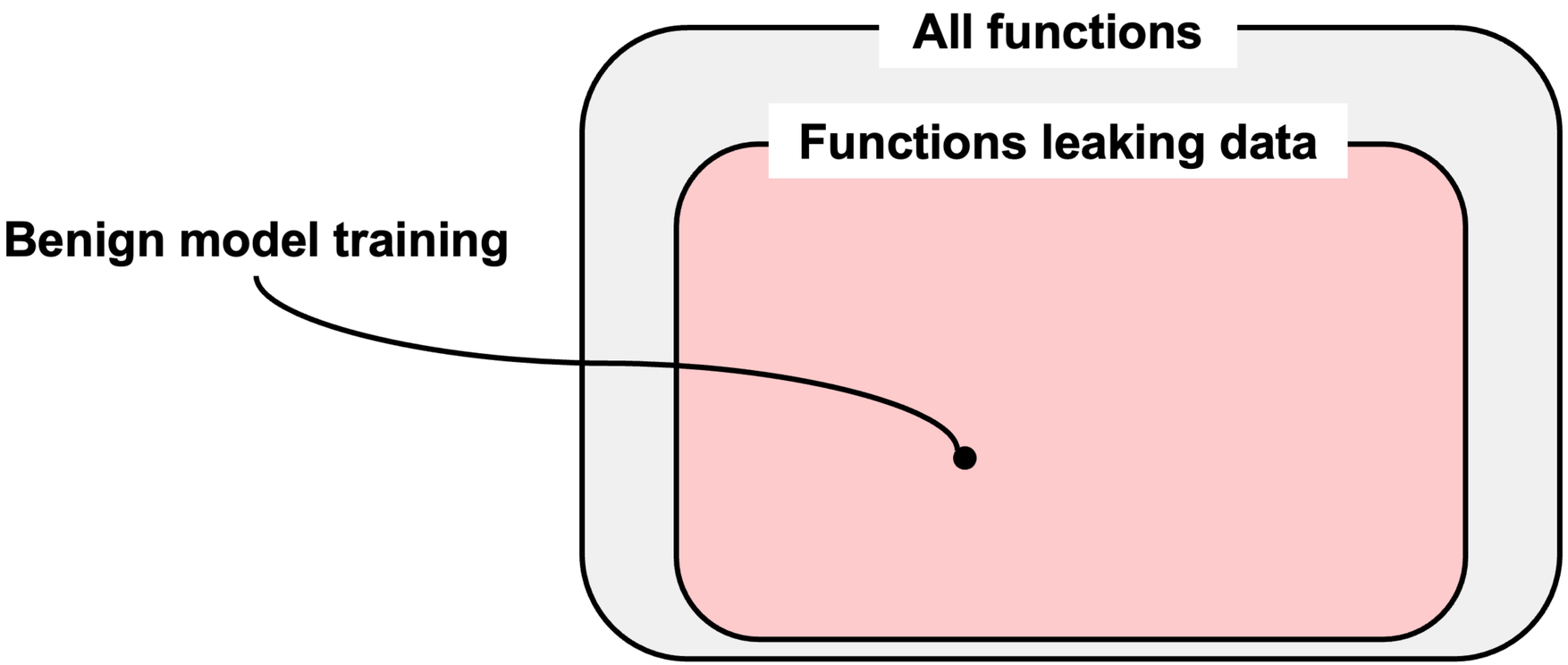
Trained model always accompanies some sort of data leakage

What DLBox Wants to Do

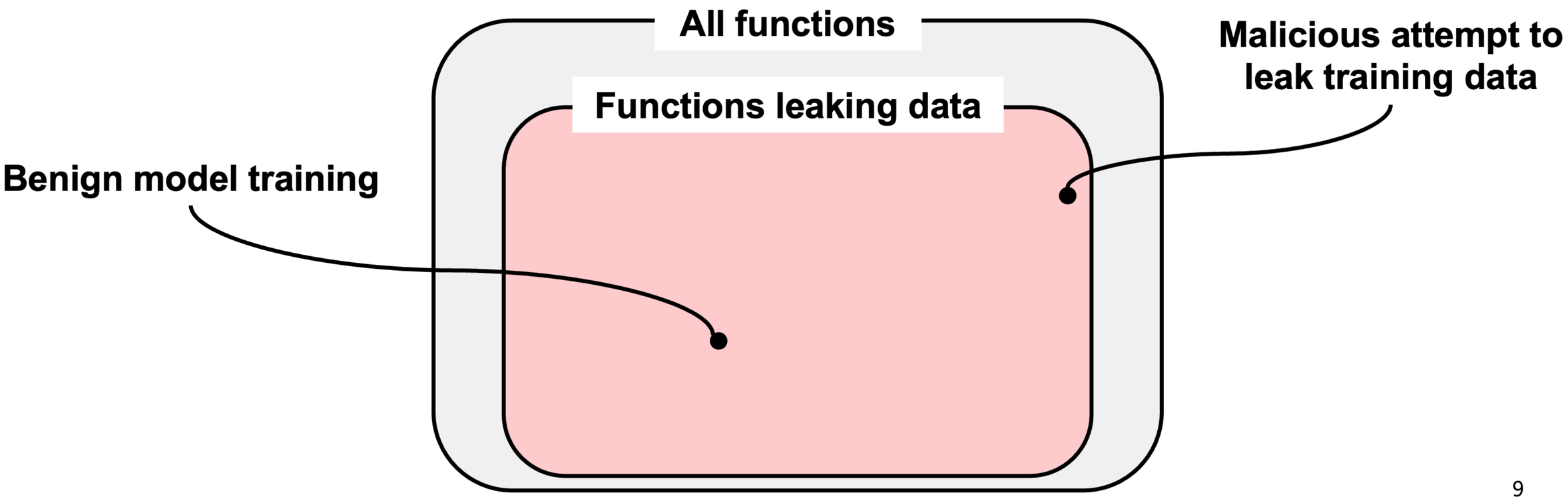
What DLBox Wants to Do



What DLBox Wants to Do

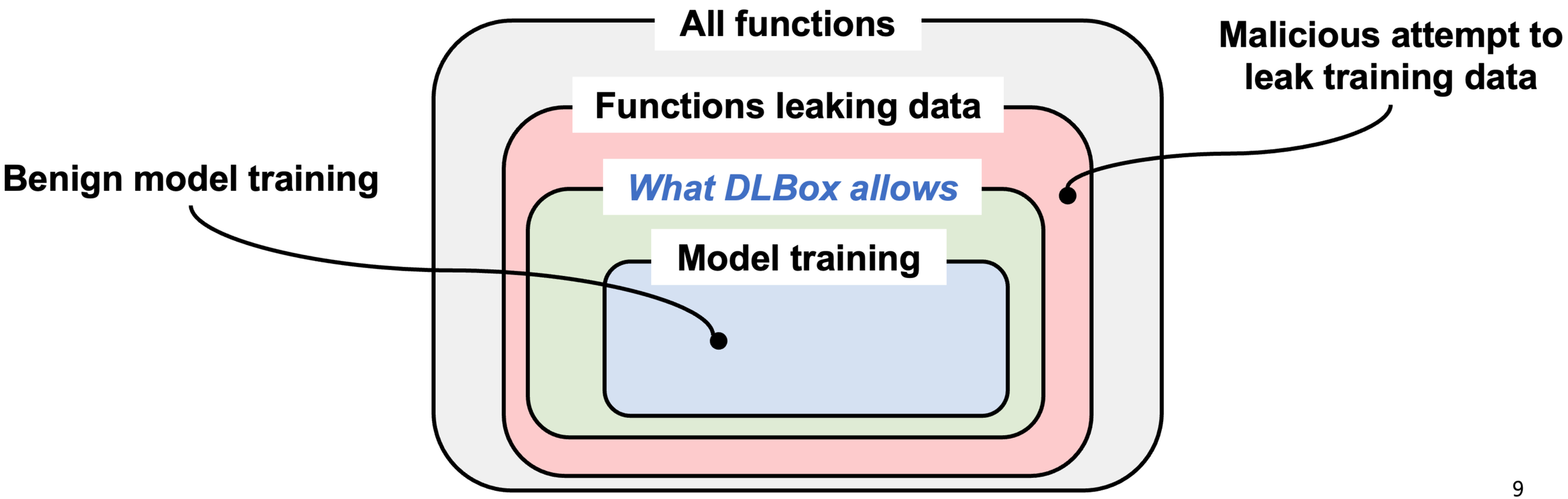


What DLBox Wants to Do



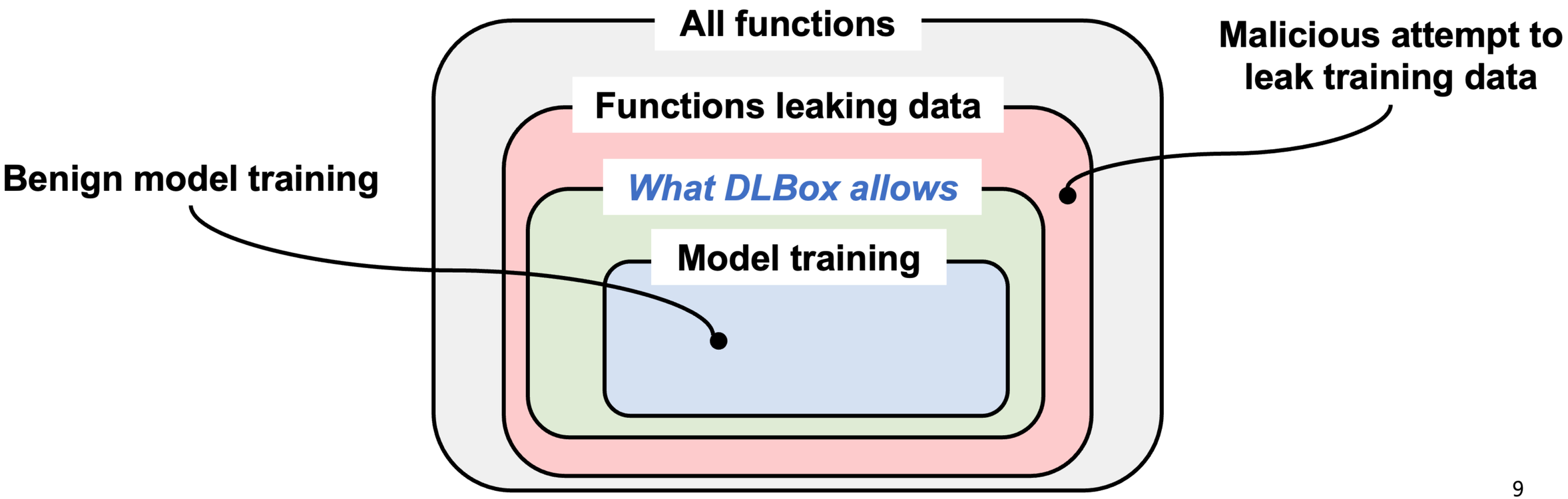
What DLBox Wants to Do

- Allow **Benign Model Training**



What DLBox Wants to Do

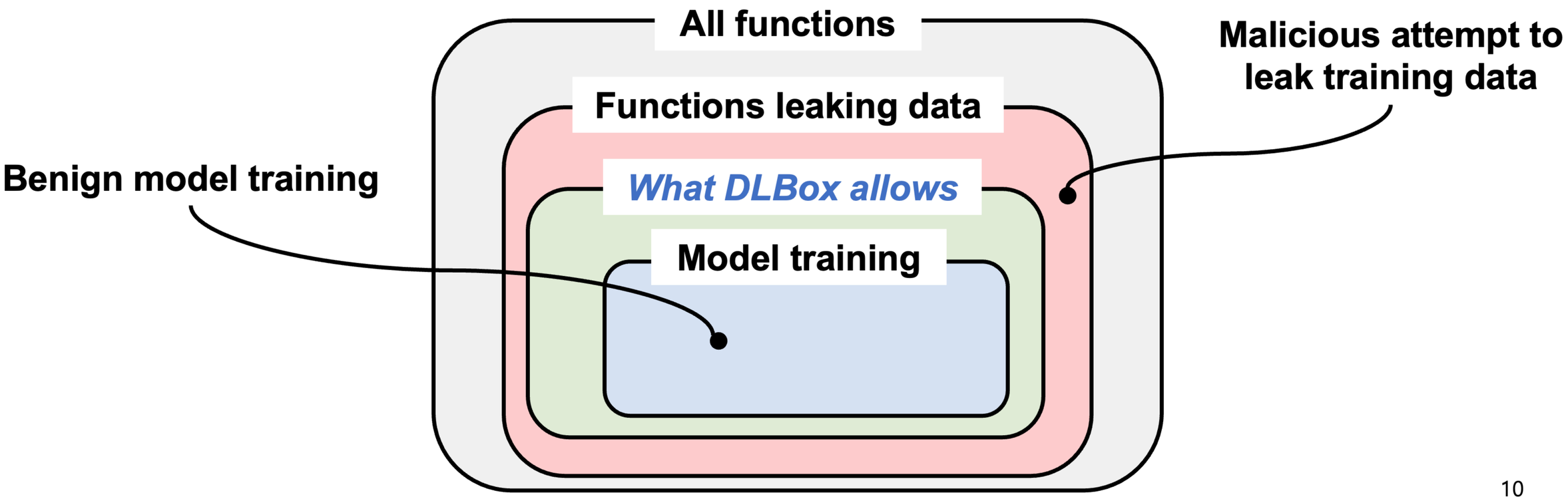
- Allow **Benign Model Training**
- Prevent **Malicious Attempts**



What DLBox Wants to Do

Question What is the Benign Model Training?

