Threat Modeling for AI: The Case for an Asset-Centric Approach

Jose Rodrigo Sanchez Vicarte
Intel Security Research
jose.sanchez.vicarte@intel.com

Marcin Spoczynski
Intel Labs
marcin.spoczynski@intel.com

Mostafa Elsaid
Intel AI Cloud Security
mostafa.elsaid@intel.com

I. Introduction

Recent advances in AI are transforming AI's ubiquitous presence in our world from that of standalone AI-applications into deeply integrated AI-agents. These changes have been driven by agents' increasing capability to autonomously make decisions and initiate actions, *using* existing applications; whether those applications are AI-based or not. This evolution enables unprecedented levels of AI integration, with agents now able to take actions on behalf of systems and users - including, in some cases, the powerful ability for the AI to write and execute scripts as it deems necessary [5]. With AI systems now able to autonomously execute code, interact with external systems, and operate without human oversight, traditional security approaches fall short.

This paper introduces an asset-centric methodology for threat modeling AI systems that addresses the unique security challenges posed by integrated AI agents. Unlike existing top-down frameworks that analyze individual attacks within specific product contexts, our bottom-up approach enables defenders to systematically identify how vulnerabilities—both conventional and AI-specific—impact critical AI assets across distributed infrastructures used to develop and deploy these agents. This methodology allows security teams to: (1) perform comprehensive analysis that communicates effectively across technical domains, (2) quantify security assumptions about third-party AI components without requiring visibility into their implementation, and (3) holistically identify AI-based vulnerabilities relevant to their specific product context. This approach is particularly relevant for securing agentic systems with complex autonomous capabilities. By focusing on assets rather than attacks, our approach scales with the rapidly evolving threat landscape while accommodating increasingly complex and distributed AI development pipelines.

A. AI-based Vulnerabilities

Integrating AI introduces vulnerabilities into development and deployment pipelines [4], [14], [26], [35]. These can be separated into two types:

1) Conventional hardware and software vulnerabilities. These vulnerabilities are introduced by the hardware and software scaffolding necessary to build, run, and interact with the AI [3], [15].
2) AI-based vulnerabilities. These vulnerabilities occur when an adversary manipulates an AI to compromise product functionality or user data [16], to steal the AI itself [27], [33], or to steal the AI's training data [11], [13].

While distinct in nature, these vulnerability types are often interconnected. Conventional vulnerabilities can serve as entry points that enable AI-based attacks, while AI-based vulnerabilities can amplify the impact of conventional security breaches. For example, a conventional code injection vulnerability might allow an attacker to manipulate model inputs, enabling an AI-specific prompt injection attack which affects downstream users and agents.

Today's security processes are designed to protect development and deployment pipelines from the first type of vulnerability. These processes must be augmented, however, to help defenders identify and reason about the second type; the ways in which adversaries can influence or manipulate an integrated AI.

Adversaries can exploit both Adversarial AI [35] and conventional hardware and software techniques [12], [29], [36] to compromise an AI. It's critical that defenders consider both techniques in their security analysis. That is, beyond considering the novel Adversarial AI techniques, defenders must reason about the ways in which conventional vulnerabilities can be exploited to influence AI. To understand *how* a vulnerability can be exploited to affect an AI, however, defenders must first understand *what* components adversaries seek to influence. The components – often termed Assets within threat models – which adversaries seek to manipulate inherently stem from AI's computing paradigms.

AI represents a shift in our computing paradigms towards data driven compute. Under traditional compute, a developer would create a program – i.e. logic rules – to generate a desired output from unseen input data. AI's computing paradigm instead analyzes massive amounts of data to infer – i.e. learn or create – logic rules. These inferred rules are then used to generate a desired output when applied to unseen input data. The AI's underlying decision making logic is, therefore, algorithmically inferred rather than explicitly encoded [9], [22]. This data driven nature is why AI's decisions are often difficult to explain, and why AI is difficult to validate. Evasion, or jailbreak, attacks are an example of adversaries finding and exploiting corner cases or incorrectly correlated features within the AI's inferred logic; i.e. the model. To compromise an AI's decision, however, adversaries are not constrained to direct attacks on the model [8], [17], [23], [28]. Adversaries are also able to introduce vulnerabilities into an AI by affecting its training data, training process or algorithm, and validation process or goals. To augment their security policies against emerging AI-based vulnerabilities, defenders must reason about the many ways in which adversaries can affect their AI.

B. Threat Modeling Challenges

Threat modeling for AI refers to the process of performing security analysis to identify vulnerabilities which could be exploited to impact the AI. Defenders face three key challenges when threat modeling AI's development and deployment pipelines.

Challenge 1. An AI's implementation and dependencies are increasingly distributed across many disjoint, often cloud-based, infrastructure contexts. Each with a unique hardware and software stack. Defenders must consider vulnerabilities within each context.

Challenge 2. Understanding all potential impacts of a vulnerability requires defenders to reason about how that vulnerability can be combined, i.e. chained, with other vulnerabilities. To accomplish this, however, defenders must reason holistically. Those other vulnerabilities may exist at different levels of the stack – and within different infrastructure contexts.

Challenge 3. It is increasingly common for defenders to have little to no visibility into large portions of an AI's training or deployment infrastructures. Creating an AI "from scratch" is an incredibly expensive process so developers will often *consume* AI components – such as datasets or pre-trained models. However, defenders currently lack a process for quantifying the risks they inherit by integrating those components.

Various frameworks have been proposed to facilitate top-down security analysis for AI. Unfortunately, a top-down perspective is not well suited for addressing these challenges.

C. The case for a bottom-up perspective

Existing frameworks leverage a top-down perspective to individually map AI attacks within the scope of a specific product. MITRE's ATLAS, for example, broadly describes the steps an adversary takes to perform an attack – i.e. within the attack's lifecycle – including common techniques for achieving each step. As depicted on the top half of Figure 1, this perspective allows defenders to reason about a specific attack by contextualizing each step in the attack's roadmap down onto their product. The defender identifies vulnerabilities which, if exploited, would allow an adversary to perform that step. If all attack steps can be satisfied, the attack is deemed to be in scope for the product. To contextualize a step, however, defenders must have enough visibility into every context of the development and deployment pipelines so as to reason about the vulnerabilities within them. Reasoning about an attack – or even a single step – also requires reanalyzing the

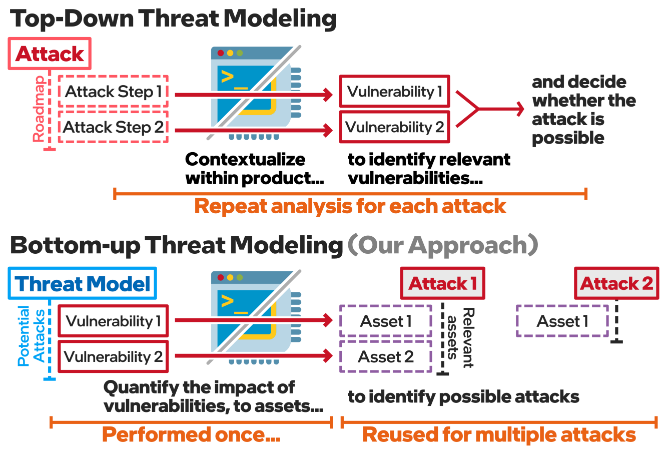

Top-Down Threat Modeling
Attack
Roadmap
Attack Step 1
Attack Step 2
Vulnerability 1
Vulnerability 2
and decide whether the attack is possible
Contextualize within product...
to identify relevant vulnerabilities...
Repeat analysis for each attack
Bottom-up Threat Modeling (Our Approach)
Threat Model
Potential Attacks
Vulnerability 1
Vulnerability 2
Attack 1
Asset 1
Asset 2
Relevant assets
Attack 2
Asset 1
Quantify the impact of vulnerabilities, to assets...
to identify possible attacks
Performed once...
Reused for multiple attacks

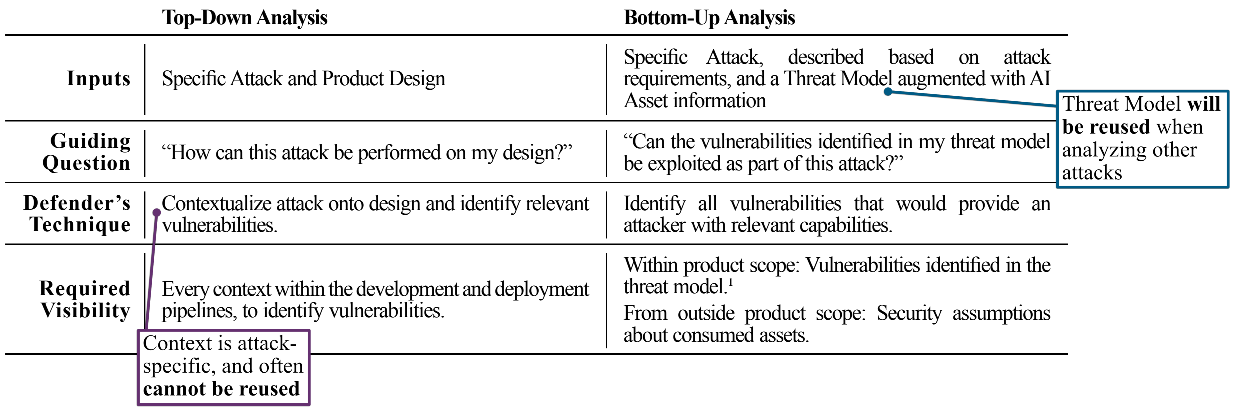
Fig. 1. Existing frameworks enable a top-down approach (top) which allows defenders to reason about a specific attack by contextualizing it within their product. This work advocates for a bottom-up approach (bottom) which allows defenders to analyze their product once, from an asset-centric perspective, and reuse that analysis to contextualize *multiple* attacks.

product. These limitations hinder defenders from applying top-down approaches across increasingly complex AI pipelines, and prevent these approaches from scaling with a quickly growing and evolving attack space.

