The Importance of Randomness in ML

dr. Richárd Ádám Vécsey

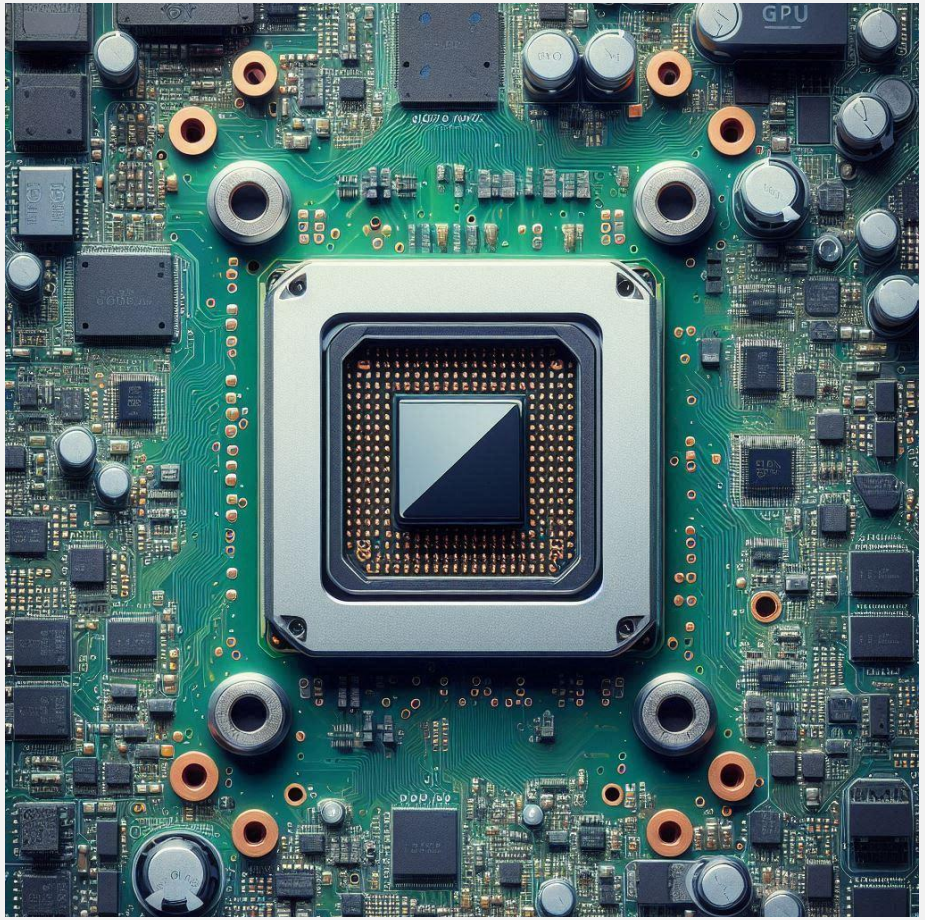
GPU DAY 2025

23. 05. 2025.

Objectives

- **Understanding the (un)importance of random**
- Perspective of a developer
- Use cases
- Pitfalls
- Prevention
- Consequences
- Examples

Use-cases



- Fixed random
  - **Reproducibility – experiments**
  - Proving a scientific result
  - Dataset validation
  - Model structure validation
- Randomness
  - **Variability – real life scenarios**
  - Providing noise
  - Generalization of models
  - Getting out of local minima

Where Randomness Happens I.

- Weight & bias initialization
- Dataset shuffling
- Dropout
- Floating point operations
- Pooling operations
- Backpropagation

Where Randomness Happens II.

- OS level
- Python level
- Library level
  - Numpy
  - TensorFlow
  - PyTorch
- Hardware level
  - CUDA

Examples I.

- Python

```
import random
random.seed(0)
```

- PyTorch

```
import torch
torch.manual_seed(0)
```

Examples II.

- TensorFlow

```
import tensorflow as tf
tf.random.set_seed(0)
```

- Numpy

```
import numpy as np
np.random.seed(0)
```

Examples III.

- OS (new process)

```
os.environ['PYTHONHASHSEED'] = str(0)
```

Examples IV.

- Sklearn

```
from sklearn.model_selection import
train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=0, shuffle=True)
```

Examples V.

- CUDA

```
torch.backends.cudnn.benchmark = False
```

```
torch.use_deterministic_algorithms(True)
```

```
CUBLAS_WORKSPACE_CONFIG=:[SIZE]:[COUNT]
```

Examples VI.

