AI, Latent Reasoning, Pickle Risks

Gemini Deep Research
Ronni Ross
15 June 2025

Prompt

AI models, latent reasoning and its synergy with Python's Pickle module, the danger and potential.

Reasoning

Researching Core Concepts
I'm starting my research by focusing on the foundational concepts. I'll be defining what AI models are and exploring the concept of latent reasoning, including examples of AI architectures that demonstrate this capability. My aim is to gather a solid understanding of these core areas before moving on to their intersection with the `pickle` module.
Next Steps in Research
My next steps will involve understanding the functionality of Python's `pickle` module. Following that, I'll delve into the synergy between AI models, particularly those utilizing latent representations, and `pickle`. This will set the stage for investigating the security implications, practical benefits, and best practices associated with this interaction, as well as exploring alternative serialization methods.

Researching websites

Understanding AI Models and Latent Reasoning
I've begun by establishing a clear understanding of AI models, recognizing them as programs that identify patterns in data to draw conclusions or perform actions. This process typically involves three stages: modeling, training, and inference. A key insight I've gained is the concept of 'latent reasoning,' which is a more advanced approach where AI models process and refine their thoughts internally within a 'latent space' before generating any output. This contrasts with 'Chain-of-Thought' reasoning, which externalizes these steps. Latent reasoning allows for dynamic computational allocation and can handle complex, non-verbalizable cognitive tasks like spatial reasoning.
Architectures Employing Latent Reasoning
My research has revealed several AI architectures that leverage latent reasoning. These include recurrent depth transformers, which can scale computation by iterating a recurrent block in latent space without needing specialized training data. I also explored variational autoencoders (VAEs) and generative adversarial networks (GANs), both of which utilize latent spaces for data compression, generation, and ensuring continuity or diversity in generated outputs. Furthermore, looped transformers demonstrate how iterative processing in latent space can achieve performance comparable to much deeper models, and transformer-based language models show an ability to disentangle and use latent concepts for complex reasoning.
Next Steps: Exploring Python's Pickle Module
Having established a foundational understanding of AI models and latent reasoning, my next step is to investigate Python's `pickle` module. I will focus on its functionalities and common applications for object serialization and deserialization, which is crucial for understanding its synergy with AI models.

Researching websites

Deepening Understanding of AI and Latent Reasoning
I've successfully gathered foundational information on AI models and the concept of latent reasoning. I now have a clearer picture of how AI models operate, from their initial modeling and training phases to the final inference stage. A significant insight is the distinction between explicit 'Chain-of-Thought' reasoning and the more internal, 'latent reasoning' approach, which allows models to refine their understanding silently within a hidden space. This internal processing enables dynamic computation and the handling of complex cognitive patterns that are difficult to verbalize.
Architectures and Applications of Latent Reasoning
My exploration into AI architectures has highlighted how various models leverage latent reasoning. I've seen how recurrent depth transformers can scale computational abilities by iterating within latent space, and how variational autoencoders (VAEs) and generative adversarial networks (GANs) utilize latent spaces for efficient data representation and generation. Furthermore, I've learned that looped transformers can achieve impressive reasoning capabilities with fewer parameters by iteratively processing in latent space, and that transformer-based language models are capable of disentangling and utilizing latent concepts for multi-hop reasoning.
Next Steps: Interfacing with Python's Pickle Module
My immediate next steps involve investigating the specific ways AI models, particularly those that use or generate latent representations, interact with Python's `pickle` module for saving and loading. I will also begin to analyze the security vulnerabilities and potential dangers associated with using `pickle` for serializing AI models, especially considering the complexities introduced by latent reasoning components. Concurrently, I will explore the practical benefits and applications of using `pickle` for the persistence, deployment, and sharing of these models.

Researching websites

Refining Understanding of AI and Latent Reasoning
I've established a solid foundation for understanding AI models and the concept of latent reasoning. I now have a clearer picture of how AI models operate, from their initial modeling and training phases to the final inference stage. A significant insight is the distinction between explicit 'Chain-of-Thought' reasoning and the more internal, 'latent reasoning' approach, which allows models to refine their understanding silently within a hidden space. This internal processing enables dynamic computation and the handling of complex cognitive patterns that are difficult to verbalize.
Architectures and Applications of Latent Reasoning
My exploration into AI architectures has highlighted how various models leverage latent reasoning. I've seen how recurrent depth transformers can scale computational abilities by iterating within latent space, and how variational autoencoders (VAEs) and generative adversarial networks (GANs) utilize latent spaces for efficient data representation and generation. Furthermore, I've learned that looped transformers can achieve impressive reasoning capabilities with fewer parameters by iteratively processing in latent space, and that transformer-based language models are capable of disentangling and utilizing latent concepts for multi-hop reasoning.
Next Steps: Deepening the Pickle Connection and Security
My immediate next steps involve investigating the specific ways AI models, particularly those that use or generate latent representations, interact with Python's `pickle` module for saving and loading. I will also begin to analyze the security vulnerabilities and potential dangers associated with using `pickle` for serializing AI models, especially considering the complexities introduced by latent reasoning components. Concurrently, I will explore the practical benefits and applications of using `pickle` for the persistence, deployment, and sharing of these models, and then move on to best practices and alternative serialization methods.

Researching websites

Understanding AI Models and Latent Reasoning

I've established a foundational understanding of AI models, recognizing them as programs that identify patterns in data for decision-making, involving modeling, training, and inference. A key insight is the concept of 'latent reasoning,' where models internally process and refine thoughts within a 'latent space' before generating output. This approach, unlike 'Chain-of-Thought,' allows for dynamic computation and handles complex, non-verbalizable cognitive tasks.

Architectures and Applications of Latent Reasoning

My research has highlighted how various AI architectures leverage latent reasoning. Recurrent depth transformers scale computation by iterating in latent space, while variational autoencoders (VAEs) and generative adversarial networks (GANs) use latent spaces for data compression and generation. Looped transformers achieve strong reasoning with fewer parameters through iterative latent processing, and transformer-based language models disentangle and utilize latent concepts for multi-hop reasoning.

Next Steps: Deepening the Pickle Connection and Security

My immediate next steps involve a deeper dive into how AI models, especially those with latent representations, interact with Python's `pickle` module for saving and loading. I will specifically investigate the internal serialization mechanisms of popular deep learning frameworks like PyTorch and TensorFlow/Keras. I also plan to explicitly link the 'black box' nature of latent reasoning to the amplification of security risks when using `pickle`, exploring how malicious code might manipulate latent representations. Concurrently, I will elaborate on the practical benefits of `pickle` for these complex models and refine best practices, as well as compare `pickle` with alternative serialization methods.

Researching websites

Serialization of AI Models

I've gained a clearer understanding of how different AI frameworks handle model serialization. PyTorch frequently uses `pickle` internally, often within a zipfile, to save the model's state dictionary and architecture. In contrast, TensorFlow and Keras primarily rely on the SavedModel format (using `protobuf` and `HDF5`) or the H5 format, explicitly advising against direct `pickle` usage due to compatibility concerns. This distinction is crucial for understanding the varying levels of `pickle` exposure across frameworks. For models employing latent reasoning, the parameters defining these latent spaces are implicitly saved as part of the overall model serialization, whether through `pickle` in PyTorch or HDF5 in TensorFlow/Keras.