This work describes a methodology for *bottom-up* analysis, driven by an Asset-Centric approach. As depicted on the bottom half of Figure 1, our methodology calls for defenders to augment their existing threat models and security analysis processes to identify *how* each vulnerability can impact AI assets. That is, defenders will identify the manner and extent – specific to the context of their product – to which an adversary can compromise those assets. They do this for every vulnerability identified in their threat model. Inverted from a top-down approach, defenders then use this information to identify the AI attacks which are enabled by any of the identified vulnerabilities. Figure 2 summarizes the two perspectives. This work describes *how* defenders can reason about vulnerabilities' impact to AI. Our methodology calls for defenders to:

1) Augment their current security analysis to quantify whether a vulnerability – conventional or AI-based – could impact some aspect of their AI. This methodology is specifically designed so that defenders can easily communicate their findings across the stack and across distributed systems.
2) Quantify, and justify, their security assumptions about the AI components consumed by their system. Even if the producing systems are completely closed-box.
3) Combine the insights from their security analysis and security assumptions to identify vulnerabilities – and their root cause – relevant to the product's AI.

Section II will introduce common types of AI Assets and describe their relationships. Section III will elaborate on the challenges defenders face when threat modeling AI. Section IV



	Top-Down Analysis	Bottom-Up Analysis
Inputs	Specific Attack and Product Design	Specific Attack, described based on attack requirements, and a Threat Model augmented with AI Asset information
Guiding Question	"How can this attack be performed on my design?"	"Can the vulnerabilities identified in my threat model be exploited as part of this attack?"
Defender's Technique	Contextualize attack onto design and identify relevant vulnerabilities.	Identify all vulnerabilities that would provide an attacker with relevant capabilities.
Required Visibility	Every context within the development and deployment pipelines, to identify vulnerabilities.	Within product scope: Vulnerabilities identified in the threat model.[1] From outside product scope: Security assumptions about consumed assets.

Threat Model **will be reused** when analyzing other attacks
Context is attack-specific, and often **cannot be reused**

Fig. 2. Compares top-down perspectives (existing frameworks) with bottom-up perspectives (our methodology) for security analysis.

provides background on asset-centric threat modeling. Section V describes our bottom-up, asset-centric, approach for threat modeling AI. Sections VI and VII introduce Enterprise RAG and apply this analysis to it, as a case study. Section VIII describes existing frameworks for threat modeling AI and why they naturally complement an asset-centric approach. Section IX describes our ongoing work and next steps. Finally, Section X concludes.

II. Relationships between AI Assets

The relationships between AI assets are inherent to AI's data-driven computing paradigm because, under this paradigm, developers do not directly change the AI's logic rules – i.e. *Model*. Instead, they indirectly affect the model by changing its training *Dataset*. This is what allows AI to scale and generalize far beyond what traditional compute could achieve: Instead of being encoded by developers, the training process simultaneously infers features in the training dataset and correlates them with their expected outputs – i.e. the data's labels. The AI's accuracy and ability to generalize is therefore a function of the quality and size of its training dataset, and of its training algorithm. This allows AI to scale and generalize at a level far beyond what traditional compute could achieve. However, because developers do not directly change the model, it also requires a significant shift in development methodologies.

Consider, for example, some of the techniques used by developers and adversaries to affect the model. Developers use deep analysis of the model and dataset to guide dataset changes or guardrail selection [2]. This deep analysis can include Explainable-AI (XAI) techniques, which extrapolate the model's learned features and relationships [22]. Adversarial techniques – which are similar in spirit – can reveal key weights or activations to affect model behavior around specific data or features [8], [17], [23], [28].

This section presents a high level overview of the relationships of AI assets spanning across three representative stages:

1) Dataset Collection & Assembly (Figure 3)
2) Training (Figure 4)
3) Deploy & Inference (Figure 5)

Each stage consists of multiple processes, depicted in the left-side column of each figure. Common operations or components within each process are denoted in the right-side column. Note that these stages are illustrative – it is not common for all operations to be implemented within a single monolithic system or process [24]. Circular transitions between processes or stages are also common. For example, within the Training stage, the Training and Model Analysis processes interleave many times before the AI is deemed ready to be deployed. Model Analysis could also reveal a need for additional data – requiring additional instances of Dataset Collection & Assembly. Accordingly, different variants of stage – or even of each process or operation – may exist within development and deployment pipelines. For example, developers add their own variants of the Dataset Collection & Assembly and Training stages when fine-tuning a pre-trained model.

Sections II-A, II-C, and II-B describe each stage, and the relationships between various AI assets within them. Section II-D then describes various noteworthy security implications of these relationships.

A. Dataset Collection & Assembly

Dataset Collection & Assembly encompasses the operations for collecting *Raw Data* and assembling it into a *Dataset*. Common operations are outlined in Figure 3.

Types of assets, and their relationships, that exist within this stage include:

- **Raw Data** (Consumed, Modified). Raw Data can be in the analog or digital domains. It may also be processed before consumption, outside project scope. For example, images collected from a public page may have been resized upon upload. Raw Data has various intermediate states as it undergoes Pre-Processing and Transformation.
- **Dataset** (Produced). The dataset is assembled from raw data which has been encoded, transformed, and labeled.
- **Validation Results** (Produced, Consumed). Validation results influence subsequent iterations of Dataset Collection & Assembly. That is, they can cause changes to feature selection or augmentation, or motivate collection of ad-

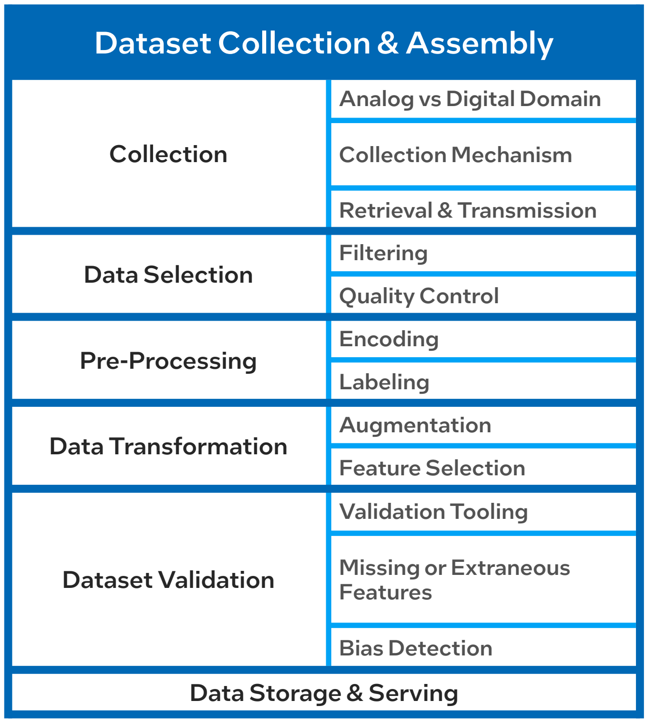


Fig. 3. Common processes (left column) and operations or components (right column) used for Dataset Collection & Assembly. It can be useful to consider 'Validation' as a separate stage when analyzing a specific pipeline's implementation, because it is often implemented on separate infrastructure or even outsourced to validation teams and services.