- PyTorch DataLoader

```
def worker_seed(worker_id):
    seed = torch.initial_seed() % 2**32
    numpy.random.seed(seed)
    random.seed(seed)

g = torch.Generator()
g.manual_seed(0)

DataLoader(train_dataset,
           batch_size=batch_size,
           num_workers=num_workers,
           worker_init_fn=worker_seed,
           generator=g,)
```

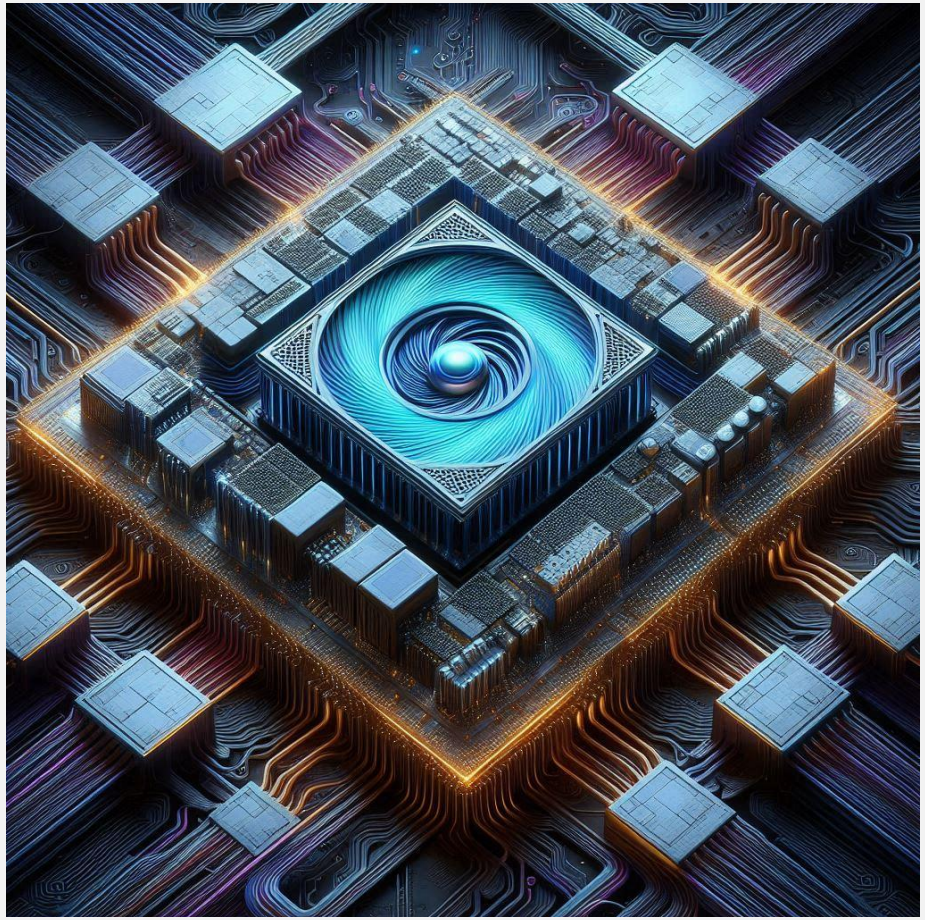
Source: https://pytorch.org/docs/stable/notes/randomness.html#dataloader)

The Randomness Trap

- **Unshuffled dataset**
  worse converge or worse generalization
- **Fixed init and random**
  suboptimal local minima
- **Train_test_split without random**
  different test sets each run

