Atlas: A Framework for ML Lifecycle Provenance & Transparency

Marcin Spoczynski
Intel Labs
Hillsboro, Oregon, USA
marcin.spoczynski@intel.com

Marcela S. Melara
Intel Labs
Hillsboro, Oregon, USA
marcela.melara@intel.com

Sebastian Szyller
Intel Labs
Helsinki, Finland
contact@sebszyller.com

***Abstract*—We propose *Atlas*, a framework that enables fully attestable ML pipelines to address ML supply chain risks. *Atlas* leverages runtime pipeline monitoring and open specifications for data and software provenance to collect model artifact integrity and end-to-end lineage metadata. *Atlas* combines trusted hardware and transparency logs to enhance metadata integrity and enable efficient verification of ML pipeline operations, from training through deployment. Our prototype implementation of *Atlas* integrates open-source tools to build an ML lifecycle transparency framework.**

1. Introduction

In recent years, machine learning (ML) models, have become increasingly popular. The pervasive use of large language models (LLMs), in particular, and multi-stakeholder involvement in model creation and deployment exacerbate security and privacy risks. These considerations are emphasized by the global nature and the complexity of large-scale ML deployments with different lifecycle stages (e.g., training dataset collection, execution of training).

Each stage is vulnerable to malicious or dishonest parties. For example, data can be poisoned [1], [2] during collection or training. Service providers executing outsourced training can shorten or omit critical steps to reduce their cost. Popular model hubs hosting pre-trained models are vulnerable to compromises that may result in corrupted, reduced, or malicious model distributions [3], [4]

On the other hand, recent regulations [5], [6] require model builders and other stakeholders to provide evidence of ML model security and trust. They may need to prove low bias in their training data, offer easily verifiable performance claims, or demonstrate end-to-end integrity of model creation in high risk domains.

To address these challenges, integrity of the entire ML lifecycle must be recorded verifiably – beginning with the data, through the training, and finally, the evaluation and deployment. Was the data modified? Did the hardware and software environment adhere to the specification? Did the contractor follow the specified training procedure? Can I trust the evaluation? How can I guarantee that I am interacting with the intended model? These are example questions that showcase the breadth of the involved challenges that must be tackled to provide end-to-end security.

We introduce *Atlas*, a framework for enhancing the security and transparency of the lifecycle of ML models. *Atlas* establishes the baseline of fundamental components and capabilities needed for comprehensive provenance tracking at each stage of the ML lifecycle. Thus, rather than preventing attacks entirely, *Atlas* detects tampering by verifying the ML lifecycle.

Atlas addresses two challenges unique to ML lifecycle transparency. First, in contrast to the clear dependency trees of traditional software, datasets and algorithm code are tightly coupled in ML models, creating significantly more intricate provenance graphs [3]. Attesting the integrity of these relationships requires mechanisms that can track cross-organizational transformations across heterogeneous artifacts of varying sizes and formats [7].

Hence, *Atlas* monitors ML pipelines during execution and automatically collects ML system information using several data and software provenance frameworks. To strengthen the integrity of provenance generation, *Atlas* relies on hardware trusted execution environments.

Second, models are not static binaries – they can be further customized by downstream users. That is, ML models undergo a non-linear lifecycle where deployment results often necessitate refinement, creating feedback loops between inferencing and data processing [8].

This iterative process requires special adaptations to provenance tracking mechanisms. *Atlas* represents these non-linear development paths and enables cryptographic auditing using Merkle trees [9].

We claim the following contributions:

1. We introduce *Atlas*, a framework designed for end-to-end ML lifecycle transparency.
2. We instantiate *Atlas* using Intel Trust Domain eXtensions [10] and metadata-based provenance tracking.
3. We evaluate our *Atlas* prototype through a fine-tuning case study with a BERT model [11], [12].

2. Background & Related Work

Data Provenance & Authenticity. Provenance and attribution of media has recently received attention due to the online proliferation of manipulated or forged content using generative ML models [13], [14]. Prior work relies primarily on cryptographic hashing and digital signatures to provide data authenticity and integrity.

The Coalition for Content Provenance and Authenticity (C2PA) specification [15], [16] digitally signs assertions about content origin to provide tamper-evident data audit trails [17], [18]. C2PA's extensible metadata format also makes it suitable for ML model artifacts [19].

The Open Source Security Foundation (OpenSSF) Model Signing project [20] is a parallel effort focusing on

integrity and authenticity of trained models. Other prior work [14], [21], [22] builds upon hashing and signing with distributed ledger technologies to create transparent and immutable content or provenance records.

These techniques are crucial building blocks for verifying ML model artifact authenticity and provenance, but each alone is insufficient for end-to-end ML supply chain transparency. In contrast, *Atlas* aims to integrate such techniques into ML systems to track model artifact provenance directly where the transformations occur.

Supply Chain Integrity. Recent cybersecurity regulations [23], [24] have shifted industry focus toward detecting supply chain threats via software dependency tracking with SBOMs [25]. Similarly, the AIBOM framework [26], [27] focuses on ML model supply chain management.

Complementing BOM, efforts like OpenSSF Supply Chain Levels for Software Artifacts (SLSA) [28] and SPDX Build [29] collect build provenance, i.e., metadata describing how a particular artifact was produced. This approach is also being considered for ML model fine tuning [30]. Building on such supply chain metadata efforts, a number of frameworks [31]–[33] provide mechanisms for collecting, digitally signing and verifying authenticated claims *across* supply chain steps.

Atlas borrows concepts from supply chain integrity to support multiple types of software artifact provenance at any stage of the ML lifecyle, providing a more comprehensive view of a model's supply chain. Works about evidence for other properties of the ML lifecycle such as assurance [8] are complementary.

Model Lineage Tracking. The EQTY Lineage Explorer [34] tracks model artifacts throughout the training process, capturing relationships between datasets, model checkpoints and hyperparameters. However, unlike *Atlas*, it lacks cryptographic authenticity properties and focuses primarily on manually collected development-time lineage, rather than automatically capturing and linking information across the entire ML lifecycle.

ML experiment trackers like Weights and Biases [35], Neptune [36] and Kubeflow Pipelines [37] offer detailed run-time logging of model metadata about training runs, metrics, and model artifacts. These tools do not integrate transparently with common ML frameworks, and they typically provide only unauthenticated metadata. *Atlas*, on the other hand, seeks to make model lineage verifiable and support integration into ML frameworks like PyTorch [38].

Hardware-Based Security for ML. Recent developments in trusted execution environment (TEEs) technologies have made it more practical to run large-scale systems and workloads [39], [40], including ML pipelines. Chrapek et al. [41] deployed and optimized a large language model (LLM) inside a TEE, showing how secure enclaves help protect LLM code and data while *in use*. They maintain practical performance in two TEE configurations based on Intel Software Guard eXtensions (Intel SGX) [42] and Intel Trust Domain eXtensions (Intel TDX) [10].

Laminator [43] and PraaS [44] demonstrate the application of TEEs to ML model or dataset property attestation and verification. Several efforts [7], [45], [46] use TEEs to build confidentiality frameworks for different ML lifecycle stages. These works are complementary to *Atlas* and may enable us to extend our framework.

Mo et al.'s survey [47] evaluates 38 works that use various TEE implementations to enhance the privacy and integrity of ML training and inference operations. The survey highlights several gaps, including the protection of full ML lifecyles, which is the primary focus of *Atlas*.

We provide additional background in Appendix A.

3. System Overview & Threat Model

3.1. Terminology

In *Atlas*, an ML model is composed of several **artifacts** that include the training dataset, ML algorithm, ML framework (e.g., PyTorch), model configuration (e.g., hyperparameters, weights), and metadata (e.g., license).

The **ML lifecycle** consists of various stages, including data preparation, training, evaluation and deployment. A common synonym for ML lifecycle is "ML supply chain", so we use these terms interchangeably in the paper.

The **ML pipeline** defines the sequence of operations or a workflow that transforms a model artifact at a particular stage of the ML lifecycle [48]. To support standardization and repeatability, the pipeline also facilitates workflow management and automation.

The **ML system** is the set of hardware and software components that implement and execute an ML pipeline. For example, an ML system for training may include orchestration tools, an authentication service, storage systems, automation infrastructure, and specialized compute hardware (e.g., GPUs, TPUs, or custom accelerators).

