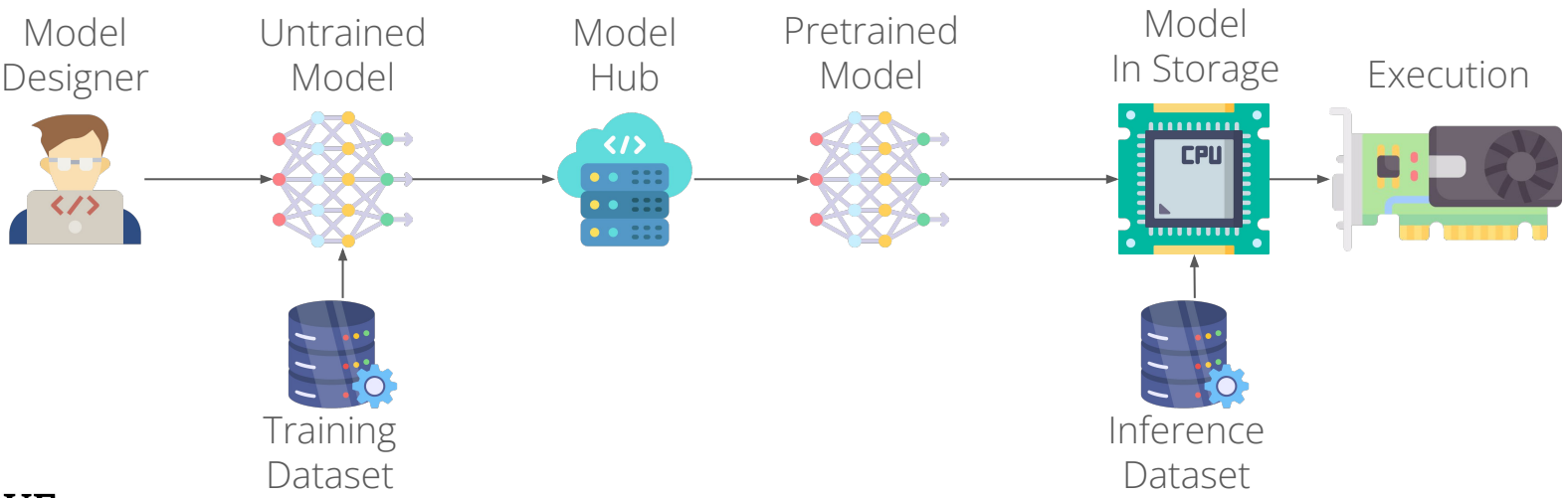
Sentry: Authenticating Machine Learning Artifacts on the Fly