- Prevent **Malicious Attempts**



What DLBox Wants to Do

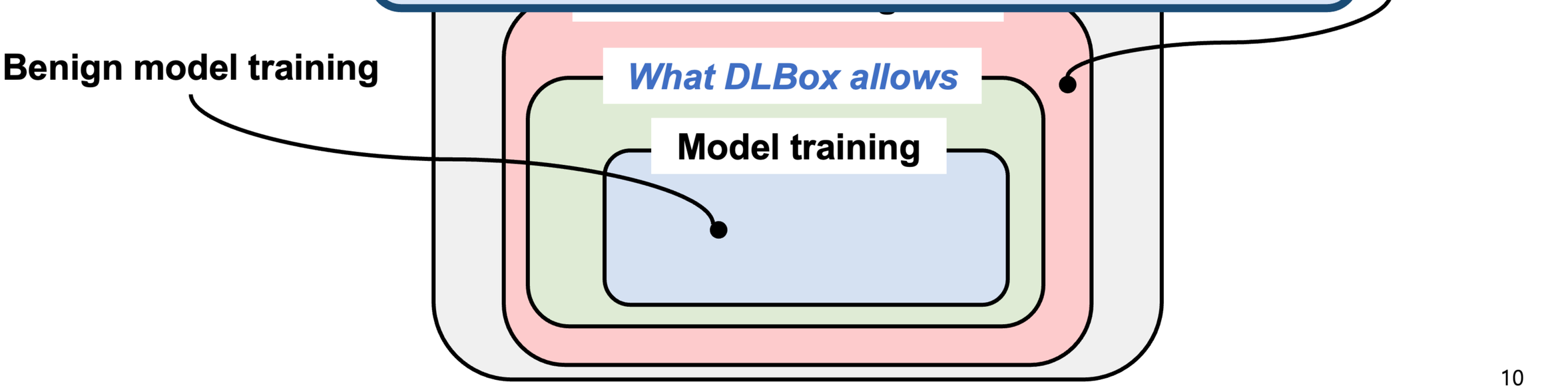
Question

What is the Benign Model Training?

- Prevent Malicious Attempts

Observation

It is a statistical process of learning common patterns from a dataset



What DLBox Wants to Do

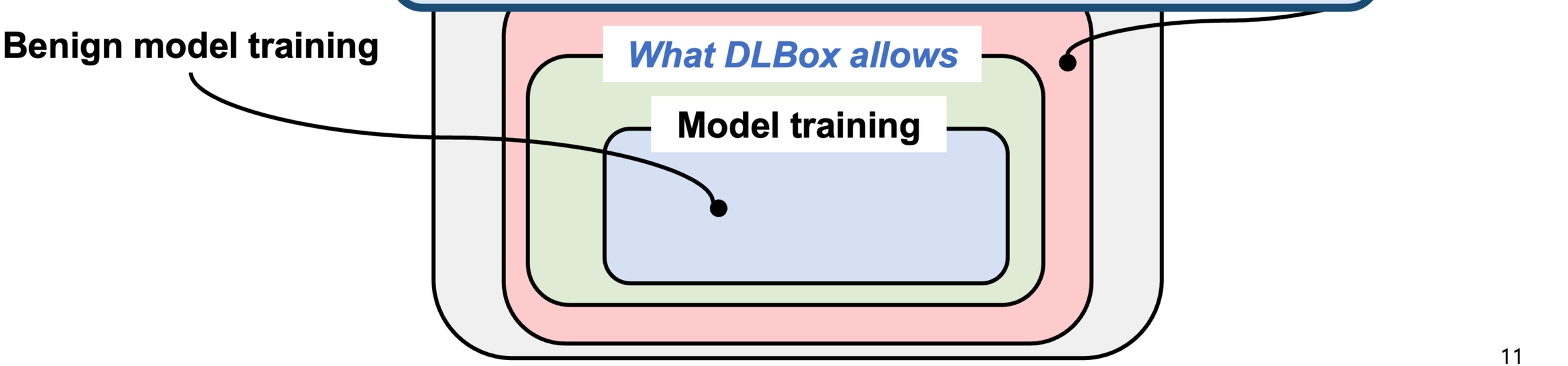
Question

What is the Benign Model Training?

- Prevent Malicious Attempts

Observation

There is no need to prioritize or discriminate a specific sample



What DLBox Wants to Do

Question

What is the Benign Model Training?

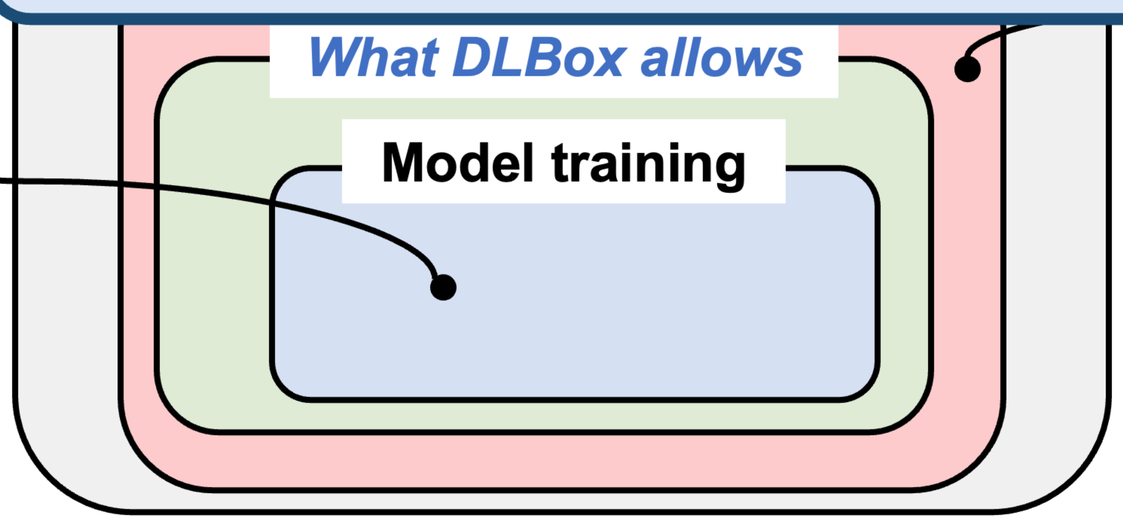
- Prevent Malicious Attempts

Observation

All samples in the dataset should have fair contribution to the model

attempt to
ing data

Benign model training



What DLBox Wants to Do

Question

What is the Benign Model Training?

- Prevent Malicious Attempts

Observation

All samples in the dataset should have fair contribution to the model

attempt to
ing data

Benign model training

DGM-Rules

What DLBox Wants to Do

Question

What is the Benign Model Training?

- Prevent Malicious Attempts

Observation

All samples in the dataset should have fair contribution to the model

attempt to
ing data

Benign model training

DGM-Rules

D ata samples receiving the same augmentation

What DLBox Wants to Do

Question

What is the Benign Model Training?

- Prevent Malicious Attempts

Observation

All samples in the dataset should have fair contribution to the model

attempt to
ing data

Benign model training

DGM-Rules

D ata samples receiving the same augmentation

G radients back propagated from loss value

What DLBox Wants to Do

Question

What is the Benign Model Training?

- Prevent Malicious Attempts

Observation

All samples in the dataset should have fair contribution to the model

attempt to
ing data

Benign model training

DGM-Rules

D ata samples receiving the same augmentation
G radients back propagated from loss value
M odel updated by adjusting all gradients

DGM-Rules: Rules for Benign Model Training

Sample 1

Sample 2

…

Sample **N**

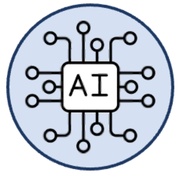
DGM-Rules: Rules for Benign Model Training

Sample 1

Sample 2

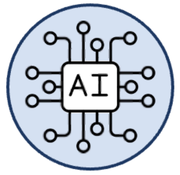
...

Sample **N**

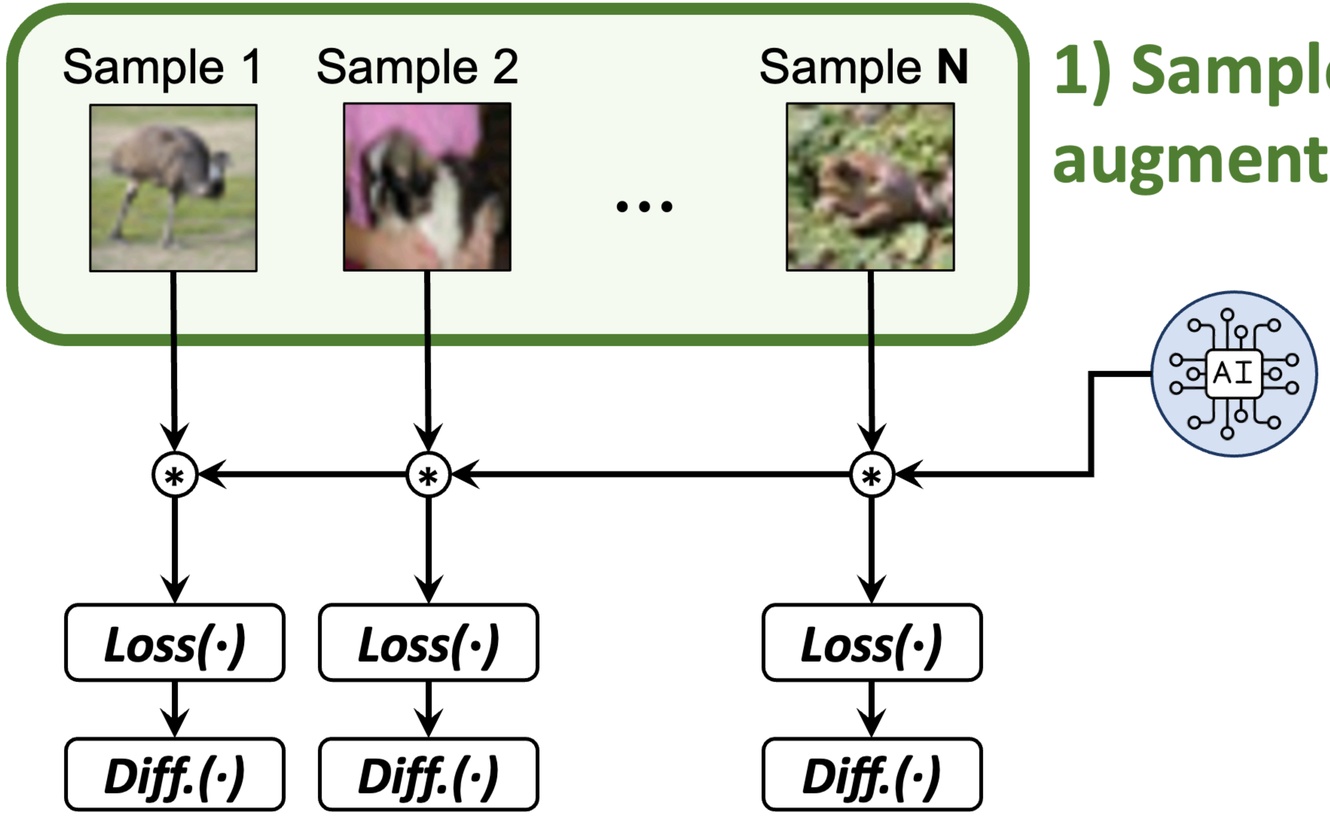


DGM-Rules: Rules for Benign Model Training

1) Samples augmented.