In *Atlas*, **metadata** describes two mains aspects about an ML model. First, provenance metadata refers to the origin and history of custody of a model artifact, including its history of transformations as it traverses the ML lifecycle. Pipeline metadata, in turn, describes the ML systems and specifics about the operations that produced a model artifact.

An **attestation** in *Atlas* refers to any digitally signed metadata and serves as evidence for model artifact or ML system authenticity, integrity and provenance. Attestations may be generated by hardware or software.

3.2. System Model

In *Atlas*, we target the *multi-stakeholder* ML model lifecycle setting.

3.2.1. Stakeholders. The **Artifact Producers** are individuals and organizations that create ML model artifacts to provide or sell them to other parties. Since the artifacts may represent intellectual property and/or make use of personally identifying information (PII), producers have business and regulatory reasons to preserve their confidentiality. To save costs, artifact producers often outsource the operation of ML systems to external service providers.

ML-as-a-Service (MLaaS) providers operate and maintain the compute infrastructure needed to run ML systems. MLaaS providers may offer ML-specific services that leverage general-purpose compute (e.g., [49]–[51]), or provide special-purpose systems (e.g., [37], [52]) that can build and run third-party ML systems.

A **Hub** is a system that stores and distributes model artifacts. Thus, model pipelines typically ingest and output

artifacts to and from hubs during their execution, enabled by interfaces exposed by MLaaS providers. Hubs may be operated by artifact producers themselves or by third-party vendors, containing open or closed source artifacts (e.g., [53], [54]).

A **Transparency Service** in *Atlas* is responsible for generating, storing and distributing the metadata necessary to verify the authenticity, integrity and provenance of model artifacts. We envision model vendors and independent parties operating transparency services in practice.

Transparency services interface with MLaaS providers through *attestation clients* that run alongside ML systems to obtain and attest provenance and pipeline metadata. On the server side, a *transparency log* contains the known good values (i.e., golden values) of model artifacts and ML system components, submitted by model producers and MLaaS providers, as well as attestations collected by the clients throughout the ML lifecycle.

Verification Services evaluate or audit a particular ML model's lifecycle with the goal of detecting unintended or malicious tampering with the model at any stage. In practice, model users, vendors or regulatory entities may operate verification services.

Using the golden values[1] and attestations obtained from a transparency service, a verification service evaluates each ML pipeline and artifact of interest against a set of *expectations*. For example, a model producer may check that the MLaaS provider ran the expected pipeline code, or a model user may verify that a fine-tuned model was produced from the expected foundation model.

A **Model User** interacts with a model in an inferencing pipeline, or in a *downstream* ML pipeline as a dependency, such as a fine-tuning or evaluation pipeline (see §3.2.2).

3.2.2. ML Lifecycle. In *Atlas*, we consider four high-level stages in the ML lifecycle. Each builds upon the outputs and feedback from the others, forming a continuous cycle in which models evolve based on real-world usage.

1. Data processing: Raw data is collected, sanitized and processed into smaller units (e.g., tokens) and collated into a structure ingestable during training or evaluation.

2. Training: A training algorithm processes a given dataset using an ML system. The output is an ML model.

3. Evaluation: Following training, model properties like its performance and accuracy undergo further fine-tuning and evaluation using a testing dataset.

4. Deployment: After training and evaluation, an ML model is deployed to a production system configured for inferencing. New data obtained from clients during inference are sent back to a data processing pipeline to enhance the training dataset and the model. Model use must comply with local laws or corporate policies.

3.3. Threat Model

We consider an adversary whose goal is to produce a tampered artifact, e.g., containing a hidden malicious component, so that a transparency service generates a legitimate signature on the artifact or its metadata.

Thus, *Atlas* aims to *detect* such tampering introduced via the ML supply chain.[2] Specifically, we consider tampering by MLaaS providers, hubs and artifact producers, while model users, transparency and verification services are trusted in *Atlas*.

Compromised MLaaS providers and hubs may involve malicious insiders, or external adversaries seeking to subvert these systems by exploiting vulnerable components. Given their central position in the lifecycle, MLaaS providers and hubs may thus be able to compromise model *integrity* at various stages.

For example, a malicious MLaaS provider can poison the training data during the curation step of the data processing stage leading to backdoors. A compromised hub may, for instance, present a dataset or model with a mismatched signature (e.g., to a different model, or any of its component artifacts) to a model user or MLaaS provider, so that pipelines in subsequent stages of the ML lifecycle may ingest compromised dependencies.

As a result, these compromises propagate through the ML lifecycle if they go undetected, ultimately leading to vulnerable ML models at the deployment stage. This risk is exacerbated if a hub colludes with an MLaaS provider to introduce or accept compromised ML pipeline inputs.

Artifact producers, on the other hand, may seek to compromise ML models to bypass regulatory requirements, introduce exploitable vulnerabilities or steal private information for profit (e.g, [60]). Thus, producers may collude with other untrusted stakeholders, or intentionally inaccurately declare their dependencies, to undermine the integrity of their artifacts.

3.3.1. Trusted Parties. *Atlas* considers the model users, transparency and verification services in an ML lifecycle to be trusted. We make the following assumptions about the systems supporting these stakeholders: 1) the hardware and cryptographic primitives are implemented correctly and do not contain known vulnerabilities; 2) a separate PKI system exists and organizations representing the stakeholders follow best practices for key management, network security and access control.

Further, running attestation clients in TEEs allows us to trust *Atlas* metadata generation, or detect attempts of tampering by malicious MLaaS providers. Similarly, model users and verification services can trust the integrity of golden values and attestations stored in *Atlas* transparency logs (or detect tampering) via their tamper-evident construction (see §4.2). Verification services are trusted to properly evaluate attestations, including their digital signatures, against pre-specified model user expectations.

3.3.2. Out of Scope. Many available TEEs provide confidentiality features, but addressing PII and model intellectual property concerns *end-to-end* requires a more comprehensive framework (e.g., [7], [61]). We plan to explore confidentiality within *Atlas* as future work.

1. Golden values should be independently verified to establish their trustworthiness. Current approaches for auditing golden values include reproducibility [55] and endorsements [56]. *Atlas* is agnostic to the chosen method and assumes that evidence of golden value verification can be made available via a transparency service.

2. Analogously to software correctness (which also applies to ML algorithms and systems), establishing dataset benignity, model quality and safety is a complementary area of research that relies on certifications, e.g., adversarial robustness [57], differential privacy [58], or poisoning [59]. We leave extending *Atlas* with such mechanisms as future work.

While a critical threat to deployed AI applications, *Atlas* does not address inference time black-box attacks (e.g., evasion attacks, model extraction, membership inference) caused by malicious users; solutions to reduce this risk [62]–[64] are complementary.

Side-channel attacks against hardware enclaves, physical attacks on hardware infrastructure, and network-level denial of service attacks are also beyond the scope of *Atlas*. These attacks are the subject of a large body of prior work [47], and these complementary security measures could be added to deployments of *Atlas*.

3.4. Design Requirements

We define the following integrity and operational requirements for *Atlas*:

R1: Artifact tampering is detectable. To provide model artifact integrity, *Atlas* must enable verification services to detect unexpected modifications to model artifacts.

R2: Every model transformation is attested. Because adversaries may seek to tamper with model artifacts after they are produced by a pipeline, *Atlas* attestation clients must record every model transformation in authenticated metadata as evidence for the process, including all inputs to the transformation.

R3: Verifiable model lineage. *Atlas* verification services must be able to detect unintended/malicious changes by MLaaS providers to the expected stages of the lifecycle (e.g., pipelines operating out of order, or being omitted), from initial data processing through model deployment.

R4: Strongly isolated ML systems. To detect tampering with a pipeline during its execution, *Atlas* must restrict access to its ML system by malicious MLaaS providers, and contain compromises from propagating beyond the execution environment.

R5: Pipeline agnostic. To facilitate adoption, *Atlas* must be agnostic to any ML pipeline that integrates it.

R6: Efficiency. We seek to minimize the computational and storage overheads incurred by *Atlas* to enable the implementation and deployment of *Atlas* in ML systems using commodity platforms and services.

4. Atlas Framework

Atlas introduces two core components to the ML lifecycle: 1) the transparency service interacting with MLaaS providers; 2) the verification service for validating model integrity and provenance.

The core techniques underlying the transparency and verification services are designed to be general, allowing them to remain agnostic to the particular ML lifecycle stage or pipeline they are applied to (**R5**). Fig. 1 depicts an example ML model lifecycle with *Atlas*.

