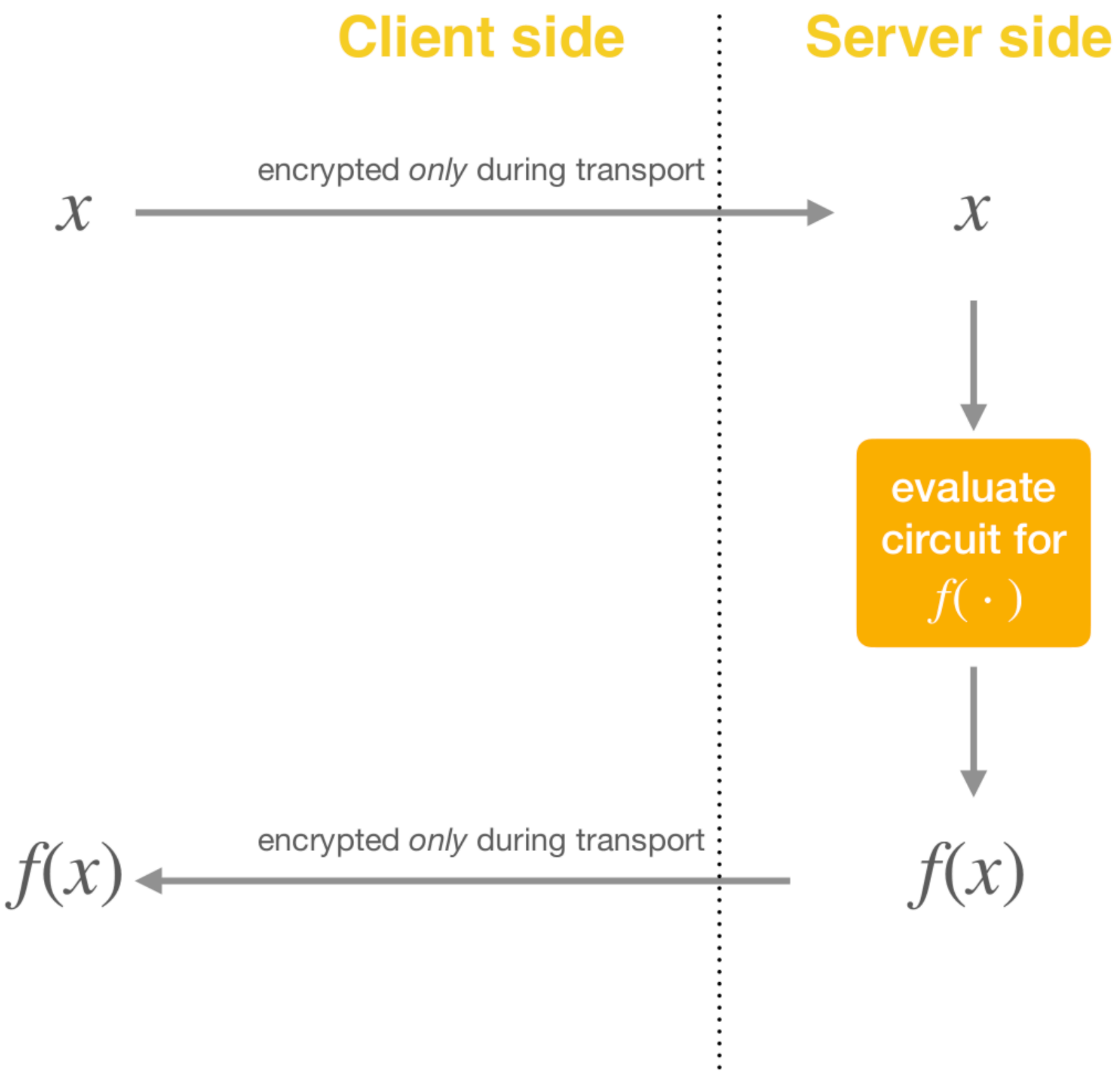
May 2024

Privacy-Preserving ML with Fully Homomorphic Encryption

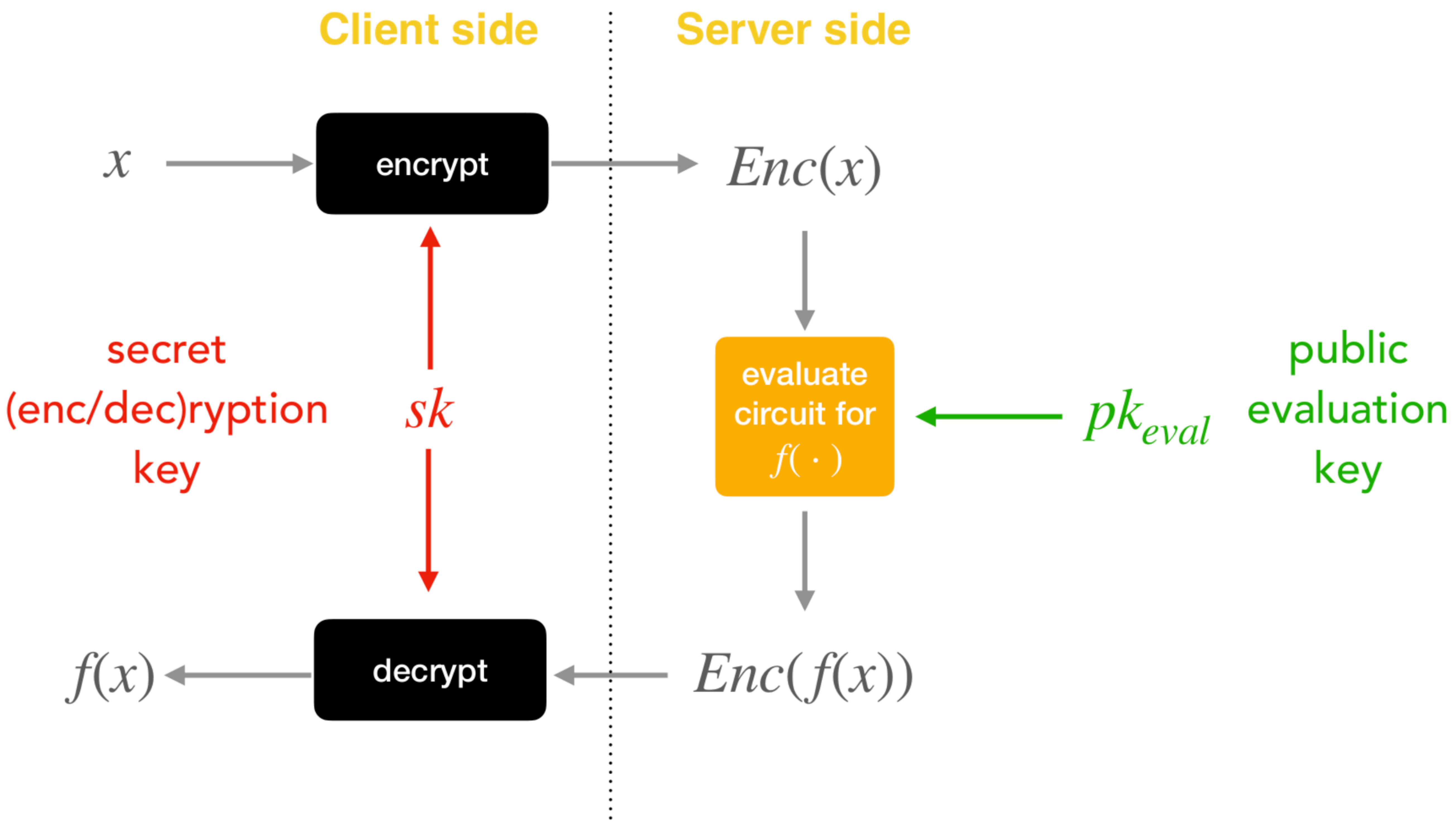
ZAMA

Data is encrypted only during the transport



Fully Homomorphic Encryption

FHE: encrypting the data also during processing



TFHE: the cryptographic scheme we use

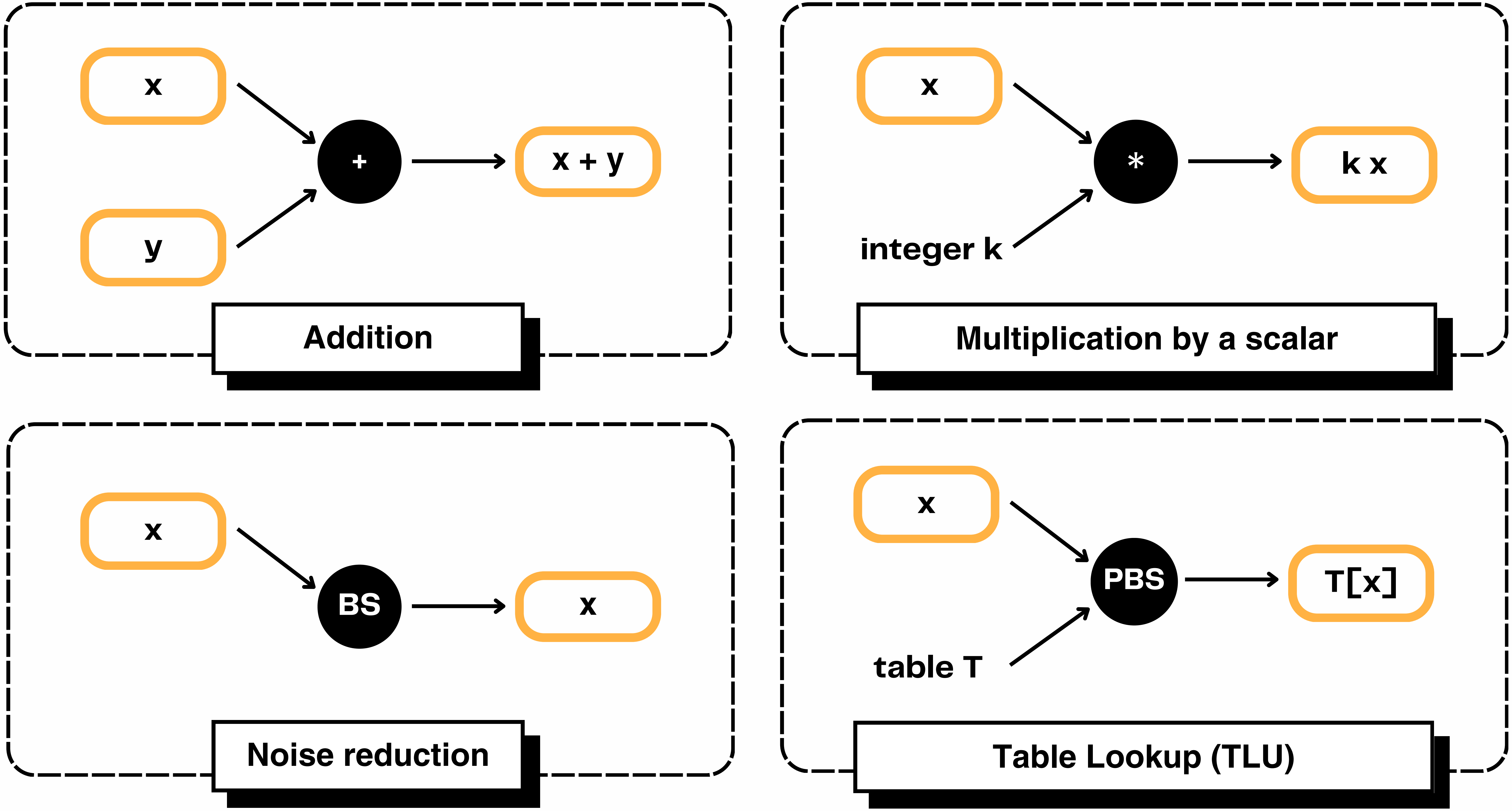
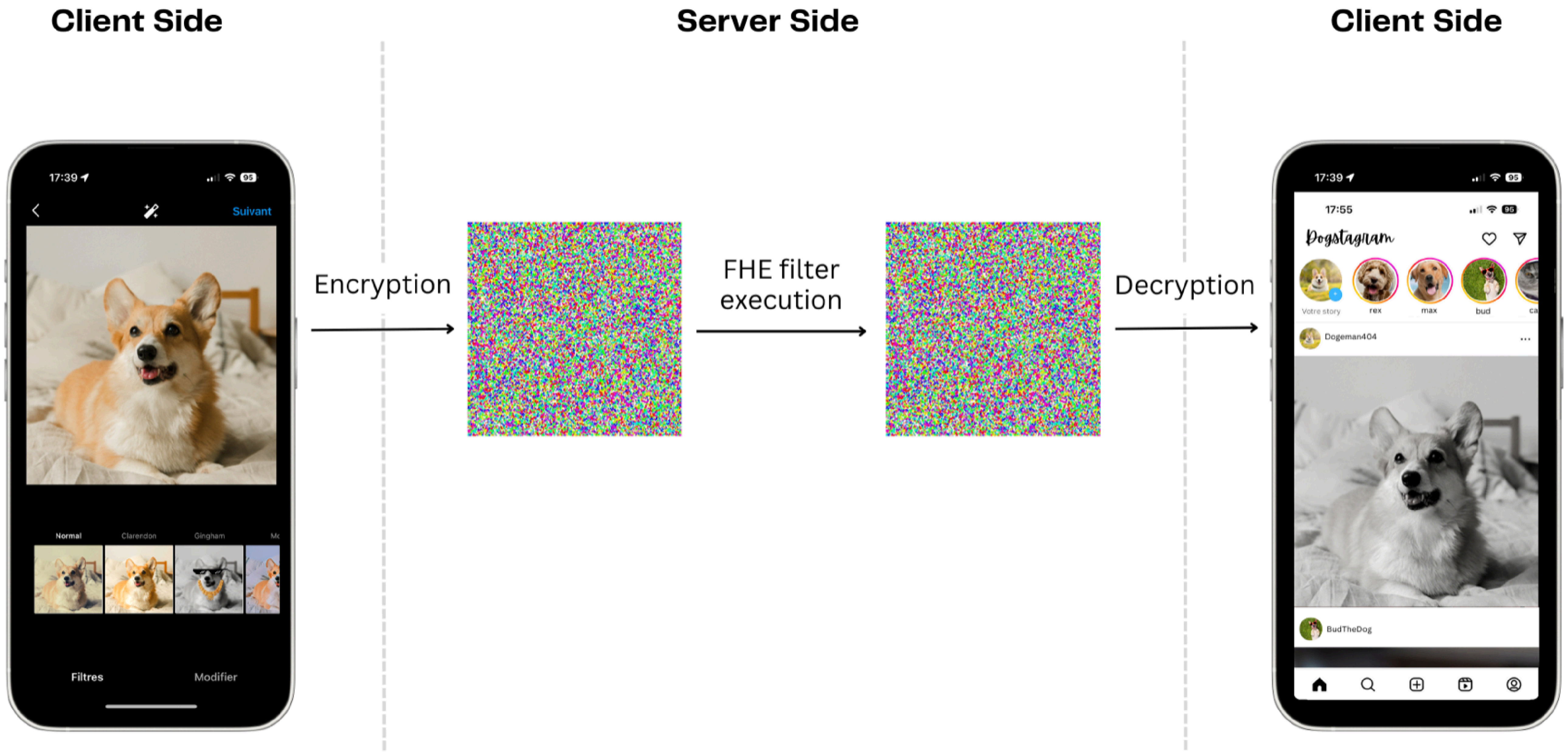