Latent Reasoning and Security Amplification

I'm synthesizing how the complexity of latent reasoning might amplify `pickle`-related security risks. While `pickle`'s vulnerability to arbitrary code execution is well-known, the 'black box' nature of latent reasoning models could make it significantly harder to detect malicious payloads. If a subtle alteration to the latent space mapping or a backdoor is introduced via `pickle`, designed to activate under specific, rare latent conditions, it would be extremely difficult to identify through standard model evaluation or static analysis. This is a critical area I'm focusing on to understand how the inherent opacity of latent reasoning could be exploited.

Practical Benefits of Pickle for AI Models

I'm also considering the practical advantages of `pickle` for AI models, especially those with latent reasoning. Its ability to save the entire Python object hierarchy, including custom layers, complex data structures, and even internal states of optimization algorithms, offers significant convenience. This is particularly beneficial in research and development for rapid iteration, checkpointing, and sharing complex, custom-built models that might feature unique latent space architectures or custom activation functions.

Next Steps: Deepening Security and Comparing Alternatives

My next steps will involve a deeper investigation into the direct connections between latent space, model complexity, and the difficulty of detecting `pickle`-borne malware or backdoors. I will specifically look for research on adversarial attacks on latent space and covert channels in deep learning models. Additionally, I will prepare a structured comparison of alternative serialization methods like `joblib`, ONNX, HDF5, and Safetensors, evaluating their advantages and disadvantages against `pickle` in terms of security, performance, and interoperability.

Researching websites

Comprehensive Understanding Achieved
I've successfully gathered and synthesized all the necessary information to address the user's request. I now have a complete picture of AI models, including their definitions and the intricacies of latent reasoning, supported by examples from various architectures like VAEs, GANs, and Transformers. I've also thoroughly explored Python's `pickle` module, its functionality, and its common applications in serialization and deserialization.
Synergy and Security of Pickle with AI Models
I've gained a clear understanding of how `pickle` interacts with AI models, noting its internal use in PyTorch for saving model parameters and architecture, while frameworks like TensorFlow/Keras advise against it, preferring HDF5/Protobuf. A critical insight is how the 'black box' nature and complexity of latent reasoning can exacerbate `pickle`'s inherent security risks, making it harder to detect subtle manipulations or malicious code embedded within the model's internal states. I've also detailed the practical benefits of `pickle` for convenience in model persistence and sharing, especially for custom AI models during research and development.
Mitigation and Alternatives
I've also compiled comprehensive best practices for securely using `pickle` with AI models and thoroughly compared it with alternative serialization methods such as `joblib`, ONNX, HDF5, and Safetensors, evaluating them based on security, performance, and interoperability. This allows for a well-rounded understanding of the dangers and potential, along with practical solutions.
Concluding Research
I have reached saturation on all sub-topics and am confident that the gathered information is comprehensive and detailed enough to generate a compelling report.

AI Models, Latent Processing, and Python's Pickle Module: A Critical Analysis of Potential and Peril

Executive Summary

The rapid evolution of Artificial Intelligence (AI) models, particularly the advent of advanced paradigms like latent processing, has revolutionized problem-solving across diverse domains. Python's Pickle module, a ubiquitous tool for object serialization, has played a significant role in enabling the persistence and deployment of these complex AI constructs. This report provides a comprehensive examination of this synergy, highlighting the profound utility that Pickle offers in the AI lifecycle while critically analyzing the severe security vulnerabilities it introduces. The inherent design of

Pickle, coupled with the increasing opacity of latent processing in AI models, creates a fertile ground for sophisticated and often undetectable attacks, posing a substantial risk to the integrity and trustworthiness of the AI supply chain. Robust security practices and the strategic adoption of safer serialization alternatives are imperative to safeguard the future of AI innovation.

1. Introduction: Navigating the Intersection of Advanced AI and Data Persistence

The landscape of artificial intelligence is undergoing a profound transformation, with AI models becoming increasingly sophisticated and integrated into critical infrastructure. This widespread adoption, from predictive analytics to autonomous systems, underscores an escalating demand for efficient and secure methods to manage these complex computational entities. The ability to save, load, and reuse trained AI models is not merely a convenience; it is a foundational requirement for fostering reproducibility, accelerating development cycles, and enabling real-world deployment.

1.1 The AI Revolution and the Need for Model Management

The proliferation of AI models across various industries necessitates robust methods for their storage, retrieval, and deployment. As AI systems become more sophisticated, the ability to preserve their learned state is paramount for reproducibility, efficiency, and real-world application.[1] Model serialization, the process of converting a trained model into a format that can be stored and later reconstructed, is crucial for saving and loading trained models, preserving their weights and architecture, and enabling their reuse, sharing, deployment, and fine-tuning.[1] Loading saved models eliminates the need for repeated retraining, significantly easing their deployment into web applications or APIs, and enhancing scalability across various use cases.[2]

1.2 Overview of Latent Processing and Python's Pickle Module

Within this evolving AI ecosystem, two distinct yet interconnected concepts warrant close examination: latent processing in AI models and Python's Pickle module. Latent processing represents a cutting-edge AI capability where models engage in internal, non-explicit "thinking" before generating an output.[3] This approach promises enhanced efficiency and deeper cognitive abilities. Concurrently, Python's Pickle module serves as a pervasive mechanism for serializing and deserializing Python object structures into byte streams, a capability widely leveraged for model persistence.[4] The intersection of these two elements—advanced AI processing and a fundamental serialization tool—forms a complex synergy that is both immensely promising and inherently precarious.

2. Understanding AI Models and the Power of Latent Processing

To fully appreciate the implications of their interaction with serialization, a foundational understanding of AI models and the advanced concept of latent processing is essential.

2.1 What are AI Models? A Brief Overview

AI models are sophisticated programs designed to detect specific patterns within large datasets, enabling them to draw conclusions or perform actions based on those inferences.[5] Their operation typically involves three fundamental stages:

- **Modeling:** This initial phase involves the development of an artificial intelligence model, which employs complex algorithms or layers of algorithms to analyze data and make informed judgments.[6] A well-designed AI model can effectively mimic human expertise.
- **Training:** In the second stage, the AI model undergoes training, which often entails processing vast quantities of data through recurrent test loops. The results are meticulously inspected to confirm accuracy and ensure the model performs as anticipated. Training methodologies include supervised learning, where models

learn from labeled datasets with pre-existing input-output relationships, and unsupervised learning, where models independently identify patterns in unlabeled data.[6]
- **Inference:** The final stage involves deploying the trained AI model into real-world scenarios. Here, it consistently draws logical inferences from new, unforeseen data inputs, leveraging the patterns and correlations discovered during its training phase to forecast or act upon subsequent information.[6]

AI models find application in a wide array of activities, from image and video recognition to natural language processing (NLP), anomaly detection, recommender systems, predictive modeling, forecasting, and robotics.[6] Machine Learning (ML) and Deep Learning (DL) models are specialized subsets of AI that utilize intricate algorithms and techniques to process and analyze data, generating predictions or decisions in real-time.[6]

2.2 Latent Processing: Thinking in the Model's "Silent Space"

Latent processing represents a significant evolution in AI's cognitive capabilities, moving beyond explicit, step-by-step externalization of thought.