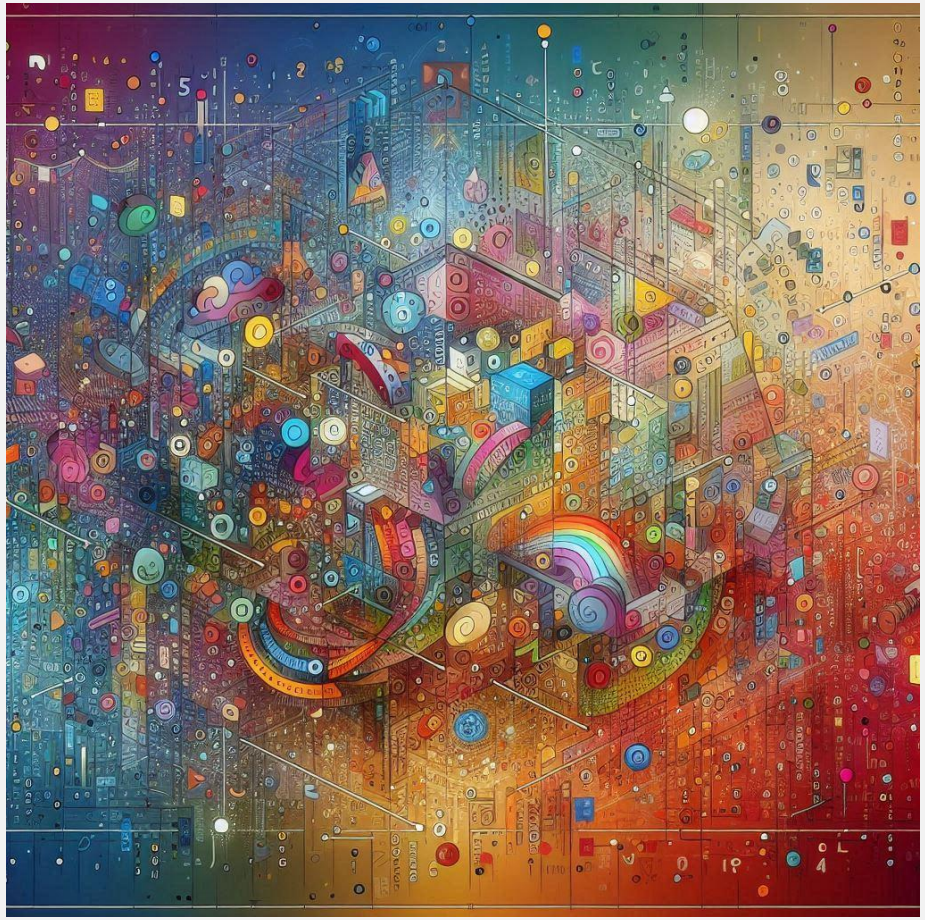
Where Things Can Still Go Wrong

- CUDA non-determinism due to backend optimizations
- Floating-point math order not guaranteed across hardware
- CPU – GPU operations
- DataLoading
- Specific layers