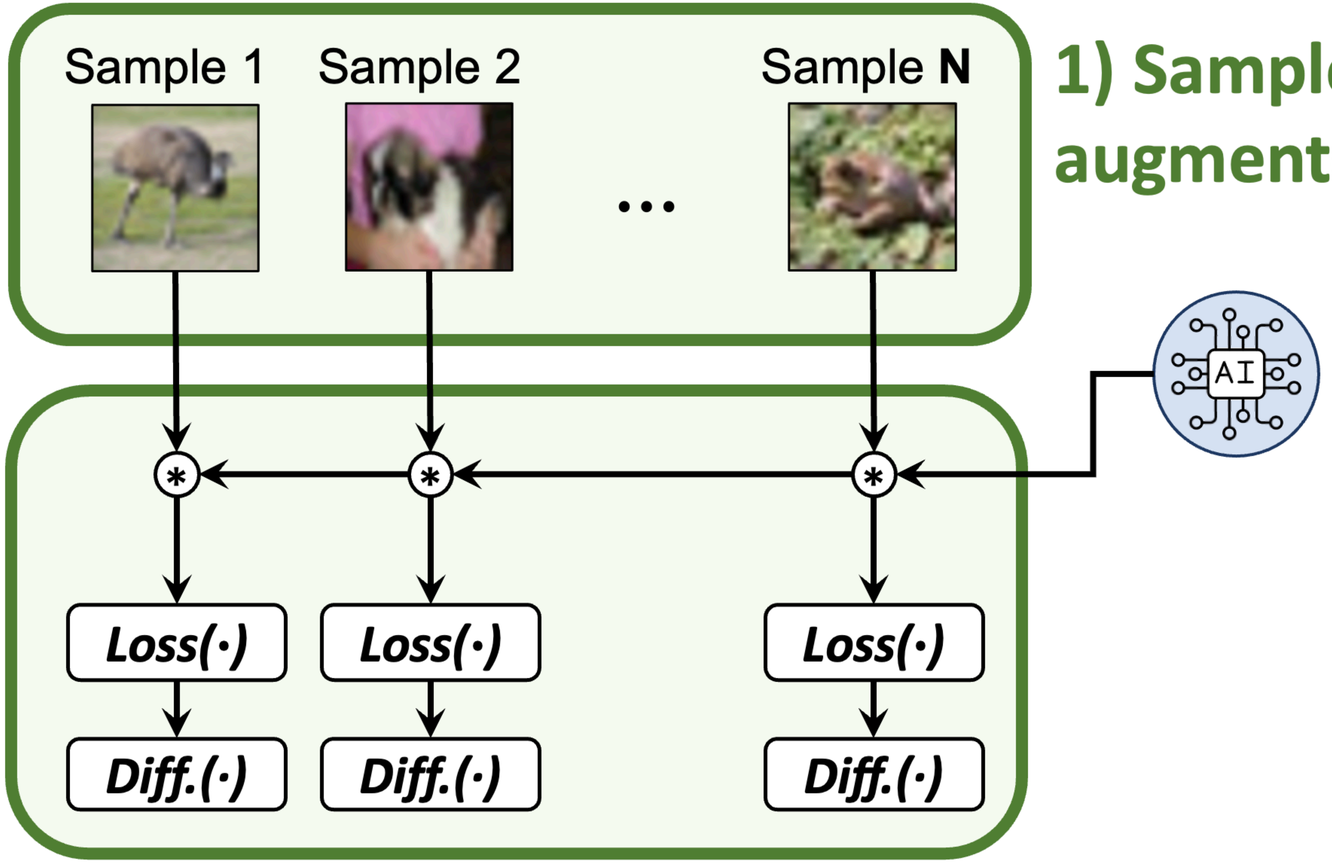


DGM-Rules: Rules for Benign Model Training



1) Samples augmented.

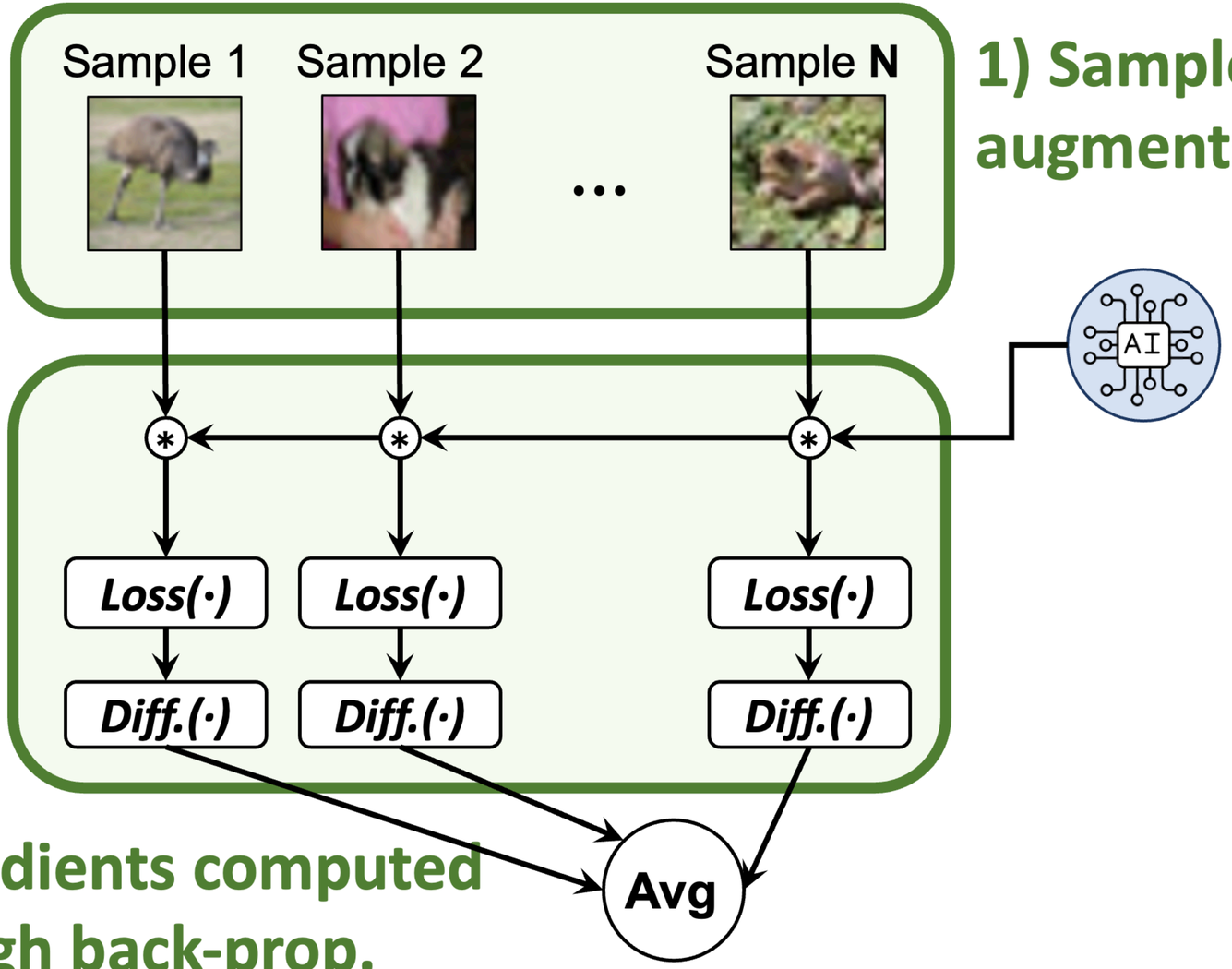
DGM-Rules: Rules for Benign Model Training



1) Samples augmented.

2) Gradients computed through back-prop.

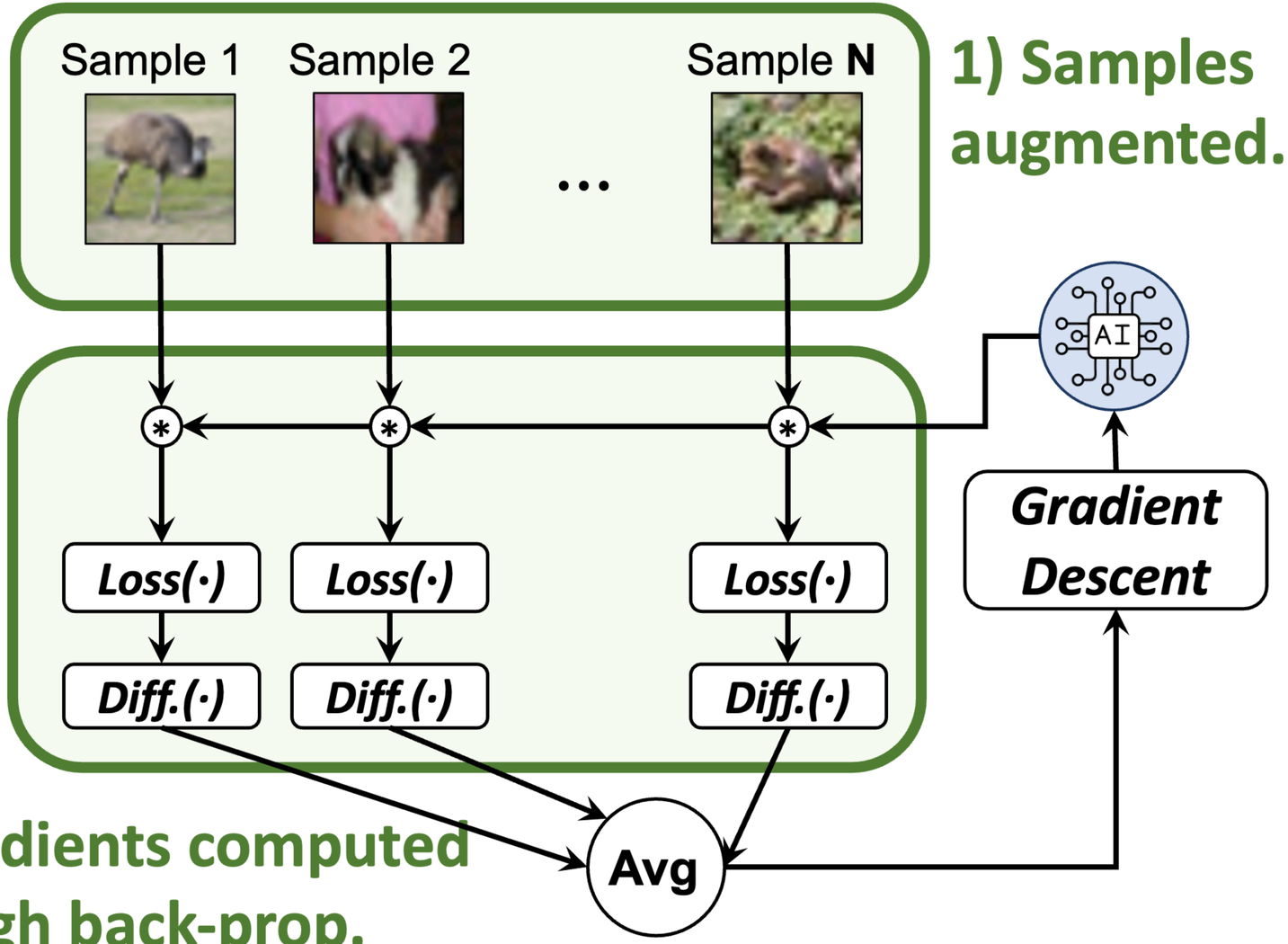
DGM-Rules: Rules for Benign Model Training



1) Samples augmented.

2) Gradients computed through back-prop.