2.2.1 From Chain-of-Thought to Internal Iteration

A novel approach to AI cognition, latent processing enables a model to process and refine its thoughts internally within its "latent space" before producing any observable output.[3] This stands in stark contrast to "Chain-of-Thought" (CoT) methods, which necessitate the model to verbalize its intermediate reasoning steps as explicit tokens in a response.[7] Instead of "thinking out loud," models employing latent processing operate in silence, iterating on potential solutions within their hidden layers.[7] This internal processing often involves iterating a recurrent block to an arbitrary depth during test-time inference, allowing for dynamic and flexible computation.[8]

2.2.2 The Role of Latent Space in AI Cognition

The "latent space" refers to the model's internal, lower-dimensional representation of data, where a substantial portion of its complex processing and refinement takes place.[7] This internal representation facilitates structured, iterative processing that mirrors human cognition, where ideas are often refined internally before being articulated.[7]

Generative models, such as Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs), inherently leverage latent spaces. VAEs, for instance, encode input data into a probabilistic distribution (characterized by mean and variance) within this latent space, from which new data can be sampled and reconstructed.[10] Similarly, GANs utilize a generator component that maps random noise sampled from a latent space into synthetic data that resembles real-world examples.[14]

Furthermore, advancements in transformer models, particularly "looped transformers," demonstrate an ability to implicitly generate internal thoughts and simulate CoT processing through recurrent depth.[9] Recent research indicates that large language models (LLMs) can indeed "think" within their latent space, effectively decoupling their internal cognitive processes from the visible context tokens they generate.[19]

2.3 Key Advantages of Latent Processing in Modern AI

Latent processing offers several compelling advantages for contemporary AI systems. It empowers AI to perform complex, multi-step internal processing without relying on explicit reasoning chains, which can lead to more accurate and nuanced responses.[7] This approach allows for dynamic allocation of computational resources based on the complexity of a given task, thereby ensuring greater efficiency.[7] Models can scale their test-time computation and enhance performance with additional processing, even competing effectively with models possessing significantly more parameters and training data by engaging in deeper "thinking".[8]

A crucial benefit is its capacity to capture intricate cognitive patterns such as spatial reasoning, physical intuition, or abstract mathematical thinking, which are often challenging to express verbally.[7] This enables AI to develop problem-solving mechanisms that extend beyond conventional language-based logic.[7] Unlike CoT models, which demand labeled step-by-step reasoning examples, models employing latent processing can learn to process information without direct supervision on how to break down problems.[7] Additionally, they generally require less memory for both training and inference compared to their CoT counterparts.[8]

The reliance of latent processing models on internal, non-verbalized computation within a "silent space" [7] yields substantial gains in performance and efficiency. This internal processing facilitates more nuanced and complex thought patterns.[7] However, this very "silence" inherently reduces the transparency of the internal processing, making it more challenging to interpret or debug compared to explicit Chain-of-Thought approaches.[7] This presents a fundamental trade-off: while the approach offers increased efficiency and capability, it comes at the expense of interpretability and explainability. This reduced transparency, in turn, renders latent processing a particularly susceptible target for subtle, difficult-to-detect malicious manipulation, as the model's core "thought process" is not readily observable or verifiable.

Furthermore, the foundational research on scaling test-time computation with latent processing explicitly articulates the aspiration that this approach will capture aspects of human cognition that defy verbalization. It encourages models to solve problems by engaging in deeper "thinking," leading to the acquisition of meta-strategies, logic, and abstraction, rather than mere memorization.[8] This signifies a shift from simple pattern recognition to a more generalized and adaptable form of intelligence.[5] If models are indeed learning such generalized strategies within their latent space, then compromising this space could have far-reaching and unpredictable consequences on the model's overall behavior, extending beyond its performance on specific, trained tasks. This makes attacks targeting latent representations exceptionally potent and difficult to contain, as the corruption affects the very essence of the model's learned intelligence.

3. Python's Pickle Module: A Foundational Tool for Object Serialization

Python's Pickle module is a cornerstone in the Python ecosystem for managing object persistence. Understanding its core mechanics is crucial for discerning both its utility and its inherent limitations.

3.1 Core Functionality: Serialization and Deserialization

Pickle is a Python-specific module that implements binary protocols for the serialization and deserialization of Python object structures.[4] "Pickling" refers to the process of converting a Python object hierarchy into a byte stream. This byte stream is a compact, machine-readable format suitable for various purposes, including storage in files or databases, maintaining a program's state across different sessions, or transmitting data over a network.[4] Conversely, "unpickling" is the inverse operation, where a byte stream is converted back into the original Python object hierarchy, restoring its structure and data.[4]

The module provides convenient functions for these operations: pickle.dump() is used to serialize an object and write it directly to a file, while pickle.dumps() returns the serialized data as a byte string. For deserialization, pickle.load() reads a pickled object from a file, and pickle.loads() deserializes an object from a bytes-like object.[23]

3.2 How Pickle Works: Object Reconstruction and the __reduce__ Method

The process of unpickling involves more than just reading raw data; the byte stream contains explicit instructions that guide the unpickler in reconstructing the original object's structure and populating it with the correct data.[4] A notable feature of Pickle is its ability to track objects that have already been serialized. This prevents redundant serialization of shared or recursive objects, ensuring that when the byte stream is unpickled, all references correctly point to the master copy, preserving the original object hierarchy and its internal relationships.[22]

A critical mechanism in Pickle's operation is the __reduce__ method. Python objects can define this special method to declare precisely how they should be pickled.[4] When an object is pickled, its

__reduce__ method is invoked. This method is expected to return a callable object and a tuple of arguments. During unpickling, the Pickle module will then call this specified callable with the provided arguments to construct the new object, effectively rebuilding the original instance.[4]

3.3 Common Applications Beyond AI Models

Beyond its extensive use in AI, Pickle is widely applied for general Python object persistence, particularly in scenarios where other serialization formats, such as JSON, fall short.[23] JSON, for example, struggles with complex Python types like tuples, datetime objects, or custom classes, and cannot preserve the exact state of an object or handle recursive data structures.[23] Pickle, however, can serialize almost every commonly used built-in Python data type and custom objects, retaining their precise state.[23] This versatility makes it a preferred choice for saving program states, creating game save files, or persisting any complex data structures that would be cumbersome to serialize manually.[25]

Pickle's broad compatibility and its capacity to preserve the exact state and internal structure of virtually any Python object, including the complex relationships within recursive data structures and shared objects [22], establish it as an incredibly versatile and convenient tool for Python object persistence. This comprehensive utility, stemming from its design to serialize

behavior—specifically, through the execution of callable objects via the __reduce__ method—alongside raw data [4], creates a paradox. The very design principle that endows Pickle with its "Swiss Army knife" capability for handling complex Python objects is precisely the root cause of its fundamental security vulnerability. This intrinsic link means that the profound utility of Pickle is inextricably tied to its primary danger, forcing developers to actively navigate a deeply ingrained tension between convenience and security in their applications.