4.1. Attestation Client

Atlas combines mechanisms for artifact and runtime environment integrity to provide transparency across the stages of the ML lifecycle. Thus, MLaaS providers integrate an *Atlas* attestation client with an ML system, each running within a dedicated trusted execution environment (TEE) (e.g., Intel TDX [10] or AMD SEV-SNP [65]).

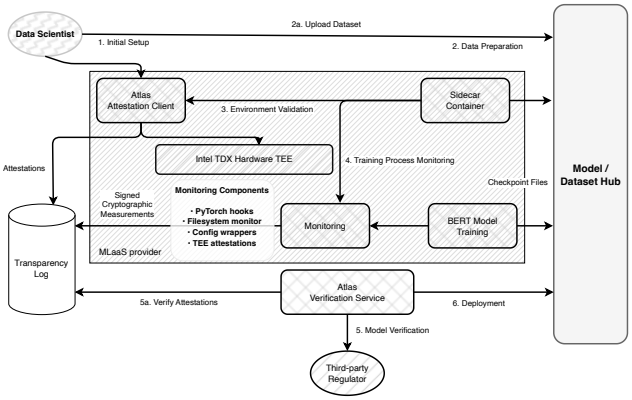

Figure 1. *Atlas* workflow for ML lifecycle transparency in a BERT Meta [11], [12] fine-tuning process. The attestation client monitors ML systems running in TEEs throughout the pipeline stages, collecting provenance metadata from initial deployment through verification.

TEEs serve two purposes in *Atlas*. First, TEE hardware-enforced memory integrity detects runtime tampering with the attestation client and ML system components by the MLaaS host (including privileged software like the OS or hypervisor). Second, TEEs provide a hardware-based root of trust for provenance and pipeline metadata.

4.1.1. Artifact Measurements. For every artifact that is ingested into and output by an ML pipeline, the attestation client computes a cryptographic measurement using a collision resistant hash function resulting in a unique, immutable identifier. If an artifact is tampered with, its measurement will differ from its golden value, allowing *Atlas* verification services to detect modifications between lifecycle stages (**R1**). Artifact producers are expected to publish digitally signed measurements as golden values whenever an artifact is first created.

4.1.2. Model Transformation Integrity. The attestation client is also responsible for generating provenance and pipeline metadata describing the transformation process and ML system that produced a new model artifact. Before pipeline execution begins, the client obtains a hardware attestation to its initial state from its TEE, which includes measurements of the client's execution environment.

At the start of pipeline execution, the attestation client remotely attests the ML system's TEE [10], [65], verifying the integrity of the *pipeline's* compute environment. Specifically, the client checks that the ML system's firmware, OS and pipeline code match the golden values published by the MLaaS provider to ensure that a pipeline starts from a known good state.[3]

Throughout pipeline execution, TEE hardware enforces memory integrity checks and isolation of executing attestation client/ML system code and in-memory data, reducing the risks of interference by compromised MLaaS providers or any co-located ML systems (**R4**). Further, the attestation client continuously monitors an ML system's execution, which allows it to determine when artifacts move into or

3. Because only select TEE implementations [66] provide built-in support for attested interactions with I/O devices or ML accelerators like GPUs, it is challenging to distinguish between benign and malicious *runtime* modifications to pipelines via the network, disk, etc (see §6.4).

outside of the system, and to collect information about operations that transform the input artifacts (see §5). For example, during data processing, the client tracks dataset modifications and preprocessing operations; during model training, *Atlas* captures state changes in model weights, hyperparameters, and configurations.

When pipeline execution concludes, the attestation client generates pipeline metadata containing all collected ML system runtime information and the ML system's TEE attestation. Then, the client creates provenance metadata including 1) artifact measurements, 2) operations producing the outputs, 3) TEE attestation for the client, 4) pipeline metadata.

The attestation client then digitally signs the provenance metadata with keys it generates within the TEE, cryptographically binding ML artifacts to the pipeline and precursor artifacts that created them in a *transformation attestation* (**R2**). The client uploads this attestation to the transparency service.

4.1.3. Provenance Chains. To enable ML model provenance tracking throughout all of its lifecycle stages, the attestation client embeds the cryptographic hash of precursor artifact attestations into every artifact's transformation attestation. These hash values are digitally signed as part of the transformation attestation, enabling detection of unexpected/malicious modifications between attested ML artifact transformations. Thus, *Atlas* attestation clients establish an authenticated, verifiable *provenance chain* representing a model's lineage relationships (**R3**).

4.2. Transparency Log

The transparency service's log makes all published golden values and attestation client-generated metadata available to verification services. To enable efficient insertion and provenance verification while accommodating the cyclical nature of the ML lifecycle, *Atlas* relies on two data structures (**R6**).

First, to provide cryptographic tamper-evidence for the stored values, the transparency log is constructed using an *append-only* Merkle tree [9], meaning that pipeline metadata can be efficiently inserted in the right-most empty leaf node of the tree (e.g., as in [32]). Second, to enable more efficient verification of provenance *across* pipelines (or even cycles of the ML lifecycle), *Atlas* can represent each discrete pipeline/cycle using a different Merkle tree. These separate trees are linked by embedding the Merkle root hash of the preceding pipeline or cycle into the "latest" Merkle tree structure (e.g., as in [67], [68]), providing a temporal cryptographic *tree chain*. Optimizations to chained Merkle trees have been developed in prior research [68]–[70].

4.3. Verification Service

Stakeholders in *Atlas* seek to validate that artifacts have not been tampered with and that they were produced by expected pipelines running in high-integrity execution environments. To respond to verification requests, the *Atlas* verification service obtains golden values and transformation attestations from transparency logs relevant to an artifact in question.

First, the verification service validates the digital signatures on attestations and golden values to authenticate their producers. It then checks whether the artifact matches its golden value. If these checks pass, the service inspects the transformation attestations to confirm the ML system and pipeline operations ran as expected based on TEE attestations and golden values. *Atlas* validates artifact lineage by traversing the provenance chain, enabling efficient verification through batching related artifact types and maintaining a cache of verified transformations. This avoids repeated inspection of unchanged artifacts (**R6**), which particularly benefits iterative ML pipelines.

5. Implementation

Our proof-of-concept implementation integrates with PyTorch [38] and Kubeflow [37] through standard APIs for metadata tracking and execution monitoring within ML pipelines. This integration approach enabled us to avoid significant modifications to our case study pipelines (§6), while maintaining *Atlas*' security and transparency enhancements. We leverage Intel TDX [10], a virtualization-layer TEE, to provide the hardware-based security primitives for ML systems and attestation clients in *Atlas*.

The attestation client is implemented as two components. First, a continuous ML system monitor integrates with PyTorch to collect metadata for a given pipeline. Second, the metadata sidecar (§5) running inside a dedicated Intel TDX TEE generates ML artifact and pipeline metadata in C2PA manifest format [16].

Due to current poor support for automated C2PA manifest generation for ML models, we implemented a Rust-based library and CLI[4] that captures artifact measurements, Intel TDX attestations, and digital signatures in C2PA format. Supporting other software provenance formats [28], [29] is future work.

We extend Sigstore's Rekor [32] to support *Atlas* C2PA-based model transformation attestations, validating signatures and measurements to ensure only properly signed artifacts are stored.

Out of space considerations, we provide additional details about the implementation in App. B.

***Atlas* Workflow Example.** We illustrate *Atlas*'s end-to-end operations through an example with fine-tuning a BERT model for sentiment analysis:

1) **Pipeline Environment Provisioning:** MLaaS provider sets up *Atlas* attestation client and ML system monitor in Kubeflow.
2) **Data Preparation:** Data scientist prepares and uploads custom dataset, with *Atlas* metadata sidecar measuring and attesting the dataset using Intel TDX, submitting attestation to the transparency log.
3) **Environment Validation:** *Atlas* sidecar verifies training environment integrity, adding TDX-based attestation to the C2PA manifest.
4) **Training Process Monitoring:** Attestation client tracks (App. B.3):
   - Model weight changes via PyTorch hooks
   - Checkpoint creation and modifications
   - Hyperparameter updates

4. Available at github.com/IntelLabs/atlas-cli

5) **Model Verification:** Third-party regulator verifies model provenance using *Atlas* verification service.
6) **Deployment:** Model vendor deploys verified model with provenance chain for user and application integrity validation.