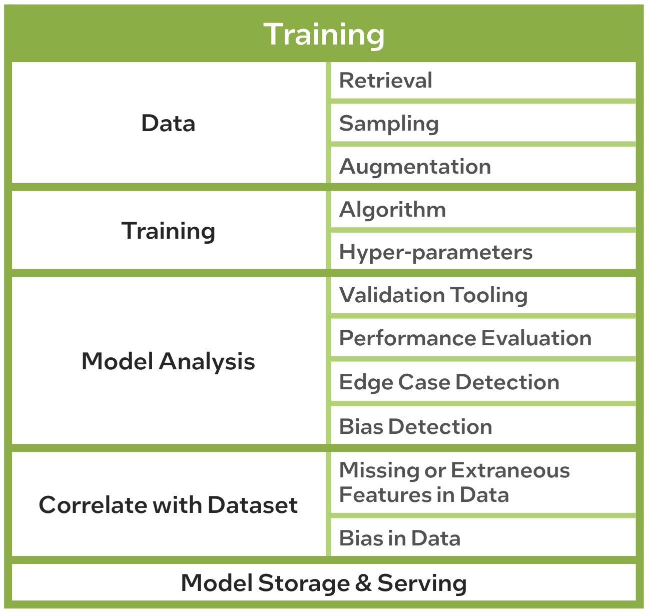


Fig. 4. Common processes (left column) and operations or components (right column) used for Training. For the same reasons as within Dataset Collection & Assembly, it is often useful to consider Model Analysis and Correlation with Dataset as a separate 'Validation' stage when analyzing a specific pipeline's implementation.

ditional data. They can also be useful to an adversary, as they reveal data types – and therefore features – the AI may have overfit on, may not recognize, or bias it may have internalized.

B. Training

Training encompasses the operations through which a Model learns from a training Dataset. Common operations are outlined in Figure 4.

Types of assets, and their relationships, that exist within this stage include:

- **Datasets** (Consumed, Transiently Modified). Datasets are consumed from the Dataset Collection & Assembly stage. These include the **Training Datasets** from which the model will extract features and infer input/output relationships, and the **Validation Datasets** which are used to produce validation results. It is common practice for training data to be dynamically – and transiently – augmented [6].
- **Validation Results** (Produced, Consumed). Validation results characterize the model's performance and behavior, serving as guideposts for changes to training or other stages. They could, for example, motivate the use of specific guardrails (security mechanisms that constrain AI behavior within predefined boundaries) during Deploy & Inference. To an adversary, they can serve as a guide for attacks by revealing corner cases, unintended behaviors, or biases.
- **Model** (Produced). Once the model's behavior meets pre-determined criteria, training is considered complete and the model is finalized; this criteria could be considered as a separate asset. It can subsequently be deployed for inference or used as a foundational model, for fine-tuning. Model checkpoints may also be produced and tracked, for provenance.

C. Deploy & Inference

This stage describes the process for deploying a model, including serving inference requests, monitoring, and model maintenance. Common operations are outlined in Figure 5.

Types of assets, and their relationships, that exist within this stage include:

- **Model** (Consumed and Exposed). Obtained from the training stage. The model is typically considered confidential, but its confidentiality can be impacted as adversaries can infer properties about it through their interactions. They can correlate the model behavior on inputs and outputs to steal or attack the model.
- **Inference or Prompt Inputs** (Consumed, Modified) and **Outputs** (Produced). Inputs are processed with the model to generate corresponding outputs. The source of inputs is product specific. They might originate from users, sensors, or system calls. Input pre-processing might

augment them with additional information; for example, from API calls or system state. Similarly, outputs may be transmitted to users, other AIs or services, etc. Conventional security best practices call for inputs and outputs to be secured in transit and during any pre-processing. It is increasingly common practice to collect inputs for potential use during AI retraining. Users typically expect that their inputs and outputs are isolated from those of other users; to maintain their privacy.

- **RAG Dataset** (Consumed). Represents a dataset deployed alongside a large language model, used to augment user inputs with data deemed to be relevant. This dataset can contain confidential or sensitive data. Attackers could also influence or change outputs by affecting this dataset.
- **Performance Metrics** (Produced). These metrics describe the model's performance and behavior on real world data, potentially motivating changes in this and other stages. These metrics are often used to detect drift or decide which input data to collect and use for retraining. They can include user feedback provided on model responses.
- **Raw Data** (Produced). Inputs which are collected for retraining are consumed as Raw Data by subsequent Dataset Collection & Assembly stages.
- **Training Dataset** (Exposed). The training dataset, unlike a RAG dataset, is not deployed alongside the model. This prevents adversaries from *directly* stealing or compromising it. However, the training dataset is, to some extent, encoded into the AI because of memorization [13]. Adversaries can, therefore, characterize the model to infer properties about the dataset or to extract information about individual data samples [11], [13].

D. Noteworthy Security Implications: Influencing AI Outputs

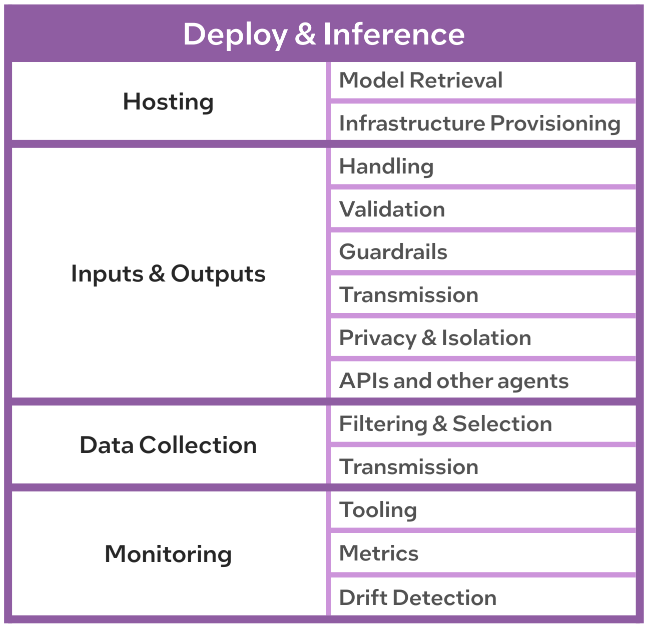
There are various noteworthy ways in which adversaries can exploit the relationships introduced above to influence AI Outputs by compromising the dataset or model. An adversary could trick the model into learning false features by changing the dataset [10], [38] or by directly compromising the model itself [8], [23].

Compromising the Dataset.

- **Permanent Writes** to the training data, at any point during Dataset Collection & Assembly or Training [10], [19], [34].
- **Permanent Additions** to the training data, by injecting malicious data into the data collection pipelines within Dataset Collection & Assembly or Deploy & Inference [30].
- **Transient Writes**, by modifying data as the model consumes it, during Training. For example, by compromising a library for Dataset Augmentation.

Compromising the Model.

- **Permanent Writes** to the model, during Training or Deploy & Inference [8].



Deploy & Inference	
Hosting	Model Retrieval
	Infrastructure Provisioning
Inputs & Outputs	Handling
	Validation
	Guardrails
	Transmission
	Privacy & Isolation
	APIs and other agents
Data Collection	Filtering & Selection
	Transmission
Monitoring	Tooling
	Metrics
	Drift Detection

Fig. 5. Common processes (left column) and operations or components (right column) used for deploying an AI and enabling its use. It is often useful to consider Data Collection and Monitoring as a separate 'Monitoring' stage when analyzing a specific implementation, as it is often implemented on separate infrastructure – to facilitate aggregation – and can also be outsourced to separate teams and services.

- **Transient Writes**, such as fault injections [12], [17], [23], [28], during Deploy & Inference.

The subtle relationships between AI assets, stemming from AI's computing paradigm, result in unique challenges for security analysis.

III. Challenges to Overcome

Security analysis for AI must overcome three key challenges. First, systems are increasingly distributed and specialized. Second, understanding all possible impacts of a vulnerability requires a holistic evaluation. Finally, aggravating the first two challenges, the vast majority of AI products are limited in scope but inherit risks in components consumed from outside their scope.

Challenge 1. AI Systems are increasingly disjoint and distributed. Training famously heavily benefits from scale out compute [7], [21], however, training is far from only operation implemented in a distributed fashion [24]. Consider Model Validation; the union of Model Analysis and Correlate with Dataset in the Training stage (Figure 4). The complexity of validation scales alongside growing data and model sizes [2], [37]. Increasingly, this complexity is addressed by specialized actors, using specialized tooling, on separate infrastructure. Similarly, it is unlikely that the data-science experts who train the AI will be the ones who deploy and maintain its inference infrastructure.

Challenge 2. Understanding the impact of a vulnerability requires a holistic evaluation. Distributing an AI's implementation and dependencies distributes its assets accordingly. Consider all the processes outlined in Figures 3, 4, and 5. Every process can exist within a separate hardware and software context, and each performs unique operations or transformations on a subset of the system's AI Assets. This creates many, often disjoint, attack surfaces. Reasoning about all possible implications of a vulnerability – identified in isolation, within a single context – requires that defenders map out how that vulnerability can be chained with vulnerabilities across other contexts.