Andrew Gan, Zahra Ghodsi
Purdue University

PURDUE UNIVERSITY

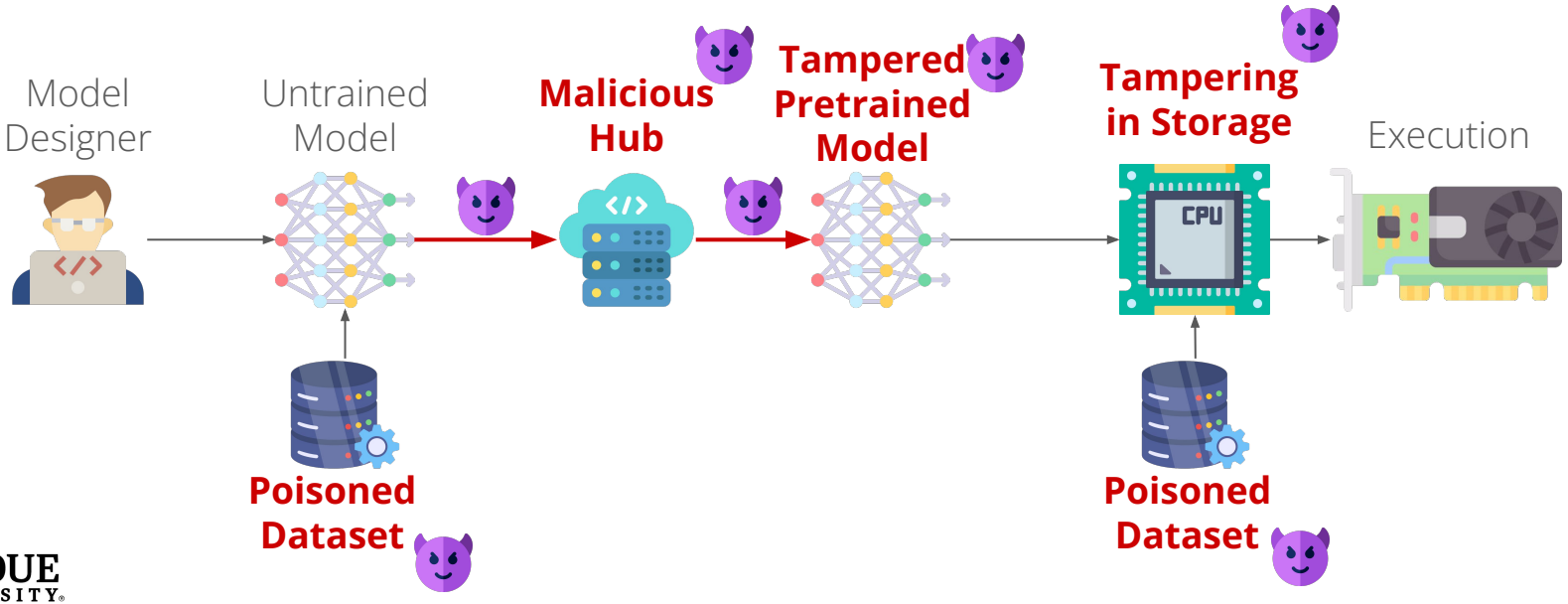
Machine Learning Supply Chain

- ML models are increasingly complex, leading to a rise in compute demand
- Multiple parties handle different parts of the MLSC
- Vulnerabilities exist in every step of the supply chain, targeting algorithms and ML artifacts such as models and datasets



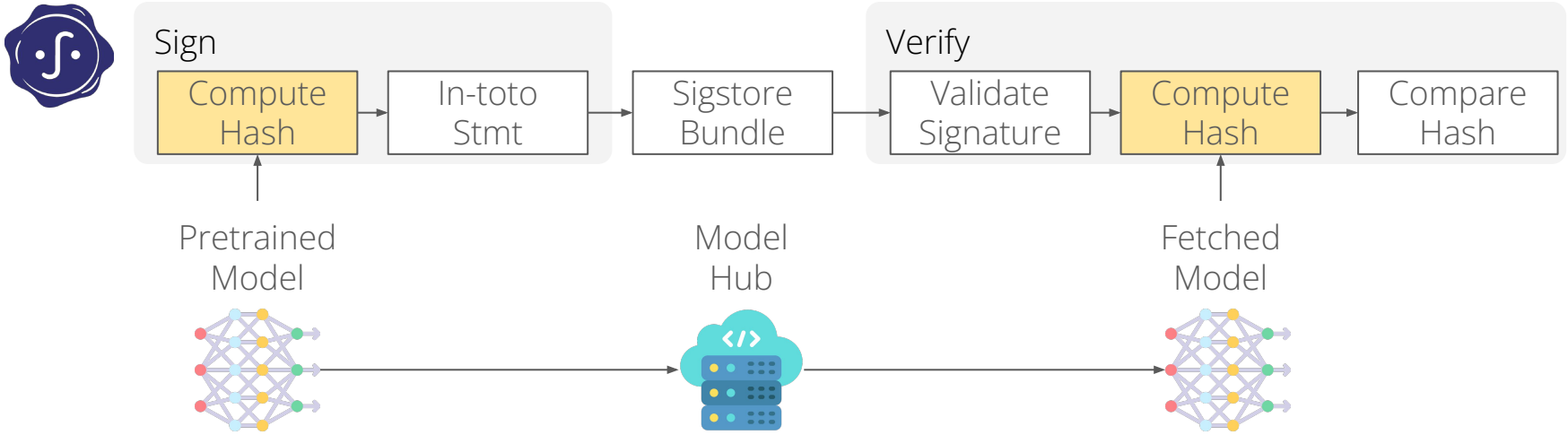
Threat Model

The poisoning of ML Artifacts impact downstream performance and security



Existing Work Not Scalable to Large Scale ML

- Apply Sigstore to ensure ML artifacts not tampered
- Apply SLSA to attest ML model build and training procedures were valid
- Current hashing strategy can be slow for the scale of state-of-the-art artifacts
  - Hashing GPT2-XL with 1.6 billion parameters takes 26s using SHA3



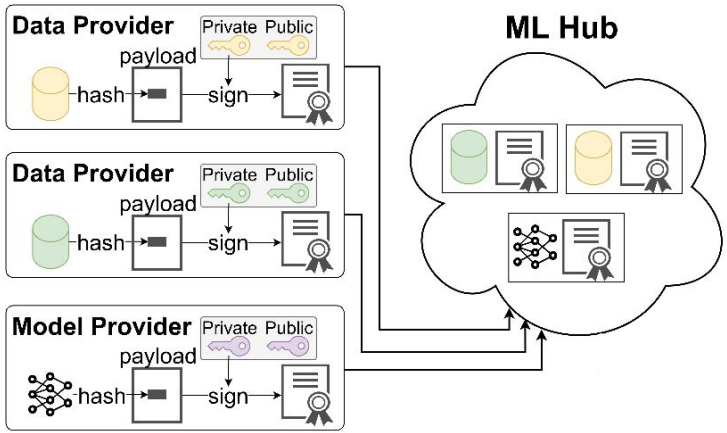
Existing Work Does Not Support Authentication on GPU