Image filtering as a demo

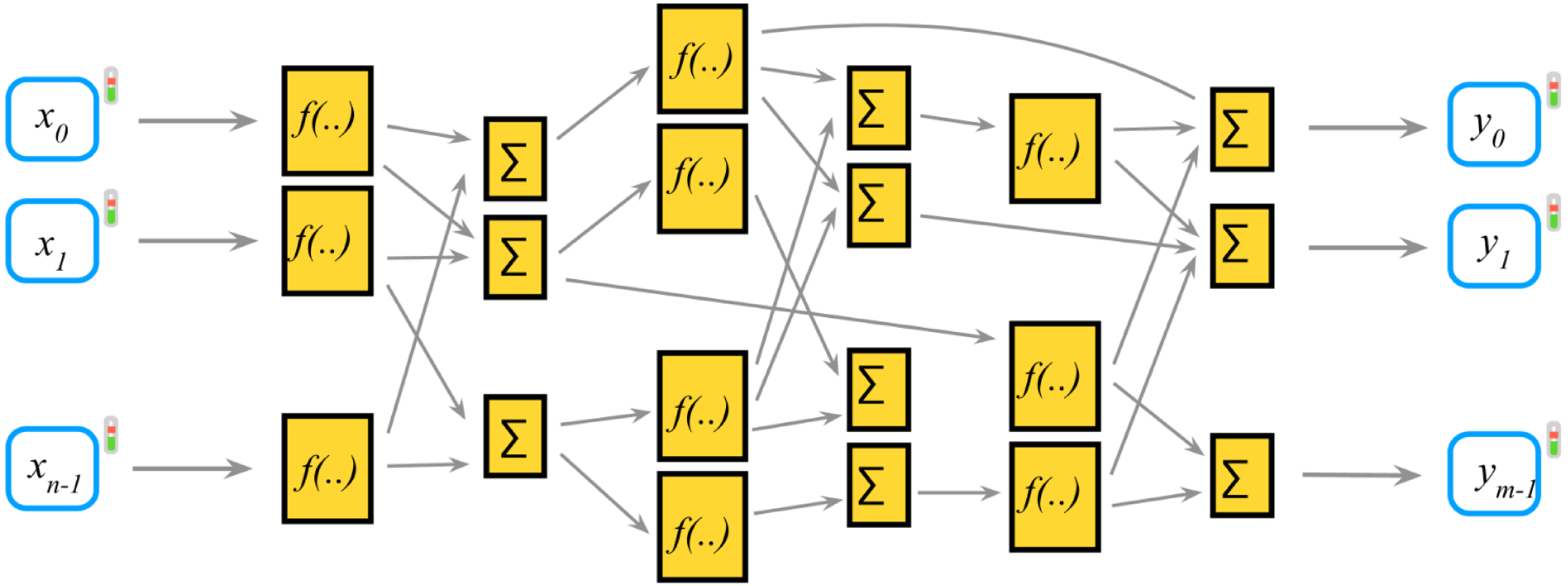


https://huggingface.co/spaces/zama-fhe/encrypted_image_filtering

How Does It Work?

Computation paradigm

circuit of univariate functions



= graph mixing univariate functions and linear combinations

Linear Models

Linear models

1. **Train**
   - in clear: linear solver, l-bfgs, quadratic solver
   - on encrypted data: SGD

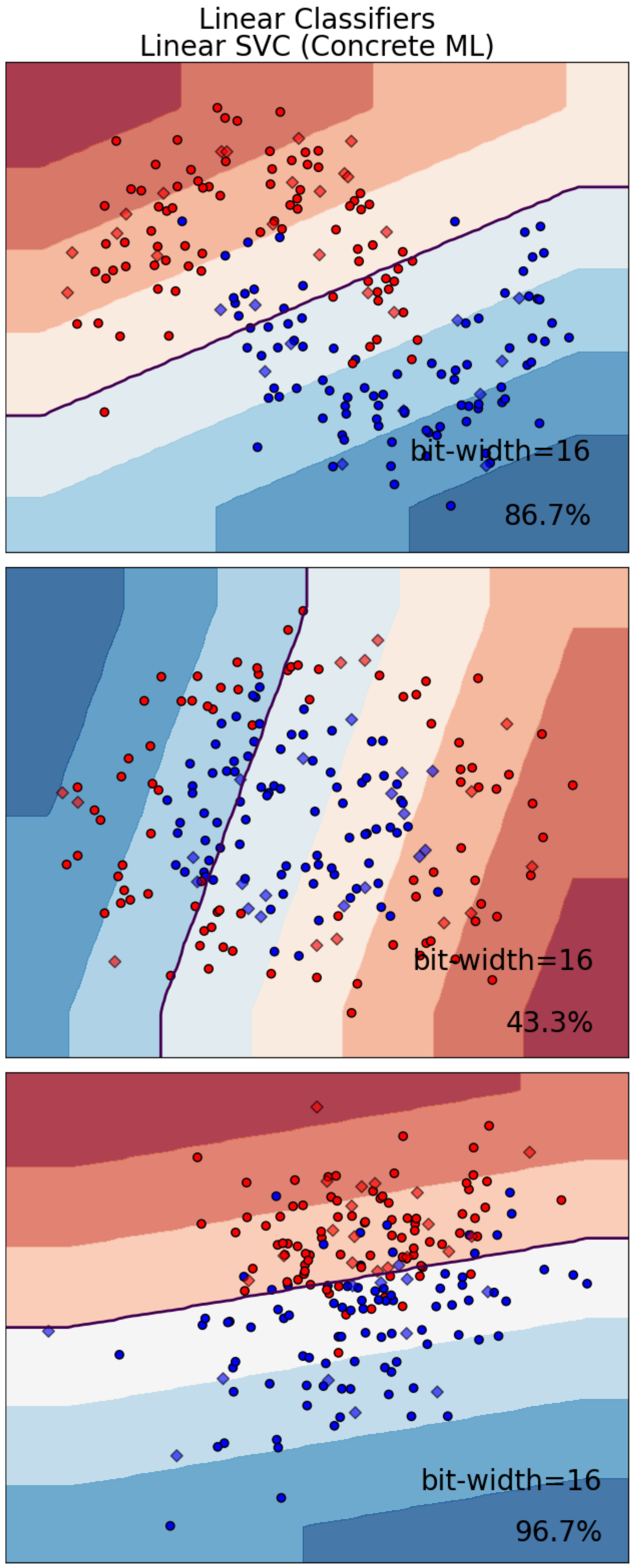
2. **Apply post-training quantization to both inputs and weights**

Asymmetric quantization

$$q_a(x) = \text{round}\left(\frac{x}{\Delta}\right) + z \qquad \Delta = \frac{\max(x) - \min(x)}{2^p - 1} \qquad z = -\left\lfloor \frac{\min(x)}{\Delta} \right\rfloor$$

3. **Execute fully-levelled in FHE**
   - execution time: 10-100ms
   - can use up to 8 bits weights and inputs

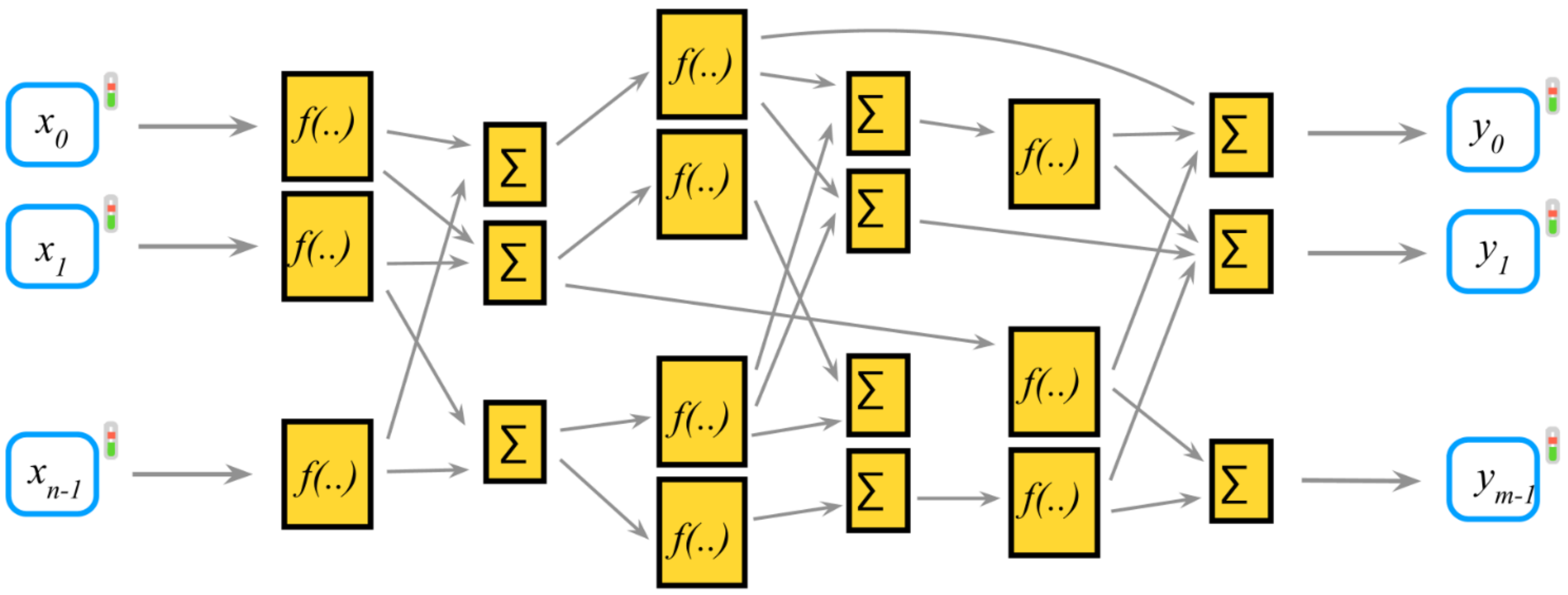
8 bits quantization – Should be good for all datasets