Challenge 3. The vast majority of AI products are limited in scope. It is increasingly rare for real world products to develop an entire AI "from scratch." That is, for real world products to implement every stage, process, or operation necessary to train, deploy, and monitor an AI. Instead, developers leverage existing products and services to consume components necessary for their product. Pre-built datasets and pre-trained models epitomize this scenario. Software libraries, including those for validation or monitoring, are another common example of consumed components.

These components and the processes used to create them are often proprietary, however, so defenders often have little to no visibility into their validation or development. This greatly complicates defenders' ability to identify and quantify the risks they inherit by consuming those components. Defenders currently lack a clear method for quantifying security assumptions about consumed components – limiting their ability to reason about vulnerabilities stemming from their use.

Our methodology addresses these challenges using an asset-centric approach. Section IV provides background on asset-centric threat modeling, and Section V describes how we build on it.

IV. Asset-Centric Threat Modeling

The goal of Asset-Centric threat modeling is to identify vulnerabilities which, if exploited, allow an adversary to compromise an *Asset*.

Asset

A valuable object which requires protection. Defenders often distinguish between assets which are inherently valuable and those which are valuable because they protect, or serve as a stepping stone towards, other assets.

Using this methodology, defenders also quantify the degree of control (Integrity or Availability) or visibility (Confidentiality) an exploit provides over an Asset.

Adversarial Capability over an Asset

Describes a specific degree of control or visibility an adversary has, or could gain, over a specific asset.

Naturally, the amount of control or visibility gained depends on the vulnerability – the potential threat – being exploited. For example, compromising a user account might allow an adversary to Read a specific file, while compromising an admin account allows them to both Read and Write to it.

A. Asset-Centric Reasoning about Threats

Although some assets are inherently valuable, others are valuable because they protect, or serve as a stepping stone towards, other assets. Adversaries must typically compromise this second class of assets through individual exploits to move towards their true goal. It is useful, therefore, to identify the set of assets an adversary must compromise to achieve their goal. Including the minimum capabilities they must obtain over each asset.

Consider, as an example, theft of encrypted Personally Identifiable Information (PII) Data from storage. The data itself is an inherently valuable asset. The cryptographic key used to encrypt the data only has value because it protects that data. To compromise data confidentiality, however, the adversary must obtain the capabilities listed below over both assets.[1]

- Cryptographic Key (Asset): Read access (Capability, compromising confidentiality).
- Encrypted PII Data (Asset): Read access (Capability, compromising confidentiality).

In practice and in threat models, the vulnerabilities which provide each capability can be reasoned about – and exploited – as individual steps in a chain of attacks. Section V will describe how we leverage this property to develop our methodology.

V. Asset-Centric Threat Modeling for AI

Our asset-centric analysis, depicted in Algorithm 1, is comprised of four steps to identify relevant threats TM and a fifth step to identify mitigations M. This section discussions the four steps for identifying TM as two phases, each comprised of two steps.

In phase one (Algorithm 1, line 3) defenders map AF by identifying capabilities adversaries could obtain over the AI assets relevant to their $system$. Defenders reason about these capabilities based on whether they originate within product scope (Section V-A) or are inherited from consumed assets (Section V-B). Our analysis implements this reasoning as two steps, however, there is no strict dependence between them. They can be performed in parallel with each other and alongside existing security analysis.

Phase two performs threat analysis (Algorithm 1, line 2) to generate a knowledge base of attacks KB, and then identifies which threats from KB are relevant based on AF (Algorithm 1, line 4). Section V-C describes how threats are broken down into their asset-centric requirements and added into KB. Section V-D describes how those threats are contextualized

[1]We'll ignore exfiltration throughout this example, leaving it as an exercise for the reader to consider the additional assets – and capabilities over those assets – which the adversary must gain to enter the system and exfiltrate the data.

within a product, using AF, to determine if, and how, they are feasible.

KB is ordered first in Algorithm 1 because defenders can *reuse* AF (Algorithm 1, line 4) as KB changes – i.e. as new threats are discovered. Similarly, KB can also be reused as $system$ or AF change; even across different products. These properties allow defenders to efficiently scale their analysis with the growing and evolving threat landscape.

Algorithm 1 Asset-Centric AI Threat Modeling: Overall Process

```
1: procedure AssetCentricModel($system$)
2:     $KB \leftarrow$ InitializeKnowledgeBase()
3:     $AF \leftarrow$ MapAdversaryFootprint($system$)
4:     $TM \leftarrow$ MapThreats($AF, KB$)
5:     $M \leftarrow$ IdentifyMitigations($TM$)
6:     return threat analysis results
7: end procedure
```

Phase 1, where defenders perform security analysis to generate AF for their $system$, is outlined in Algorithm 2 and described in Sections V-A and V-B.

A. *Identifying Adversarial Influence within Product Scope (Phase 1)*

This step guides defenders through AI-centric analysis of vulnerabilities within product scope. During this process, defenders will reason about vulnerabilities from their existing security analysis but may also identify additional related vulnerabilities, such as those described in [18], [24]. As outlined in Algorithm 2, line 4-line 8, this step guides defenders through identifying the AI assets that exist within each in-scope stage, identifying vulnerabilities which could impact each asset, quantifying the capabilities each vulnerability could provide, and aggregating them into AF.

Repeating this process within each stage – i.e. each system boundary – allows AF to describe capabilities which could be obtained across across the entire development and deployment pipelines. This allows defenders to address Challenge 1 (Section III). An example output from this step is depicted on the top right of Figure 6.

1) Holistic Evaluation: Within Algorithm 2, line 19, defenders may identify immediate risks based on gained capabilities. For example, the risk of dataset theft through a vulnerability which provides the ability to read the dataset in storage. However, identifying *all* risks enabled by this vulnerability, as described in Challenge 2, requires that defenders consider the aggregated capabilities an adversary may gain across the entire product lifecycle.

2) Enabling Collaboration: Threat modeling and security experts – who are not AI or AI security experts – are already applying variants of Algorithm 2, line 19 when reasoning about conventional assets. It is, therefore, more straightforward for them to reason about AI *assets* than it is for them to identify the full range of AI *threats* enabled by an exploit. For example, a hardware security expert can identify that a fault injection attack would provide an adversary the ability to “perform transient writes to model weights, during inference." Leveraging KB, they use that information to identify in scope AI threats; e.g. an evasion attack. This information would also allow an AI security expert – who may not be a hardware security expert – to reason about the risks stemming from an exploit of that fault injection.

Algorithm 2 Phase 1: Mapping Adversary Footprint

```
1: procedure MapAdversaryFootprint($system$)
2:     $AF \leftarrow \emptyset$
3:     for each stage $s$ in system.stages do
4:         $inScopeAssets \leftarrow$ IdentifyAssets($s$)
5:         for each asset $a$ in $inScopeAssets$ do
6:             $cap \leftarrow$ DetermineCapabilities($a, s$)
7:             Add $(a, cap)$ to $AF$
8:         end for
9:         $consumedAssets \leftarrow$ IdentifyConsumedAssets($system, s$)
10:        if $consumedAssets \neq \emptyset$ then
11:            for each asset $a$ in $consumedAssets$ do
12:                $assumptions \leftarrow$ JustifyAssumptions($a, s$)
13:                Add $(a, assumptions)$ to $AF$
14:            end for
15:        end if
16:    end for
17:    return $AF$
18: end procedure
19: procedure DetermineCapabilities($asset, stage$)
20:    Identify attack surfaces related to $asset$ through existing security analysis
21:    For each attack surface, determine adversarial capabilities (Read, Write, Execute, etc.)
22:    Consider different adversary types with access to attack surface (insider, remote, etc.)
23:    return list of capabilities with constraints
24: end procedure
25: procedure JustifyAssumptions($asset, stage$)
26:    Consider producer reputation, provenance information, and asset properties
27:    Document justifications for trust or distrust in the asset
28:    return list of capabilities with constraints
29: end procedure
```

3) Reasoning about Different Adversaries: It is common practice to threat model from the perspective of multiple adversaries, each with a distinct level of access to the system or product. In scenarios where the obtained capabilities depend on each adversary's level of access, defenders would identify each adversary-specific outcome. Subsequent analysis, in Phase 2, would leverage this information to contextualize and distinguish each adversary's specific techniques.

B. Quantifying Assumptions about Adversarial Influence Outside Product Scope (Phase 1)

The second step in phase one, depicted in Algorithm 2, line 9-line 15, calls for defenders to identify the capabilities adversaries may have obtained over consumed assets while they were outside product scope.