- They do not work with GPU data movement solutions
  - GPUDirect Storage: NVMe □ GPU
  - GPUDirect RDMA: NIC □ GPU

- We propose a GPU implementation for artifact authentication

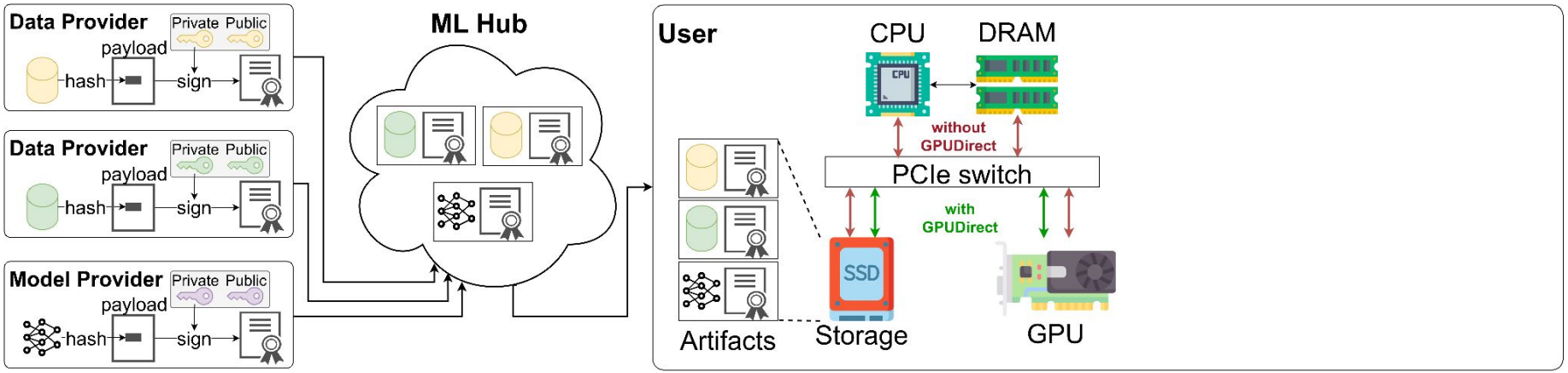
Sentry: Authenticating ML Artifacts on the Fly

- Attest ML dataset and model integrity after it is loaded into GPU memory for training or inference



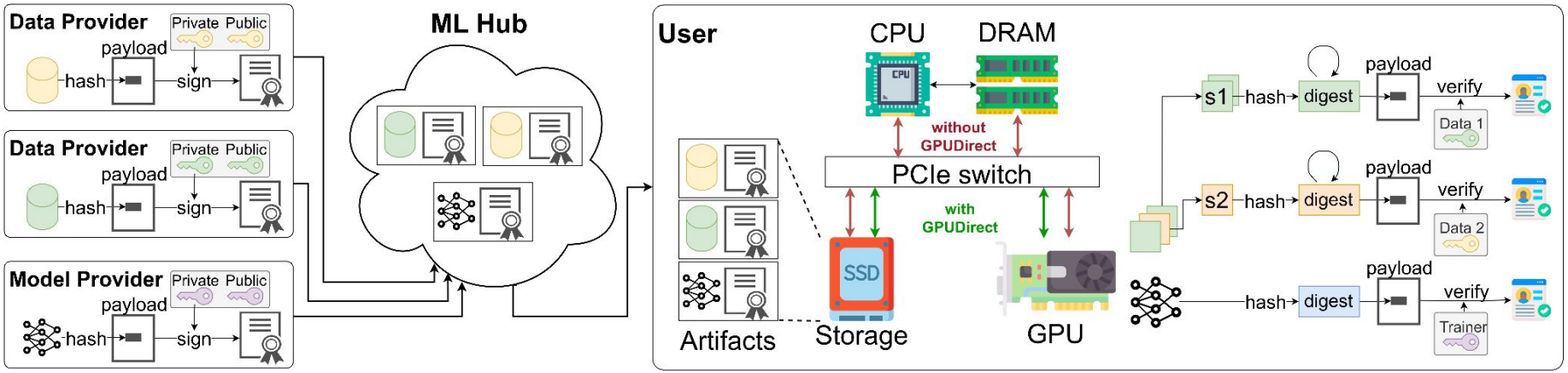
Sentry: Authenticating ML Artifacts on the Fly

- Attest ML dataset and model integrity after it is loaded into GPU memory for training or inference