Deep Neural Networks

FHE computation paradigm vs neural networks operation graphs

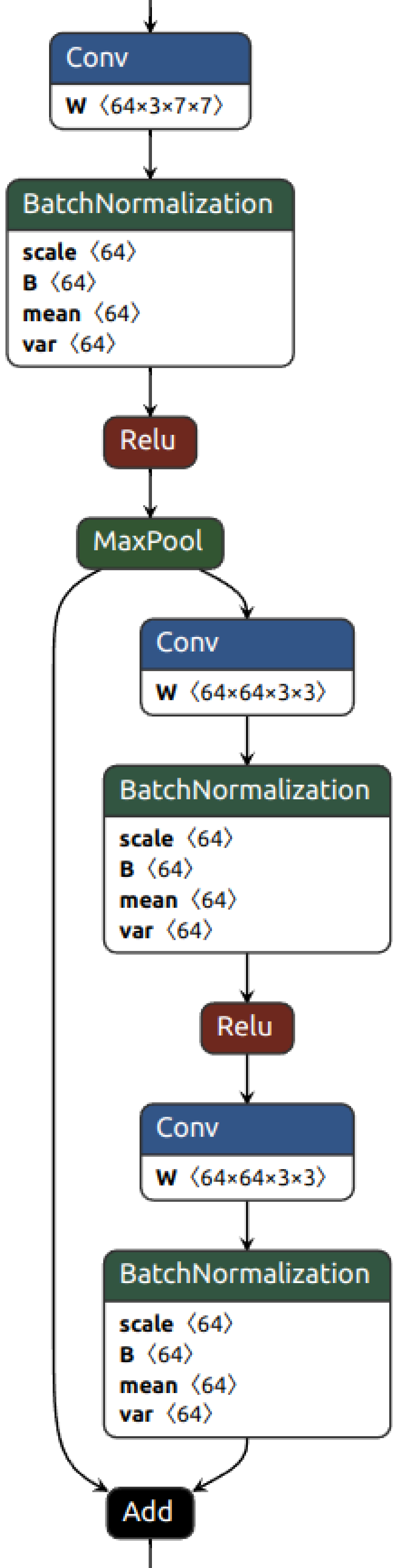
circuit of univariate functions



= graph mixing univariate functions and linear combinations

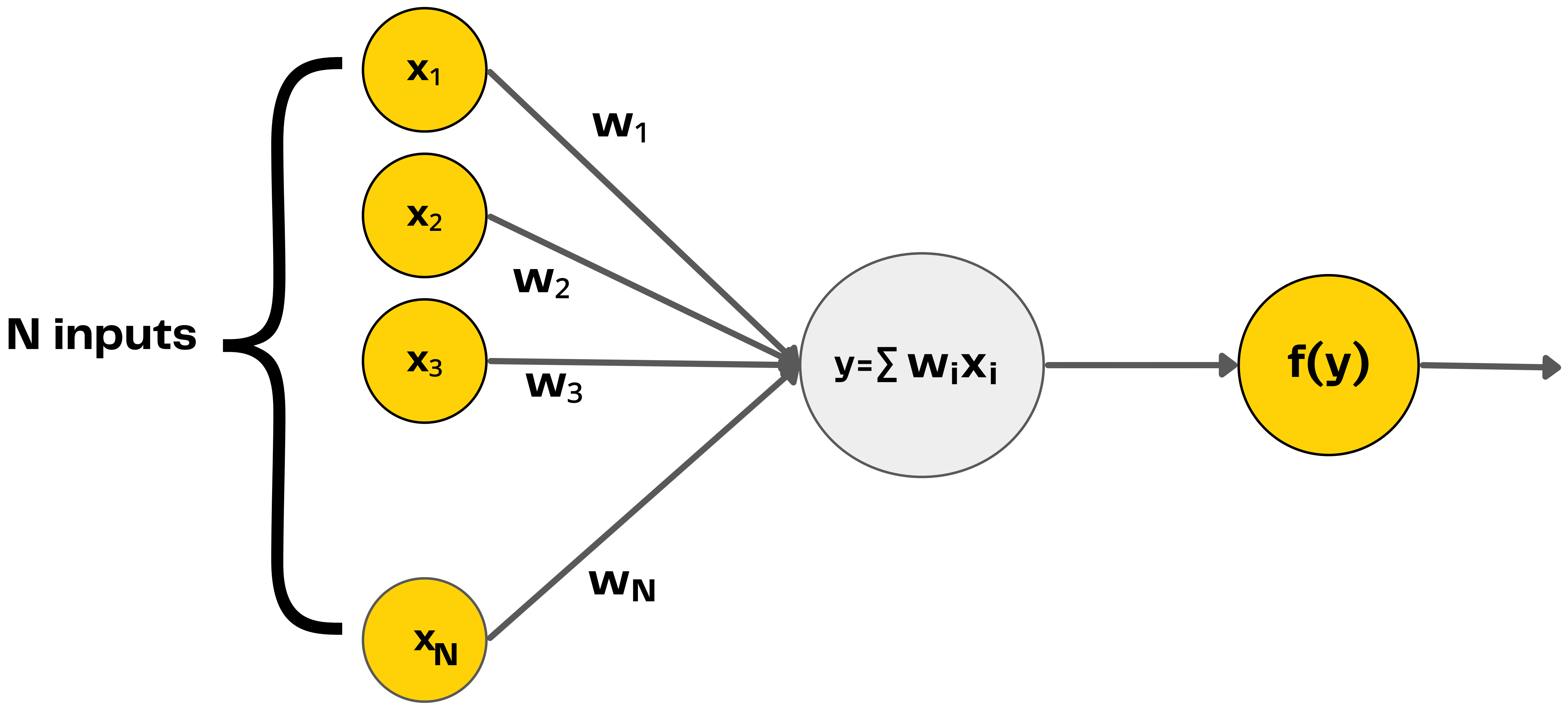
f(...)
TLU/PBS

Σ
LINEAR OPERATION ON INTEGERS



Quantization for integer computation

- Inputs, weights and activations are float32
- Need to convert to integer computation
- All values must have at most P bits



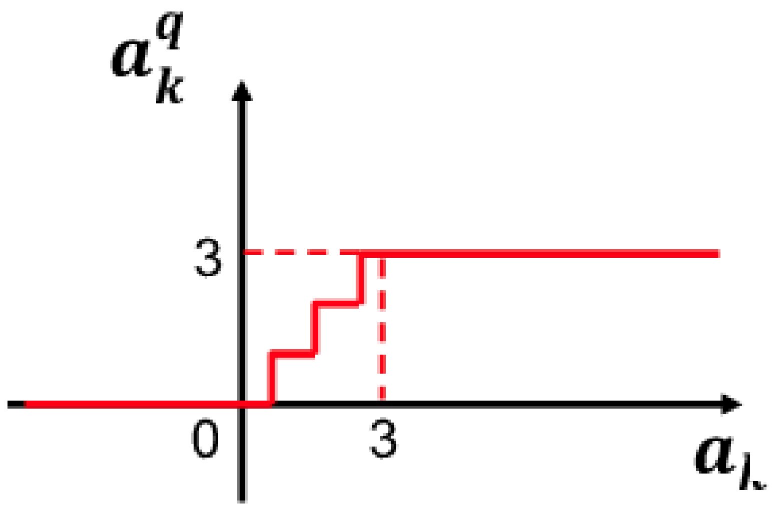
Quantization Aware Training (QAT)

Training:
find the weights that minimize the objective function when applying the model on the inputs

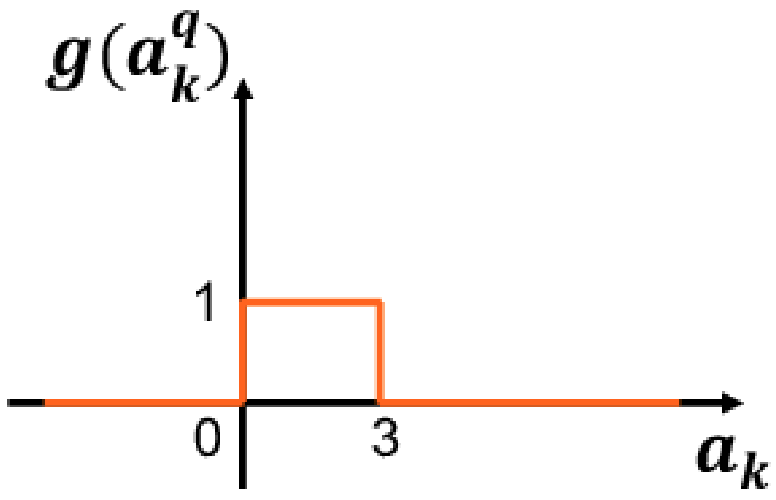
Quantization aware training:
+ the weights are constrained to be representable by integers of a certain bit-width
+ the inputs are represented by integers

What about gradients under QAT?