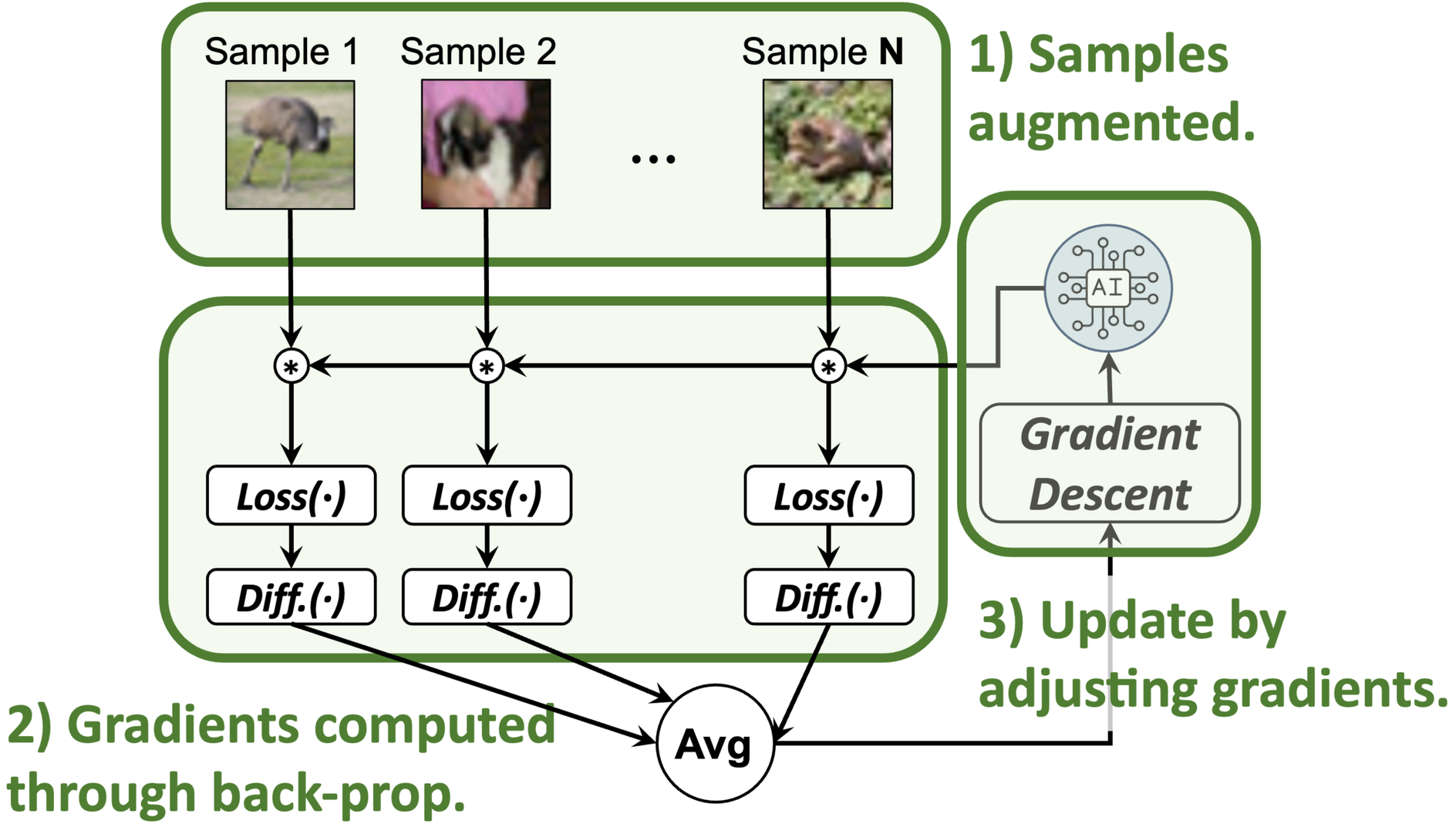
DGM-Rules: Rules for Benign Model Training



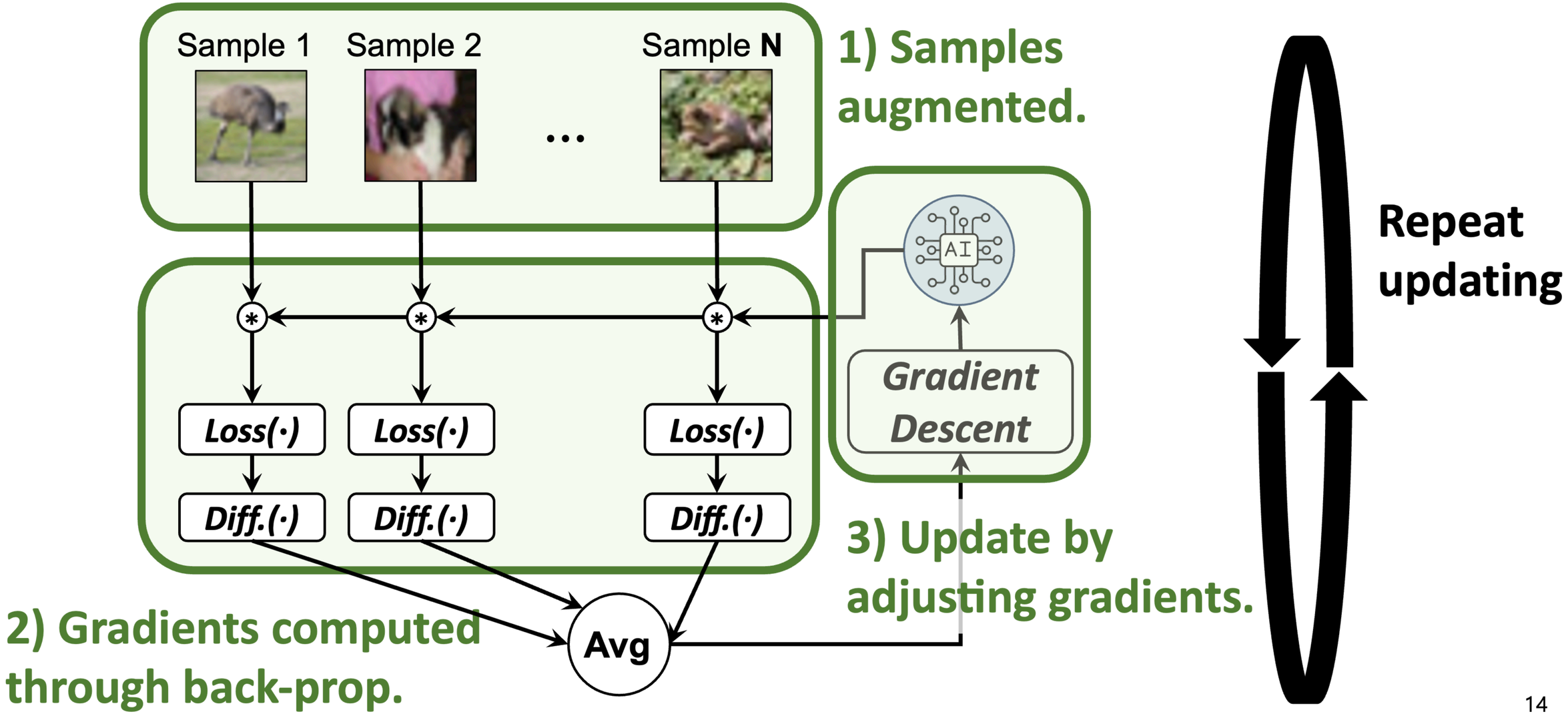
1) Samples augmented.

2) Gradients computed through back-prop.

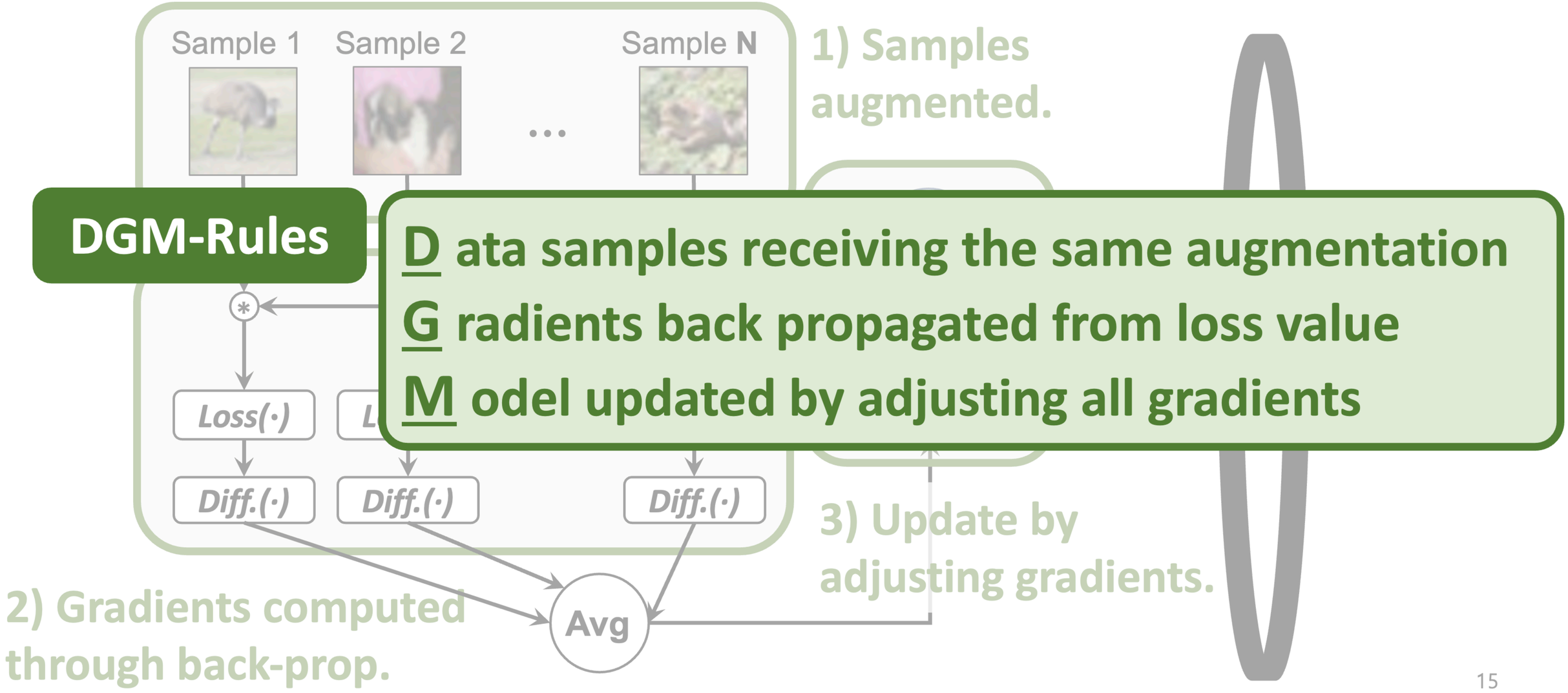
DGM-Rules: Rules for Benign Model Training