At first glance, the lack of visibility into each consumed asset's source appears to invite pessimistic assumptions. That is, defenders could argue that consuming any asset is a massive risk because they cannot guarantee that an adversary did not obtain full control over the asset outside product scope. However, defenders *are* able to justify constraints on assumed capabilities. As described in Algorithm 2, line 25, defenders could leverage supplemental material – such as provenance information – to justify constraints on assumed capabilities. Effective provenance information should be end-to-end, capturing the complete lifecycle of AI assets from creation through deployment, as proposed in frameworks like Atlas [32]. Defenders could also leverage the producer's reputation to justify constraints. For example, defenders might justify a larger degree of trust in a producer's infrastructure if that producer is a large AI corporation instead of an anonymous account on a public code repository or model zoo. Similarly, properties about the asset itself may justify a larger degree of trust. For example, it is easier to justify trust in the validation of a pre-trained model that has widespread adoption and has been the subject of academic and industry research.

The resulting assumptions are framed in the same way as if they had been identified in the previous step; an example output is depicted on the top left of Figure 6. This allows defenders to aggregate them with the capabilities obtained within product scope. The top half of Figure 6, encompassing both steps in Phase 1, then depicts a complete AF.

C. Threat Analysis (Phase 2)

The first step in phase two (Algorithm 1, line 2) generates KB – a prerequisite for mapping attacks (Algorithm 1, line 4). Defenders repeat the first portion of the procedure in Algorithm 3, line 15 for every attack they wish to add into KB. Fortunately, this process is product-independent. KB can be reused when analyzing other products because it is not dependent on $system$. The cards on the bottom left of Figure 6 represent a knowledge base of threats.

As described in Section IV-A and depicted in Algorithm 3, line 16, an attack's requirements can be described using the set of assets an adversary must gain capabilities over. Papernot et. al.'s work in [26] presented a first step towards describing adversarial AI attacks based on their requirements; albeit less rigorously than would be necessary for bottom-up analysis. Section V-C1 describes the process for mapping the requirements for Backdoor attacks. Table I summarizes the identified requirements.

1) Example Breakdown: Backdoor attacks : Consider, as an example, a backdoor attack [20], [30], [34] where the adversary's goal is to circumvent a content filter; an image classifier for detecting inappropriate visual ads (e.g. those that promote racism or political extremism). Backdoor attacks manipulate a classifier's training process so that any input with an attacker-designed trigger pattern will be classified as benign. A trigger pattern could be visual noise which is imperceptible to humans. For a backdoor attack to be successful, an adversary must:

1) Manipulate the Training Data. Some subset of the training data for the adversary's target class – i.e. ads labeled as "benign" – must include the trigger pattern.
2) Modify inference inputs so that they include the backdoor trigger. This is how the adversary exploits the backdoor. They overlay their chosen trigger pattern on non-benign ads, allowing those ads to circumvent the content filter and reach millions of internet users.

Note that solely achieving either of these requirements is insufficient. There will be no backdoor for the adversary to exploit if they cannot modify the training data. Similarly, the attack will fail if the adversary successfully modifies the training data but cannot submit ads which include the trigger. For example, because the adversary is not allowed, or cannot afford, to purchase the ad space. Table I and the top card on the bottom left of Figure 6 depict the requirements to perform a backdoor attack. The rest of this section describes how those requirements were determined.

The Adversarial AI literature has identified many techniques which will allow an adversary to modify training data. Four examples are listed below. A description of the required capability has been included alongside each. The first three describe techniques which allow an adversary to change benign training data. That is, the adversary has obtained some form of Write capability and, implicitly, a Read capability. The fourth technique is of particular note because the adversary has no access to benign training data. They are unable to Read or Write over benign training data.

Example Techniques to Compromise Training Data:

1) Apply trigger to benign examples in a public dataset. Victims which consume this dataset will become vulnerable to the trigger pattern [31], [34]. It is often assumed that the adversary themselves hosts this dataset. The adversary must also, somehow, ensure that the intended victim consumes the dataset.
   Capability: Permanently modify training data. Modifications must be imperceptible to avoid detection.
2) Manipulate training data during collection or in storage. An adversary with a foothold on a victim's data collection infrastructure can apply the trigger to benign examples as they are collected, transformed, or encoded. Similarly, an adversary could compromise the data at rest, in storage. These modifications are permanent and persist across all future uses of the data – unless the victim is able to rolls the dataset back to a previous version.
   Capability: Permanently modify training data. Modifications must be imperceptible to avoid detection.
3) Manipulate training data during use. The adversary

transiently applies the trigger to benign data as it is used for training. For example, by compromising a data augmentation library [6].
These modifications may not need to be imperceptible because augmented training data is typically not saved. Saving augmentation data would essentially require saving a modified version of the dataset for every single epoch of training. Doing so would incur massive storage costs while generating too much data for analysis to be feasible.
Capability: Transiently modify training data.
4) Apply the trigger to benign-appearing data and inject it into a data collection or retraining pipeline. For example, by leveraging techniques from [30], [38].
This technique exemplifies an AI-specific way for adversaries to interact with data: by exploiting data collection or re-training pipelines. This capability is also the minimal required capability, for a backdoor attack, because the adversary has no read or write access to any data besides their own. Notably, they also also do not need a foothold on any victim infrastructure.
Capability: Contribute training data. Modifications must be imperceptible to avoid detection.

Table I summarizes the relevant assets and *minimum* capabilities an adversary must obtain over each. Critically, obtaining *stronger* capabilities enables an adversary to perform more powerful variants of backdoor attacks. Using the minimum capabilities during their analysis allows a defender to identify the full range of backdoor attacks which are in-scope for their system. Therefore, the minimum requirements are used when mapping threats to AF in Figure 6.

TABLE I
THIS TABLE SUMMARIZES BACKDOOR ATTACKS. ROW 3 IDENTIFIES THE *asset-centric attack requirements* FOR PERFORMING A BACKDOOR ATTACK.

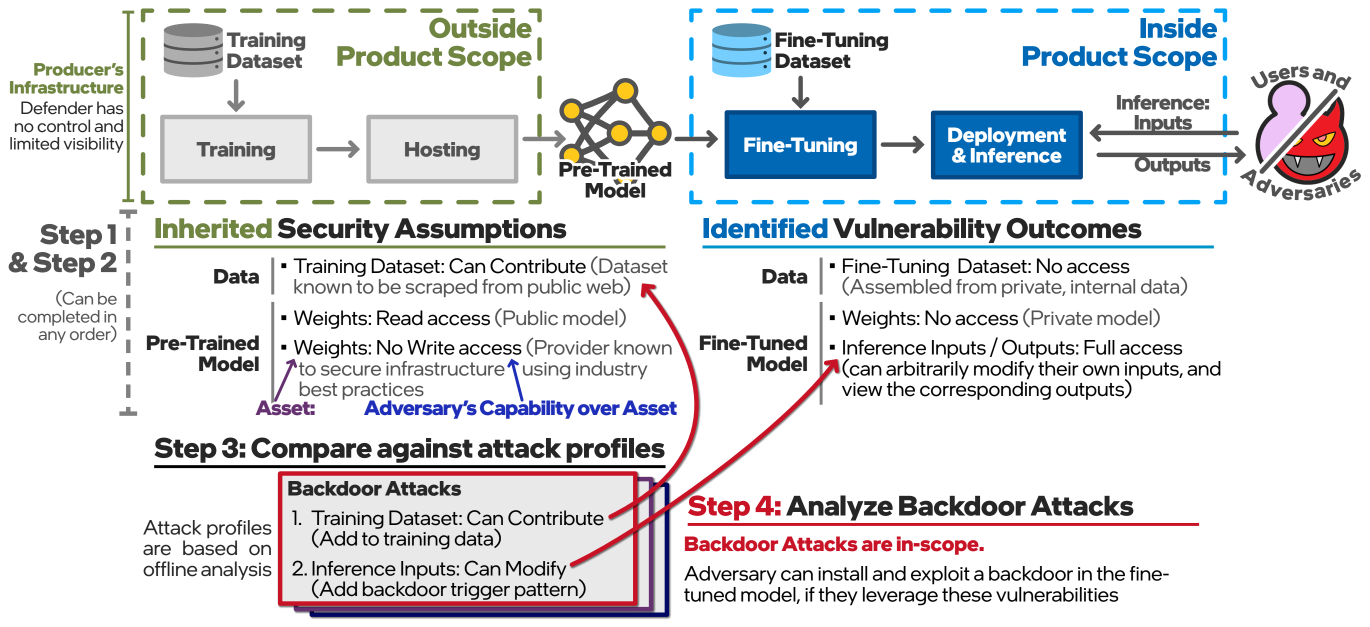
Backdoor Attacks	Backdoor attacks allow an adversary to manipulate inference output.
Technique	Attacker manipulates classifier's training process so that any input with an attacker-designed trigger pattern will classify to an attacker-chosen target class.
Attack Requirements	Adversary must compromise the **Training Dataset** and **Inference Inputs**. 1) **Training Dataset:** Add to the training data. 2) **Inference Inputs:** Modify inputs so that they include the backdoor trigger.
Impact	AI Output becomes adversary controlled, but only for inputs which include the backdoor trigger.