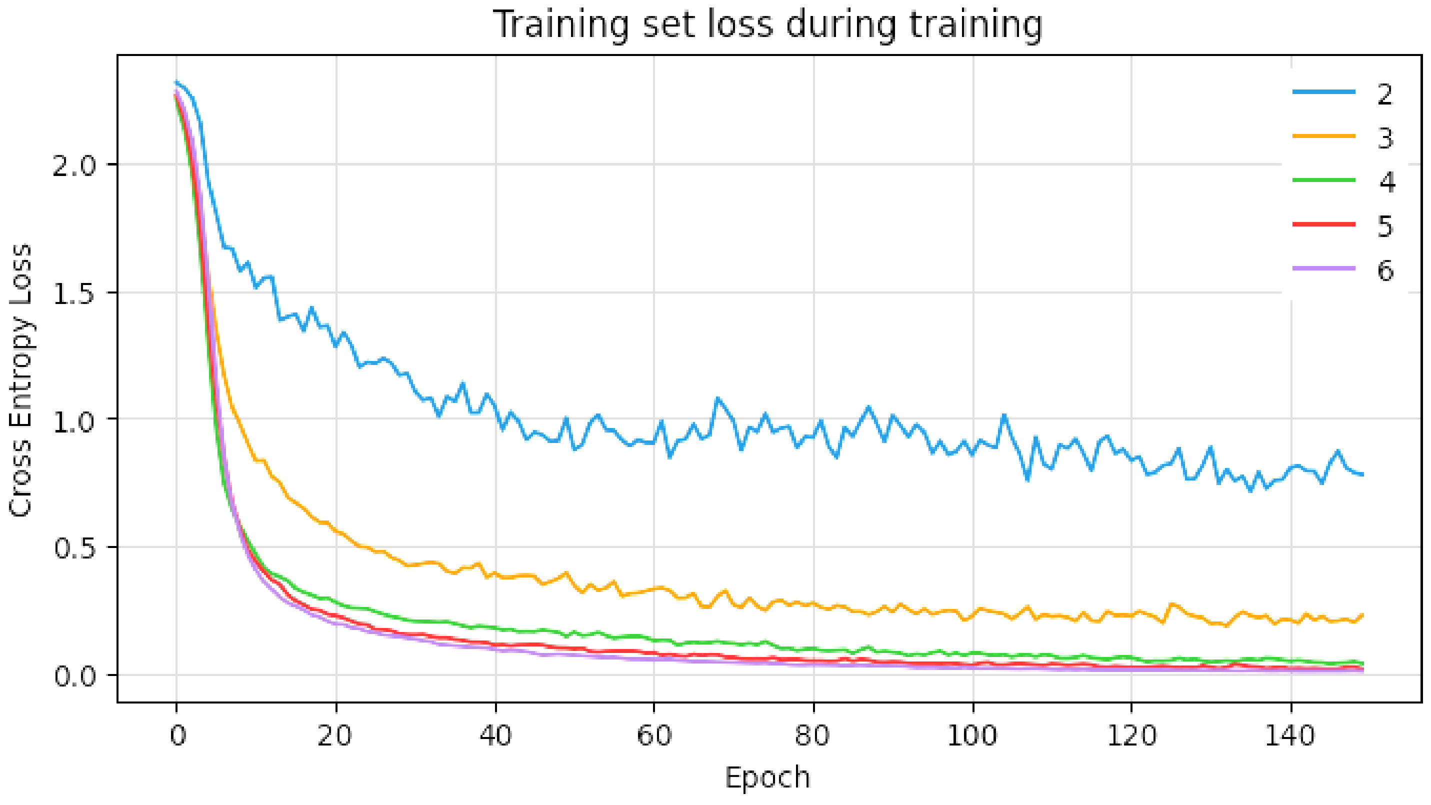
Quantizer function



Straight through estimator



QAT is (much) more difficult with very low bitwidth

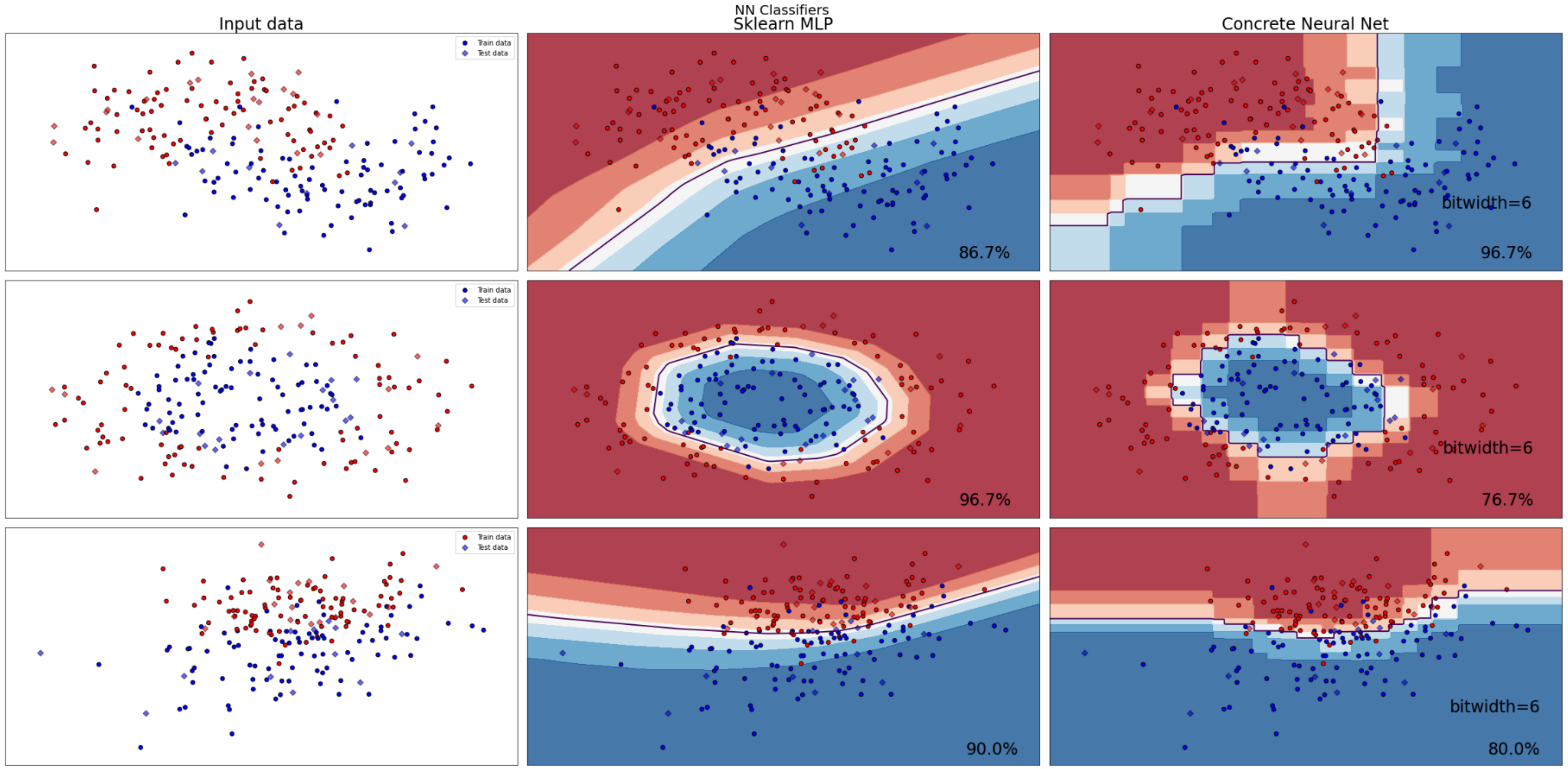


Built-in neural networks for toy datasets

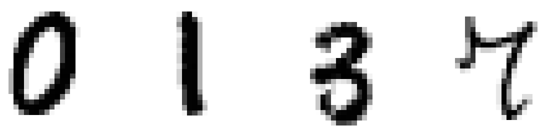
- 3-layer MLP
- ReLU Activation

- 2 bits weights
- 4 bits inputs & activations

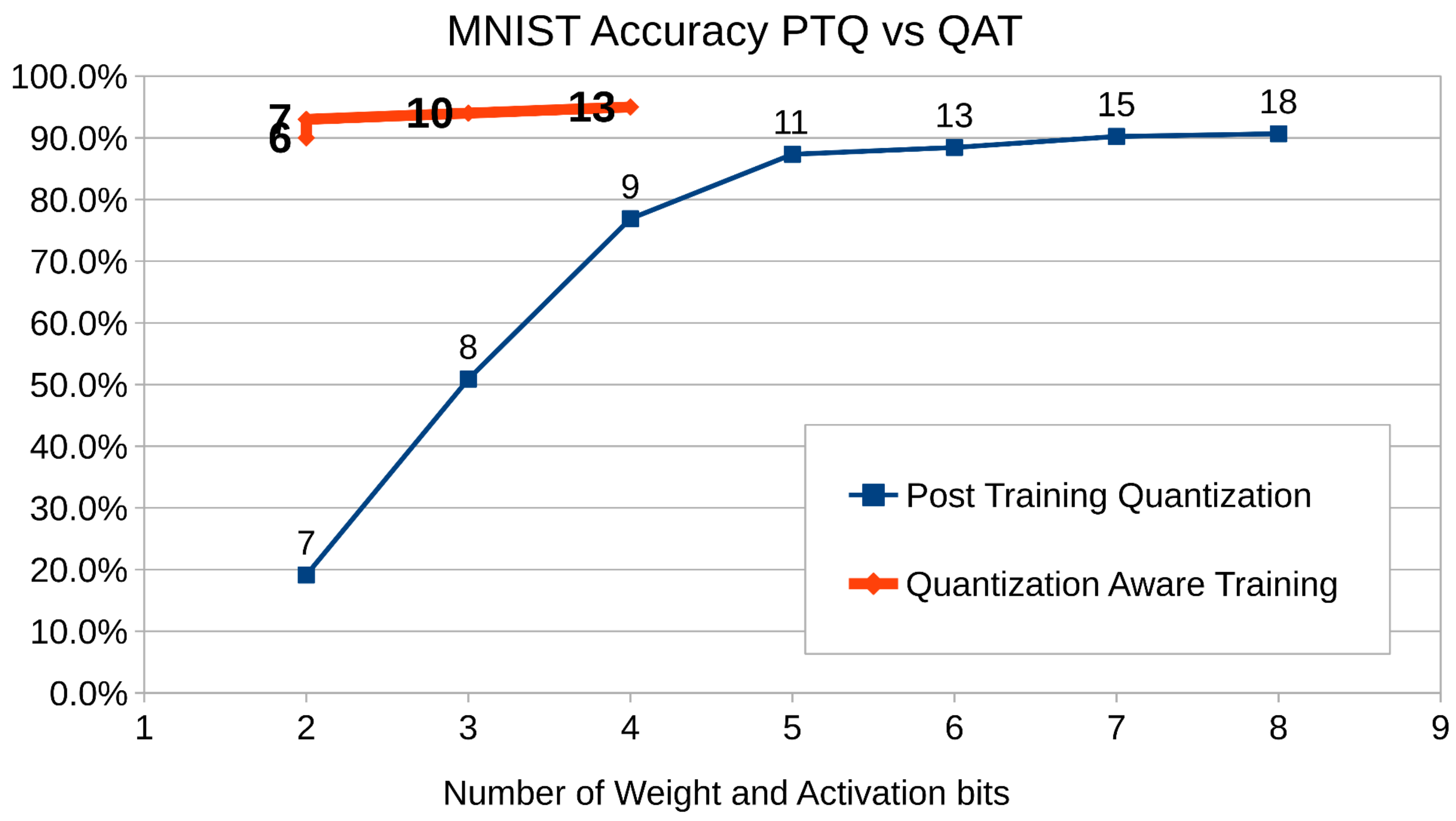
- Maximum accumulator: 6 bits



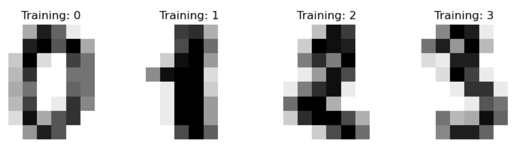
Custom QAT models: results on a tiny dataset



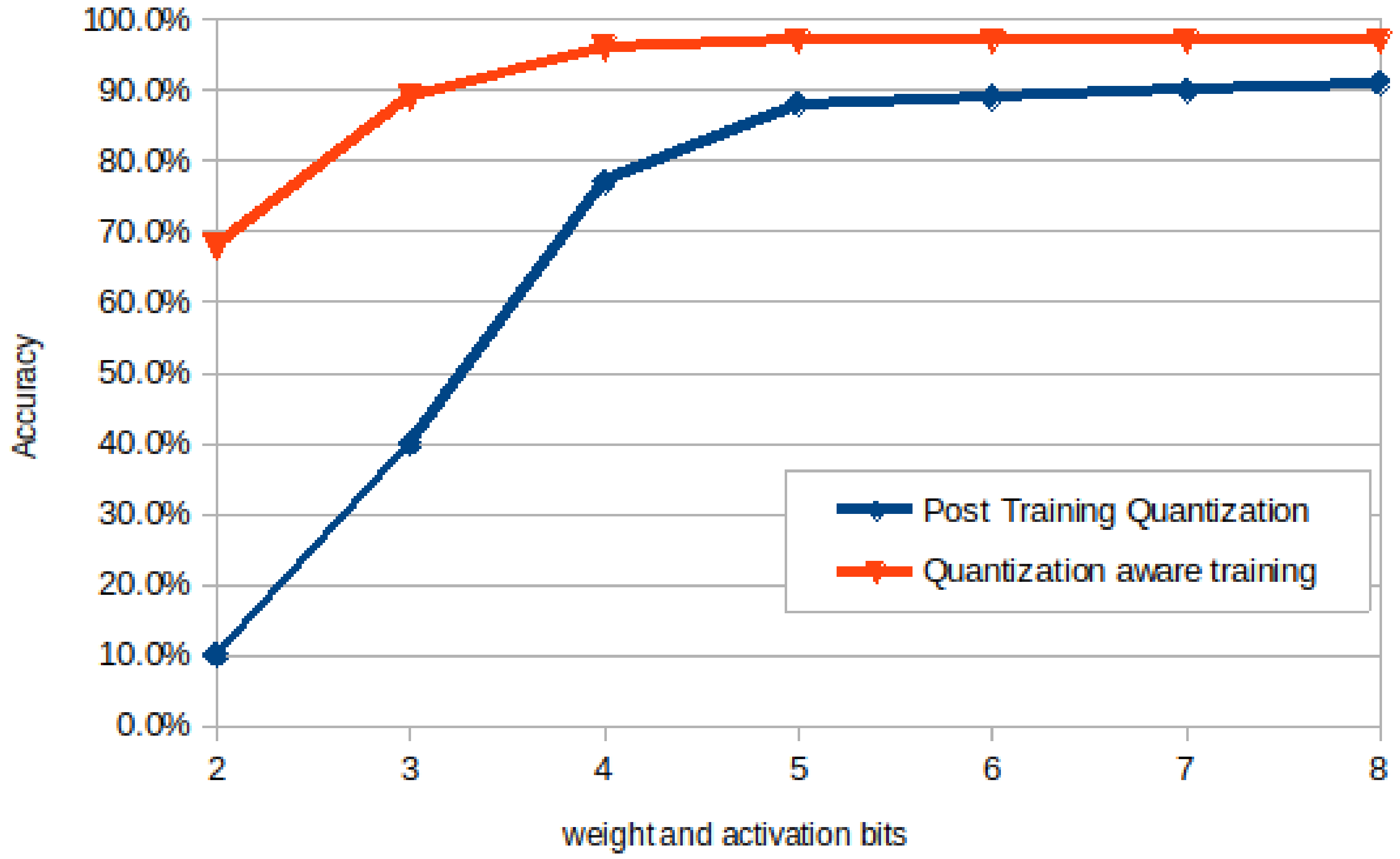
Fully Connected Model:
3 layers with 192 neurons



optimized: 4 seconds / image



CNN:
3 layers with 8-32 3x3 filters, pruned to 12 max active neurons

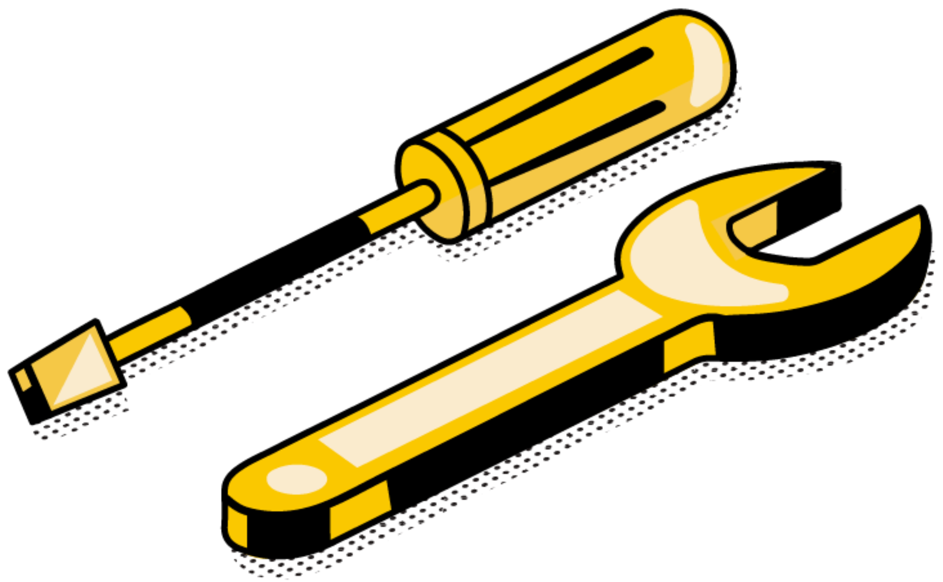


unoptimized: 3 seconds / image

Optimizations

Using round PBS / truncate PBS / approximate PBS:

- replace T[i], where i is on n bits by T'[i'], where:
  - i' is on (n-r) bits
  - i' corresponds to most significant bits of i
- depending on the case:
  - i' = round(i) or truncate(i)