DGM-Rules: Rules for Benign Model Training



DGM-Rules: Rules for Benign Model Training



DLBox: Enforcing DGM-Rules

DLBox: Enforcing DGM-Rules

DL-Sandbox

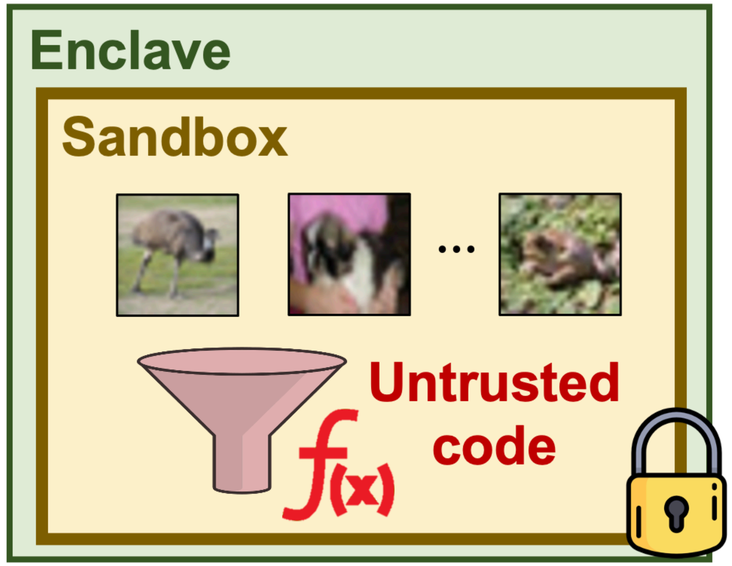
DL-Oracle

DLBox: Enforcing DGM-Rules

DL-Sandbox

Sandbox AI developer's code

- Sanitizing all outgoing data flows of training data



Cannot pull the data without approval

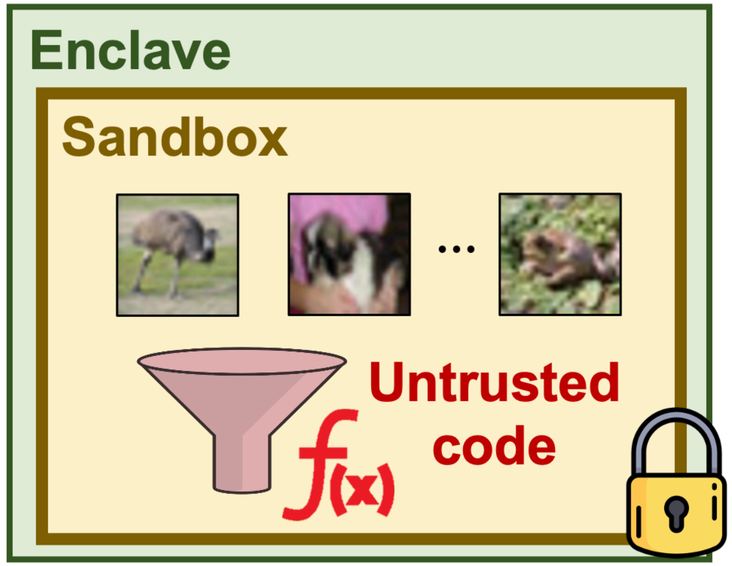
DL-Oracle

DLBox: Enforcing DGM-Rules

DL-Sandbox

Sandbox AI developer's code

- Sanitizing all outgoing data flows of training data

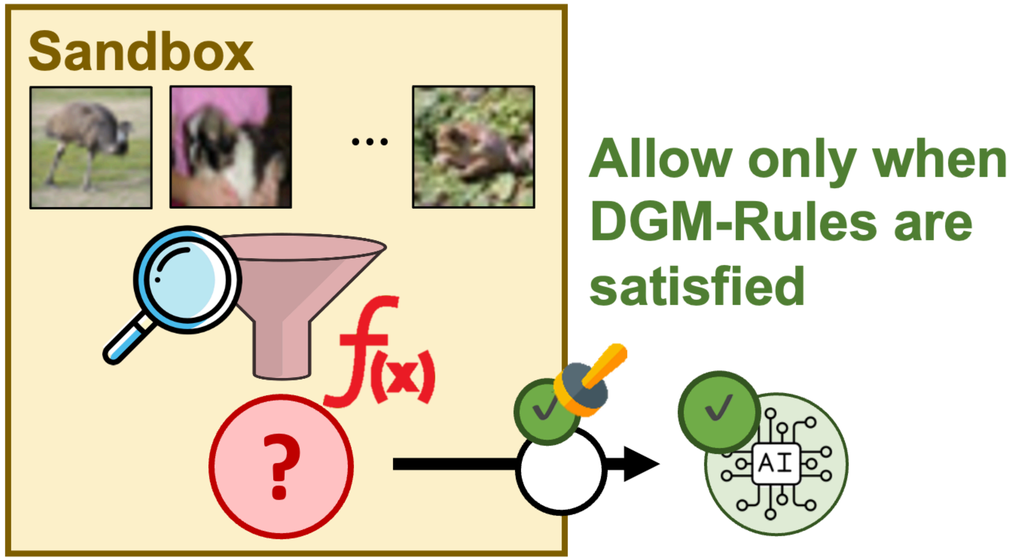


Cannot pull the data without approval

DL-Oracle

Taint analyze computations on data

- Checking the data flows satisfy **DGM-Rules**



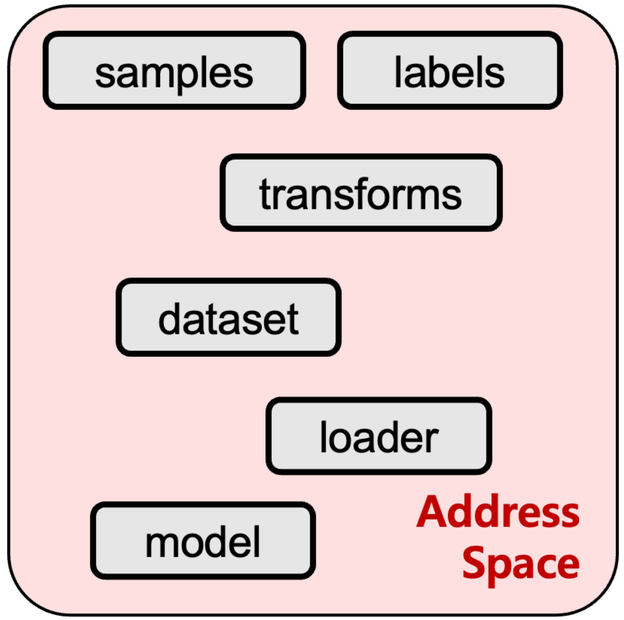
Allow only when DGM-Rules are satisfied

DL-Sandbox: Sandboxing AI Developer's Code

```
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```

DL-Sandbox: Sandboxing AI Developer's Code

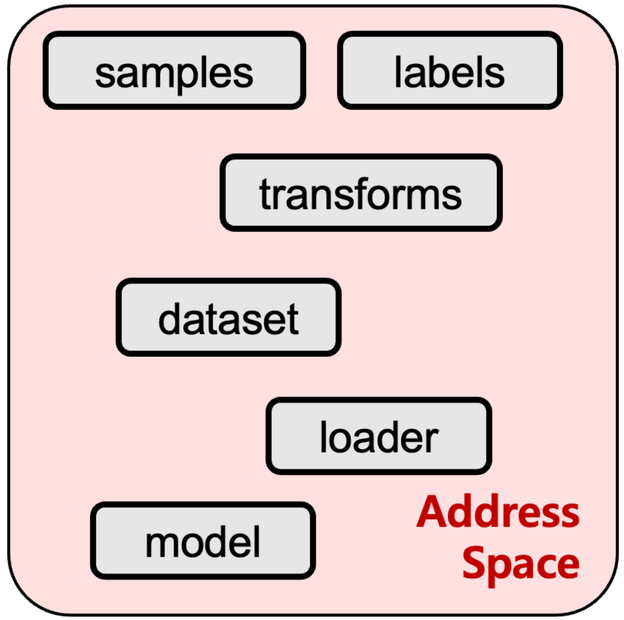
```python
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```



Address Space w/o DL-Sandbox (**UNSAFE**)

DL-Sandbox: Sandboxing AI Developer's Code

```python
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```

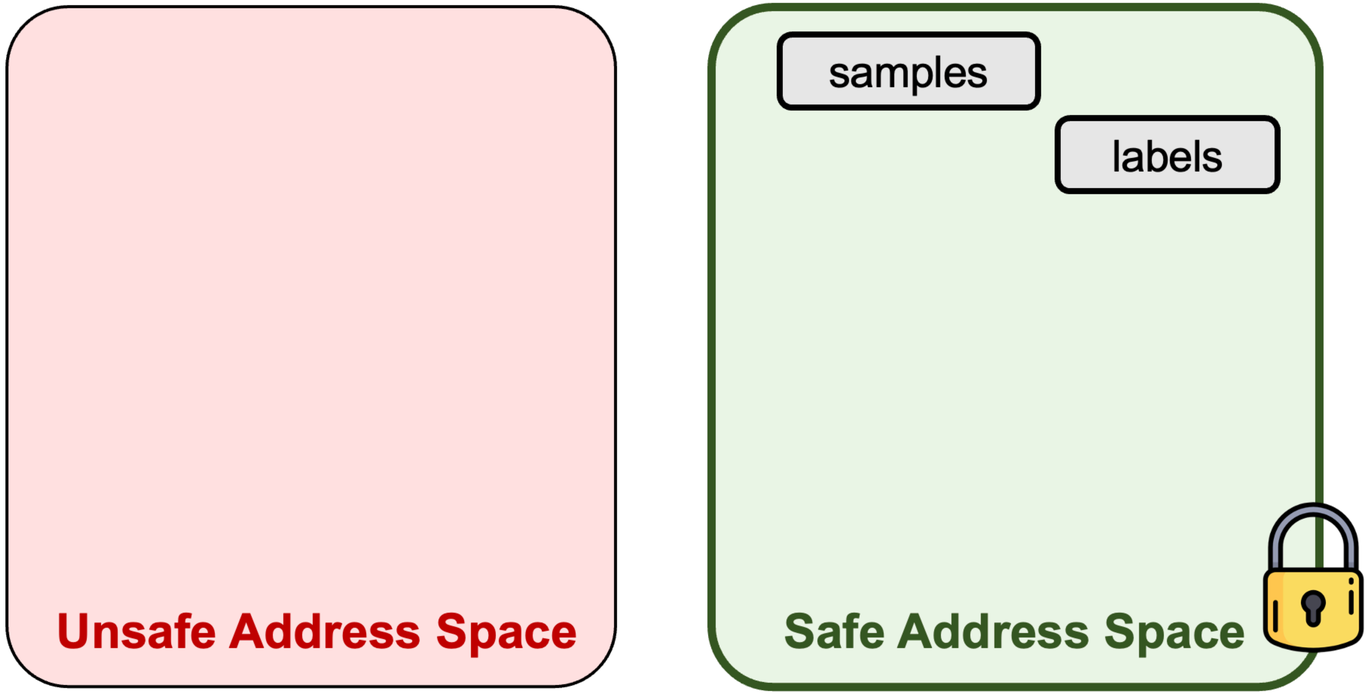


Address Space w/o DL-Sandbox (**UNSAFE**)

AI developers have
full control over the data

DL-Sandbox: Sandboxing AI Developer's Code

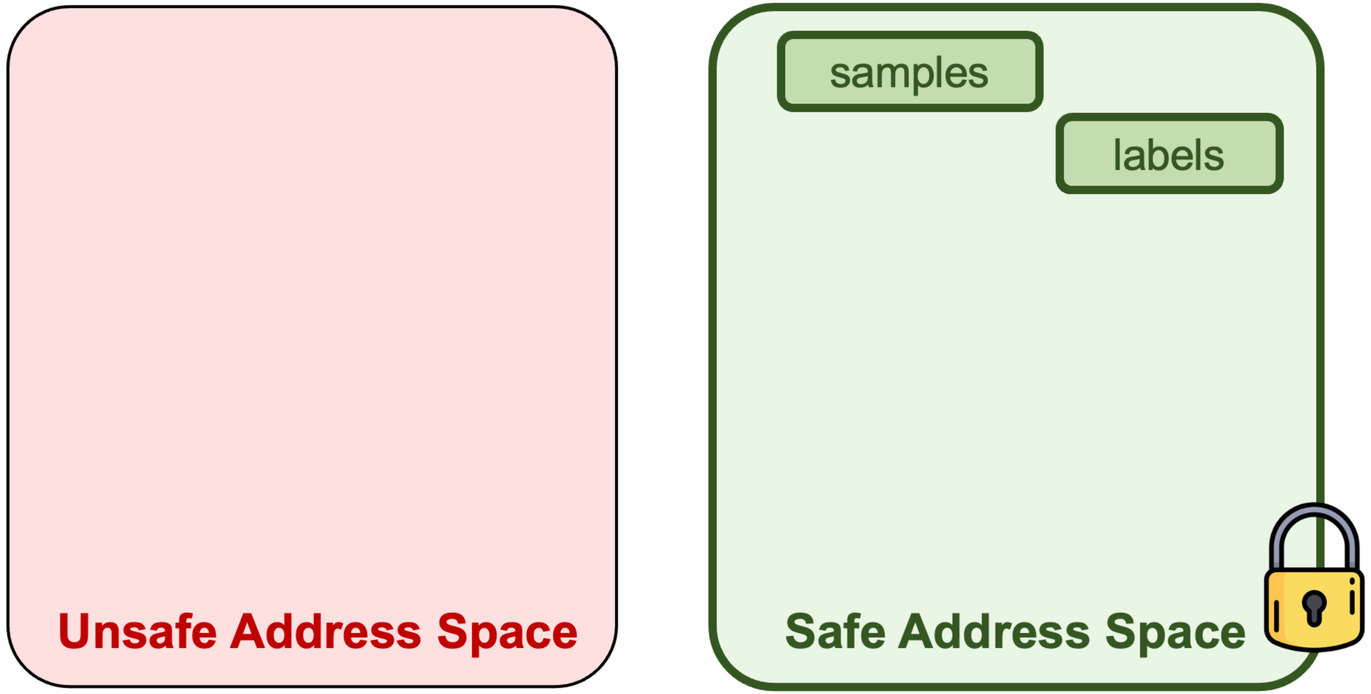
```python
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```



Isolated Address Space by DL-Sandbox (**SAFE**)