D. Mapping an attack into a product (Phase 2)

The second step in phase 2 (Algorithm 1, line 4) compares the aggregated capabilities in AF against the threats in KB to identify in-scope threats. Having broken down an AI attack as described in Section V-C and added it into the knowledge base KB, defenders determine whether that attack could be enabled by obtaining any combination(s) of the capabilities identified in AF. As depicted in Algorithm 3, line 24, this is done by comparing the capabilities an adversary could gain within the AI's lifecycle against the attack's requirements. If the attack's requirements can be met by some combination of vulnerabilities, the attack is deemed to be in scope and added into TM. This analysis is caricatured by the red arrows in Figure 6, The defender may identify *multiple* vulnerabilities, across the stack and the distributed pipeline, which provide the required capability. Knowing that the adversary could exploit *any* of these vulnerabilities or assumptions, as a link in their chain of attacks, facilitates the defender's risk analysis.

Algorithm 3 Phase 2: Mapping Threats Based on Adversary Footprint (Scalable Analysis)

```
1: procedure MAPTHREATS($AF, KB$)
2:     $inScope \leftarrow \emptyset$
3:     $outScope \leftarrow \emptyset$
4:     for each threat $t$ in $KB$ do
5:         if RequirementsMet($t.requirements, AF$) then
6:             $vectors \leftarrow$ IdentifyAttackVectors($t, AF$)
7:             Add $(t, vectors)$ to $inScope$
8:         else
9:             $just \leftarrow$ ExplainUnmetRequirements($t, AF$)
10:            Add $(t, just)$ to $outScope$
11:        end if
12:    end for
13:    return $(inScope, outScope)$
14: end procedure
15: procedure ANALYZENEWTHREAT($newThreat, AF$)
16:    Breakdown $newThreat$ into its requirements
17:    Add $newThreat$ to knowledge base
18:    if RequirementsMet($newThreat.requirements, AF$)
   then
19:        return "In scope" with attack vectors
20:    else
21:        return "Out of scope" with justification
22:    end if
23: end procedure
24: procedure REQUIREMENTSMET($requirements, AF$)
25:    for each requirement $r$ in $requirements$ do
26:        $met \leftarrow$ false
27:        for each capability $c$ in $AF$ do
28:            if $c$ satisfies $r$ then
29:                $met \leftarrow$ true
30:                break
31:            end if
32:        end for
33:        if not $met$ then
34:            return false
35:        end if
36:    end for
37:    return true
38: end procedure
```



Producer's Infrastructure
Defender has no control and limited visibility
Training Dataset
Outside Product Scope
Training
Hosting
Pre-Trained Model
Fine-Tuning Dataset
Inside Product Scope
Fine-Tuning
Deployment & Inference
Inference: Inputs
Outputs
Users and Adversaries
Step 1 & Step 2
(Can be completed in any order)
Inherited Security Assumptions
Data
▪ Training Dataset: Can Contribute (Dataset known to be scraped from public web)
Pre-Trained Model
▪ Weights: Read access (Public model)
▪ Weights: No Write access (Provider known to secure infrastructure using industry best practices
Asset:
Adversary's Capability over Asset
Identified Vulnerability Outcomes
Data
▪ Fine-Tuning Dataset: No access (Assembled from private, internal data)
Fine-Tuned Model
▪ Weights: No access (Private model)
▪ Inference Inputs / Outputs: Full access (can arbitrarily modify their own inputs, and view the corresponding outputs)
Step 3: Compare against attack profiles
Attack profiles are based on offline analysis
Backdoor Attacks
1. Training Dataset: Can Contribute (Add to training data)
2. Inference Inputs: Can Modify (Add backdoor trigger pattern)
Step 4: Analyze Backdoor Attacks
Backdoor Attacks are in-scope.
Adversary can install and exploit a backdoor in the fine-tuned model, if they leverage these vulnerabilities

Fig. 6. This figure depicts an asset-centric analysis across an example AI lifecycle. The top left lists various security assumptions, including justifications, made about the pre-trained model being consumed. The steps necessary to train are outside product scope, preventing defenders from directly inspecting them. The top right lists various vulnerabilities identified during security analysis of the steps which are in product scope. The bottom depicts how a defender can combine the insights from an attack profile with the security assumptions and identified vulnerabilities to determine if, and how, a specific attack is possible.

Notably, because TM is generated using AF, the defender does not need in depth knowledge of each vulnerability or its originating system architecture when implementing Algorithm 3. The defender may even choose not mitigate a threat at the "root cause" vulnerabilities in AF. For example, a defender may opt to mitigate the risks of a backdoor attack (Table I) by implementing a consensus system instead of performing high-cost and high-complexity model retraining. This mitigation instead opts to increases confidence in AI outputs by augmenting them with information from a non-AI source – such as a LIDAR sensor in an autonomous vehicle – or a second model.

VI. Enterprise RAG: A Secure Retrieval Augmented Generation Architecture

The Enterprise RAG architecture implements a secure Retrieval Augmented Generation system for enterprise deployments. As illustrated in Figure 7, the system follows an asset-centric security approach, where each component is treated as a potential attack surface with specific security controls [25].

The architecture comprises three primary functional domains: Data Ingestion, RAG Query Processing, and Configuration/Metrics. At the entry point, an NGINX Ingress layer provides initial request handling, with all traffic flowing through a comprehensive authentication layer that leverages OpenID/SAML protocols. A React-based UI interfaces with users, while an API SIX Gateway mediates all communication between frontend components and backend services. This gateway serves as a critical security boundary, enforcing access controls and traffic validation.

The data flow begins with enterprise data sources (documents, email, chat, and technical databases) being processed through an extraction and data preparation pipeline. The processed data is then embedded and stored in a secure Vector Database. When users submit queries, the system enhances standard LLM processing with a retrieval mechanism that incorporates relevant contextual information. This involves several secured microservices: Embedding, Rerank, and LLM components, with LLM Guard modules providing content filtering and safety validation at two critical points - before prompt construction and after generation.

VII. Applying Asset-Centric Threat Modeling to Enterprise RAG

To demonstrate the application of asset-centric threat modeling to AI systems, we onboard the Enterprise RAG [25] (Section VI) architecture into this framework. Following the asset-centric methodology we begin by identifying the key AI assets within the system and the potential adversarial capabilities over these assets.

A. Asset Identification and Capability Mapping

For our Enterprise RAG system, we identify the following critical AI assets:

- **Enterprise Data**: Source documents containing sensitive corporate information
- **Embedding Model**: Transforms raw text into vector representations
- **Vector Database**: Stores embeddings with semantic search capabilities

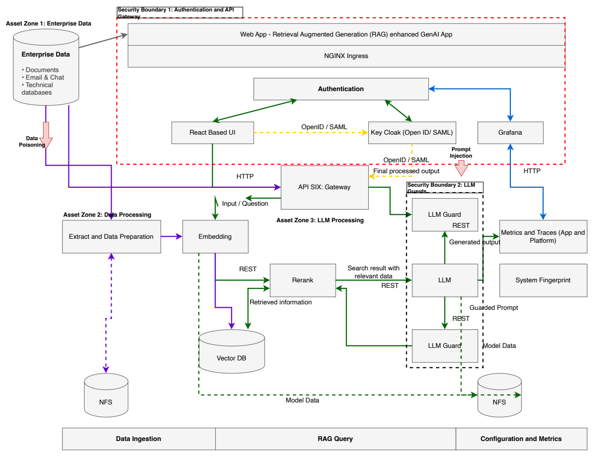


Fig. 7. Enterprise RAG Architecture with Asset-Centric Security Analysis. The diagram illustrates a complete Retrieval Augmented Generation system with four distinct asset zones: (1) Enterprise Data (top-left), containing source documents and databases; (2) Data Processing (bottom-left), including extraction, embedding, and vector storage components; (3) LLM Processing (center-right), featuring reranking, LLM, and guard components; and (4) Monitoring (far-right) with metrics and system fingerprinting. Security boundaries (authentication layer and LLM guards) protect critical assets. Red arrows indicate potential attack vectors: data poisoning, prompt injection, vector manipulation, and retrieval tampering. This asset-centric approach enables systematic vulnerability identification across distributed components, regardless of deployment context. Each asset is evaluated for adversarial capabilities concerning confidentiality, integrity, and availability.