Metadata Sidecar. Because the *Atlas* ML system monitor runs alongside untrusted MLaaS provider code, the attestation client's metadata sidecar leverages TEE remote attestation to detect tampering with the ML system monitor. That is, the sidecar interfaces with the ML system to obtain the Intel TDX-based compute environment attestations that capture TEE state and ML system component measurements, which are cryptographically anchored in hardware. We use the Confidential Containers (CoCo) framework [71] to implement the remote attestation procedure in the sidecar and ML system monitor.

Once the ML system monitor's integrity has been validated, the sidecar generates and digitally signs C2PA manifests. These include the sidecar's and ML system's Intel TDX attestations, the received ML system metadata, the measurements for the pipeline's input and output artifacts, and hashes for any linked transformation attestations. We describe a storage optimization in App. B.5.

Verification Service Implementation. For ease of implementation, the metadata sidecar also serves as a verification endpoint allowing pipeline components to validate artifact integrity against stored attestations. We optimize the performance of our staged verification system in three ways: 1) by processing changes incrementally and caching to avoid re-verifying unchanged components, 2) via batch processing of verification operations, and 3) parallel verification paths for independent component classes. App. B.6 provides additional details.

6. Evaluation

We validate our framework through a security analysis, preliminary performance testing and a case study with a BERT Meta [11], [12] fine-tuning pipeline.

6.1. Security Analysis

Atlas provides measures against the threats outlined in §3.3 through multiple security mechanisms.

For MLaaS provider threats, the hardware-rooted TEEs in *Atlas* isolate sensitive computations and detect malicious insider tampering with executing ML pipelines. The attestation client continuously validates the runtime environment, generating ML system measurements that are cryptographically bound to model artifacts.

Atlas counters hub threats by verifying artifact integrity through cryptographic measurements and signatures, maintaining a provenance chain that identifies mismatched signatures or tampered dependencies before they propagate.

Atlas mitigates artifact producer threats through comprehensive provenance tracking, providing an immutable record of pipeline operations that detects undeclared dependencies and intentional omissions.

6.2. Preliminary Performance Analysis

We conduct our experiments on Intel® Xeon® Gold 5520+ processors with 256 GB of RAM running Ubuntu 24.04 beta. Employing *Atlas* with the BERT Meta case study's CPU-only PyTorch-based fine-tuning pipeline, where the provenance chain covers 20 artifacts (up to 120 for more complex pipelines). Our measurements demonstrate near-linear scalability of verification time across different chain lengths and model sizes.

Preliminary tests show *Atlas* security mechanisms introduce minimal training overhead (under 8%), with each C2PA provenance manifest (8KB) containing artifact measurements, TEE attestations, and pipeline metadata. Verification processes scale linearly with model size, and our caching strategies reduce verification latency by up to 50%, achieving near-constant time for cached (see §5) component verification.

For large-scale operations, performance is maintained through concurrent verification operations, cache optimization, and selective invalidation for error handling. We plan to conduct more extensive benchmarking in future work.

6.3. Case Study

BERT was selected for its complex architecture and widespread production use. Our implementation covers the complete lifecycle from pre-trained model fine-tuning through deployment, with Intel TDX TEEs for attention computations and weight updates.

Atlas secures instruction-based configuration using JSON records with query-positive-negative text triplets, tracking model adaptations and maintaining verifiable records of hyperparameters and training progressions.

The BERT Meta implementation demonstrates performance consistent with our analysis in §6.2.

6.4. Discussion & Limitations

Our implementation reveals limitations in current hardware security. While *Atlas* provides TEE-based protection for CPU operations [10], ML workloads rely on GPUs and accelerators without equivalent security features [72], creating security and trust boundaries between protected and unprotected environments [73]. Emerging solutions like confidential GPU computing show promise but have performance trade-offs [74], [75].

Additionally, ML lifecycle transparency creates competing requirements between verification and confidentiality of intellectual property and sensitive data [76], an important challenge that *Atlas* needs to address in a future version. Organizations must balance security requirements with operational efficiency, considering factors like verification frequency, attestation depth, and computational overhead.

7. Conclusion

The combination of hardware-backed security with runtime provenance tracking in *Atlas* provides a foundation for securing ML pipelines. Our case study shows *Atlas*'s ability to integrate into existing ML frameworks with reasonable performance. Interesting directions for future work include: 1) provenance tracking of ML accelerator-based computations, 2) end-to-end ML lifecycle confidentiality, and 3) algorithmic verification methods and model guardrails against attacks targeting model behavior.

References

[1] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 1467–1474.

[2] N. Carlini, M. Jagielski, C. A. Choquette-Choo, D. Paleka, W. Pearce, H. Anderson, A. Terzis, K. Thomas, and F. Tramèr, "Poisoning web-scale training datasets is practical," 2024. [Online]. Available: https://arxiv.org/abs/2302.10149

[3] W. Jiang, N. Synovic, R. Sethi, A. Indarapu, M. Hyatt, T. R. Schorlemmer, G. K. Thiruvathukal, and J. C. Davis, "An empirical study of artifacts and security risks in the pre-trained model supply chain," in *Proc. ACM SCORED*, ser. SCORED'22. New York, NY, USA: Association for Computing Machinery, 2022, p. 105–114. [Online]. Available: https://doi.org/10.1145/3560835.3564547

[4] D. Cohen, "Data Scientists Targeted by Malicious Hugging Face ML Models with Silent Backdoor," JFrog Blog, Feb 2024.

[5] Executive Office of the President, "Executive Order 14144: Strengthening and Promoting Innovation in the Nation's Cybersecurity," Tech. Rep., Jan 2025.

[6] The European Parliament and the Council of the E.U., "The EU Artificial Intelligence Act," Tech. Rep., Jul 2024.

[7] W. Ozga, D. L. Quoc, and C. Fetzer, "Perun: Confidential multi-stakeholder machine learning framework with hardware acceleration support," in *Proc. DBSec*. Berlin, Heidelberg: Springer-Verlag, 2021, p. 189–208. [Online]. Available: https://doi.org/10.1007/978-3-030-81242-3_11

[8] R. Ashmore, R. Calinescu, and C. Paterson, "Assuring the machine learning lifecycle: Desiderata, methods, and challenges," *ACM Comput. Surv.*, vol. 54, no. 5, May 2021. [Online]. Available: https://doi.org/10.1145/3453444

[9] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the theory and application of cryptographic techniques*. Springer, 1987, pp. 369–378.

[10] Intel Corporation, "Intel® Trust Domain Extensions (Intel® TDX)," https://software.intel.com/content/www/us/en/develop/articles/intel-trust-domain-extensions.html.

[11] P. Lin, "Meta-BERT: A pre-trained language model with multi-layer attention mechanism," *arXiv preprint arXiv:2306.12937*, 2023.

[12] ——, "Meta-BERT implementation and fine-tuning framework," *arXiv preprint arXiv:2306.12938*, 2023.

[13] K. J. K. Feng, N. Ritchie, P. Blumenthal, A. Parsons, and A. X. Zhang, "Examining the impact of provenance-enabled media on trust and accuracy perceptions," *Proc. ACM Hum.-Comput. Interact.*, vol. 7, no. CSCW2, 2023.

[14] P. England, H. S. Malvar, E. Horvitz, J. W. Stokes, C. Fournet, R. Burke-Aguero, A. Chamayou, S. Clebsch, M. Costa, J. Deutscher, S. Erfani, M. Gaylor, A. Jenks, K. Kane, E. M. Redmiles, A. Shamis, I. Sharma, J. C. Simmons, S. Wenker, and A. Zaman, "AMP: Authentication of media via provenance," in *Proc. 12th ACM Multimedia Syst. Conf.*, 2021, pp. 108–121.

[15] ISO/TC 171/SC 2, "Document management – content authenticity," Int. Org. Standardization, Tech. Rep. ISO/WD 24138, 2024.

[16] Coalition for Content Provenance and Authenticity, "C2PA specification 2.1," *C2PA Specifications*, 2024.

[17] B. Laurie and A. Newman, "Adapting content authenticity standards for digital media provenance," *Proc. Int. Conf. Digital Media Integrity*, pp. 45–52, 2023.