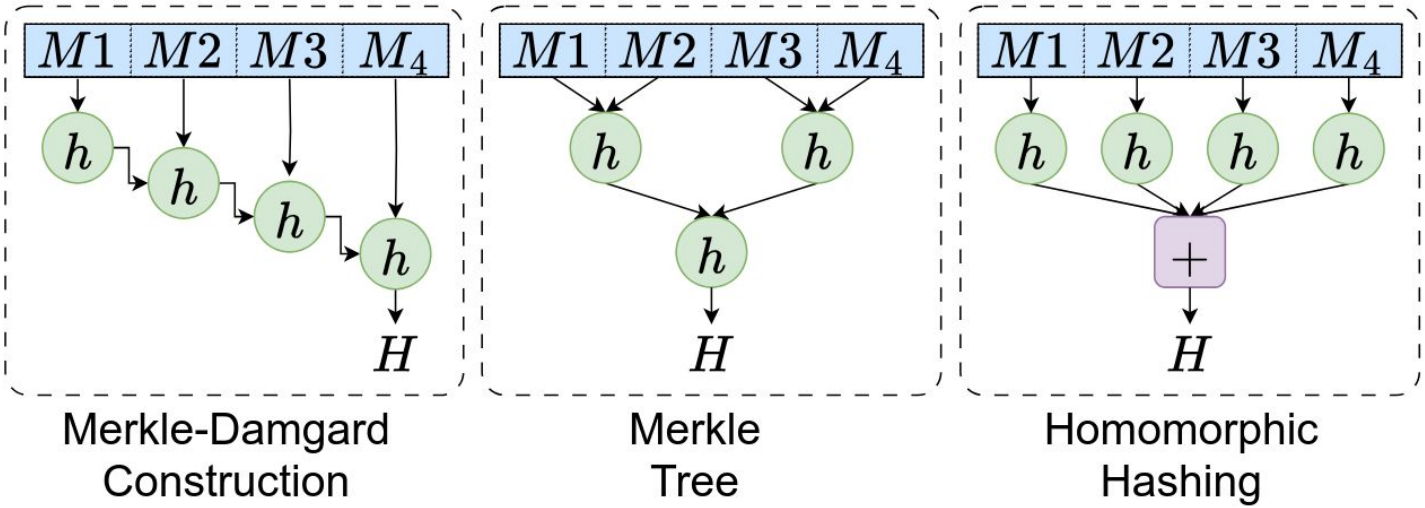


Sentry: Authenticating ML Artifacts on the Fly

- Attest ML dataset and model integrity after it is loaded into GPU memory for training or inference



Targeted Hashing Constructions

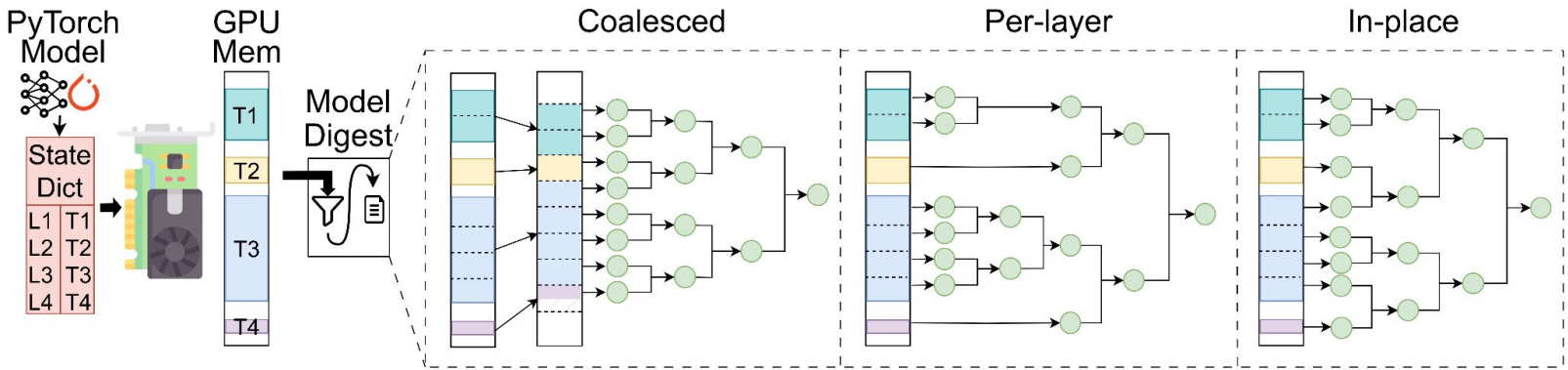


Homomorphic hashing is an additive hashing scheme where the hash digest of a group of items may be obtained by adding the individual digests of each item in the group.

$$\text{LtHash}_{n,d}(\{M1, \ldots, Mk\}) = {}^{k}\sum_{i=1} \hbar(M_i) \bmod q$$