- **LLM**: Large language model for generating responses
- **Fine-tuning Dataset**: Data used to adapt the LLM to enterprise context
- **User Queries**: End-user inputs that could be manipulated
- **Generated Responses**: Output text that must maintain accuracy and safety

For each asset, we map the potential adversarial capabilities. For instance, with the Vector Database:

- **Read Access**: An adversary could extract sensitive embeddings
- **Write Access**: An adversary could poison the database with malicious vectors
- **Contribute Data**: Limited to adding new entries without modifying existing ones

B. *Vulnerability Analysis and Threat Contextualization*

Through security analysis, we identified several potential vulnerabilities in our implementation:

- **API Gateway**: Potential for request manipulation if authentication is bypassed
- **LLM Guard**: Risk of prompt injection if guardrails are insufficient
- **Vector DB**: Potential for embedding extraction if access controls are compromised

By correlating these vulnerabilities with our asset-capability mapping, we can contextualize specific attacks. For example, a jailbreaking attack on our Enterprise RAG would require:

1) Capability over **User Queries**: Ability to craft specially formatted inputs
2) Capability over **LLM Guard**: Ability to bypass prompt filtering mechanisms
3) Capability over **Generated Responses**: Ability to receive unfiltered LLM output

Our security analysis identified that while the system has strong protections for the Vector Database and enterprise data sources, the LLM Guard component could potentially be bypassed through sophisticated prompt engineering. This leads to a focused mitigation strategy targeting this specific vulnerability rather than implementing broad, untargeted security measures.

TABLE II
ASSET-CENTRIC SECURITY ANALYSIS OF ENTERPRISE RAG ARCHITECTURE

Asset Zone	**Key Assets**	**Adversarial Capabilities**	**Security Controls**
Enterprise Data	Source documents, Email, Technical databases	Read (data theft), Write (data poisoning)	Access controls, Data validation, Integrity checks
Data Processing	Embedding model, Vector DB, NFS	Read (model stealing), Write (embedding poisoning), Contribute (fake data)	Embedding validation, Anomaly detection, Distribution checks
LLM Processing	Reranker, LLM, LLM Guards	Read (prompt leakage), Execute (jailbreaking)	Input sanitization, Output filtering, Context boundaries
Monitoring	Metrics, System Fingerprint	Write (log tampering), Read (security insights)	Immutable logs, Real-time monitoring

VIII. Related Work

An asset-centric approach is highly complementary to existing AI threat modeling frameworks because existing frameworks take a top-down perspective while an asset-centric approach is bottom-up. In practice, defenders will benefit from blending the two perspectives.

1) Perspective for Analysis: Today's AI threat modeling frameworks map the space of potential AI attacks, broadly describing the techniques adversaries might use to perform each one [1], [4], [26]. MITRE's ATLAS framework [1], for example, categorizes attack techniques based on their role within MITRE's attack lifecycle; allowing defenders to reason about a specific attack based on the steps an adversary takes to achieve that attack. Defenders use this approach to contextualize a specific attack within their product by individually mapping every step the adversary takes, from Reconnaissance and Resource Development through to Exfiltration and Impact. This is a top-down perspective because the defender's analysis maps the attack onto their product.

A defender seeking to prevent an adversary from manipulating the output of their AI, for example, would need to consider every possible attack technique which could lead to output manipulation. As potential techniques span across the AI's lifecycle, defenders must individually contextualize many attack types within their product. This approach, therefore, does not scale with a quickly growing and evolving attack space. It also does not inherently leverage the insights from a defender's existing security analysis.

Defenders cannot quickly determine whether a vulnerability identified in their threat model could enable an AI attack. That is, whether it could serve as a step in an attack's lifecycle. Contextualizing a step in an attack's lifecycle could require reanalyzing every vulnerability identified in the threat model, to determine potential relevance. This is a particularly challenging task when reason about the ways in which conventional hardware or security vulnerabilities can facilitate AI threats.

Asset-centric analysis helps invert this perspective. It's *centered* around allowing defenders to determine whether the vulnerabilities they've already identified in their threat model can serve as steps in an AI attack's lifecycle. That is, by allowing them to identify how a vulnerability impacts AI assets and reason about whether that impact enables subsequent attacks. Importantly, defenders can reason about many attacks by *reusing* their analysis. Finally, it is more accessible for product domain and security experts – who may not be AI security experts, but do have asset-centric threat modeling experience. Their security analysis can scale with a quickly evolving, and often nuanced, AI threat landscape by reasoning about threats at an asset level.

2) Analysis Across System Boundaries: Defenders lack a straightforward way to apply today's frameworks to AI systems distributed across multiple actors. Consuming AI assets from external, producing, actors – such as datasets or pretrained models – allows developers to leverage state of the art AI assets without the costs of creating them from scratch. However, due to their proprietary nature, defenders often lack visibility into the systems which produced those assets. Defenders are unable to contextualize attacks within those systems, as required by a top-down approach. This prevents defenders from leveraging today's frameworks to identify and quantify the security assumptions they make about assets consumed from external, closed box, systems.

Defenders can overcome this challenge through a *boundary based* approach, enabled by asset-centric analysis. This approach does not require defenders to reason about individual vulnerabilities within systems they lack visibility over. That is, defenders can identify, quantify, and justify their security assumptions about consumed assets at system boundaries – at the point where assets transition across contexts. As described in Section V-B, there are many ways for defenders to justify constraints on their security assumptions. They might, for example, leverage provenance information, the producer's reputation, or properties about the asset itself. Conversely, producers could leverage an asset-centric perspective to communicate the security guarantees they provide – without needing to open their systems. Even AI-enabling products – such as processors, accelerators and GPUs, or cloud infrastructures – could leverage this perspective to describe the security guarantees they provide for AI assets.

3) Complementary Analysis: Top-down and bottom-up perspectives are not mutually exclusive. Ultimately, defenders will benefit most from incorporating both top-down and bottom-up perspectives into their security analysis. Existing frameworks, including ATLAS and [18], [24], could also augment their attack knowledge bases with asset-centric information. Defenders could then leverage these knowledge bases for whichever perspective – or combination of perspectives – best fit their unique needs.

IX. Ongoing Work and Next Steps

We have identified, and are pursuing, various opportunities to facilitate the adoption and automation of asset-centric analysis for threat modeling AI. The security analysis in Phase 1 can be facilitated by standardizing the types of AI assets, the range of potential capabilities adversaries can obtain over them (Sections V-A and V-B). The threat analysis in Phase 2 can be facilitated by building and augmenting AI attack knowledge bases with asset-centric information. Defenders can reference such knowledge bases instead of reanalyzing attacks (Section V-C). Finally, threat mapping can be automated using these knowledge bases (Section V-D). These opportunities are, in large part, enabled by standardizing the types of AI assets and mapping out the range of possible capabilities adversaries may obtain over each.

While our formalized knowledge base and standardization efforts continue to develop, organizations may consider taking initial steps toward this methodology. Security teams could begin by identifying key AI assets within their existing environments and considering how these assets might be affected by current threats. Even a simplified version of the asset-capability mapping can provide valuable insights when integrated into existing security processes. By bringing together security practitioners and AI specialists for collaborative threat analysis sessions, organizations can build the cross-functional understanding needed for effective AI security, laying groundwork that will align well with more comprehensive asset-centric approaches as they mature.

A. Standardizing Assets

As a foundation for facilitating, optimizing, and automating asset-centric analysis, we have enumerated the different types of AI assets and have mapped out the extents to which each can be compromised. Just like for non-AI assets, the range of potential capabilities that an adversary can gain over each type of AI asset can be statically mapped out for each asset type. Today, defenders often denote whether a vulnerability would compromise an asset's Confidentiality, Integrity, or Availability. These ranges are particularly interesting for AI assets because adversaries can gain previously unseen, nuanced, capabilities over them. One such capability, discussed in the example within Section V-C, is an adversary's ability to *add* data to an AI's training or re-training datasets but lack the ability to read or write the rest of the dataset. Standardization also makes threat models easier to write, interpret, and reuse. Especially for those who are product domain or security experts but not AI security experts. It also facilitates reuse of security assumptions and threat analysis.

B. Reusing Security Assumptions

Security assumptions about a consumed asset can be reused across different products, because those assumptions are not specific to the defender's product. Defenders' security assumptions about that asset are based on their assumptions about the *producer's* systems (Sectio V-B).

Consider, for example, two products which consume the same pre-trained model. Security assumptions about the pre-trained model are based on how that model is *created* – they are not based on how that model is *used* once consumed. It's the *interpretation* of those assumptions – i.e. their potential impact – which is product-specific. As with conventional hardware and software security, the security requirements of two products can be significantly different. Even if they share common components. Critically, security assumptions must be based on up-to-date justifications for them to be reused. However, it's easier for defenders to verify that previous justifications still hold than it is for them to repeat the analysis from scratch.

C. Automating Threat Analysis

We are building a knowledge base of AI attacks augmented with asset-centric information. To maximize re-usability, this knowledge base uses our standardized asset types.

Defenders could *programmatically* compare each entry in this database – rather than manually – against the potential capabilities they identified in AF (Section V-D).