4. The Synergy: Pickle's Ubiquitous Role in the AI Model Lifecycle

Python's Pickle module has become deeply embedded within the AI/ML ecosystem, serving as a pervasive tool for saving and loading trained models across various frameworks. Its practical benefits have significantly driven its widespread adoption in development, deployment, and research workflows.

4.1 Saving and Loading Trained AI Models (ML/DL Frameworks)

Pickle is a common and often default format for persisting trained machine learning models across numerous Python libraries. For instance, in Scikit-learn, pickle.dump() and pickle.load() are directly utilized to save and restore models.[28] Deep learning frameworks also leverage Pickle, albeit sometimes indirectly. PyTorch's

torch.save() function, widely used to save model state dictionaries or entire models, implicitly relies on the pickle module internally for serializing metadata and objects.[30] While TensorFlow and Keras primarily employ their own native formats like SavedModel (

.pb) or HDF5 (.h5), there are specific scenarios or workarounds where pickle or joblib (a pickle-based alternative optimized for large NumPy arrays) can be used to save Keras models or their weights.[2] Beyond models, Pickle is also employed for saving and loading training data in deep learning workflows, providing an efficient way to store and retrieve large datasets.[37]

4.2 Practical Benefits for AI Development, Deployment, and Sharing

The widespread adoption of Pickle in the AI community is driven by several practical advantages:

- **Model Persistence and Reuse:** Pickle enables trained models to be saved to disk and reloaded later, eliminating the need to retrain them from scratch for every use.[1] This capability preserves the model's exact state, including all learned variables and configurations, ensuring consistent behavior upon reloading.[41]
- **Ease of Deployment:** Once saved, models can be seamlessly integrated into various applications, such as Flask web applications or other APIs, facilitating real-time inference and prediction in production environments.[1]
- **Scalability:** The ability to load a single trained model across multiple applications or sessions contributes significantly to overall operational efficiency and scalability.[2]

- **Sharing and Collaboration:** Pickled models can be easily shared among researchers, developers, or organizations, fostering collaboration and accelerating knowledge exchange within the AI community.[1]

4.3 Pickle's Utility in Research and Rapid Prototyping of Complex AI Models

The simplicity and convenience offered by Pickle make it particularly appealing for rapid prototyping and iterative development in research environments.[23] It excels at saving custom objects and complex data structures, including bespoke deep learning architectures and their intricate internal states. This allows researchers to save the state of their custom classes and reuse them efficiently, streamlining experimental workflows.[23] For smaller models with fewer parameters, Pickle also offers notably fast saving times.[43] Real-world applications further underscore its utility; companies like Pickle Robot leverage serialization for their physical AI systems, which contributes to faster model iteration cycles and improved precision in robotic grasping accuracy.[44]

The deep integration of Pickle into major AI frameworks like PyTorch and Scikit-learn, coupled with its fundamental ease of use for saving virtually any Python object [23], has led to its pervasive adoption for model persistence, deployment, and rapid prototyping.[1] While this convenience undeniably boosts productivity in research and development, it has inadvertently allowed organizations to accumulate a significant "security debt." This accrual occurs because the widespread reliance on Pickle implicitly accepts a format known for its susceptibility to arbitrary code execution.[4] The pervasive use of Pickle, driven by its undeniable convenience, means that the associated security risks are not isolated incidents but rather represent a systemic vulnerability spanning a vast portion of the Python-based AI ecosystem. This makes the problem considerably more challenging to address retrospectively, often necessitating substantial and costly migration efforts.

Moreover, the ability to easily share and reuse pickled models [1], a key benefit for collaborative AI development, simultaneously creates a significant attack surface. When models are exchanged, particularly on open platforms like Hugging Face [27], the implicit trust often placed in the source of these models can be exploited. Adversaries can capitalize on this by embedding malicious payloads within seemingly benign model files.[49] This situation highlights a critical supply chain security gap where the "product" itself—the AI model—can serve as the primary vector for an attack, rather than merely the underlying infrastructure on which it operates. This transforms the very mechanisms designed to foster innovation and collaboration in AI into potent avenues for adversarial exploitation.

5. The Dangers: Security Vulnerabilities and Exploitation in the AI Supply Chain

Despite its practical advantages, Python's Pickle module introduces severe security implications, particularly when integrated into AI workflows. This section delves into the fundamental flaws and advanced exploitation techniques that pose a significant threat to the AI supply chain.

5.1 Arbitrary Code Execution: The Fundamental Flaw of Pickle

The most critical danger associated with Python's Pickle module is its inherent susceptibility to **arbitrary code execution (RCE)** during the deserialization process.[4] This vulnerability arises from Pickle's design, which allows it to execute instructions embedded within the byte stream to reconstruct an object. This includes the invocation of an object's

__reduce__ method, which can be manipulated by an attacker.[4] A malicious actor can craft a specially designed pickled byte stream that, upon unpickling, forces the execution of arbitrary shell code on the target system.[4]

A significant contributing factor to this risk is the absence of effective built-in mechanisms within the Pickle module to verify the integrity or safety of the incoming byte stream. Consequently, pickle.loads() will attempt to reconstruct whatever object is described in the stream, irrespective of its origin or potentially malicious content.[4] Common attack scenarios include accepting raw pickle data received over an unencrypted network, which allows for data modification in transit, or an attacker gaining access to and modifying stored pickle files in caches, file systems, or databases.[4]

5.2 Advanced Attack Vectors: From "Sleepy Pickle" to Undetectable Malware

The threat landscape extends beyond basic RCE, encompassing sophisticated techniques designed to evade detection and embed persistent malicious capabilities.

5.2.1 Bypassing Security Scanners and Evasion Techniques

Even with the deployment of security tools like Picklescan, which are intended to detect malicious content in pickle files, numerous vulnerabilities have been discovered that allow attackers to circumvent these defenses.[46] Picklescan's reliance on blacklists of "dangerous" functions proves to be an insufficient and non-scalable defense against evolving threats.[46] Attackers can exploit this by embedding secondary, malicious pickle files with non-standard extensions within legitimate model archives (e.g., PyTorch's ZIP-based format). These hidden files often remain undetected by Picklescan but are still successfully loaded by

torch.load().[31] Furthermore, sophisticated ZIP archive manipulation attacks, such as altering filenames in ZIP headers or flipping specific flag bits, can cause scanning tools to crash or fail to detect malicious payloads altogether.[31] The sequential nature of pickle deserialization presents another critical vulnerability: malicious payloads inserted at the very beginning of the stream can execute

before security tools encounter a broken instruction or flag the file as invalid, thereby effectively evading detection.[46]