Playing with p-error parameters:

- probability of by-one error can be tuned
- eg, with an error probability of 0.01, it is ~10x faster than a probability of 1E-6

Custom QAT models: VGG9 on CIFAR

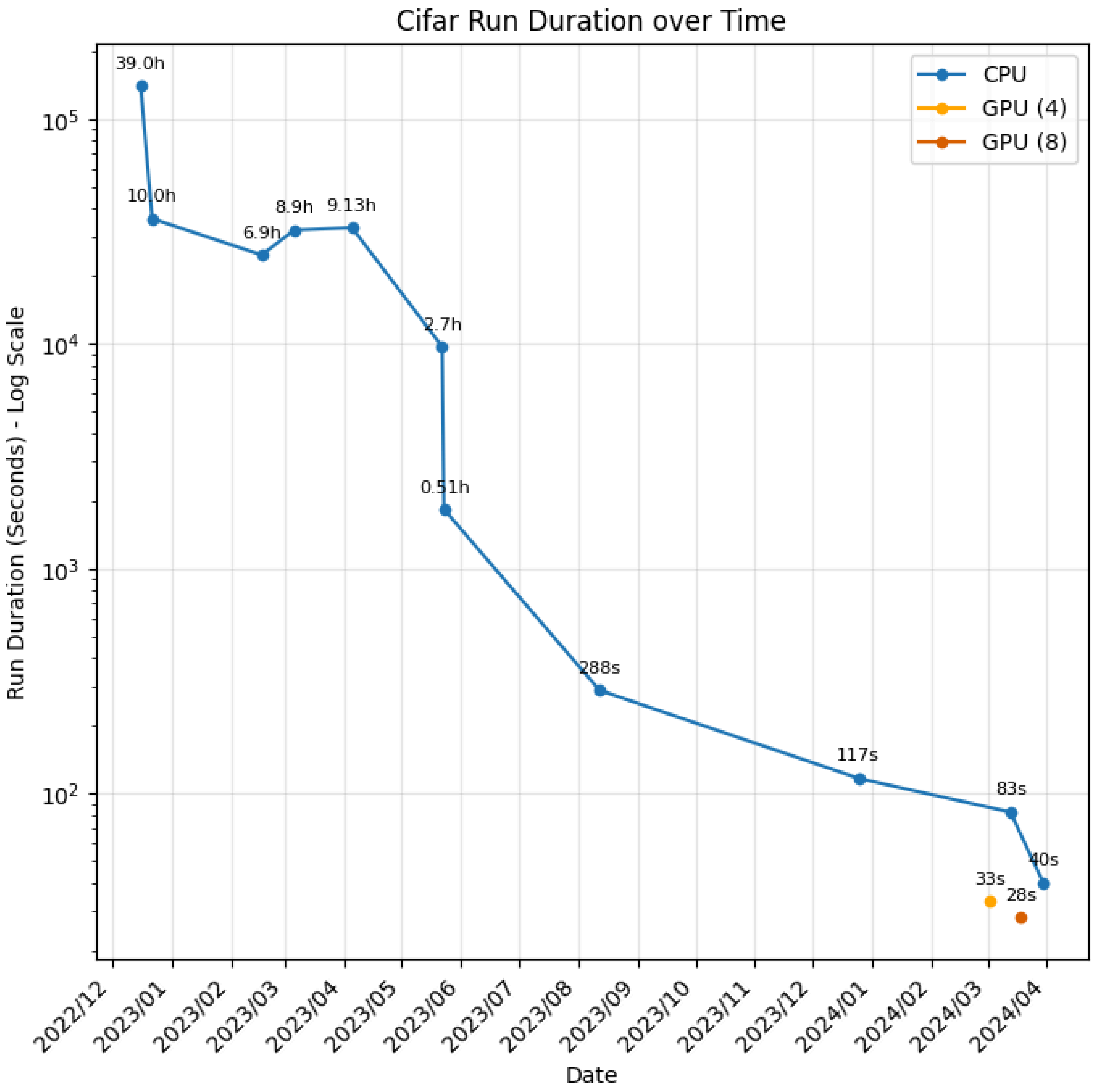
CIFAR10 - 32x32 - VGG-9 with AveragePooling

2b weights, 2b activations, 8b inputs

Runtime	Rounding	Accuracy
VGG Torch	None	88.7
VGG FHE (simulation)	None	88.7
VGG FHE (simulation)	8 bits	88.0
VGG FHE (simulation)	7 bits	87.2
VGG FHE (simulation)	6 bits	86.0
VGG FHE	6 bits	86.0

Privacy-Preserving ML with FHE

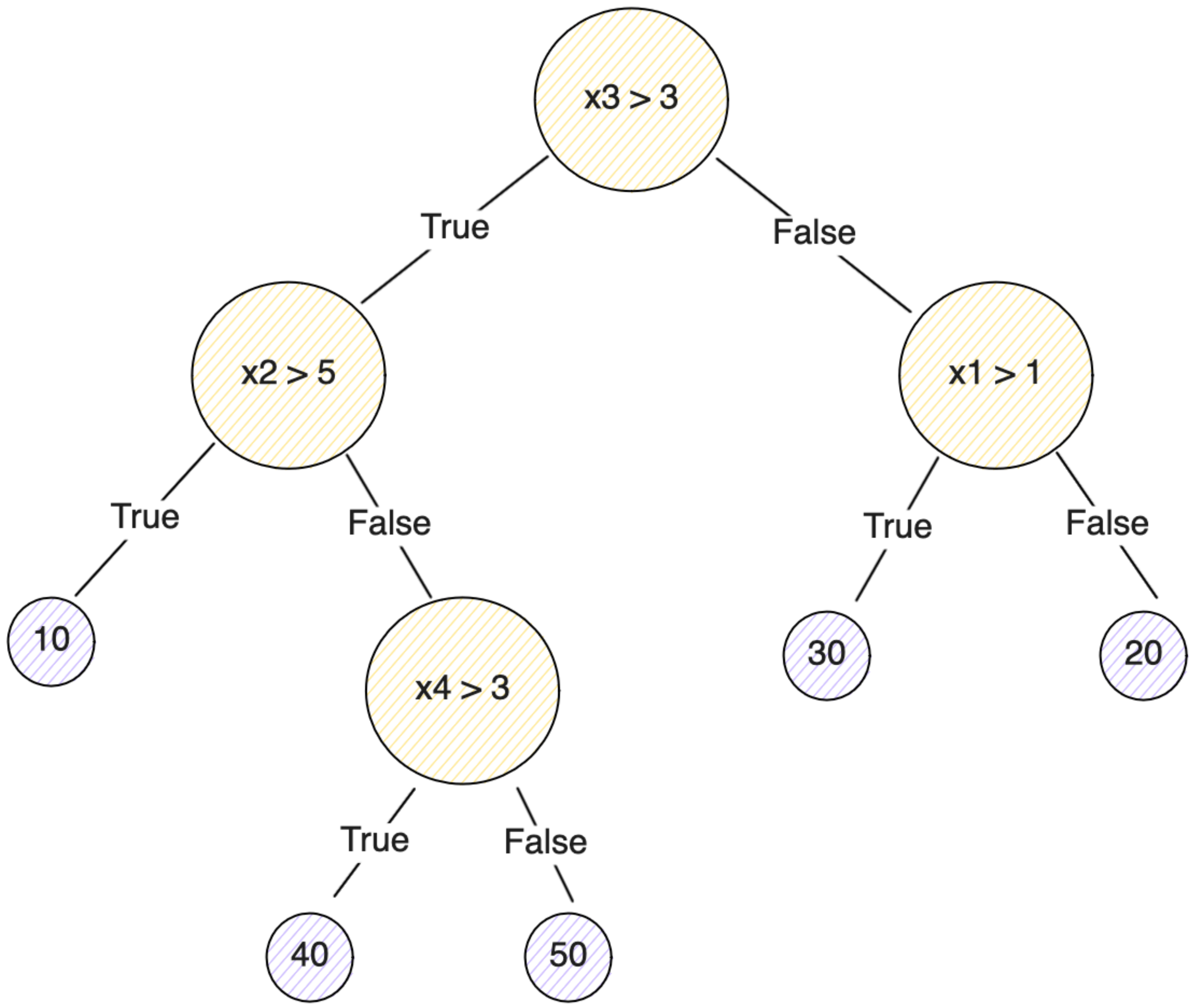
Custom QAT models: VGG9 on CIFAR



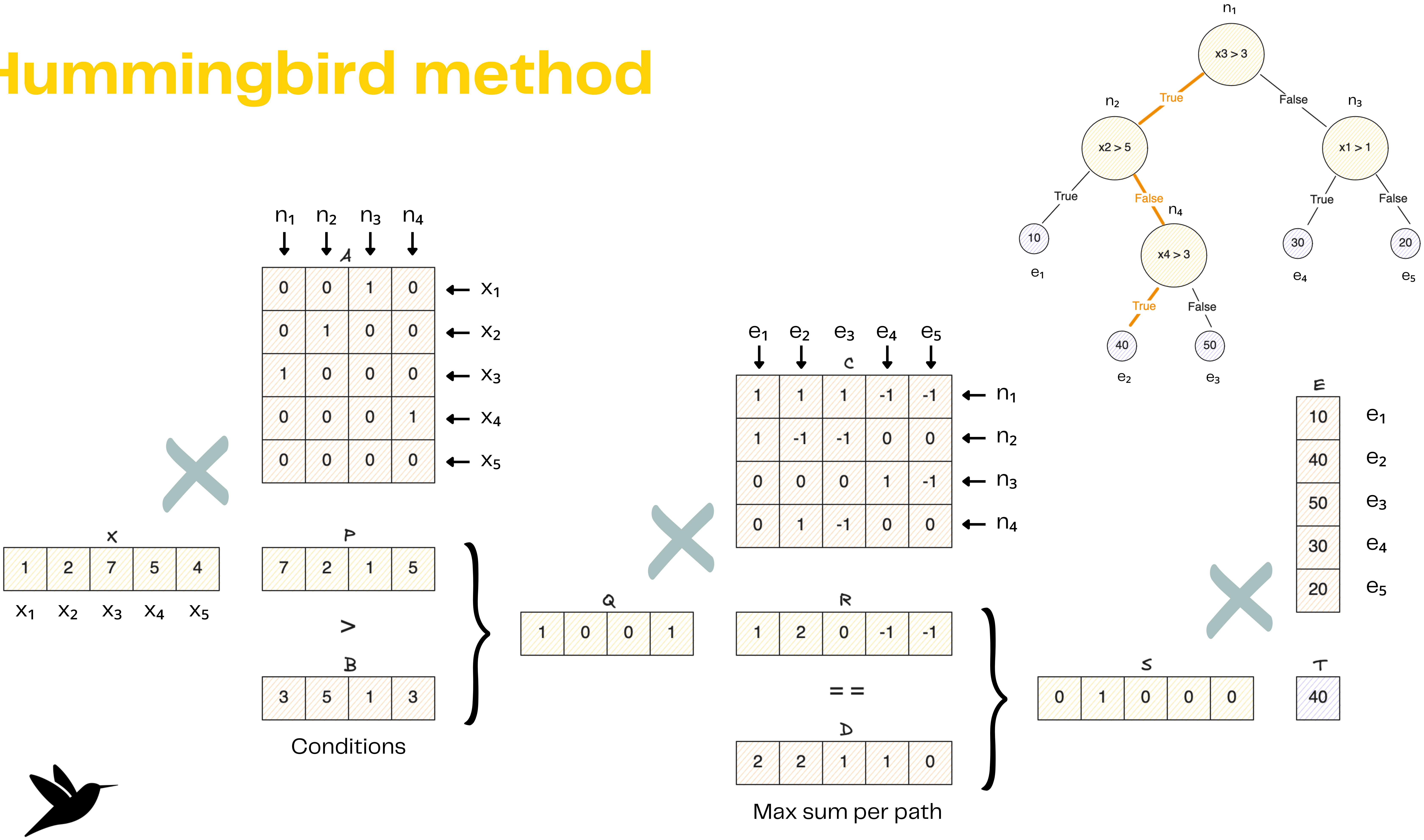
Tree-Based Models

Tree-based models and FHE?