[18] E. Sidnam-Mauch, B. Ivancsics, A. Monroe, E. Washington, E. Francis II, K. Caine, J. Bonneau, and S. E. McGregor, "Usable cryptographic provenance: A proactive complement to fact-checking for mitigating misinformation," in *Proc. 16th Int. AAAI Conf. Web Social Media Workshops*, P. O. S. Vaz de Melo, W. Jeng, and C. Buntain, Eds., June 2022.

[19] J. Collomosse and A. Parsons, "To authenticity, and beyond! building safe and fair generative AI upon the three pillars of provenance," *IEEE Comput. Graph. Appl.*, vol. 44, pp. 82–90, 2024.

[20] The Model Transparency contributors, "Model signing," 2024, [Online] Available: github.com/sigstore/model-transparency.

[21] The New York Times, "The news provenance project," 2019, [Online] Available: open.nytimes.com.

[22] Treeverse, Inc., "Welcome to the lake!" 2025, [Online] Available: docs.lakefs.io.

[23] The White House, "Executive order on improving the nation's cybersecurity," The White House Briefing Room, Presidential Actions, May 2021.

[24] European Commission, "Cyber resilience act," European Commission, Tech. Rep., Sep 2022.

[25] National Telecommunications and Information Administration, "Software bill of materials (SBOM)," 2025, [Online] Available: ntia.gov/page/software-bill-materials.

[26] M. Brown, A. Travers, and H. Khlaaf, "Trail of bits's response to PEO IEWS automated AIBOM RFI," Trail of Bits, Tech. Rep., 2024.

[27] D. Bardenstein, N. Kulkarni, and J. Frick, "Driving AI transparency: The AI bill of materials," Manifest, Tech. Rep., 2023.

[28] The Linux Foundation, "Safeguarding artifact integrity across any software supply chain," 2025, [Online] Available: slsa.dev.

[29] ——, "SPDX build," 2023, [Online] Available: spdx.dev/learn/areas-of-interest/build.

[30] The Model Transparency contributors, "SLSA for models," 2024, [Online] Available: github.com/sigstore/model-transparency.

[31] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *Proc. 28th USENIX Security Symp.*, 2019, pp. 1393–1410.

[32] Sigstore, "Overview," 2025, [Online] Available: docs.sigstore.dev.

[33] H. Birkholz, A. Delignat-Lavaud, C. Fournet, Y. Deshpande, and S. Lasker, "An architecture for trustworthy and transparent digital supply chains," 2024, [Online] Available: datatracker.ietf.org.

[34] EQTY Team, "EQTY lineage explorer: A framework for ML model provenance," *HuggingFace Blog*, 2023.

[35] Weights and Biases Team, "Weights and biases artifacts: ML experiment tracking," *Weights and Biases Tech. Rep.*, 2023.

[36] Neptune Labs, "Introduction to neptune.ai," 2025, [Online] Available: docs.neptune.ai.

[37] The Kubeflow Authors, "ML metadata," 2025, [Online] Available: kubeflow.org.

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[39] I. E. Akkus and I. Rimac, "Duet: Combining a trustworthy controller with a confidential computing environment," in *Proc. IEEE SysTEX*, 2024, pp. 436–442.

[40] A. Galanou, K. Bindlish, L. Preibsch, Y.-A. Pignolet, C. Fetzer, and R. Kapitza, "Trustworthy confidential virtual machines for the masses," in *Proc. Middleware*, ser. Middleware '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 316–328. [Online]. Available: https://doi.org/10.1145/3590140.3629124

[41] M. Chrapek, A. Vahldiek-Oberwagner, M. Spoczynski, S. Constable, M. Vij, and T. Hoefler, "Fortify your foundations: Practical privacy and security for foundation model deployments in the cloud," *arXiv preprint arXiv:2410.04766*, 2024.

[42] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative Instructions and Software Model for Isolated Execution," in *Proc. Hardware and Architectural Support for Security and Privacy*, 2013.

[43] V. Duddu, O. Järvinen, L. J. Gunn, and N. Asokan, "Laminator: Verifiable ML property cards using hardware-assisted attestations," *arXiv preprint arXiv:2406.17548*, 2024.

[44] I. E. Akkus, I. Rimac, and R. Chen, "Praas: Verifiable proofs of property as-a-service with intel sgx," in *Proc. IEEE SysTEX*, 2024, pp. 199–207.

[45] M. Russinovich, "Azure AI Confidential Inferencing: Technical Deep-Dive," Azure Confidential Computing Blog, Sep 2024. [Online]. Available: https://techcommunity.microsoft.com/blog/azureconfidentialcomputingblog/azure-ai-confidential-inferencing-technical-deep-dive/4253150

[46] L. Montoya Laske, "Announcing Privatemode: the AI service resolving your data privacy and security issues," Edgeless Blog, Feb 2025. [Online]. Available: https://www.edgeless.systems/blog/what-is-privatemode

[47] F. Mo, Z. Tarkhani, and H. Haddadi, "Machine learning with confidential computing: A systematization of knowledge," *ACM Comput. Surv.*, vol. 56, no. 11, 2024.

[48] Google, "ML pipelines," 2025, [Online] Available: developers.google.com.

[49] Microsoft, "Azure machine learning," 2025, [Online] Available: azure.microsoft.com.

[50] IBM, "Realize the promise of AI with watsonx," 2025, [Online] Available: ibm.com/watsonx.

[51] Google, "Innovate faster with enterprise-ready AI, enhanced by Gemini models," 2025, [Online] Available: cloud.google.com/vertex-ai.

[52] GitHub, "Using GitHub actions for MLOps & data science," 2020, [Online] Available: github.blog.

[53] HuggingFace, Inc., "The AI community building the future," 2025, [Online] Available: huggingface.co.

[54] The PyTorch Foundation, "PyTorch hub," 2025, [Online] Available: pytorch.org/hub.

[55] C. Lamb and S. Zacchiroli, "Reproducible builds: Increasing the integrity of software supply chains," vol. 39, no. 2, pp. 62–70.

[56] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan, "RFC 9334: Remote ATtestation procedureS (RATS) Architecture," IETF Data Tracker, Jan 2023. [Online]. Available: https://datatracker.ietf.org/doc/rfc9334/

[57] J. Cohen, E. Rosenfeld, and Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *international conference on machine learning*. PMLR, 2019, pp. 1310–1320.

[58] M. Wicker, P. Sosnin, I. Shilov, A. Janik, M. N. Müller, Y.-A. de Montjoye, A. Weller, and C. Tsay, "Certification for differentially private prediction in gradient-based training," *arXiv preprint arXiv:2406.13433*, 2024.

[59] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," *Advances in neural information processing systems*, vol. 30, 2017.

[60] R. Hat. (2024) Urgent security alert for fedora linux 40 and fedora rawhide users. [Online]. Available: https://www.redhat.com/en/blog/urgent-security-alert-fedora-41-and-rawhide-users

[61] Apple Inc., "Private Cloud Compute: A new frontier for AI privacy in the cloud," Security Research Blog, Jun 2024. [Online]. Available: https://security.apple.com/blog/private-cloud-compute/

[62] N. Carlini, F. Tramér, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson, A. Oprea, and C. Raffel, "Membership inference attacks from first principles," 2022, iEEE S&P 2022, 18 pages, 8 figures.

[63] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High accuracy and high fidelity extraction of neural networks," 2020, uSENIX Security 2020, 18 pages, 6 figures.

[64] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," 2016.

[65] Advanced Micro Devices, Inc., "AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More," https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, Jan 2020.

[66] Intel Corporation, "Intel® TDX Connect Architecture Specification," https://www.intel.com/content/www/us/en/content-details/773614/intel-tdx-connect-architecture-specification.html, 2023.

[67] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, "CONIKS: Bringing Key Transparency to End Users," 2015.

[68] Y. Hu, K. Hooshmand, H. Kalidhindi, S. J. Yang, and R. A. Popa, "Merkle 2: A low-latency transparency log system," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 285–303.

[69] H. Malvai, L. Kokoris-Kogias, A. Sonnino, E. Ghosh, E. Oztürk, K. Lewi, and S. Lawlor, "Parakeet: Practical Key Transparency for End-to-End Encrypted Messaging," 2023. [Online]. Available: https://eprint.iacr.org/2023/081

[70] Y. Sun, Y. Hu, and Y. Yu, "Advanced Transparency System," Cryptology ePrint Archive, Paper 2024/1788, 2024. [Online]. Available: https://eprint.iacr.org/2024/1788

[71] Confidential Containers Project, "Confidential containers attestation service," 2024, [Online] Available: github.com/confidential-containers.