5.2.2 The Threat of Undetectable Backdoors and Covert Channels in Latent Space

The "Sleepy Pickle" technique exemplifies a new breed of attack that weaponizes the pickle format to directly corrupt the AI model itself. This involves inserting payloads that modify the model in-place—for example, by injecting backdoors, altering control outputs, or tampering with processed data—during the deserialization process, often without triggering conventional security alerts.[49] An even more advanced variant, "Sticky Pickle," incorporates self-replicating mechanisms and obfuscation techniques to ensure persistence and evade detection, even if the compromised model is subsequently modified and redistributed.[49]

Research has demonstrated the alarming possibility of injecting malware into deep neural networks in a manner that is "virtually undetectable." Such embedded malware incurs little to no performance penalty and can generalize across a wide variety of architectures and datasets.[57] This is achieved by embedding malicious code directly within the model's weight parameters.[57] Adversarial attacks can also be generated by injecting subtle perturbations directly into the latent (feature) space of a model, often using generative adversarial networks, which can subtly steer the model's output towards a target class.[60]

In generative models, backdoor attacks can be introduced through poisoned training data, leading to unintended outputs when specific triggers are present. Critically, these triggers can be embedded within intermediate latent representations, making them difficult to trace.[62] While techniques like Spatial Attention Unlearning (SAU) are being developed to leverage latent space manipulation for removing backdoor triggers [62], the threat remains significant. Moreover, deep models can be exploited as "covert channels" to transmit hidden messages by subtly altering artificial data generated by the model or by perturbing global models in federated learning systems.[64] The compact latent representation extracted by generative models can serve as an ideal vehicle for such clandestine communication.[67] While latent space representations are also being explored for enhancing malware detection [68], they simultaneously present a potent target and medium for embedding undetectable malware.[57]

5.3 The Amplification of Risk with Complex AI Architectures

The increasing complexity and inherent "black box" nature of advanced AI models, particularly those that employ latent processing, significantly amplify the difficulty of detecting subtle malicious manipulations.[46] Since latent processing operates "in silence" [7], without producing explicit intermediate steps or verbalized thoughts, it creates an ideal environment for attackers to embed and trigger malicious payloads that are exceedingly difficult to observe or verify through traditional means.[7]

The fundamental arbitrary code execution vulnerability of Pickle [4] is critically exacerbated by the open and collaborative nature of AI development and sharing platforms.[46] Developers, often prioritizing productivity and simplicity, frequently utilize Pickle for models sourced from potentially untrusted origins.[46] This widespread practice creates a fundamental "trust gap" within the AI supply chain, where the convenience of sharing models directly translates into a significant vector for supply chain attacks.[49] The issue extends beyond a mere technical vulnerability in Pickle; it represents a systemic problem rooted in prevailing development practices and the implicit trust model of the AI ecosystem. Addressing this necessitates not only technical solutions but also a fundamental cultural shift towards security-by-design in AI development and distribution.

Furthermore, the internal, non-verbalized processing characteristic of latent processing [7] is a primary driver of AI performance, enabling complex, human-like cognition.[7] However, this very opacity simultaneously creates a substantial "blind spot" for security. Malicious payloads embedded within the latent space [62] or subtle manipulations designed for covert channels [64] become virtually undetectable by current security methodologies.[57] The fact that existing security scanners, such as Picklescan, can be easily bypassed due to their inherent limitations [46] further broadens this blind spot. Compromising the latent space means altering the model's core "thought process" or internal understanding, which can lead to unpredictable and far-reaching behavioral changes that extend well beyond simple misclassification.[7] This signifies that the very features that empower latent processing also render it an ideal target for sophisticated, stealthy attacks. This situation demands a paradigm shift in AI security, moving beyond surface-level code analysis to develop novel techniques for inspecting and verifying the integrity of internal model representations and behaviors. This constitutes a critical and emerging threat to the overall trustworthiness of AI systems.

6. Mitigating Risks: Best Practices and Secure Alternatives for AI Model Serialization

Addressing the inherent risks of Python's Pickle module in AI workflows requires a dual approach: implementing stringent safeguards for its unavoidable use and, more critically, migrating to inherently more secure serialization formats.

6.1 Secure Pickle Implementation: Essential Safeguards

While the pickle module is not inherently insecure, its safe implementation depends entirely on strict adherence to best practices.[4]

- **Avoid Untrusted Sources:** It is paramount to never use pickle with data originating from unknown or untrusted parties, as an untrusted client or server can directly lead to remote code execution.[4]
- **Controlled Environments:** If the use of pickle is unavoidable—for instance, in legacy systems or highly controlled in-house research and development where the environment is fully managed—pickle files should only be loaded within secure, isolated, and sandboxed environments. Such environments must have restricted permissions to minimize the potential impact of arbitrary code execution.[4]
- **Encrypted Connections:** Parties exchanging pickle data must ensure the use of encrypted network connections. This measure is crucial to prevent the alteration or replay of data while it is in transit.[4]
- **Cryptographic Signatures and Checksums:** To verify the integrity of pickled data before loading, cryptographic signatures (e.g., using HMAC) and checksums should be implemented. This helps detect any unauthorized modifications or tampering.[4]
- **File System Permissions:** For any stored pickle data, rigorous review and enforcement of file system permissions are necessary to ensure protected access.[4]
- **Multi-layered Security:** Relying on a single security tool, such as Picklescan, is insufficient to defend against evolving threats. A multi-layered approach is essential, combining static and dynamic analysis tools, utilizing multiple scanning solutions, and actively monitoring for behavioral anomalies during deserialization.[31]

6.2 A Comparative Analysis of Alternative Serialization Formats

Given Pickle's inherent security risks, adopting inherently safer serialization formats is highly recommended for AI models.[31] The landscape of model serialization has evolved, offering specialized formats that address various concerns, including security, performance, and interoperability.

Table 1: Comparison of Key AI Model Serialization Formats

Format	Security (RCE Risk)	Performance (Loading Speed, Memory)	Interoperability	Content Saved	Primary Use Case	Key Pros/Cons	
Pickle (.pkl,.pt /.pth)	**High** (Arbitrary Code Execution) [4]	Average to Slow (large models), High memory usage [23]	Python-specific [72]	Full Python object hierarchy, including behavior [4]	General Python object persistence, ML/DL model saving (legacy/ convenience) [24]	**Pros:** Versatile, handles complex Python types, preserves exact state.[23]	**Cons:** Major security risk, poor interoperability, not designed for large numerical arrays.[72]
Joblib (.joblib,.sav)	**High** (Pickle-based RCE) [74]	Fast for NumPy arrays, Efficient memory mapping, Compression [28]	Python-specific [76]	Python objects, optimized for large NumPy arrays [74]	Scikit-learn models, large numerical data [28]	**Pros:** Faster than Pickle for large NumPy arrays, built-in compression.[76]	**Cons:** Still vulnerable to RCE, less general than Pickle.[74]
Safetensors (.safetensors)	**None** (Secure by Design) [27]	Fast (zero-copy loading, mmap), Reduced memory usage [27]	Framework-agnostic, Portable (multi-language) [27]	Numerical tensors and metadata only [27]	Secure sharing of model weights, Hugging Face default [27]	**Pros:** Eliminates RCE risk, significantly faster loading, memory	**Cons:** Architecture must be defined separately, less flexible quantiza