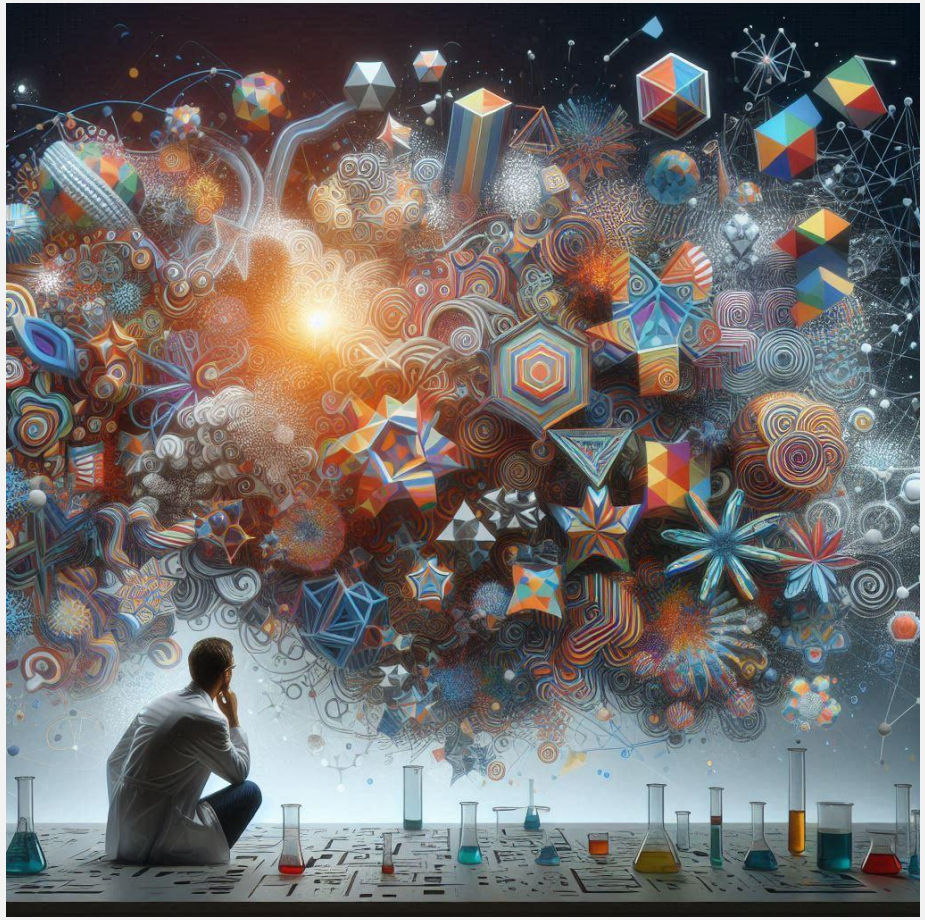
Pitfalls and Prevention

- Set random seeds globally
- Use deterministic algos
- Avoid implicit randomness
- Avoid non-det. layers
- Save the initial model state
- Save the dataset
- Be cautious of library version changes



Implications in Research & Production

- False sense of model performance from "lucky" runs
- Troubleshooting becomes harder
- In production random matters
- EU AI Act

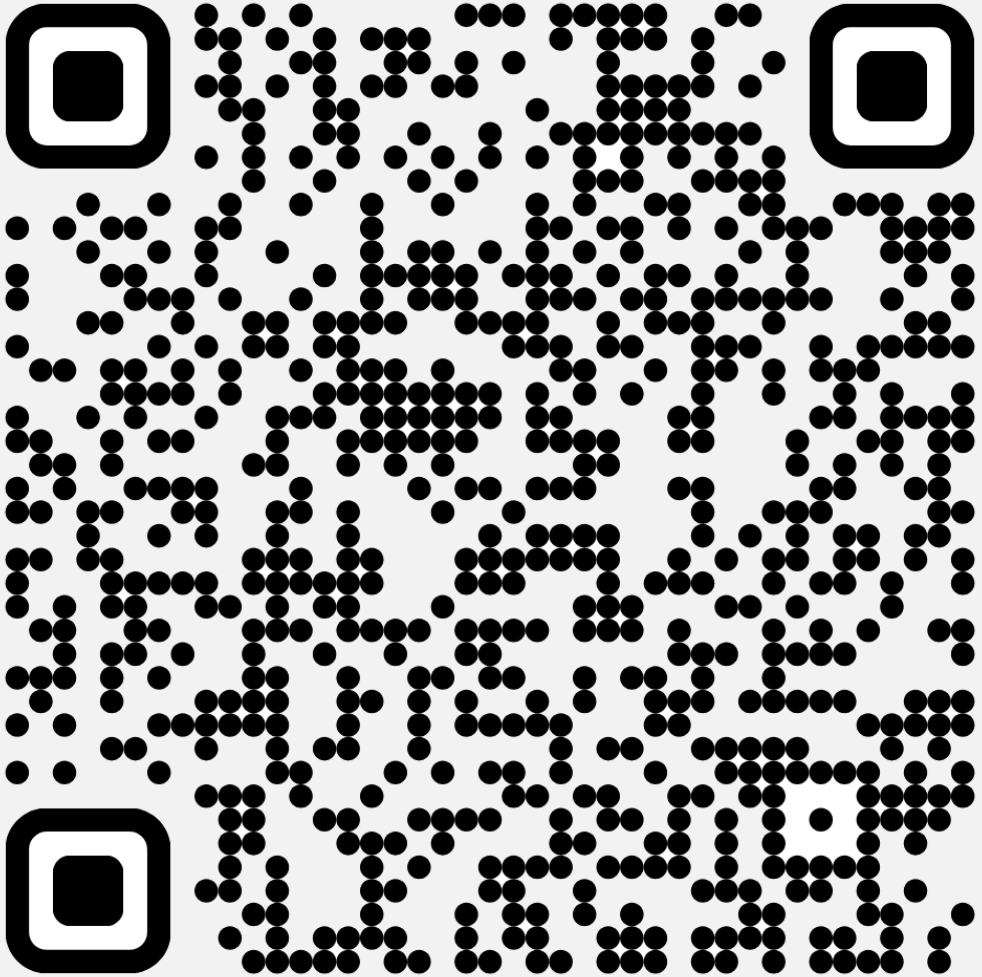


Quantum ML and Randomness

- Quantum systems = inherently probabilistic
- Pennylane
  - Numpy random
  - OS entropy
  - JAX-function
- Simulators might works with different random compared to real life

Key Takeaways

- Randomness is essential in real life scenarios, but must be managed
- Reproducibility is hard to maintain, not a binary
- Use explicit seeding across all levels if needed
- Quantum ML highlights randomness as a feature, not a bug



dr. Richárd Ádám Vécsey