[72] Intel Corporation, "Seamless attestation of Intel TDX and NVIDIA H100 TEEs with Intel trust authority," 2024, [Online] Available: community.intel.com.

[73] J. Ménétrey, C. Göttel, A. Khurshid, M. Pasin, P. Felber, V. Schiavoni, and S. Raza, "Attestation mechanisms for trusted execution environments demystified," *arXiv preprint arXiv:2206.03780*, 2022.

[74] Microsoft Community Hub, "Announcing azure confidential VMs with NVIDIA H100 Tensor Core GPUs in Preview," November 2023. [Online]. Available: https://techcommunity.microsoft.com/t5/azure-confidential-computing/announcing-azure-confidential-vms-with-nvidia-h100-tensor-core/ba-p/3975389

[75] J. Liu, Z. Xu, S. Wu, Y. Yang, and Y. Liu, "Confidential computing on nVIDIA H100 GPU: A Performance Benchmark Study," 2024. [Online]. Available: https://arxiv.org/html/2409.03992v2

[76] D. Hernandez, A. Wiesmaier, G. Gentile, M. Rak, and H. Pinto, "A survey on the (in)security of trusted execution environments," *Computers & Security*, vol. 128, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404823000901

[77] Adobe Inc., The New York Times Company, and Twitter Inc., "Content authenticity initiative: Setting the standard for digital content attribution," Adobe Inc., White Paper, 2019.

[78] Project Origin, "Project origin - protecting trusted media," 2024, [Online] Available: originproject.info.

[79] H. M. H. Hegge and J. C. Wortmann, "Generic bill-of-material: A new product model," *Int. J. Prod. Econ.*, vol. 23, no. 1, pp. 117–128, 1991.

[80] N. Zahan, E. Lin, M. Tamanna, W. Enck, and L. Williams, "Software bills of materials are required. are we there yet?" vol. 21. [Online]. Available: https://ieeexplore.ieee.org/document/10102604/?arnumber=10102604

[81] NTIA Formats and Tooling Working Group, "Software Suppliers Playbook: SBOM Production and Provision," Nov 2021. [Online]. Available: https://www.ntia.gov/sites/default/files/publications/software_suppliers_sbom_production_and_provision_-_final_0.pdf

[82] Lightning AI, "Callback — pytorch lightning 2.5.1 documentation," https://lightning.ai/docs/pytorch/stable/extensions/callbacks.html, 2025, accessed: 2025-04-03.

[83] GeeksforGeeks, "Distributed ledger technology (DLT) in distributed system," 2025, [Online] Available: geeksforgeeks.org.

[84] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3454–3469, 2020.

[85] P. Foley, M. J. Sheller, B. Edwards, S. Pati, W. Riviera, M. Sharma, P. N. Moorthy, S. Wang, J. Martin, and P. Mirhaji, "OpenFL: The open federated learning library," *IEEE Trans. Inf. Forensics Security*, 2022.

[86] H. Zheng and O. Arden, "Building secure distributed applications the DECENT way," *arXiv preprint arXiv:2004.02020*, 2020.

[87] F. Regazzoni, P. Palmieri, F. Smailbegovic, R. Cammarota, and I. Polian, "Protecting artificial intelligence IPs: A survey of watermarking and fingerprinting for machine learning," *CAAI Trans. Intell. Technol.*, vol. 6, no. 2, pp. 180–191, 2021.

[88] F. Boenisch, "A systematic review on model watermarking for neural networks," *arXiv preprint arXiv:2009.12153*, 2021.

[89] X. Cao, J. Jia, and N. Z. Gong, "Protecting intellectual property of machine learning models via fingerprinting the classification boundary," *Digital Watermarking Mach. Learn.*, 2023.

[90] H. Ben Braiek and F. Khomh, "Machine learning robustness: A primer," *arXiv preprint arXiv:2404.00897*, 2024.

[91] Y. Liang, L. Cheng, A. Payani, and K. Shu, "Beyond detection: Unveiling fairness vulnerabilities in abusive language models," *arXiv preprint arXiv:2311.09428*, 2023.

[92] E. Bagdasaryan and V. Shmatikov, "Mithridates: Auditing and boosting backdoor resistance of machine learning pipelines," *arXiv preprint arXiv:2302.04977*, 2023.

Appendix A. Background & Related Work

In addition to the works highlighted in §2, we describe further details and approaches about related work addressing ML lifecycle security and integrity.

A.1. Data Provenance & Authenticity

C2PA. The Coalition for Content Provenance and Authenticity (C2PA) specification [15], [16] was initially introduced by the Content Authenticity Initiative (CAI) [77] and Project Origin [78] as a response to the growing challenge of deepfakes [77] and digital content manipulation, gaining traction in digital photography and journalism workflows.

LakeFS. LakeFS [22] combines Git-like semantics with concepts from object stores such as S3 to provide a version control system for data, including ML datsets. Thus, LakeFS aims to capture data lineage by tracking changes to stored data over time, and allowing ML applications to reference specific versions of the stored data. This approach is meant to integrate with existing first-party data processing pipelines, but does not facilitate verification of data provenance by downstream consumers. *Atlas*' metadata centered approach, on the other hand, enables first- *and* third-party ML dataset consumers to track changes and check their provenance, even when they may not have direct access to the data.

A.2. Supply Chain Integrity

BOM. Bills of Materials (BOM) have been employed to document the list of components of a hardware or software product for over three decades [79]. Software BOM (SBOM) have been the focus of many industry and academic efforts seeking to facilitate tracking software dependencies and other metadata [25], to improve their adoption, and to enhance SBOM integrity and privacy (e.g., [80], [81]).

Similarly, the AIBOM framework [26], [27] focuses on intended ML model supply chain management. Like SBOM, AIBOM provide a mechanism for tracking model software dependencies and maintaining model metadata. At the time of writing, we are not aware of any frameworks other than *Atlas* that utilize any sort of BOM data format to track ML model components.

Authenticated claims. A number of frameworks for capturing and verifying a variety of security claims and metadata about the supply chain have been proposed. in-toto [31] collects authenticated claims *across* supply chain steps, including SBOM and SLSA metadata. In particular, in-toto enables software development pipeline owners and downstream artifact consumers to specify end-to-end supply chain policies, and validate that only the expected parties carried out specific steps in the pipeline and artifacts underwent transformations in the expected order. Given recent and upcoming enhancements that further generalize the framework, in-toto may be a suitable option for specifying and verifying end-to-end ML model pipeline integrity policies in *Atlas*.

Sigstore [32] provides a transparency log-based infrastructure for issuing signing credentials and validating digital signatures on supply chain artifacts and metadata. Similarly, Supply Chain Integrity, Transparency and Trust (SCITT) [33] is an architecture for implementing distributed ledger based supply chain integrity mechanisms, providing global visibility and auditing for supply chain operations and claims. The SCITT architecture also includes confidential computing technologies that help ensure that only authorized parties submit claims to the transparency ledger.

Appendix B. Implementation Details

B.1. Kubeflow Integration

The integration with Kubeflow is achieved through custom operators and controllers that monitor pipeline execution through Kubeflow's Metadata V2 Beta API and KFP API. Through the `//apis/v2beta1/metadata` endpoint, we track execution contexts and maintain verifiable records of pipeline runs.

By interfacing with `/apis/v2beta1/artifacts`, we track model artifacts and their lineage. The metadata store provides structured information about component dependencies and data flow through the `/apis/v2beta1/connections` endpoint. Our system correlates this information with integrity measurements and hardware attestations, creating verifiable records of pipeline execution states.

The metadata extraction leverages Kubeflow's event system through `/apis/v2beta1/events`, enabling real-time capture of pipeline state transitions, component execution details, artifact generation events, and parameter updates. This structured approach enables verification of pipeline states while maintaining compatibility with existing workflows.