						efficient. [27]	tion than GGUF.[72]
ONNX (.onnx)	**Low** (Protobuf-based, no arbitrary code execution) [72]	Average (conversion can incur loss) [72]	Cross-framework, Cross-platform (mobile, edge) [42]	Model architecture and weights (computation graph) [72]	Cross-framework deployment, production environments, mobile/edge inference [42]	**Pros:** Vendor-neutral, high portability, includes computation graph.[72]	**Cons:** Limited native quantized tensor support, complex architectures may need rewrites. [72]
HDF5 (.h5)	**Low** (Data-focused, not code execution) [77]	Fast for saving numerical data, Average for loading [78]	Good (structured numerical data) [78]	Large multidimensional NumPy arrays (weights), model architecture, optimizer state (Keras) [36]	Deep Learning model weights/checkpoints (Keras/TF), large numerical datasets [36]	**Pros:** Efficient for large numerical arrays, built-in compression, supports checkpointing.[77]	**Cons:** Being replaced by SavedModel, less efficient for very large models.[73]
TensorFlow SavedModel (.pb)	**Low** (Protobuf-based) [79]	Can be large, but optimized for TF ecosystem [41]	TensorFlow-specific, but deployable [79]	Full model (architecture, weights, optimizer state, traced subgraphs) [36]	Standard TensorFlow model saving/loading, deployment [36]	**Pros:** Comprehensive, recommended for TF, enables resuming training. [36]	**Cons:** Can be very large, potential compatibility issues across TF versions. [41]
GGUF (.gguf)	**Low** (Binary,	Fast (mmap),	llama.cpp	Weights, tokenize	Efficient inferenc	**Pros:** Highly	**Cons:** Not for

	data-focused) [72]	Efficient (quantization) [72]	ecosystem specific [72]	r, metadata (quantization-friendly) [72]	e of LLMs on CPU/GPU, resource-limited environments [72]	efficient for LLM inference, flexible quantization, single-file.[72]	training/fine-tuning, conversion often required, ecosystem-specific.[72]
TFLite (.tflite)	**Low** (FlatBuffer format) [73]	Low-latency, optimized for edge [73]	Edge/Mobile devices (Android, embedded) [73]	Highly compressed model structure [73]	On-device inference, embedded systems [73]	**Pros:** Highly compressed, low-latency, hardware acceleration.[73]	**Cons:** Not all TF operations supported, difficult debugging.[73]
Core ML (.mlmodel)	**Low** (Protobuf structure) [73]	Optimized for Apple Silicon [73]	Apple ecosystem (iOS, macOS, watchOS) [73]	Model specifications, inputs/outputs, metadata [73]	Deploying ML models on Apple devices [73]	**Pros:** Seamless integration with Apple apps, optimized for Apple hardware.[73]	**Cons:** Restricted to Apple platforms, requires conversion tools.[73]
TensorRT (.engine)	**Low** (Optimized runtime format) [73]	Ultra-fast inference [73]	NVIDIA GPUs, specific hardware/version [73]	Optimized execution graph [73]	Production deployment on NVIDIA hardware, real-time inference [73]	**Pros:** Highest performance on NVIDIA GPUs, supports precision modes.[73]	**Cons:** Hardware/version specific, not for training.[73]

The development and widespread adoption of formats like Safetensors [27], explicitly designed to counteract Pickle's arbitrary code execution vulnerability [27], signify a maturing understanding of AI supply chain security risks. This evolution represents a direct industry response to the perils identified. The heightened emphasis on model integrity verification [71] and meticulous provenance tracking [82] further indicates a strategic shift from reactive security patching to proactive, systemic security measures across the AI lifecycle. This progression suggests that the AI community is increasingly prioritizing robust security postures, moving towards more secure defaults and practices. However, this transition is not instantaneous, and organizations must actively engage in this shift by migrating away from insecure legacy practices and investing in tools and processes that align with these new security paradigms.

Moreover, effectively mitigating the risks associated with Pickle and, more broadly, ensuring comprehensive AI model security, demands more than purely technical solutions like adopting Safetensors or implementing cryptographic signatures. It also necessitates robust governance frameworks. This includes establishing clear AI-specific security policies, continuously monitoring open-source registries for emerging threats, and fostering collaborative efforts within the broader open-source community.[31] This multi-faceted requirement underscores that effective AI security is a complex challenge that cannot be resolved by technology alone. For organizations, securing AI models is not solely the purview of technical teams; it mandates a holistic approach involving policy-makers, legal experts, and cross-functional collaboration to define acceptable risk levels, enforce secure practices, and respond dynamically to the evolving threat landscape. This comprehensive view acknowledges the intricate interplay between technical controls and organizational governance in safeguarding AI systems.

6.3 Strategies for Model Integrity Verification and Reproducibility

Beyond selecting secure serialization formats, robust strategies are critical to ensure the integrity, trustworthiness, and reproducibility of AI models throughout their entire lifecycle.[71]

- **Checksums and Hashing:** Calculate and compare cryptographic hash values of models immediately after training and prior to each execution. This provides a digital fingerprint that can detect any unauthorized modifications or tampering.[71]

- **Watermarking:** Unique signatures or watermarks can be embedded directly into models. These watermarks can then be checked at various stages to validate the model's authenticity and ensure it has not been illicitly copied or altered.[71]
- **Runtime Behavior Analysis:** Continuous monitoring of a model's behavior during runtime can reveal anomalies or deviations from its expected operational patterns. Such deviations can serve as early indicators of potential integrity breaches or malicious activity.[71]
- **Provenance Tracking:** Maintaining detailed, immutable logs of all model interactions, updates, and changes is essential. This comprehensive lineage allows for tracing back any potential compromises and is fundamental for ensuring the reproducibility of results, a cornerstone of scientific and engineering rigor in AI.[71] This includes robust versioning of both data and models, allowing for precise rollback to any prior state.[82]
- **Isolation:** Deploying AI models within isolated and secure environments is a critical defense mechanism. This practice prevents cascading failures and limits the potential scope of an attack if a compromise were to occur.[71]
- **Regular Audits and Frequent Checks:** Instituting regular and frequent integrity checks, particularly in high-risk or high-value environments, is a proactive measure to detect and mitigate threats before they can cause significant harm.[71]

7. Conclusion: Securing the Future of AI Innovation

The analysis underscores a critical duality in the relationship between AI models, latent processing, and Python's Pickle module. While Pickle's inherent versatility and ease of use have undeniably fostered rapid innovation and deployment within the AI ecosystem, its fundamental design flaw—the capacity for arbitrary code execution during deserialization—introduces a profound and unacceptable security risk. This risk is further amplified by the increasing adoption of latent processing in advanced AI architectures, which, by their very nature of internal, non-verbalized computation, create an opaque environment that can conceal sophisticated and virtually undetectable malicious payloads.