DL-Sandbox: Sandboxing AI Developer's Code

```python
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```

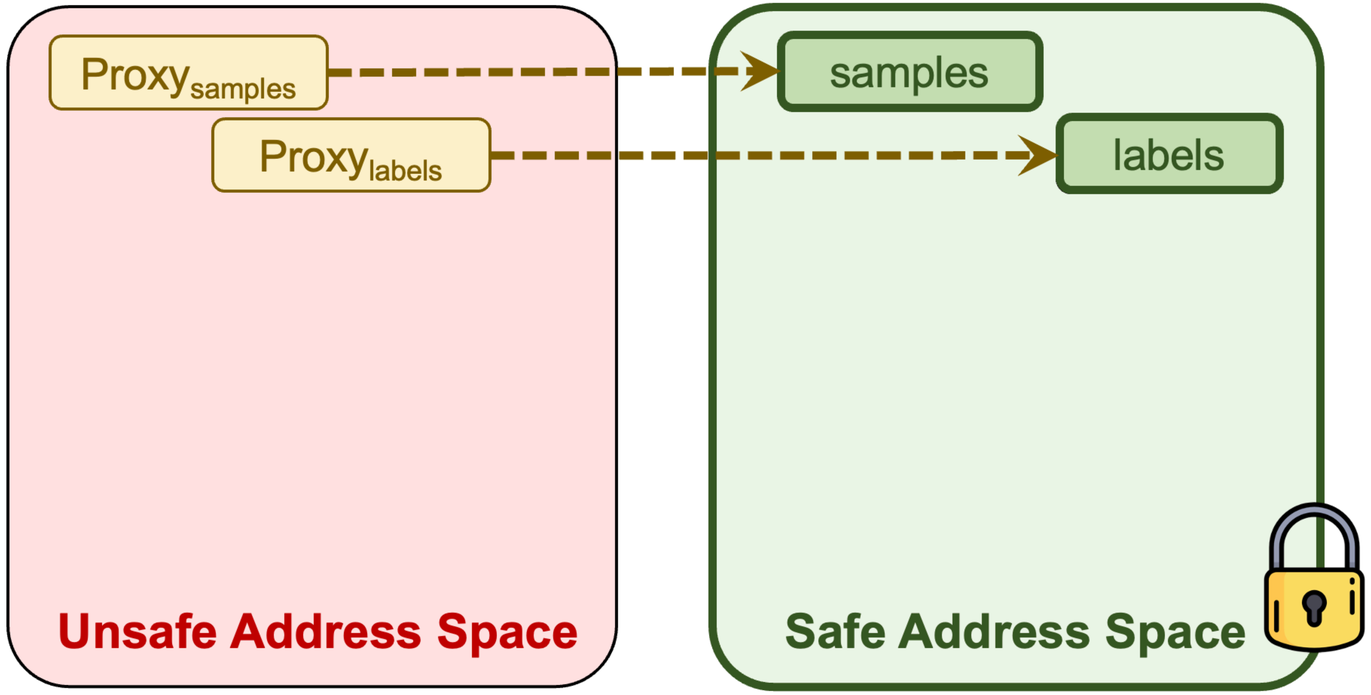


Isolated Address Space by DL-Sandbox (**SAFE**)

Attackers **cannot directly access or modify** data

DL-Sandbox: Sandboxing AI Developer's Code

```python
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```



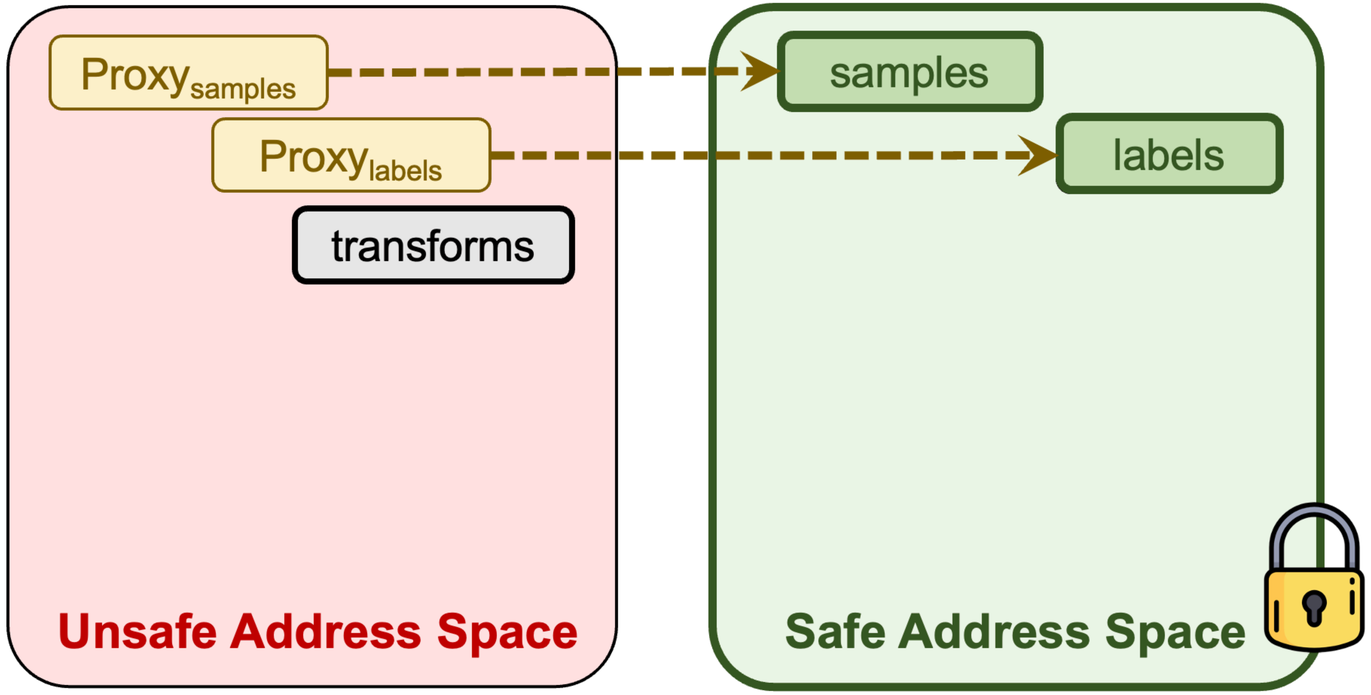
Isolated Address Space by DL-Sandbox (**SAFE**)

Attackers **cannot directly access or modify** data

Proxy to reference data

DL-Sandbox: Sandboxing AI Developer's Code

```python
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```



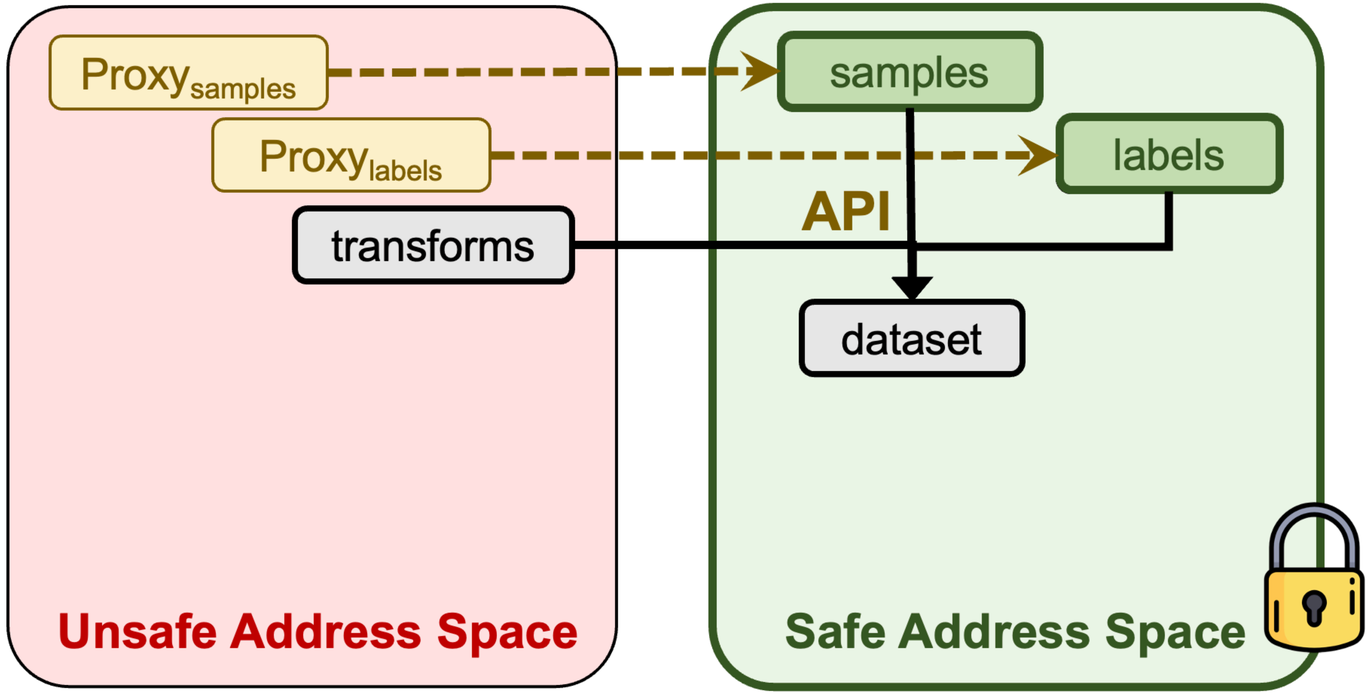
Isolated Address Space by DL-Sandbox (**SAFE**)

Attackers **cannot directly access or modify** data

Proxy to reference data

DL-Sandbox: Sandboxing AI Developer's Code

```python
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```



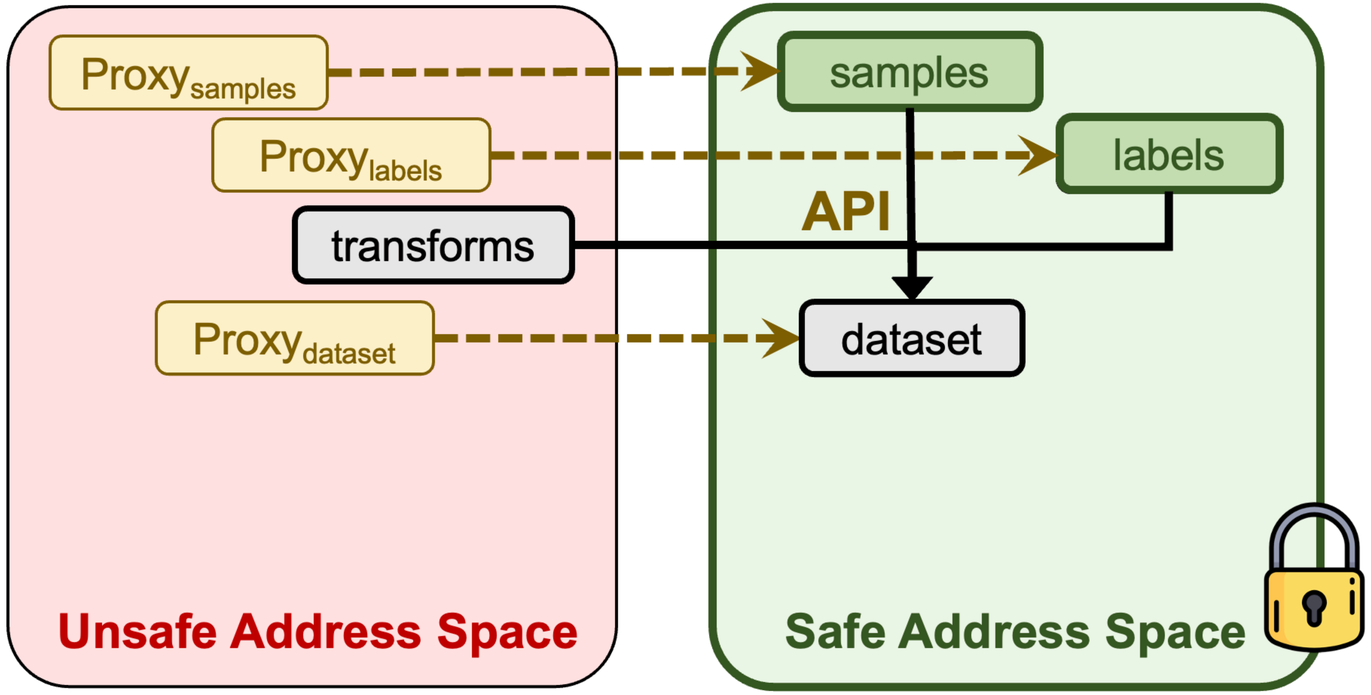
Isolated Address Space by DL-Sandbox (**SAFE**)