X. Conclusion

Security analysis for AI must overcome three key challenges: (1) increasingly distributed development and deployment pipelines across disjoint infrastructures (2) the need for holistic evaluation of vulnerabilities across system boundaries, and (3) limited visibility into external components that AI systems increasingly depend upon. Defenders face a complex task when identifying vulnerabilities in these distributed environments. They must not only locate individual vulnerabilities within separate contexts but also understand how adversaries might chain these vulnerabilities across system boundaries. This challenge is compounded when systems incorporate external AI assets like pre-trained models, where defenders lack visibility into the development processes that created these components.

Current threat modeling frameworks employ a top-down approach that focuses on contextualizing specific attacks within product boundaries. This approach becomes increasingly untenable as AI systems grow more complex for two reasons: it requires comprehensive visibility across all development contexts, and it forces defenders to repeatedly reanalyze their entire system for each new attack pattern. As a result, these approaches cannot efficiently scale to address the rapidly evolving AI threat landscape.

This work presents a bottom-up, asset-centric, analysis that can be used to overcome these challenges. This analysis is implemented in two phases. First, defenders identify and quantify all potential adversarial influence over AI assets; both for assets within their product scope and assets consumed from external, often closed-box, systems. Second, defenders analyze AI attacks – from an asset-centric perspective – so that they can contextualize those attacks within their product. This process allows defenders to determine whether the attacks are feasible within their system and to identify the root-cause vulnerabilities that enabled those attacks. Importantly, this methodology scales with a growing attack space because defenders *reuse* their analysis from the first phase when they're contextualizing attacks in the second phase. That is, unlike with top-down approaches, defenders need not reanalyze their product when evaluating each attack.

This work also describes ongoing and future work which will facilitate integrating and automating this methodology. Specifically, and externally to this work, we have identified the different possible types of AI assets and mapped out the different degrees of control and/or visibility an adversary can gain over each one. Combined with the methodology presented in this work, standardizing AI assets will allow the security assumptions about consumed assets to become portable across products, allow AI security experts to develop and augment AI attack knowledge bases with asset-centric information, and allow defenders to automate the mapping of attacks from those knowledge bases onto defenders' product-specific context.

Ultimately, defenders will benefit most from *complementing* top-down analysis – when it's feasible – with bottom-up analysis. Top-down and bottom-up perspectives are not mutually exclusive and each has unique benefits. This work takes the first step towards enabling both perspectives by presenting defenders with the first bottom-up approach for threat modeling AI.

References

[1] ATLAS Matrix | MITRE ATLAS™. https://atlas.mitre.org/matrices/ATLAS.

[2] Gpt-4-system-card.pdf. https://cdn.openai.com/papers/gpt-4-system-card.pdf.

[3] Malicious AI models on Hugging Face backdoor users' machines. https://www.bleepingcomputer.com/news/security/malicious-ai-models-on-hugging-face-backdoor-users-machines/.

[4] OWASP Top Ten for LLM. https://owasp.org/www-project-top-10-for-large-language-model-applications/assets/PDF/OWASP-Top-10-for-LLMs-2023-v1_0.pdf.

[5] Significant-Gravitas/AutoGPT at v0.4.2. https://github.com/Significant-Gravitas/AutoGPT.

[6] What is Data Augmentation? - Data Augmentation Techniques Explained - AWS. https://aws.amazon.com/what-is/data-augmentation/.

[7] Dan Alistarh. A brief tutorial on distributed and concurrent machine learning. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, 18:487–488, 2018.

[8] Mikel Bober-Irizar, Ilia Shumailov, Yiren Zhao, Robert Mullins, and Nicolas Papernot. Architectural Backdoors in Neural Networks. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 24595–24604. IEEE Computer Society, June 2023.

[9] Nadia Burkart and Marco F. Huber. A Survey on the Explainability of Supervised Machine Learning. *Journal of Artificial Intelligence Research*, 70:245–317, January 2021.

[10] Nicholas Carlini, Matthew Jagielski, Christopher A. Choquette-Choo, Daniel Paleka, Will Pearce, Hyrum Anderson, Andreas Terzis, Kurt Thomas, and Florian Tramèr. Poisoning Web-Scale Training Datasets is Practical, February 2023.

[11] Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting Training Data from Large Language Models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650, 2021.

[12] Yanzuo Chen, Zhibo Liu, Yuanyuan Yuan, Sihang Hu, Tianxiang Li, and Shuai Wang. Unveiling Single-Bit-Flip Attacks on DNN Executables, October 2023.

[13] Christopher A. Choquette-Choo, Florian Tramer, Nicholas Carlini, and Nicolas Papernot. Label-Only Membership Inference Attacks, December 2021.

[14] Lukas Euler. Hacking Auto-GPT and escaping its docker container | Positive Security. https://positive.security/blog/auto-gpt-rce#convincing-gpt-4-to-interpret-attacker-controlled-text-as-instructions.

[15] Berenice Flores Garcia. Ray, Versions 2.6.3, 2.8.0. https://bishopfox.com/blog/ray-versions-2-6-3-2-8-0.

[16] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection, May 2023.

[17] Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitraș. Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 497–514, 2019.

[18] Omar Khawaja, Arun Pamulapati, and Kelly Albano. Databricks AI Security Framework (DASF), 2024.

[19] Keita Kurita, Paul Michel, and Graham Neubig. Weight Poisoning Attacks on Pre-trained Models, April 2020.

[20] Linyang Li, Demin Song, Xiaonan Li, Jiehang Zeng, Ruotian Ma, and Xipeng Qiu. Backdoor Attacks on Pre-trained Models by Layerwise Weight Poisoning, August 2021.

[21] Eun Ji Lim, Shin Young Ahn, and Wan Choi. Accelerating training of DNN in distributed machine learning system with shared memory. In *International Conference on Information and Communication Technology Convergence: ICT Convergence Technologies Leading the Fourth Industrial Revolution, ICTC 2017*, volume 2017-Decem, pages 1209–1212. IEEE, October 2017.

[22] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. Explainable AI: A Review of Machine Learning Interpretability Methods. *Entropy*, 23(1):18, January 2021.

[23] Abdulrahman Mahmoud, Siva Kumar Sastry Hari, Christopher W. Fletcher, Sarita V. Adve, Charbel Sakr, Naresh Shanbhag, Pavlo Molchanov, Michael B. Sullivan, Timothy Tsai, and Stephen W. Keckler. HarDNN: Feature Map Vulnerability Evaluation in CNNs, February 2020.

[24] Gary McGraw, Harold Figueroa, Victor Shepardson, and Richie Bonett. AN ARCHITECTURAL RISK ANALYSIS OF MACHINE LEARNING SYSTEMS:.

[25] OPEA Project. Enterprise-rag: Retrieval augmented generation framework for enterprise applications. https://github.com/opea-project/Enterprise-RAG, 2024. Accessed: March 24, 2025.

[26] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P. Wellman. SoK: Security and Privacy in Machine Learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414, London, April 2018. IEEE.

[27] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1157–1174, May 2022.

[28] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-Flip Attack: Crushing Neural Network with Progressive Bit Search, April 2019.

[29] Jose Rodrigo Sanchez Vicarte, Benjamin Schreiber, Riccardo Paccagnella, and Christopher W. Fletcher. Game of Threads: Enabling Asynchronous Poisoning Attacks. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, pages 35–52, New York, NY, USA, March 2020. Association for Computing Machinery.

[30] Jose Rodrigo Sanchez Vicarte, Gang Wang, and Christopher W. Fletcher. Double-Cross Attacks: Subverting Active Learning Systems. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, pages 1593–1610, 2021.

[31] Shawn Shan, Wenxin Ding, Josephine Passananti, Stanley Wu, Haitao Zheng, and Ben Y. Zhao. Nightshade: Prompt-Specific Poisoning Attacks on Text-to-Image Generative Models, April 2024.

[32] Marcin Spoczynski, Marcela S. Melara, and Sebastian Szyller. Atlas: A framework for ml lifecycle provenance & transparency. *arXiv preprint arXiv:2502.19567*, 2024.

[33] Jean-Baptiste Truong, Pratyush Maini, Robert J. Walls, and Nicolas Papernot. Data-Free Model Extraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4771–4780, 2021.

[34] Alexander Turner, Dimitris Tsipras, and Aleksander Mądry. Clean-Label Backdoor Attacks.

[35] Apostol Vassilev, Alina Oprea, Alie Fordyce, Hyrum Anderson, Xander Davies, and Maia Hamin. Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations.

[36] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. {DeepHammer}: Depleting the Intelligence of Deep Neural Networks through Targeted Chain of Bit Flips. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1463–1480, 2020.

[37] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization.

[38] Xuezhou Zhang, Xiaojin Zhu, and Laurent Lessard. Online Data Poisoning Attacks. In *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, pages 201–210. PMLR, July 2020.