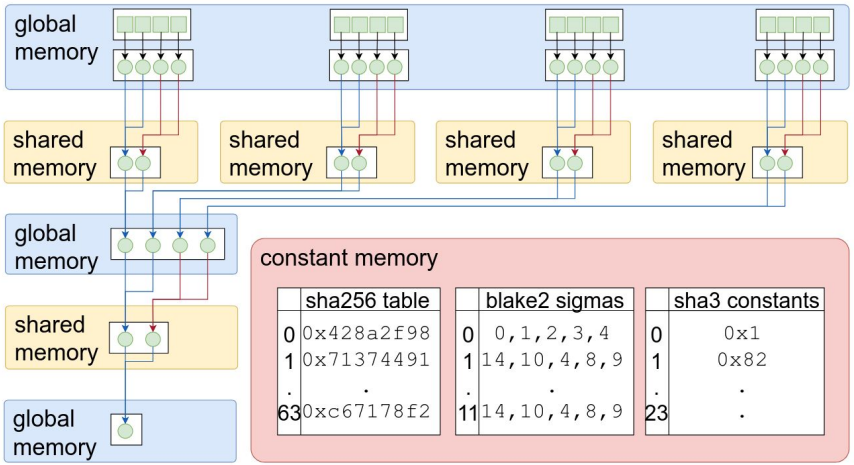
Model Hashing

- Weights of a layer are stored in one tensor; weight tensors across different layers are scattered in memory.
- This is because at any given time, only parameters belonging to the current layer need to be accessed.
- Three modes of operation for merkle tree hashing
  - What level of granularity within a model needed?
  - Is minimizing additional runtime to training / inference a priority?

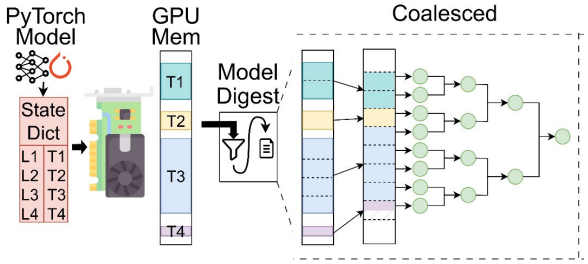


GPU Memory Types for Different Purposes

RTX A6000	Size	Access time	Access
Global	48 GB	600-800 cycles	anywhere
Constant	32 KB	300-500 cycles	anywhere
Shared	48 KB	1-3 cycles	Thread block
Register	192 B	1-3 cycles	Thread



Model Hashing Coalesced



Pros:

- Coalesced memory access during hashing
- Easier to add custom padding due to customizable buffer size

Cons:

- High granularity with a single hash of all model parameters
- High memory consumption equal to model size

```
Model: {
     “algorithm”: MERKLE-GPU-SHA256,
     “digest”: 4f4c469a2caa2f7e,
}
```

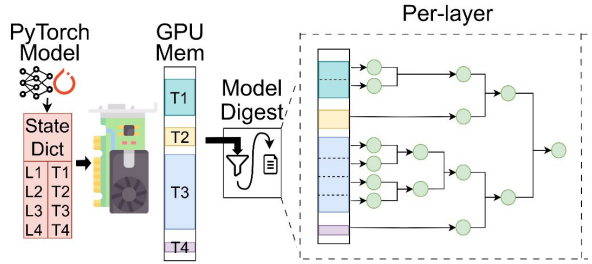
Model Hashing Per-layer

Pros:

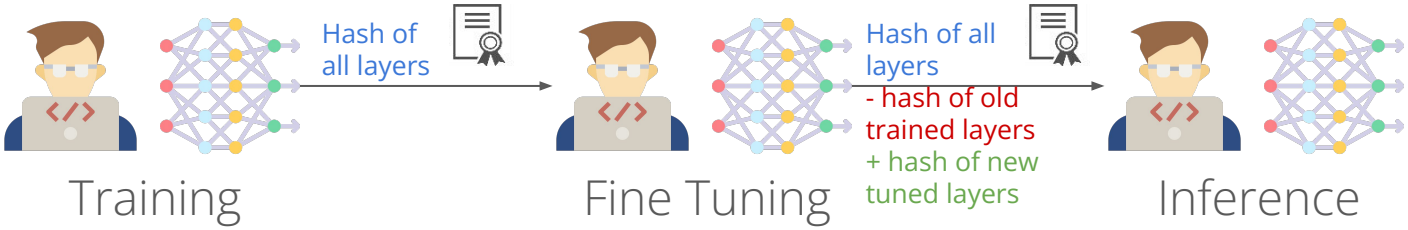
- Supports two levels of granularity (layer vs model) for tuning and inference
- Mixed use of hashing algorithms for single hashes (layer) and additive hashes (model)

Cons:

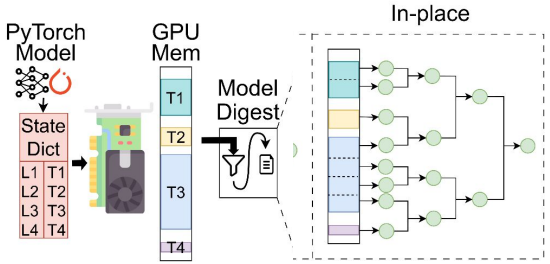
- Longer hashing time: reduction to layer digests before reduction to model digest
- Larger intoto payload to sign




```
Conv 1 weights: {
      "algorithm": MERKLE-GPU-SHA256,
      "digest": f67f7e2a4b732970,
}
FC 3 weights:   {
      "algorithm": MERKLE-GPU-SHA256,
      "digest": 8991ff3998644ea7,
}
Model: {
      "algorithm": HOMOMORPHIC-GPU-LATTICE,
      "digest": 2b83c2e9e4c86a27,
}
```