B.2. C2PA Metadata Examples

A typical execution record captured by our system looks like:

```
{
```

```
    "execution": {
        "name": "training-run-132",
        "state": "RUNNING",
        "pipeline_spec": {
            "parameters": {
                "learning_rate": 0.001,
                "batch_size": 32,
                "random_seed": 42,
                "optimizer_config": {
                    "type": "Adam",
                    "beta1": 0.9,
                    "beta2": 0.999
                }
            },
            "runtime_config": {
                "gcs_output_directory": "gs://...",
                "tensorflow_version": "2.9.0"
            }
        }
    }
}
```

For each execution, our system adds corresponding integrity measurements and verification records:

```
{
    "integrity_measurement": {
        "component_id": "training-run-132",
        "tdx_quote": "base64:...",
        "environment_hash": "sha256:...",
        "timestamp": "2024-01-15T10:30:00Z",
        "parameter_hash": "sha256:..."
    }
}
```

B.3. ML System Monitoring Procedures

Our proof-of-concept implementation leverages several techniques to monitor ML pipeline activities with minimal intrusion into existing workflows. The implementation focuses on collecting runtime data about model weights, hyperparameters, and execution context while maintaining performance and compatibility with established ML frameworks.

B.3.1. File System Monitoring in the Atlas Framework. The Atlas sidecar implements a file system monitor written in Rust that detects checkpoint creation and modification events:

procedure INITCHECKPOINTMONITOR($dir, client$)
  Initialize directory and client references
  Create empty checksum tracking map
  Setup file system watcher
end procedure
procedure SETUPWATCHER
  Create event listener for file changes
  Start background monitoring thread
  Register directory for change notifications
end procedure
procedure SCANCHECKPOINTS
  for each checkpoint file in directory **do**
    Compute file cryptographic checksum
    Store checksum in tracking map
    Register existing checkpoint in metadata
  end for
end procedure
procedure ONFILECREATED($file$)
  if file is checkpoint type **then**
    Compute checksum and record creation
    Update checksum tracking map
  end if
end procedure
procedure ONFILEMODIFIED($file$)
  if file is checkpoint type **then**
    Compute new checksum
    Retrieve old checksum from map
    if checksums differ **then**
      Update tracking map
      Record modification in metadata
    end if
  end if
end procedure

B.3.2. Callback/Hook Registration in PyTorch. For our BERT Meta case study, we implemented a callback system that integrates with PyTorch's event mechanisms. The model monitoring component operates as follows:

procedure INITMODELMONITOR($model, client$)
  Store references to model and client
  Register monitoring hooks on model
end procedure
procedure REGISTERHOOKS
  for each layer module in model **do**
    if module is neural network layer **then**
      Attach forward hook for activation capture
      if module has trainable weights **then**
        Attach gradient hook for updates
      end if
    end if
  end for
end procedure
procedure FORWARDHOOK($module, input, output$)
  Calculate unique layer identifier
  Extract statistical metrics from output
  Record activation data to metadata store
end procedure
procedure GRADIENTHOOK($gradient$)
  Calculate gradient magnitude
  Record gradient event with timestamp
end procedure

Additionally,for some cases we extended PyTorch's standard training loop with epoch-level callbacks [82]:

procedure INITTRAININGCALLBACK($client$)
  Store reference to metadata client
end procedure
procedure ONEPOCHSTART($epoch, optimizer$)
  Create hash of optimizer configuration
  Record epoch start event with config hash
end procedure
procedure ONEPOCHEND($epoch, metrics, model$)
  Capture cryptographic model state snapshot
  Record completion with metrics and snapshot
end procedure

These hooks operate with minimal overhead while providing comprehensive visibility into the model's evolution.

B.3.3. Configuration Wrappers in the Atlas Framework. To extract hyperparameter access and modifications from PyTorch, we implement transparent wrapper classes:

procedure INITTRACKEDCONFIG($config, client$)

Store config and client references
Initialize version counter to zero
Record initial configuration state
end procedure
procedure GET(key)
Log access event to metadata store
Return value for requested key
end procedure
procedure SET($key, value$)
Retrieve current value for key
Update configuration with new value
Increment version counter
Record modification in metadata
Update configuration state hash
end procedure
procedure RECORDSTATE
Generate hash of current configuration
Store versioned snapshot in metadata
end procedure
procedure GETCONFIG
Return copy of configuration
end procedure

B.4. Integration Between Framework and ML Pipeline

For the BERT Meta case study, we developed an integration layer that allows the framework components to interact with the Python ML pipeline:

procedure INITBRIDGESERVICE($endpoint, dir$)
Create metadata client connection
Initialize checkpoint monitor
Start bridge service
end procedure
procedure INITCONFIG($configJSON$)
Parse configuration from JSON
Create configuration tracking wrapper
Generate unique tracking identifier
return tracking identifier
end procedure
procedure SCANCHECKPOINTS
Scan and register existing checkpoints
end procedure
procedure RECORDEVENT($type, data$)
if $type$ is epoch start **then**
Record epoch initialization
else if $type$ is epoch end **then**
Record epoch completion with metrics
else if $type$ is layer activation **then**
Record layer output statistics
else if $type$ is gradient event **then**
Record gradient flow information
end if
end procedure

On the Python side, we implement a complementary bridge client:

procedure INITBRIDGECLIENT($socketPath$)
Connect to monitoring bridge service
end procedure
procedure SETUPMONITORING($model, config, dir$)
Serialize configuration to JSON
Initialize configuration tracking
Scan existing model checkpoints
Create model monitoring hooks
Setup training loop callbacks
return monitoring components
end procedure
procedure RECORDLAYERDATA($layer, stats$)
Package activation statistics
Send to bridge as layer event
end procedure
procedure RECORDGRADIENT($magnitude, time$)
Package gradient information
Send to bridge as gradient event
end procedure
procedure RECORDEPOCHSTART($epoch, optHash$)
Package epoch initialization data
Send to bridge as epoch start event
end procedure
procedure RECORDEPOCHEND($epoch, metrics, hash$)
Package completion metrics
Send to bridge as epoch end event
end procedure

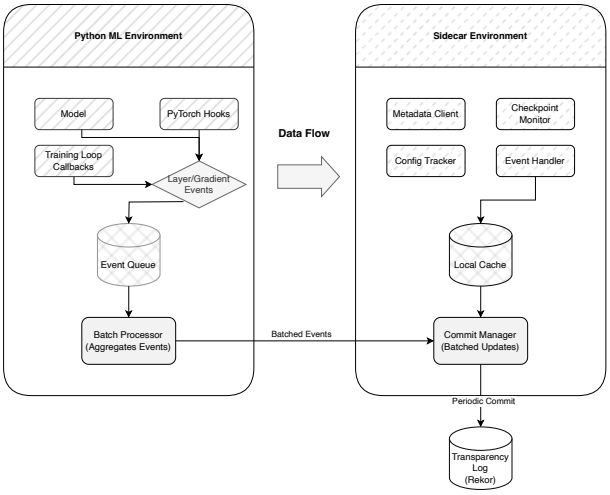

Figure 2. Altas sidecar collector showing data flow between Python ML environment and framework: The diagram illustrates how monitoring events are cached before being committed to the transparency log.

This hybrid architecture enables monitoring of the BERT training process with minimal modifications to the existing pipeline, while leveraging efficient system-level operations for the monitoring infrastructure.

B.5. Attestation Client Storage Optimization

As a storage optimization, the attestation client's metadata sidecar first stores all generated C2PA manifests in a local cache layer before being committed to the transparency log. The local cache maintains an indexed hierarchy of manifests for efficient validation during pipeline execution, before final storage in Rekor for tamper-evident provenance tracking.

More specifically, we decompose manifests into constituent components within the cache. The C2PA metadata assertions, claim signatures, and pipeline metadata are

stored separately, with relationships maintained through a reference system. This approach enables efficient updates to specific manifest components, reduced storage redundancy, optimized query performance, and scalable version tracking.