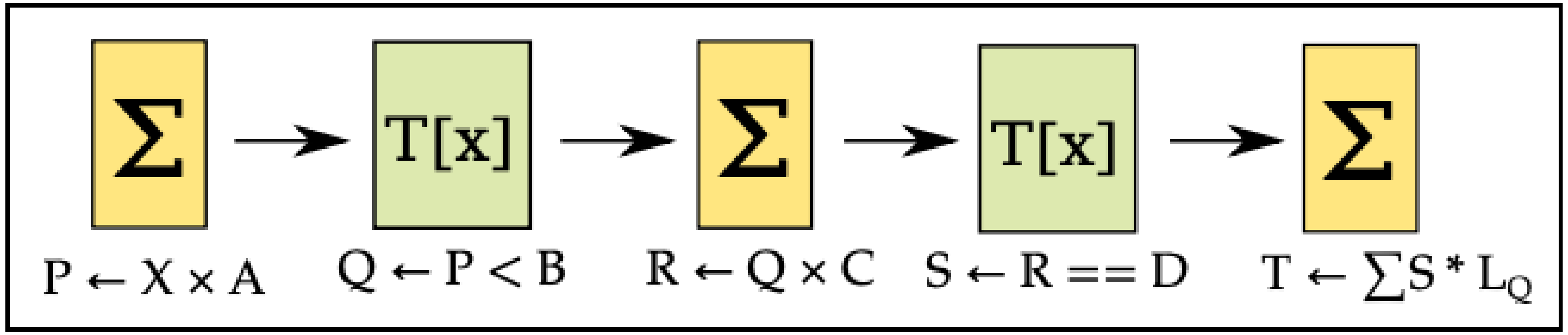
- Powerful and common ML models : DecisionTree, RandomForest, XGBoost, ...
- Not directly FHE compliant :
    - depend on control-flow operations (if statements)
    - work with floating points by default



Hummingbird method



Hummingbird method

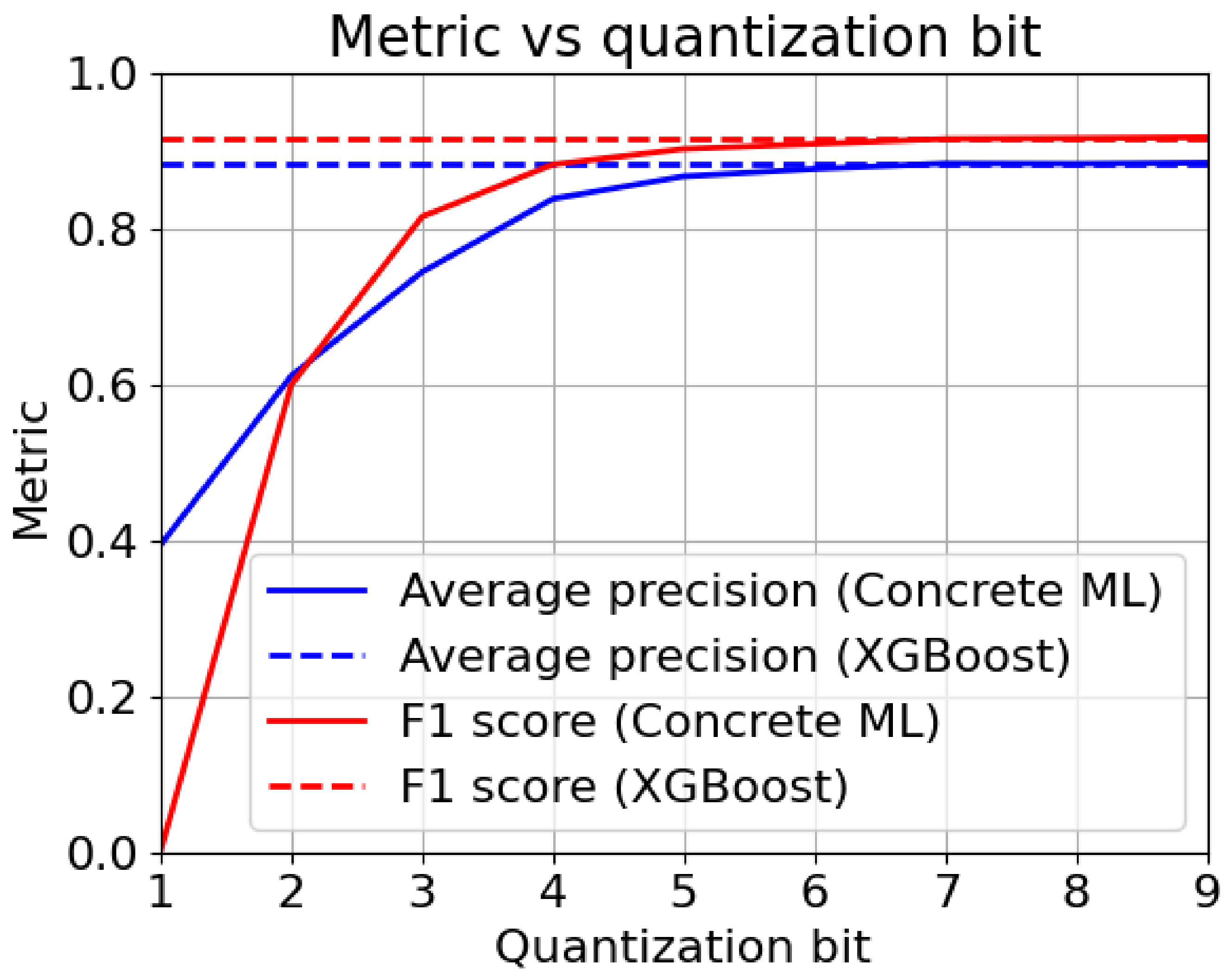


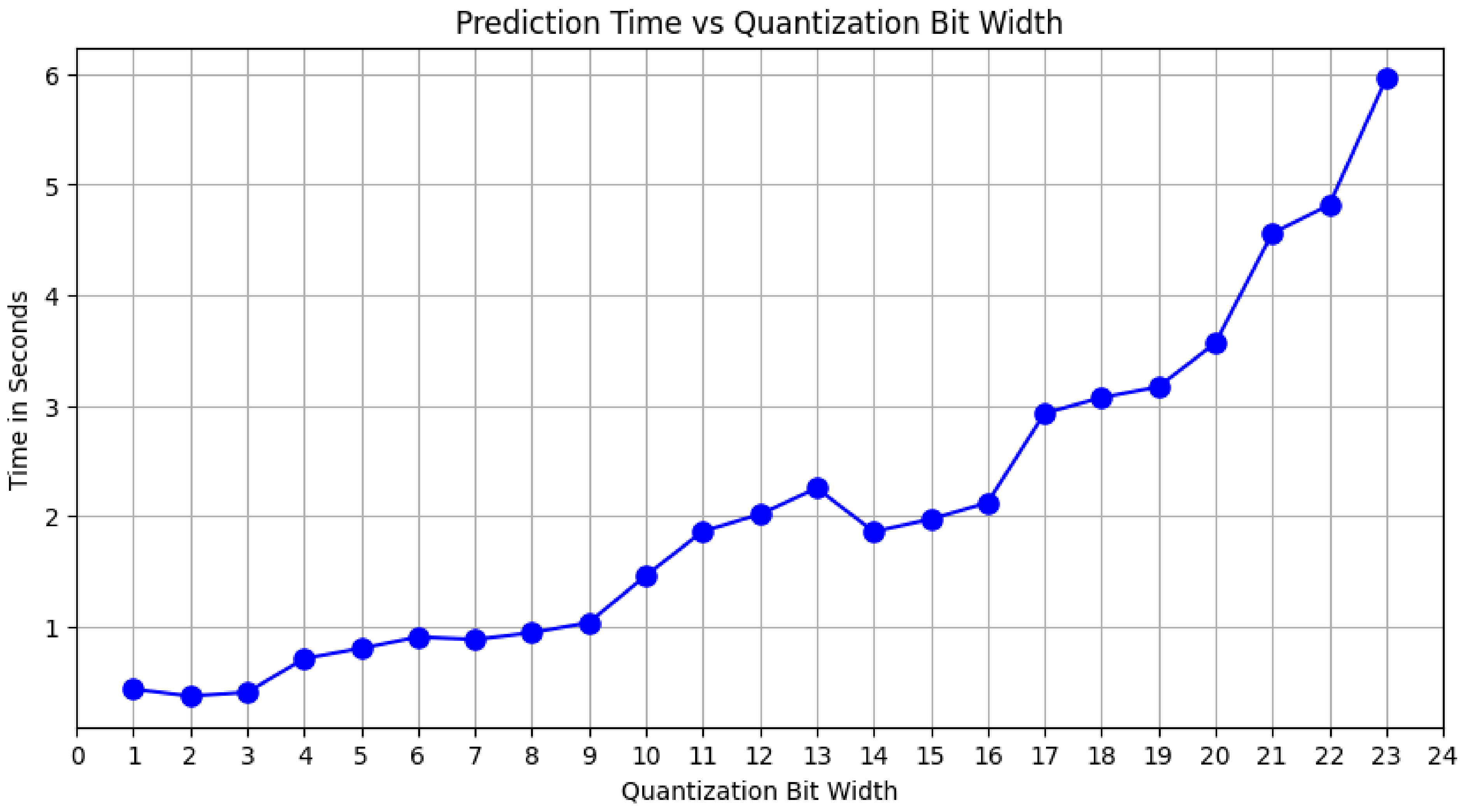
max bits :	p	p	log2(d) + 1	log2(d) + 1	p

Max bit-width reached : **max(p, $\log_2$(d) + 1)**

- p : number of bits of quantization used for inputs and outputs
- d : tree depth

Experimental results : XGBoost





XGBoost classifier model (max_depth=3, n_estimators=50) on data set “Spambase”

Experimental results : spambase

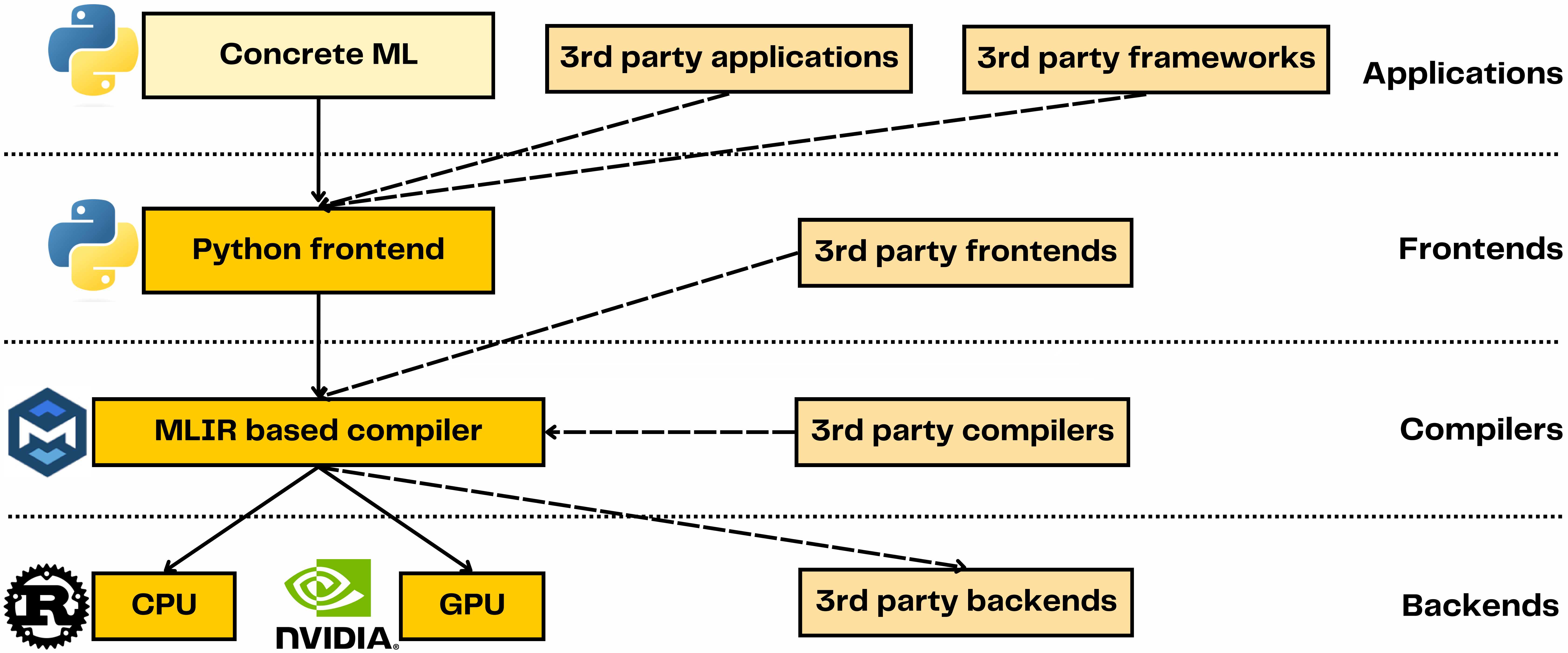
		accuracy	f1	AP	#nodes	Time (s)	FHE/Clear
spambase	FHE-DT	91.0%	88.0%	84.3%	23	1.313	825x
	FP32-DT	90.3%	87.4%	82.4%	-	0.002	-
	FHE-XGB	93.1%	90.9%	87.7%	350	7.020	4617x
	FP32-XGB	93.6%	91.7%	88.3%	-	0.002	-
	FHE-RF	90.9%	87.5%	84.6%	750	16.248	8520x
	FP32-RF	91.8%	89.0%	86.0%	-	0.002	-

n_bits=6, max_depth={4,5}, n_estimators=50

8 cores

Concrete / Concrete ML

Concrete stack



Scikit-learn API's for ML

```
from concrete.ml.sklearn import LogisticRegression

model = LogisticRegression(n_bits=12)
model.fit(X_train, y_train)
model.predict(X_test)
model.compile(X_train)
model.predict(X_test, fhe="simulate")
model.predict(X_test, fhe="execute")
```

```
from concrete.ml.sklearn import XGBClassifier

model = XGBClassifier(n_bits=8)
model.fit(X_train, y_train)
model.predict(X_test)
model.compile(X_train)
model.predict(X_test, fhe="simulate")
model.predict(X_test, fhe="execute")
```

No need to know cryptography!