Model Hashing Inplace



Pros:

- Fastest hashing mode due to no additional buffers and not computing per-layer digests
- No additional global memory allocation as data is read directly from state dictionary

Cons:

- Single high-level hash granularity

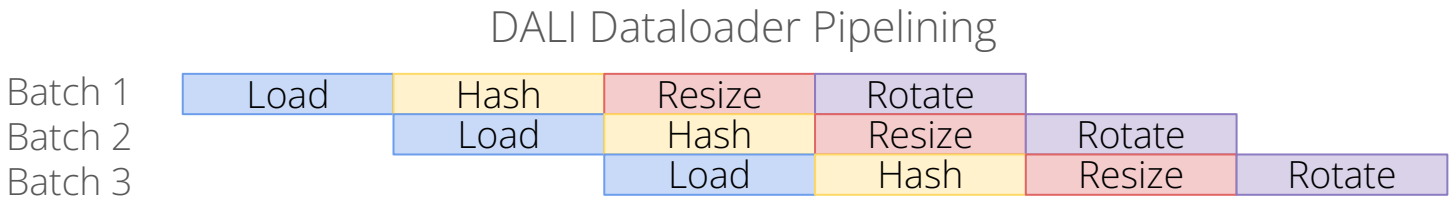
```
Model: {
     "algorithm": MERKLE-GPU-SHA256,
     "digest": 4f4c469a2caa2f7e,
}
```

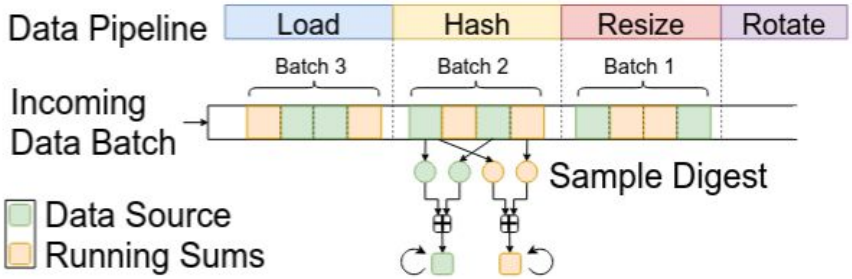
Memory Consumption

	Model Specifications			Additional Memory Usage		
	Weights (M)	Layers (num)	Size (MB)	Coalesced (MB)	Per-layer (MB)	In-place (MB)
ResNet152	60	932	270	270	2	4
Bert	109	199	538	538	4	4
GPT2	124	149	1077	1077	8	6
VGG19	143	38	1077	1077	8	6
GPT2-XL	1610	581	8623	8623	54	54

Dataset Hashing

- Data authentication implemented on GPU with support for **mixed-source datasets**
- Sentry data pipeline integrates NVIDIA DALI to offload data processing to GPU