The convenience offered by Pickle has, for too long, overshadowed its inherent peril, leading to a widespread accumulation of security vulnerabilities across the AI supply chain. The open exchange of AI models, a cornerstone of collaborative progress, inadvertently transforms these models into potent vectors for attack. The "silent thinking" of latent processing, while a leap forward in AI capability, simultaneously creates an adversarial "blind spot," making it exceedingly difficult to detect deep-seated compromises. Safeguarding the integrity and trustworthiness of AI systems in this evolving landscape is not merely a technical challenge but a strategic imperative.

8. Recommendations for AI Practitioners and Organizations

To navigate the complex interplay of utility and danger, and to secure the future of AI innovation, the following actionable recommendations are provided for AI practitioners, MLOps teams, and organizational leadership:

- **Prioritize Secure Serialization Formats:**
  - **Strongly recommend** the adoption of inherently safer formats as the default for model persistence and sharing. For models primarily comprising numerical tensors, Safetensors is the superior choice due as it prevents arbitrary code execution.[27] For scenarios requiring cross-framework portability, ONNX is highly recommended.[42] This is especially critical for models downloaded from public repositories or exchanged between parties without established trust.
  - For TensorFlow models, leverage the SavedModel format, which is comprehensive and secure within its ecosystem.[36] For PyTorch, consider TorchScript for production deployment where feasible, as it offers a more secure alternative to raw pickle-based saving.[31]
- **Implement Strict Pickle Usage Policies:**
  - If the use of Pickle is absolutely necessary (e.g., for compatibility with legacy systems or in highly controlled, in-house research and development environments where complete control over the execution environment is guaranteed), enforce its use **only within strictly controlled, isolated, and sandboxed environments**.[4]
  - Establish an unequivocal policy to **never unpickle data from untrusted or unverified sources.**[4] This is the single most critical safeguard.
- **Enhance Model Integrity Verification:**
  - Implement robust cryptographic signing and checksum verification for all AI models before they are loaded into production environments. This provides a

verifiable chain of trust and detects any unauthorized modifications.[4]
    - Establish comprehensive provenance tracking and version control systems for both models and their associated data. This ensures full traceability and reproducibility, crucial for debugging and security audits.[71]
    - Invest in advanced AI security tools capable of detecting subtle manipulations, including those specifically targeting latent representations. This requires moving beyond simplistic blacklist-based scanners to more sophisticated behavioral and structural analysis.[46]
- **Foster a Security-First AI Culture:**
    - Educate developers and data scientists on the inherent risks of Pickle and the paramount importance of secure serialization practices. This includes training on alternative formats and their secure implementation.
    - Integrate AI security policies and automated scanning into the Continuous Integration/Continuous Deployment (CI/CD) pipeline (DevSecOps). This ensures security checks are an integral part of the development workflow, rather than an afterthought.[31]
    - Actively monitor open-source registries and threat intelligence feeds for new vulnerabilities, malicious models, and emerging attack techniques.[31]
    - Contribute to and collaborate with the open-source AI security community to share knowledge and strengthen collective defenses against evolving threats.[31]
- **Consider Broader Implications:**
    - Acknowledge that undetectable malware and covert channels embedded within AI models, particularly those leveraging the opaque nature of latent processing, represent a significant and evolving threat.[57] Organizations must proactively prepare for scenarios where model integrity is compromised at a deeper, less visible level.
    - Recognize that the "black box" characteristic of latent processing amplifies these risks. This necessitates ongoing research and development into innovative approaches for AI model explainability and advanced security auditing techniques that can peer into the internal workings of these complex systems.

Referências citadas

1. What is the purpose of model serialization in deep learning? - Massed Compute, acessado em junho 15, 2025, https://massedcompute.com/faq-answers/?question=What+is+the+purpose+of+model+serialization+in+deep+learning%3F
2. Loading the Saved Model in Python - Meritshot, acessado em junho 15, 2025,

https://www.meritshot.com/loading-the-saved-model-in-python/
3. ajithp.com, acessado em junho 15, 2025, https://ajithp.com/2025/02/14/latent-reasoning-the-next-evolution-in-ai-for-scalable-adaptive-and-efficient-problem-solving/#:~:text=Latent%20reasoning%20is%20a%20new,internally%20before%20generating%20any%20output.
4. Python pickling: What it is and how to use it securely | Black Duck Blog, acessado em junho 15, 2025, https://www.blackduck.com/blog/python-pickling.html
5. www.hpe.com, acessado em junho 15, 2025, https://www.hpe.com/us/en/what-is/ai-models.html#:~:text=AI%20models%20or%20artificial%20intelligence,actions%20depending%20on%20those%20conclusions.
6. What are AI Models? | Glossary | HPE, acessado em junho 15, 2025, https://www.hpe.com/us/en/what-is/ai-models.html
7. Latent Reasoning: The Next Evolution in AI for Scalable, Adaptive, and Efficient Problem-Solving - Ajith's AI Pulse, acessado em junho 15, 2025, https://ajithp.com/2025/02/14/latent-reasoning-the-next-evolution-in-ai-for-scalable-adaptive-and-efficient-problem-solving/
8. 1 Scaling by Thinking in Continuous Space - arXiv, acessado em junho 15, 2025, https://arxiv.org/html/2502.05171v1
9. [2502.17416] Reasoning with Latent Thoughts: On the Power of Looped Transformers, acessado em junho 15, 2025, https://arxiv.org/abs/2502.17416
10. Difference between AutoEncoder (AE) and Variational AutoEncoder (VAE), acessado em junho 15, 2025, https://towardsdatascience.com/difference-between-autoencoder-ae-and-variational-autoencoder-vae-ed7be1c038f2/
11. Tutorial - What is a variational autoencoder? - Jaan Lı 李, acessado em junho 15, 2025, https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
12. Variational Autoencoders: How They Work and Why They Matter - DataCamp, acessado em junho 15, 2025, https://www.datacamp.com/tutorial/variational-autoencoders
13. Variational AutoEncoders - GeeksforGeeks, acessado em junho 15, 2025, https://www.geeksforgeeks.org/variational-autoencoders/
14. Improving Generative Adversarial Networks via Adversarial Learning in Latent Space - NIPS, acessado em junho 15, 2025, https://proceedings.neurips.cc/paper_files/paper/2022/file/3a1fc7b8e200a45110872c56f0569f61-Paper-Conference.pdf
15. Generative Adversarial Networks for Text : r/MachineLearning - Reddit, acessado em junho 15, 2025, https://www.reddit.com/r/MachineLearning/comments/40ldq6/generative_adversarial_networks_for_text/
16. Generative Adversarial Network (GAN) - GeeksforGeeks, acessado em junho 15, 2025, https://www.geeksforgeeks.org/generative-adversarial-network-gan/
17. Generative Adversarial Networks: Build Your First Models - Real Python, acessado em junho 15, 2025, https://realpython.com/generative-adversarial-networks/
18. Latent Concept Disentanglement in Transformer-based Language Models -