Attackers **cannot directly access or modify** data

Proxy to reference data & Operations through **API**

DL-Sandbox: Sandboxing AI Developer's Code

```python
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```



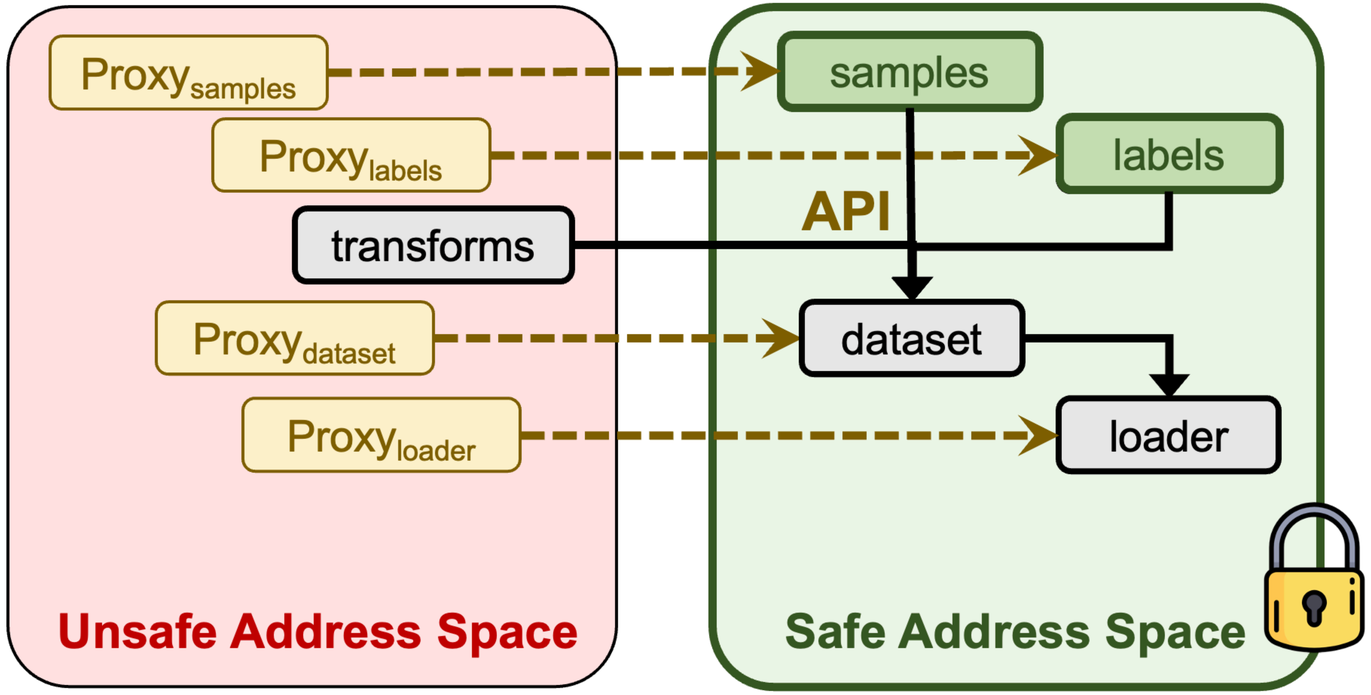
Isolated Address Space by DL-Sandbox (**SAFE**)

Attackers **cannot directly access or modify** data

Proxy to reference data & Operations through **API**

DL-Sandbox: Sandboxing AI Developer's Code

```python
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```



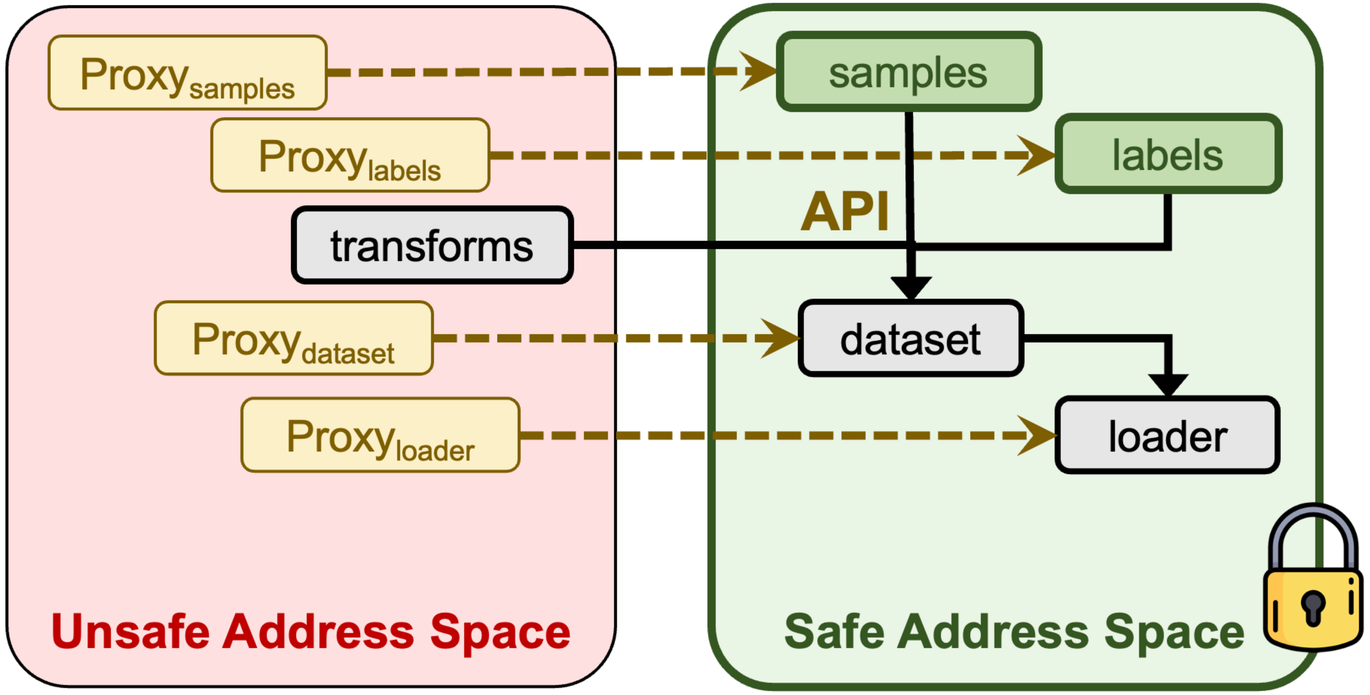
Isolated Address Space by DL-Sandbox (**SAFE**)

Attackers **cannot directly access or modify** data

Proxy to reference data & Operations through **API**

DL-Oracle: Taint Tracking Operations on Data

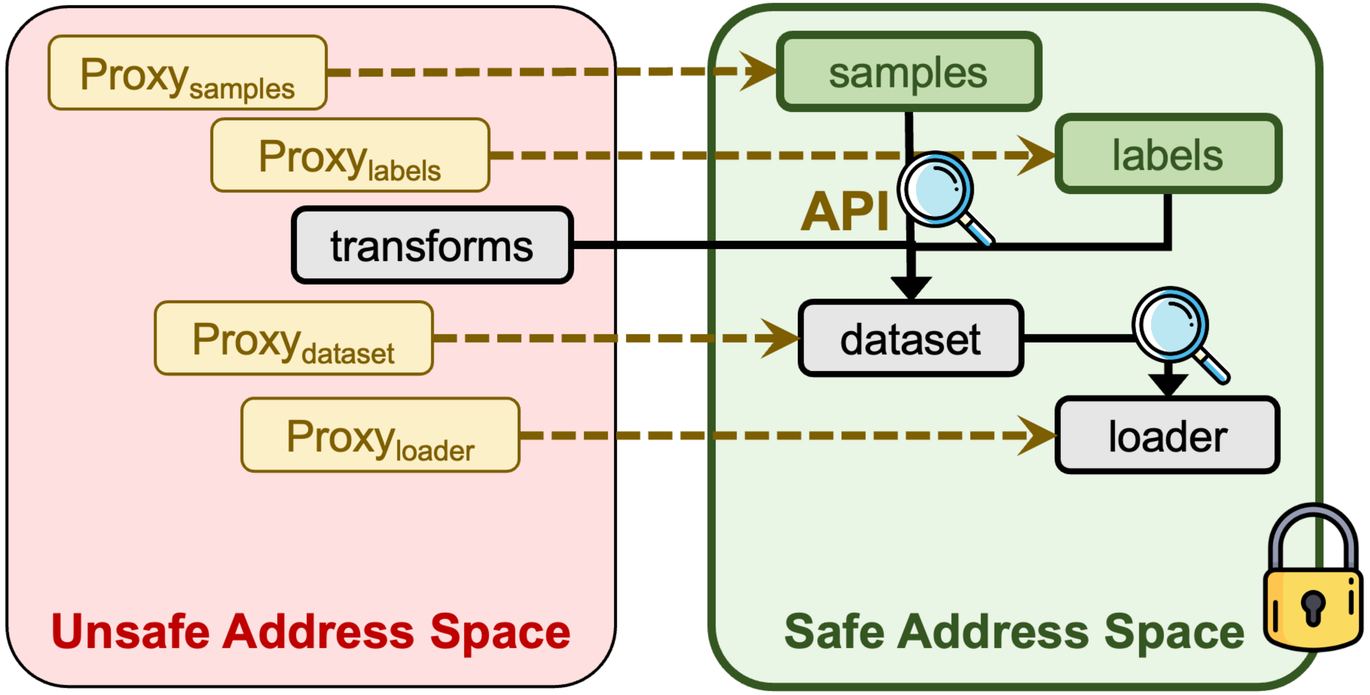
```python
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```



Isolated Address Space by DL-Sandbox (**SAFE**)

DL-Oracle: Taint Tracking Operations on Data

```python
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```

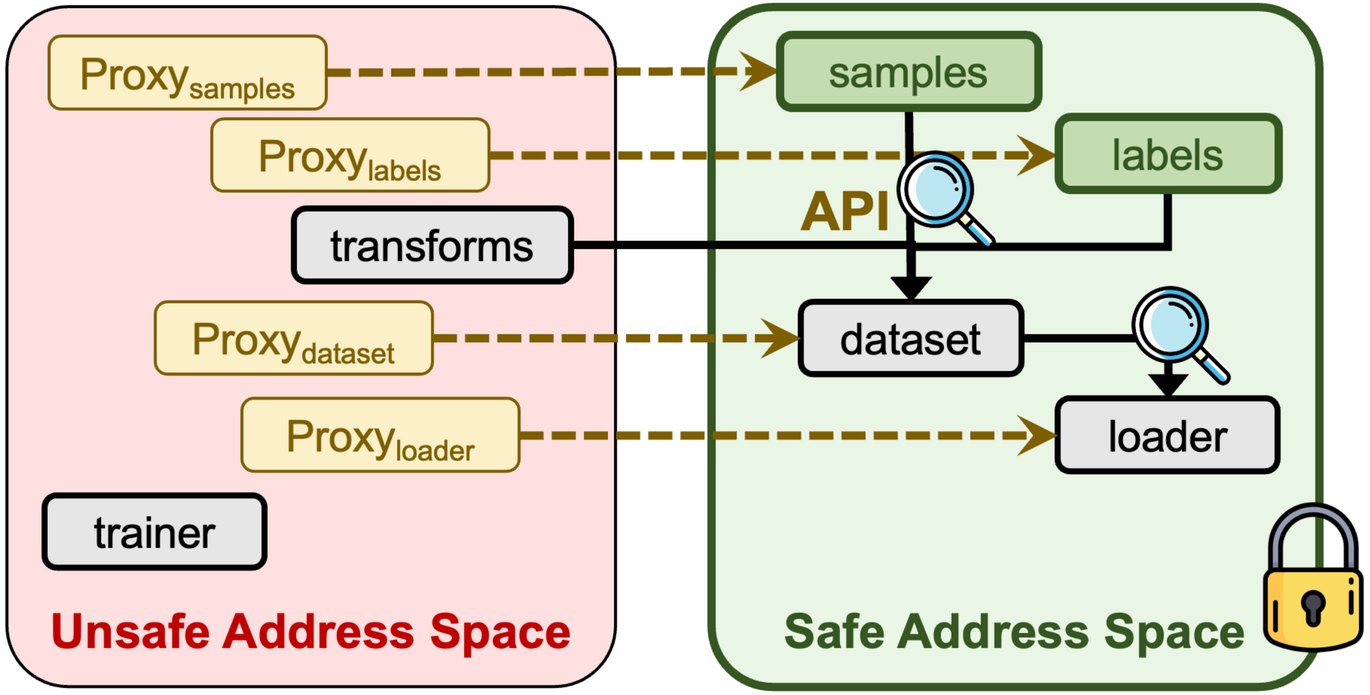


Isolated Address Space by DL-Sandbox (**SAFE**)