B.6. Verification Service Optimizations

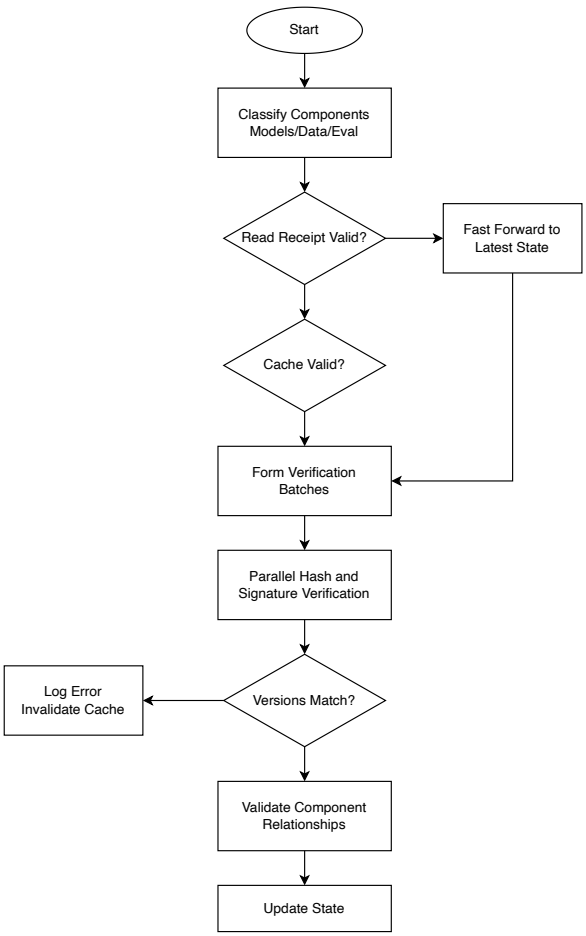

Figure 3. Verification workflow implementation for Atlas components. The system classifies artifacts and processes changes incrementally, preserving cached states for unchanged components. Related artifacts are grouped for batch processing, with parallel validation of relationships. During errors, only affected components are invalidated, reducing verification overhead while maintaining security guarantees.

For provenance chain validation, our verification service implementation parallelizes verification of cross-component dependencies, version compatibility and evaluation result consistency. The system also maintains verification checkpoints that serve as trusted reference points, enabling partial verification from the last known good state instead of complete chain recomputation.

Error handling focuses on computational efficiency through targeted cache invalidation rather than complete cache clearing. When verification failures occur, the system preserves verified states while ensuring security through selective invalidation. Preliminary testing shows these optimizations reduce verification time by up to 50% through parallelization and caching, while maintaining security guarantees.

B.7. Framework Adaptability

The BERT deployment validated our framework's flexibility across different ML environments. The abstraction layer successfully handled variations in framework-specific interfaces, from PyTorch's hook mechanisms to TensorFlow's Keras callbacks, while maintaining consistent security guarantees. Custom adapters enabled integration without modifying existing ML infrastructure, demonstrating the framework's ability to enhance security while preserving established workflows.

Appendix C. Adoption Considerations

C.1. Storage Optimizations

C.1.1. Storage and Scalability. Our analysis suggests opportunities for optimizing manifest storage through decomposed and hybrid architectures. Rather than storing complete manifests as single documents, separating components like assertions, signatures, and metadata could improve efficiency while maintaining security guarantees. Organizations should consider distributed storage strategies that balance immutability requirements with query performance needs.

C.1.2. Decomposed Storage Model. Instead of storing complete manifests as single documents, the system could decompose manifests into their constituent components. The assertion store, claim signatures, and metadata could be stored separately, with relationships maintained through a reference system. This approach would enable more efficient updates and queries of specific manifest components.

The decomposed model could leverage distributed ledger technology (DLT) [83] for critical manifest components while maintaining bulk data in optimized storage systems. This hybrid approach would:

- Store cryptographic proofs and signatures on the distributed ledger for immutability
- Maintain manifest metadata and relationships in graph databases for efficient querying
- Use object storage for large artifacts like model weights and datasets
- Link components through cryptographic references preserved in the ledger

C.1.3. Hybrid Storage Architecture. A hybrid approach could maintain critical verification data in Rekor for its transparency guarantees while storing detailed manifest data in optimized storage systems. This would balance the need for immutable proof of existence with efficient data access and management.

These storage optimizations could significantly reduce operational overhead while maintaining the security guarantees of our framework. Performance testing indicates potential reduction in storage requirements and query latency through these alternative approaches.

C.2. Deployment Guidelines

C.2.1. Organizational Adaptations. Organization-specific adaptations are necessary to align with existing infrastructure and security policies. Key considerations include:

- Integration with current MLOps platforms
- Alignment with existing security monitoring systems
- Customization of verification policies
- Adaptation to specific hardware security capabilities
- Compliance with organizational security standards

C.2.2. Security Requirement Balance. Security requirement balance directly impacts operational efficiency. Organizations must determine appropriate verification frequencies and depth based on their risk profile and performance requirements. For instance, continuous hardware attestation of all pipeline components provides maximum security but introduces significant overhead. A more balanced approach might implement full verification at critical pipeline stages while using lightweight checks during intermediate steps.

C.2.3. Computational Overhead Management. Computational overhead management becomes crucial when scaling the framework across large ML operations. Our implementation shows that intelligent caching of verification results and batch processing of integrity checks can significantly reduce overhead. Organizations should consider:

- Strategic placement of verification checkpoints - Organizations can tailor verification intensity based on their specific security needs and operational context. While financial or healthcare institutions might require comprehensive verification throughout their ML pipeline, research or development environments might focus verification efforts primarily on model publication or deployment stages. This flexible approach enables efficient resource utilization while maintaining appropriate security levels for each use case.
- Optimization of hardware attestation frequency - By analyzing pipeline characteristics and risk patterns, attestation frequency can be tuned to concentrate on high-risk operations while reducing overhead during stable processing phases.
- Efficient manifest storage and retrieval mechanisms - The system maintains an indexed store of manifests with hierarchical organization, enabling quick validation of model lineage while managing storage overhead for long-term provenance tracking.
- Parallel verification processing where possible - This approach utilizes available computational resources effectively by running verification operations concurrently when component dependencies allow.

Appendix D. Future Work

Several potential enhancements could extend our framework's capabilities and applicability:

Distributed Training Support. The current framework could be enhanced to handle multiple TEEs coordinating across training nodes, with cross-node attestation and verification protocols. This would require developing protocols for maintaining integrity across distributed components while managing the additional complexity of verifying inter-node communications and state synchronization [73].

Federated Learning Compatibility. Our current framework could be extended to support federated learning environments, particularly through integration with Intel's OpenFL (Open Federated Learning) [84] framework. OpenFL's architecture, which separates aggregator and participant nodes while maintaining model security, presents unique opportunities and challenges for provenance tracking. The framework would need to extend its attestation and verification protocols to handle distributed model updates while preserving the privacy guarantees inherent in federated learning.

Key considerations include tracking model aggregation operations, verifying participant contributions, and maintaining cryptographic proofs across federation rounds. OpenFL's existing security features, including its support for secure aggregation and TEE integration, provide natural integration points for our provenance framework. The challenge lies in extending our verification protocols to handle the partial model updates and differential privacy mechanisms [85] common in federated learning scenarios.

Enhanced Scalability Features. Support for more complex ML architectures, particularly for multi-model systems and ensemble methods, would expand the framework's utility. This would require developing verification protocols for model composition and interaction, tracking dependencies between component models, and maintaining provenance across model combinations [86].

The framework could also be extended to support dynamic trust models, allowing for flexible trust relationships between different components and participants in the ML pipeline.

Algorithmic Security Enhancements. Our framework's modular design allows for integration of additional algorithmic security methods to enhance pipeline protection. Model watermarking techniques [87], [88] could be incorporated to embed verifiable ownership proofs directly into model weights, providing an additional layer of provenance verification. These watermarks would be included in the manifest chain, creating cryptographically verifiable links between model versions and their origins.

Verification Protocol Extensions. Our staged verification system could be extended to support dynamic trust models. This would allow more flexible verification policies based on component criticality and risk levels, while maintaining our core security guarantees. The current implementation's classification system provides a foundation for such policy-based verification.

Neural fingerprinting [87], [89] methods could extend our verification capabilities by enabling detection of unauthorized model modifications or derivatives. By maintaining fingerprint signatures in our provenance records, the framework could track model lineage even when traditional hash-based verification is insufficient. This is particularly valuable for scenarios involving fine-tuning or transfer learning.

Property attestation mechanisms could verify specific algorithmic characteristics of models throughout the pipeline. For example, robustness guarantees [90], fairness metrics [91], or backdoor resistance [92] could be measured and included in the manifest chain. These properties would enhance the framework's ability to detect subtle manipulations that might not affect model hashes but could impact model behavior.

Throughout this process, *Atlas* provides tamper-evident records through its transparency log and TEE-based attestations. The verification service requires access to both the model artifacts and the cryptographic measurements in the transparency log to confirm the integrity of the complete ML pipeline, ensuring that no unauthorized modifications occurred during the model's lifecycle.