Torch API's for DL

also support for: Tensorflow

ONNX

```
from transformers import AutoModel
from concrete.ml.torch.compile import compile_torch_model

# Load model from Hugging Face Hub
model = AutoModel.from_pretrained("dacorvo/mnist-mlp")

# Convert to FHE
q_module = compile_torch_model(
    model,
    torch_inputset=data,
)

fhe_outputs = q_module.forward(test_data, fhe="execute")
```

Confidential training

Federated Learning

- Federated learning protects training data
- Works for big models

Models trained with FL can be **deployed with Concrete ML**

```
from concrete.ml.sklearn import LogisticRegression

with open("federated_trained_model.pkl", "rb") as f:
    federated_model = pickle.load(f)

fhe_model = LogisticRegression.from_sklearn_model(model)
fhe_model.compile()

model.predict(X_test, fhe="execute")
```

Training on encrypted data

- FHE protects training data
- Encrypted models are trained on encrypted data
- Works for small models like LogisticRegression, MLP, and later for larger models

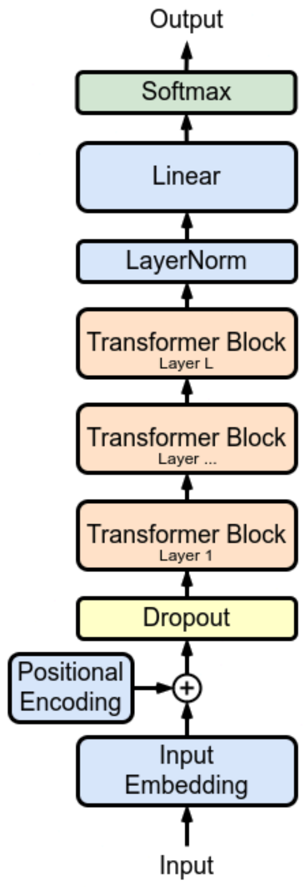
```
from concrete.ml.sklearn import SGDClassifier

sgd_clf_encrypted = SGDClassifier(fit_encrypted=True)
sgd_clf_encrypted.fit(X_binary, y_binary, fhe="execute")

y_pred = sgd_clf_encrypted.predict(X_binary, fhe="execute")
```

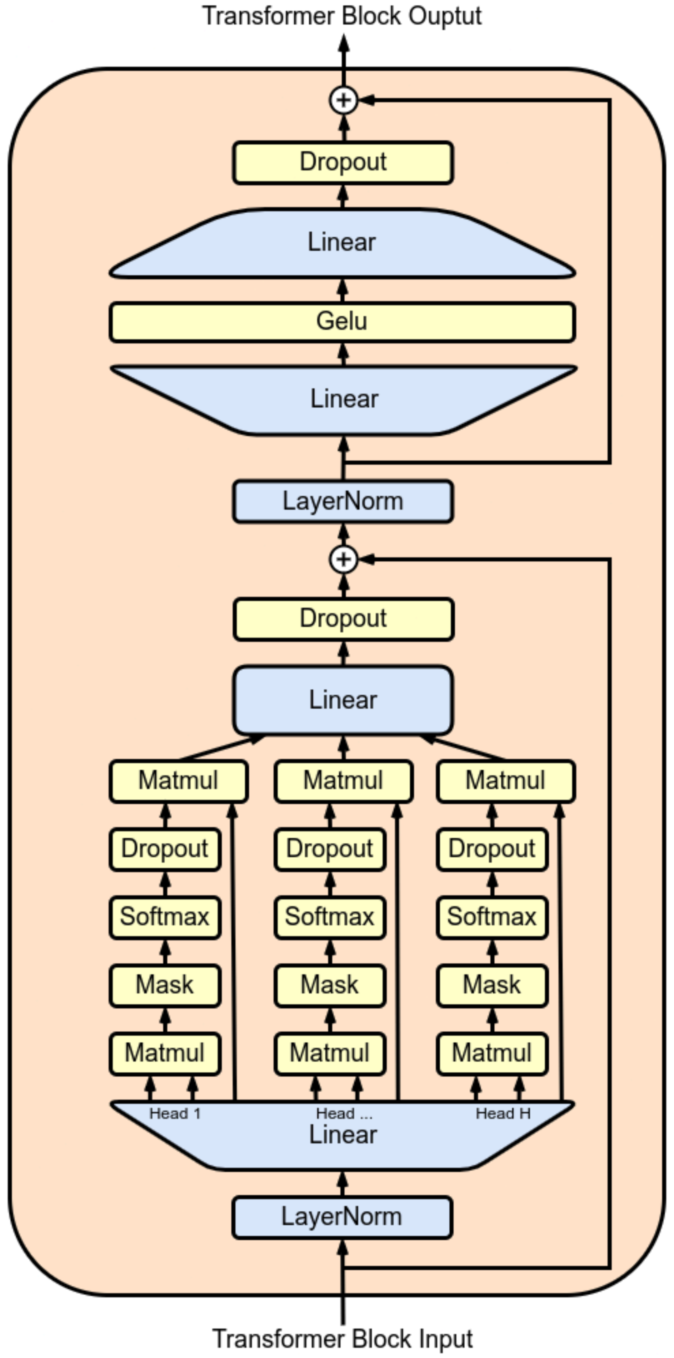
Large Language Models

LLM in FHE

Operations

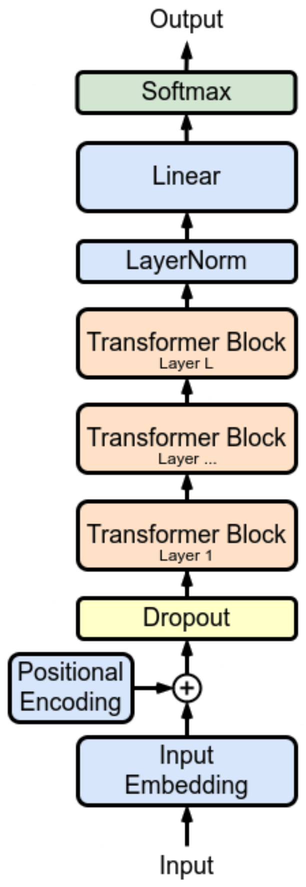
- Linear
- Embedding
- Non-Linear Activation (e.g. Gelu)
- Attention
  - Softmax
  - Encrypted matrix multiplication

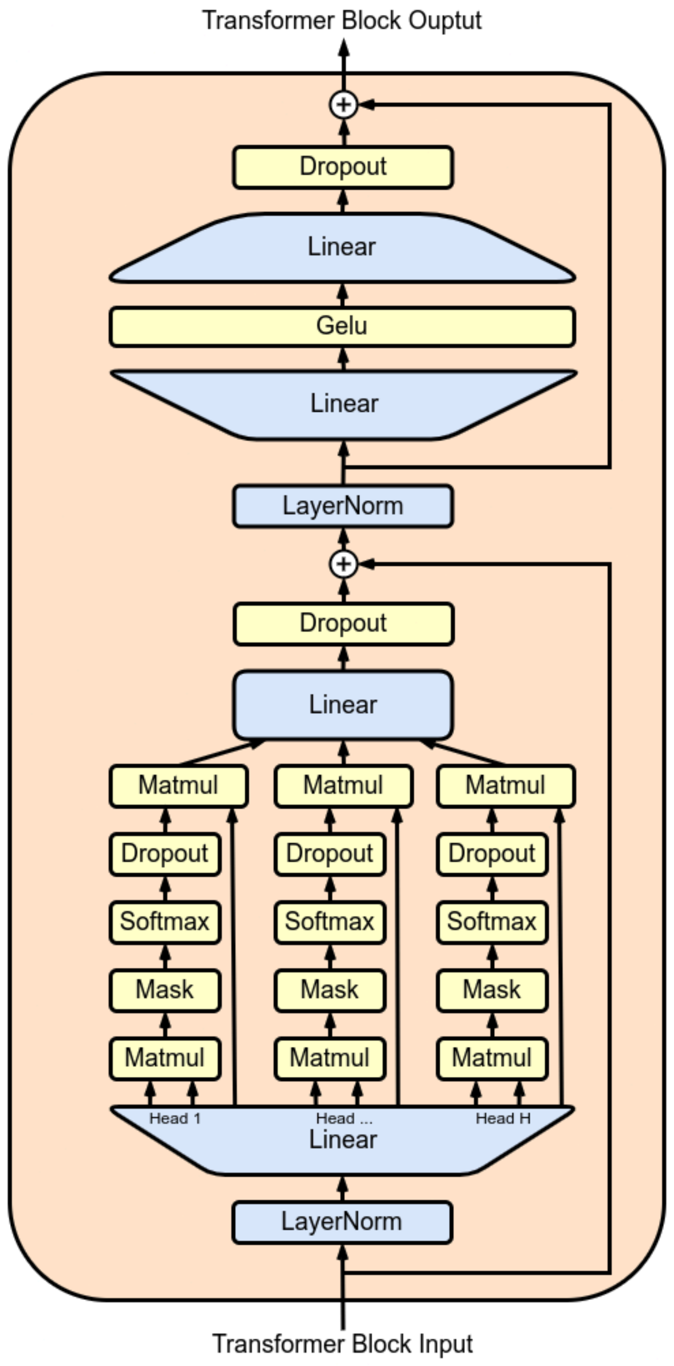




LLM in FHE

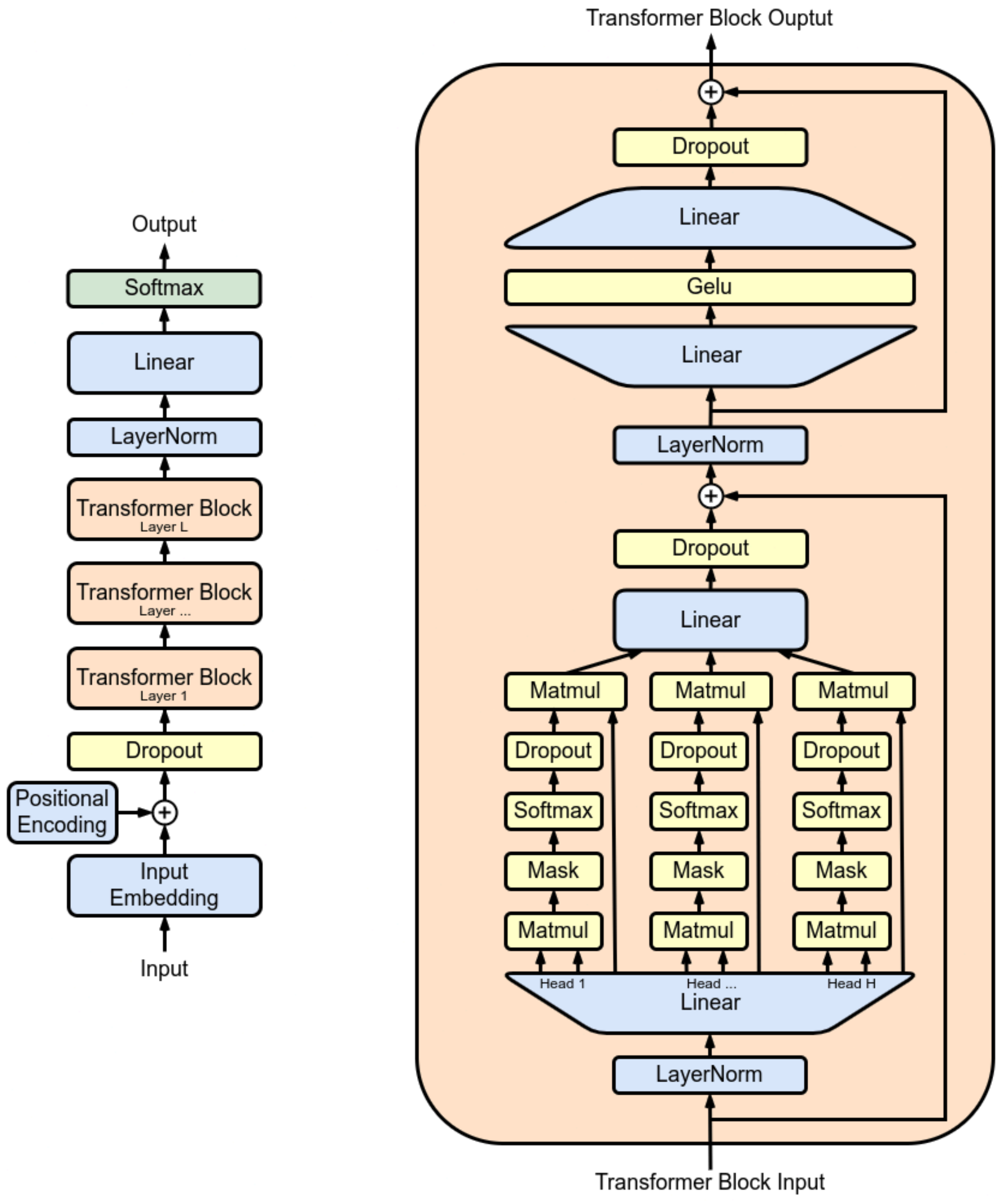
Operations	FHE Complexity
• Linear	ms
• Embedding	ms
• Non-Linear Activation (e.g. Gelu)	ms
• Attention	
◦ Softmax	seconds
◦ Encrypted matrix multiplication	minutes