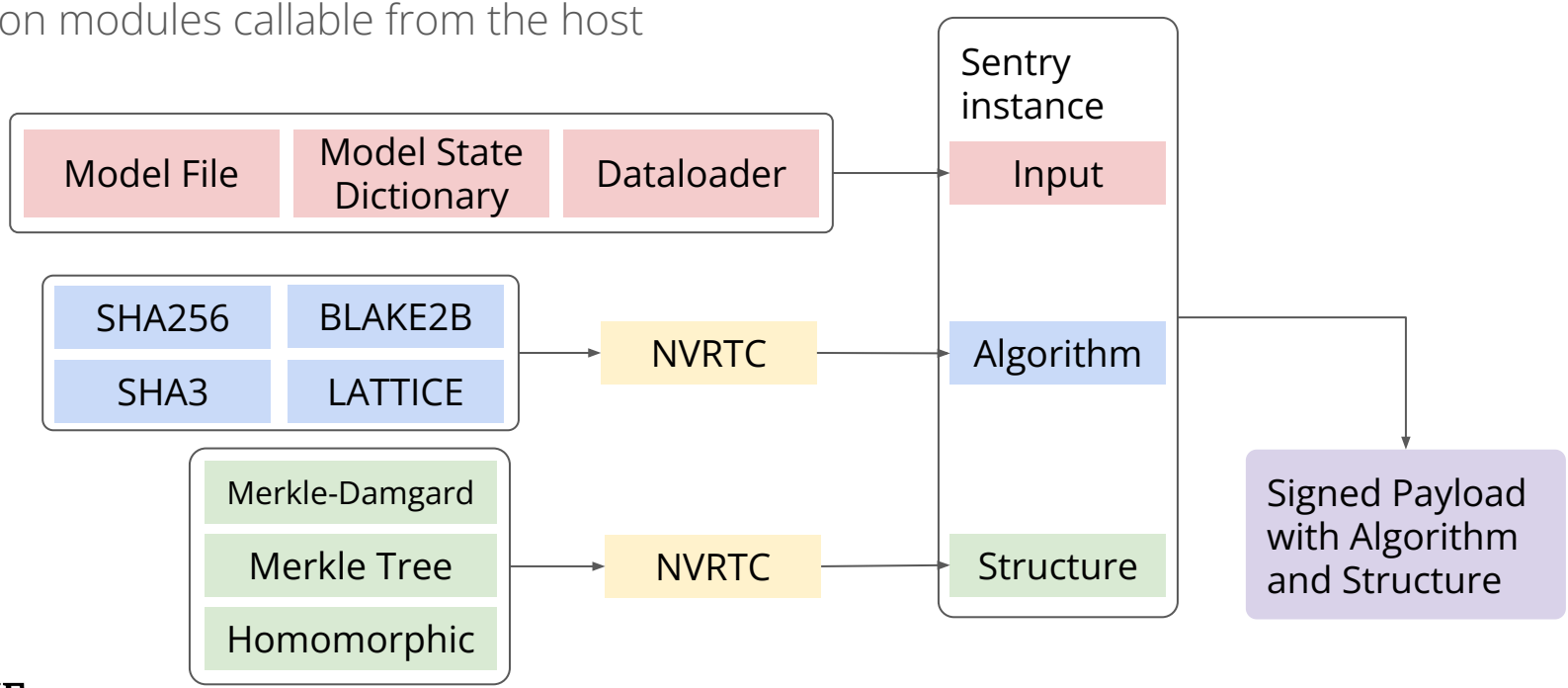



```
Dataset 1: {
  “algorithm”: HOMOMORPHIC-GPU-LATTICE,
  “digest”: f67f7e2a4b732970,
}
Dataset 2:   {
  “algorithm”: HOMOMORPHIC-GPU-LATTICE,
  “digest”: 8991ff3998644ea7,
}
```

PURDUE UNIVERSITY

Library Design

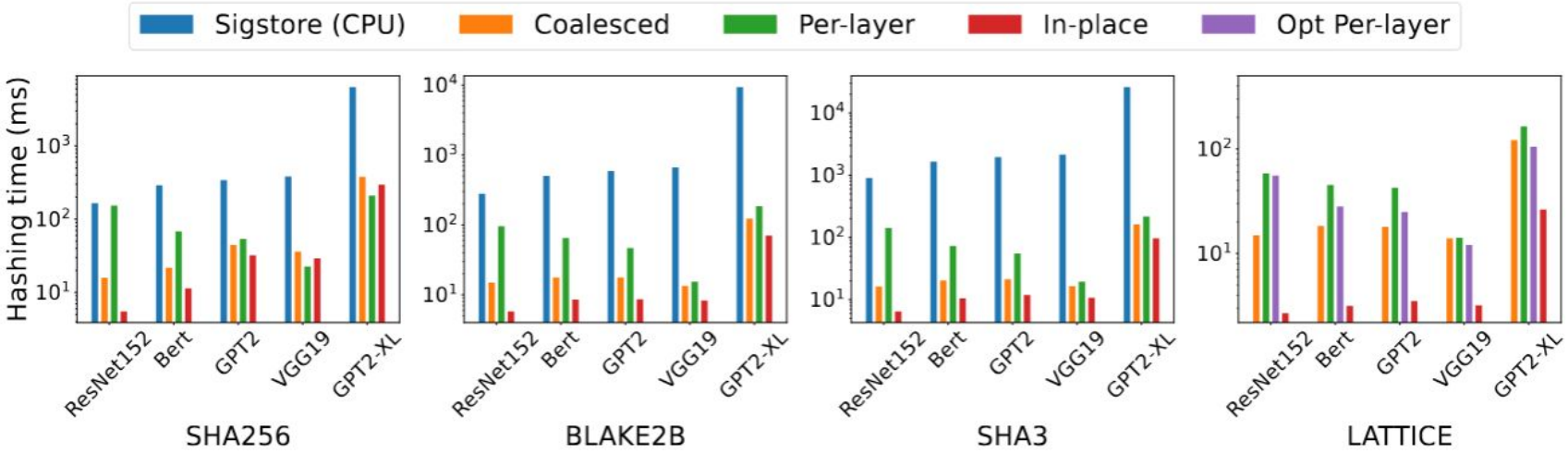
- User selects a valid combination of input, hash algorithm and structure
- Nvidia runtime compiler (NVRTC) translates device-side CUDA kernels into python modules callable from the host