Taint track all operations on data

DL-Oracle: Taint Tracking Operations on Data

```python
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```

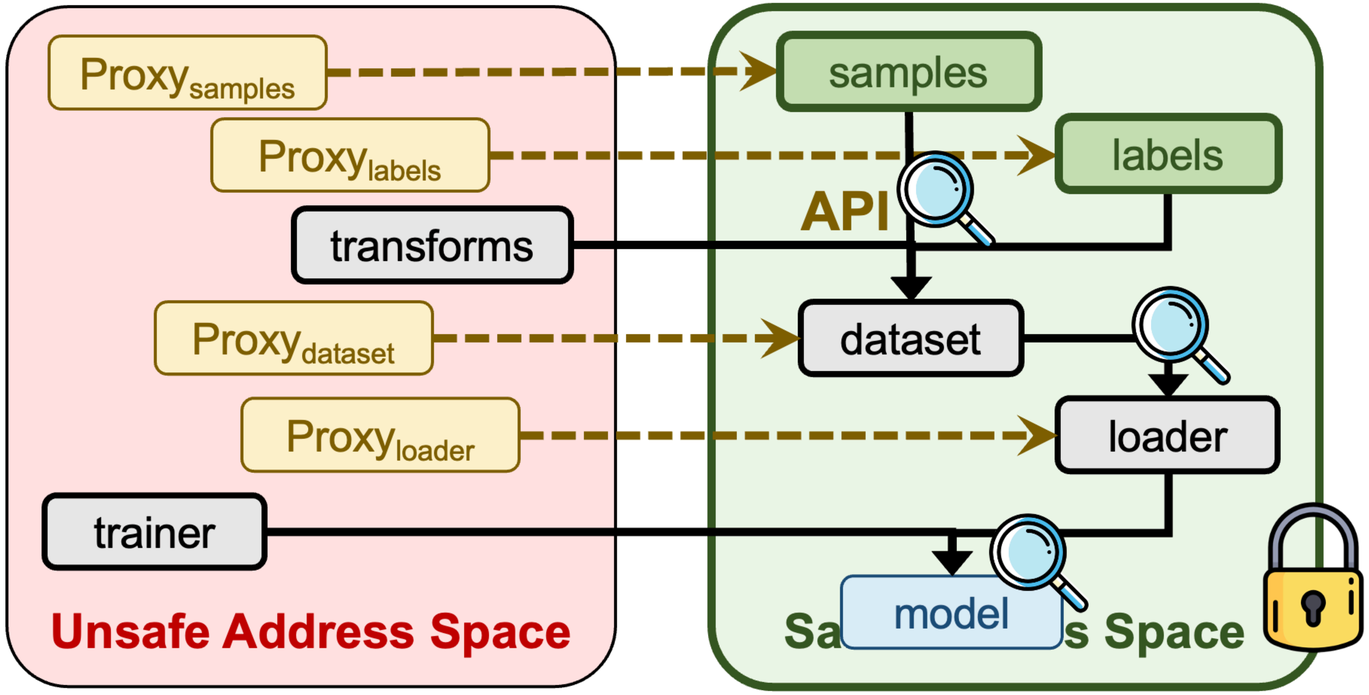


Isolated Address Space by DL-Sandbox (**SAFE**)

Taint track all operations on data

DL-Oracle: Taint Tracking Operations on Data

```
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```

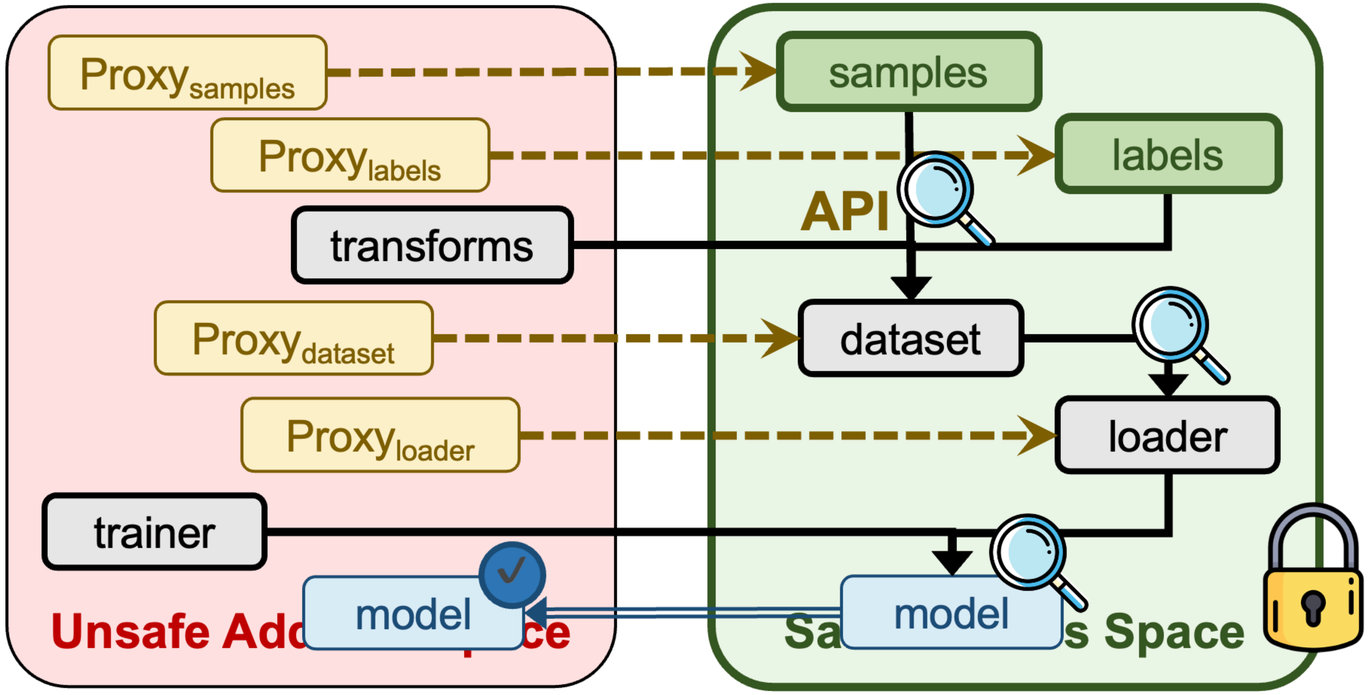


Isolated Address Space by DL-Sandbox (**SAFE**)

Taint track all operations on data

DL-Oracle: Taint Tracking Operations on Data

```python
samples, labels = sample_init()

print("[mnist] start running...")

DATA_MEAN = (samples / 255.0).mean(axis=(0, 1, 2))
DATA_STD  = (samples / 255.0).std(axis=(0, 1, 2))

print(f"[mnist] data_mean: {DATA_MEAN}, data_std: {DATA_STD}")

transforms = Compose([
  Tensor.numpy,
  Image.fromarray,
  ToTensor(),
  Normalize(DATA_MEAN, DATA_STD)
])

print("[mnist] fit model...")

dataset = TensorDataset(samples, labels, transforms=transforms)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

trainer = pytorch_lightning.Trainer(gpus=1, max_epochs=10)
model = LitClassifier()

trainer.fit(model, train_dataloaders=loader)

print("[mnist] done")
```



Isolated Address Space by DL-Sandbox (**SAFE**)

Taint track all operations on data

Return model only when **DGM-Rules are satisfied**

Implementation of DLBox

- Implemented on PyTorch library to support real-world datasets and ML models.
- Used confidential computing (i.e., AMD SEVSNP) to protect integrity and confidentiality.

Security Evaluation

- Number of samples leaked by one transfer of the (seemingly trained) model

Security Evaluation

- Number of samples leaked by one transfer of the (seemingly trained) model

Security measure	Attack method	Training dataset		
		Cifar10	UTKFace	ChestXray

Security Evaluation

- Number of samples leaked by one transfer of the (seemingly trained) model

Security measure	Attack method	Training dataset		
		Cifar10	UTKFace	ChestXray
Baseline (no security)	Leak-as-itself	60000 (100%)	20000 (100%)	15000 (100%)

Security Evaluation

- Number of samples leaked by one transfer of the (seemingly trained) model

Security measure	Attack method	Training dataset		
		Cifar10	UTKFace	ChestXray
Baseline (no security)	Leak-as-itself	60000 (100%)	20000 (100%)	15000 (100%)
FormatEnforce	Embed-to-model	17060	5685	1853

Security Evaluation

- Number of samples leaked by one transfer of the (seemingly trained) model

Security measure	Attack method	Training dataset		
		Cifar10	UTKFace	ChestXray
Baseline (no security)	Leak-as-itself	60000 (100%)	20000 (100%)	15000 (100%)
FormatEnforce	Embed-to-model	17060	5685	1853
AccuracyEnforce	Bias-to-be-data	2470	113	53

Security Evaluation

- Number of samples leaked by one transfer of the (seemingly trained) model

Security measure	Attack method	Training dataset		
		Cifar10	UTKFace	ChestXray
Baseline (no security)	Leak-as-itself	60000 (100%)	20000 (100%)	15000 (100%)
FormatEnforce	Embed-to-model	17060	5685	1853
AccuracyEnforce	Bias-to-be-data	2470	113	53
ManualCheck	Invert-gradients	4	2	1

Security Evaluation

- Number of samples leaked by one transfer of the (seemingly trained) model

Security measure	Attack method	Training dataset		
		Cifar10	UTKFace	ChestXray
Baseline (no security)	Leak-as-itself	60000 (100%)	20000 (100%)	15000 (100%)
FormatEnforce	Embed-to-model	17060	5685	1853
AccuracyEnforce	Bias-to-be-data	2470	113	53
ManualCheck	Invert-gradients	4	2	1
DLBox	Model-inversion	-	-	-

Performance Evaluation

- Performance overheads on training image & language models
  - 4% overheads of training time on average

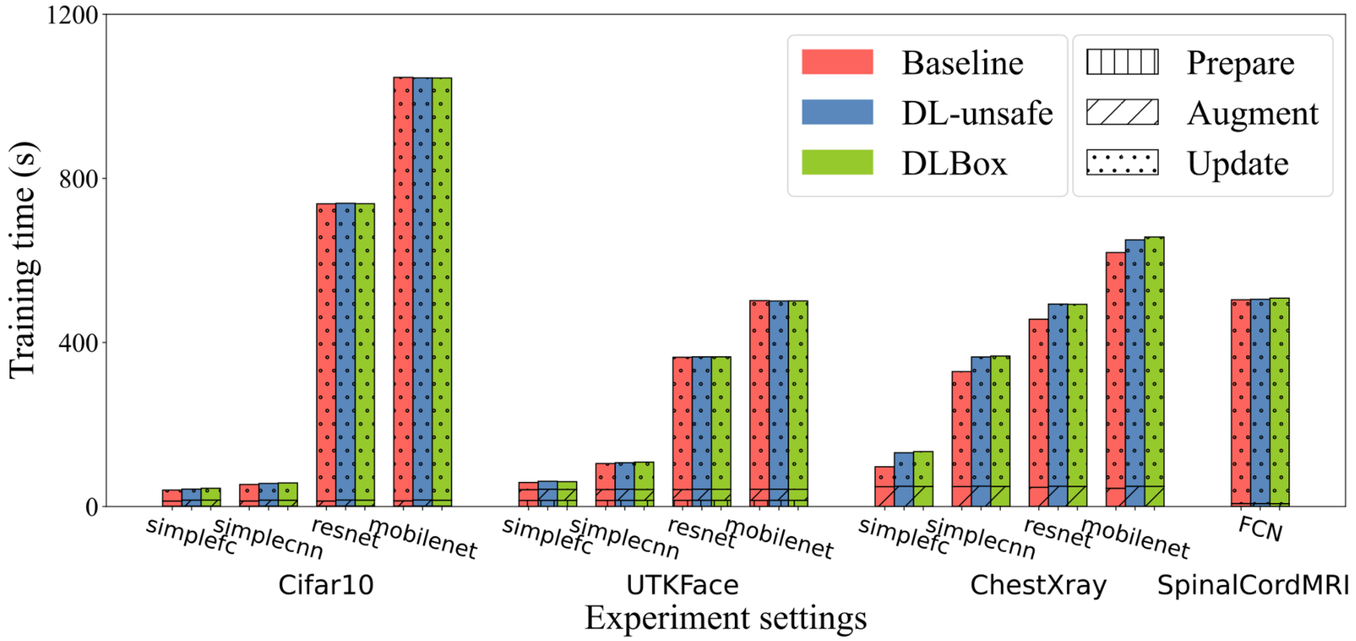
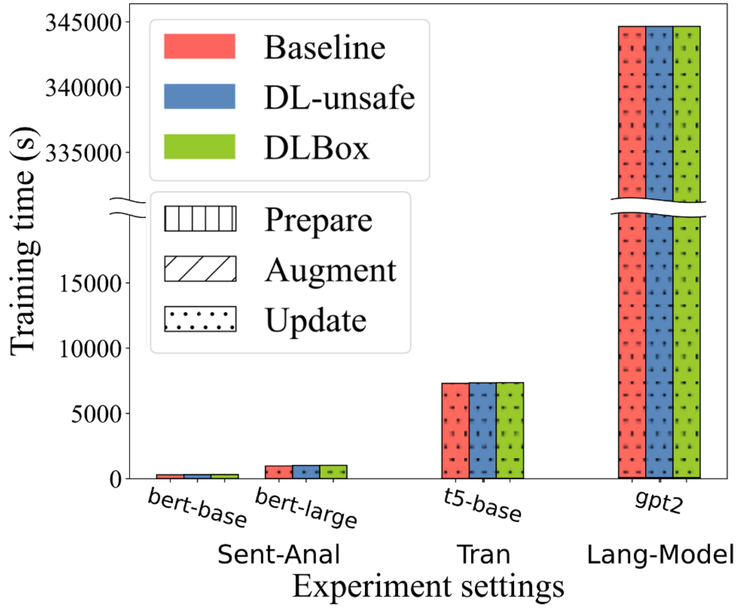


Image models



Language models

DLBox: Conclusion

DLBox: Conclusion

- We design **a new model training framework to protect training data**

DLBox: Conclusion

- We design **a new model training framework to protect training data**
- **DLBox** can foster the data sharing by protecting the right of data owners

Thank you