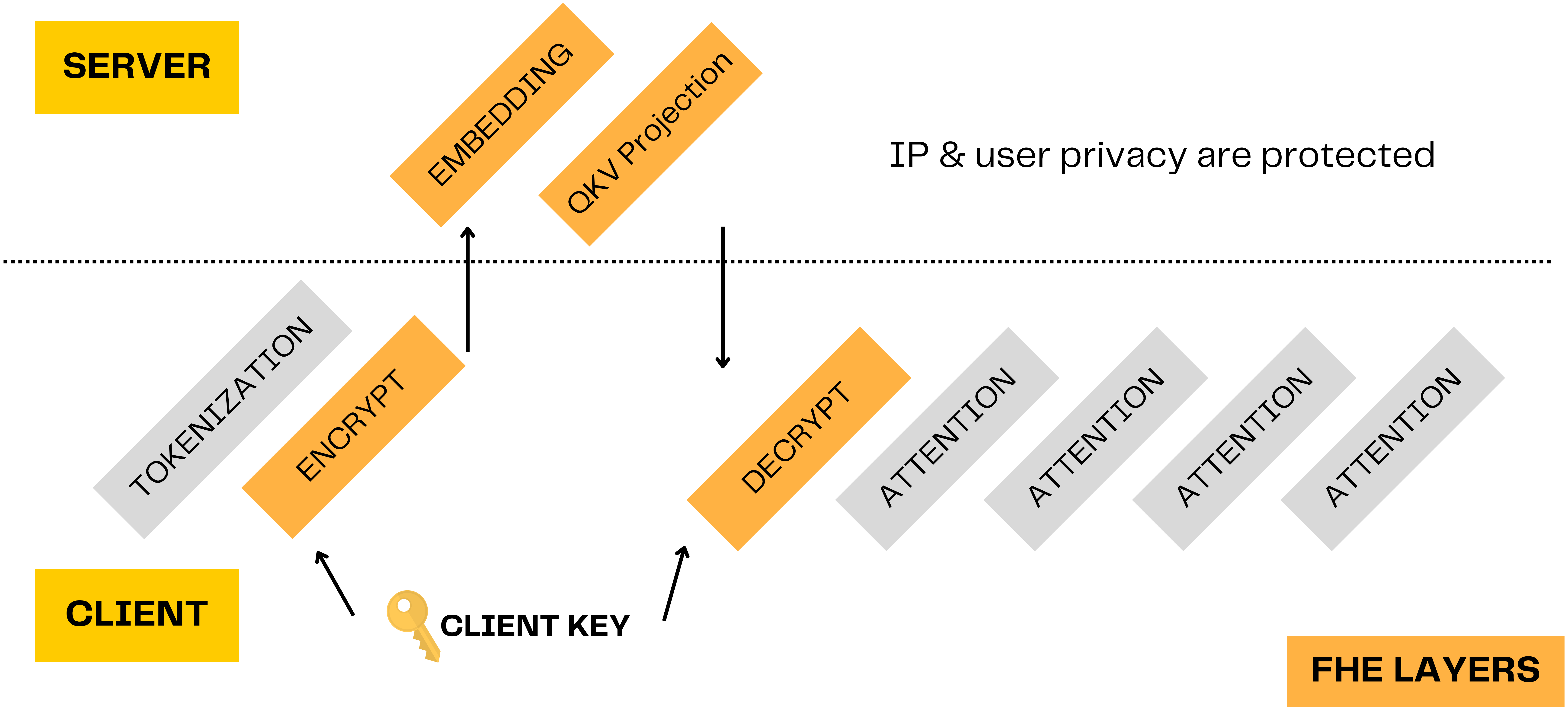


LLM in FHE

Operations	Weight distribution
• Linear	**99%**
• Embedding	1%
• Non-Linear Activation (e.g. Gelu)	0%
• Attention	
◦ Softmax	0%
◦ Encrypted matrix multiplication	0%

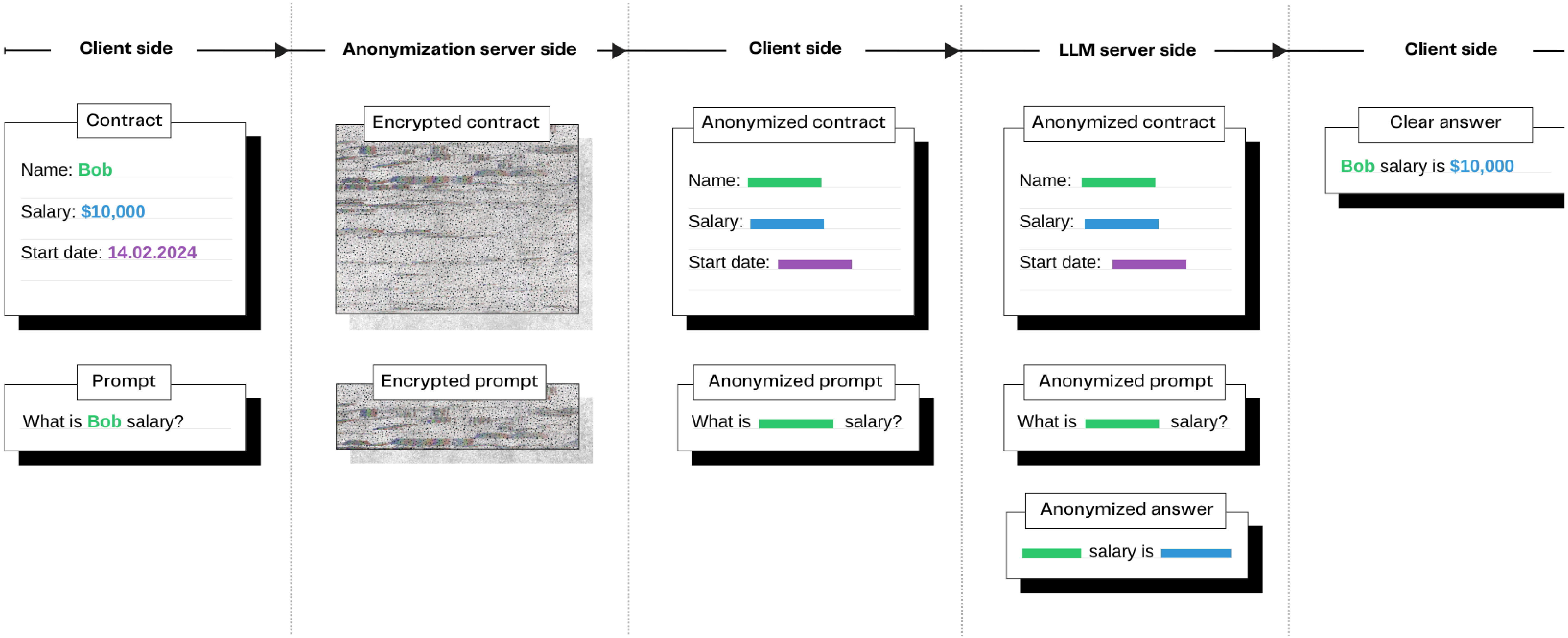


Protecting models: the hybrid approach



Work in Progress

Secure anonymization for private ChatGPT



https://huggingface.co/spaces/zama-fhe/encrypted-anonymization

Collaborative computation

- Several companies wanting to co-compute a common function, without sharing their data
- Companies manage the collaborative decryption of results thanks to treshold decryption

```
def f(enc_a, enc_b, enc_c):
  encrypted_concatenated_features = np.concatenate([enc_a, enc_b, enc_c])
  encrypted_predictions = fhe_model.run(encrypted_concatenated_features)
  return encrypted_predictions
```

Data-frames

- Persist encrypted tabular data while allowing FHE processing
- Encapsulate pre-processing before encryption to streamline FHE processing

Client Side

- Encrypt pandas data-frames

```
client = ClientEngine(keys_path=client_1_keys_path)
df_left = pandas.read_csv("df_left.csv")
df_left_enc = client.encrypt_from_pandas(df_left)
```

	index	day	time	size
0	2	Thur	Lunch	2
1	5	Sat	Dinner	3
2	9	Sun	Dinner	2

Server Side

- Join encrypted data-frames

```
df_left_enc = load_encrypted_dataframe(df_left_enc_path)
df_right_enc = load_encrypted_dataframe(df_right_enc_path)


df_joined_enc_server = df_left_enc.merge(df_right_enc, how=HOW, on=ON)
```

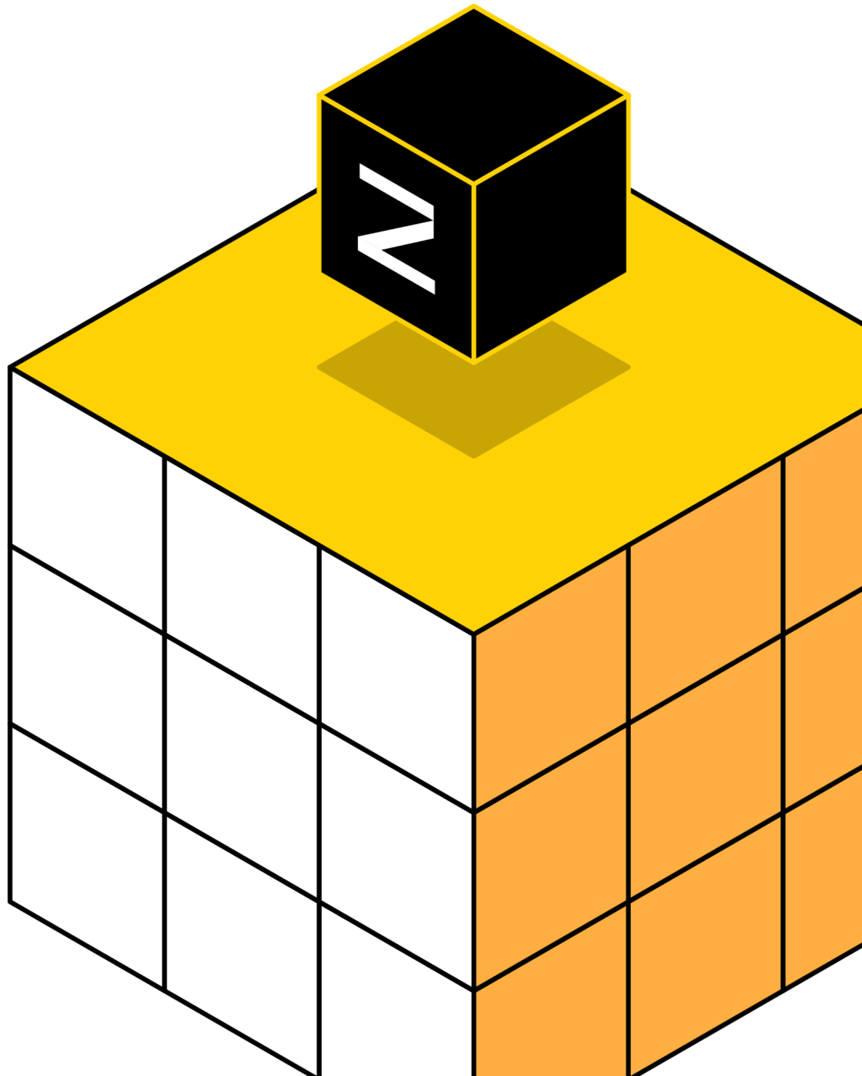
index	day	time	size
..48d4814937..	..dd6b288e52..	..497a80e2dd..	..41f496fe3a..
..0a19fbfc58..	..047a92f5bc..	..7f7a6f1167..	..5ca8e5edfc..
..79c726effe..	..6835b68ece..	..4ae3bca370..	..f4eb2bde07..

Starting Building

We are open-source

- We're open source: have a look to
  - **repo**: https://github.com/zama-ai/concrete-ml
  - **docs**: http://docs.zama.ai/concrete-ml
  - **live demos on Hugging Face**: https://huggingface.co/zama-fhe

- And if you're interested by the scientific side:
  - Neural Network Training on Encrypted Data with TFHE
  - Deep Neural Networks for Encrypted Inference with TFHE (CSCML 2023)
  - Privacy-Preserving Tree-Based Inference with TFHE (MSPN 2023)



How to learn and make yours

- Reproduce examples from the documentation
- Then:
  - create some task in ML (linear or tree-based models)
  - once it works, compile it with Concrete ML
  - once familiar, continue with an easy DL example, have a look to QAT

- We'll be happy to support you on discord.fhe.org/

Contact and Links

benoit.chevalliermames@zama.ai
jordan.frery@zama.ai

zama.ai

github.com/zama-ai

community.zama.ai/

discord.fhe.org

ZAMA