Evaluation - Model Authentication

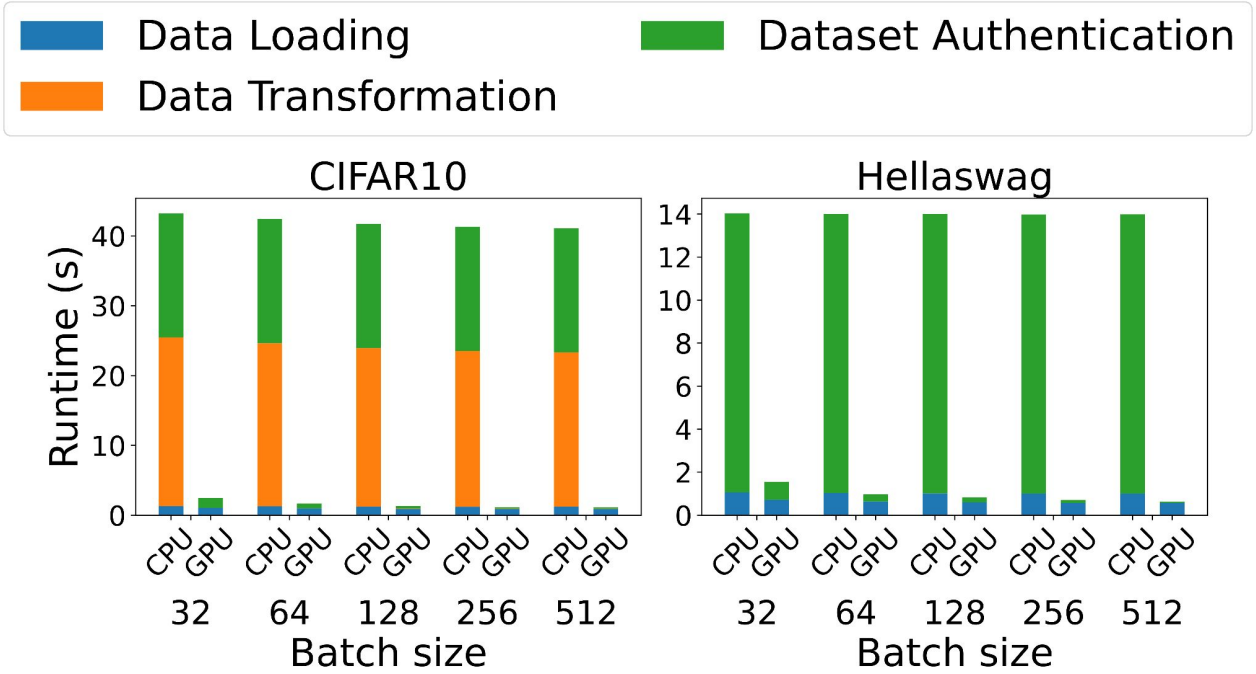
- In-place Merkle tree achieves the lowest runtime in most settings
  - 11x to 296x compared to Sigstore CPU baseline

- Largest model GP2-XL with 1.6B params observes the greatest speedup



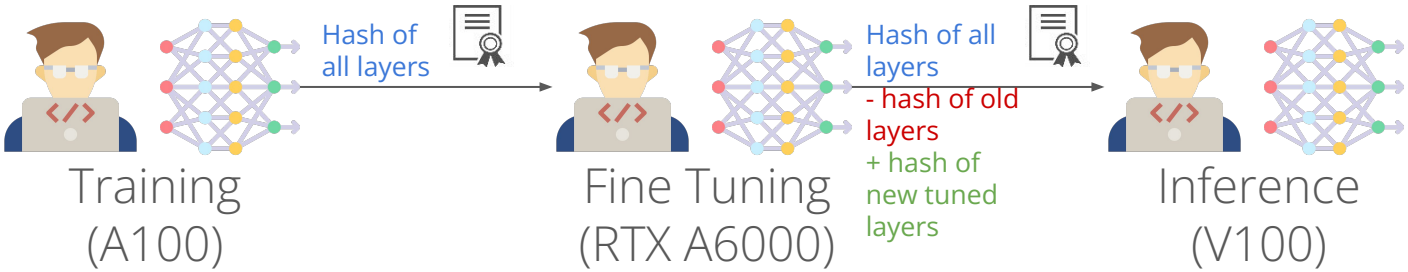
Evaluation - Dataset Authentication

- Sentry incorporates GPUDirect Storage for data loading
- For CIFAR10 and Hellaswag datasets, Sentry achieves up to 27x and 10x runtime improvements



Evaluation - Cross-platform Compatibility

- Sentry has CPU implementation that validate GPU acceleration
- Allows distribution of ML supply chain across heterogenous hardware
- Tested on CPU, V100, A100, RTX A6000



Outcome

- Attest ML model integrity after it is loaded into GPU memory for training or inference
- Attest ML dataset integrity while it is loaded into Nvidia's preprocessing pipeline