OpenReview, acessado em junho 15, 2025, https://openreview.net/forum?id=c1Qw6uDxpS
19. New paper gives models a chance to think in latent space before outputting tokens, weights are already on HF - Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach - Reddit, acessado em junho 15, 2025, https://www.reddit.com/r/LocalLLaMA/comments/1imca0s/new_paper_gives_models_a_chance_to_think_in/
20. A new paper demonstrates that LLMs could "think" in latent space, effectively decoupling internal reasoning from visible context tokens. This breakthrough suggests that even smaller models can achieve remarkable performance without relying on extensive context windows. : r/LocalLLaMA - Reddit, acessado em junho 15, 2025, https://www.reddit.com/r/LocalLLaMA/comments/1inch7r/a_new_paper_demonstrates_that_llms_could_think_in/
21. A new paper demonstrates that LLMs could "think" in latent space, effectively decoupling internal reasoning from visible context tokens. This breakthrough suggests that even smaller models can achieve remarkable performance without relying on extensive context windows. : r/singularity - Reddit, acessado em junho 15, 2025, https://www.reddit.com/r/singularity/comments/1inh7lt/a_new_paper_demonstrates_that_llms_could_think_in/
22. pickle — Python object serialization — Python 3.13.5 documentation, acessado em junho 15, 2025, https://docs.python.org/3/library/pickle.html
23. Python Pickle Tutorial: Object Serialization - DataCamp, acessado em junho 15, 2025, https://www.datacamp.com/tutorial/pickle-python-tutorial
24. Python Pickle: A Comprehensive Guide to Object Serialization - Analytics Vidhya, acessado em junho 15, 2025, https://www.analyticsvidhya.com/blog/2024/01/python-pickle-a-comprehensive-guide-to-object-serialization/
25. Pickle - Learn Data Science with Travis - your AI-powered tutor - AIgents, acessado em junho 15, 2025, https://aigents.co/learn/Pickle
26. Purpose of pickle : r/learnpython - Reddit, acessado em junho 15, 2025, https://www.reddit.com/r/learnpython/comments/4nhjap/purpose_of_pickle/
27. Understanding SafeTensors: A Secure Alternative to Pickle for ML ..., acessado em junho 15, 2025, https://dev.to/lukehinds/understanding-safetensors-a-secure-alternative-to-pickle-for-ml-models-o71
28. How to use pickle to save sklearn model - python - Stack Overflow, acessado em junho 15, 2025, https://stackoverflow.com/questions/54879434/how-to-use-pickle-to-save-sklearn-model
29. Model Serialization using pickle and joblib - Kaggle, acessado em junho 15, 2025, https://www.kaggle.com/code/tasnimniger/model-serialization-using-pickle-and-joblib
30. Save and Load the Model — PyTorch Tutorials 2.7.0+cu126 documentation,

acessado em junho 15, 2025, https://docs.pytorch.org/tutorials/beginner/basics/saveloadrun_tutorial.html
31. Unpickling Pytorch: Keeping Malicious AI Out | Sonatype Whitepaper, acessado em junho 15, 2025, https://www.sonatype.com/resources/whitepapers/unpickling-pytorch
32. torch.save — PyTorch 2.7 documentation, acessado em junho 15, 2025, https://docs.pytorch.org/docs/stable/generated/torch.save.html
33. torch.save — PyTorch 2.7 documentation, acessado em junho 15, 2025, https://pytorch.org/docs/stable/generated/torch.save.html
34. save and load keras transformer model - python - Stack Overflow, acessado em junho 15, 2025, https://stackoverflow.com/questions/78396871/save-and-load-keras-transformer-model
35. how to save tensorflow model to pickle file - python - Stack Overflow, acessado em junho 15, 2025, https://stackoverflow.com/questions/71492778/how-to-save-tensorflow-model-to-pickle-file
36. Save and load models in Tensorflow - GeeksforGeeks, acessado em junho 15, 2025, https://www.geeksforgeeks.org/save-and-load-models-in-tensorflow/
37. What is the purpose of using the "pickle" library in deep learning and how can you save and load training data using it? - EITCA Academy, acessado em junho 15, 2025, https://eitca.org/artificial-intelligence/eitc-ai-dlptfk-deep-learning-with-python-tensorflow-and-keras/data/loading-in-your-own-data/examination-review-loading-in-your-own-data/what-is-the-purpose-of-using-the-pickle-library-in-deep-learning-and-how-can-you-save-and-load-training-data-using-it/
38. Use Keras with Metaflow | Outerbounds, acessado em junho 15, 2025, https://docs.outerbounds.com/use-keras-with-metaflow/
39. How to pickle Keras model? - python - Stack Overflow, acessado em junho 15, 2025, https://stackoverflow.com/questions/48295661/how-to-pickle-keras-model
40. Creating the initial model - IBM, acessado em junho 15, 2025, https://www.ibm.com/docs/en/watsonx/saas?topic=experiment-creating-initial-model
41. How to Save Trained Model in Python - neptune.ai, acessado em junho 15, 2025, https://neptune.ai/blog/saving-trained-model-in-python
42. Deploy Machine Learning Models on Edge Devices | Moez Ali, acessado em junho 15, 2025, https://www.moez.ai/2021/05/19/deploy-machine-learning-models-on-edge-devices/
43. Save Machine Learning Model Using Pickle and Joblib - Analytics Vidhya, acessado em junho 15, 2025, https://www.analyticsvidhya.com/blog/2021/08/quick-hacks-to-save-machine-learning-model-using-pickle-and-joblib/
44. How Pickle Robot Is Using Encord To Build Physical AI For Warehouse Automation, acessado em junho 15, 2025, https://encord.com/customers/pickle-robot/

45. Technology - Pickle Robot, acessado em junho 15, 2025, https://picklerobot.com/technology
46. Malicious ML models discovered on Hugging Face platform, acessado em junho 15, 2025, https://www.reversinglabs.com/blog/rl-identifies-malware-ml-model-hosted-on-hugging-face
47. Bypassing picklescan: Sonatype discovers four vulnerabilities, acessado em junho 15, 2025, https://www.sonatype.com/blog/bypassing-picklescan-sonatype-discovers-four-vulnerabilities
48. ModelScan - Protection Against Model Serialization Attacks - SANS Internet Storm Center, acessado em junho 15, 2025, https://isc.sans.edu/diary/31692
49. New Attack Technique 'Sleepy Pickle' Targets Machine Learning Models - The Hacker News, acessado em junho 15, 2025, https://thehackernews.com/2024/06/new-attack-technique-sleepy-pickle.html
50. IA x pickle: vulnerability or feature? - GLIMPS, acessado em junho 15, 2025, https://www.glimps.re/en/resource/protectiong-ai-models/
51. PAIT-PKL-100: Pickle Model Arbitrary Code Execution Detected at Model Load Time - Protect AI, acessado em junho 15, 2025, https://protectai.com/insights/knowledge-base/deserialization-threats/PAIT-PKL-100
52. Never a dill moment: Exploiting machine learning pickle files - The Trail of Bits Blog, acessado em junho 15, 2025, https://blog.trailofbits.com/2021/03/15/never-a-dill-moment-exploiting-machine-learning-pickle-files/
53. Enhancing cloud security in AI/ML: The little pickle story - AWS, acessado em junho 15, 2025, https://aws.amazon.com/blogs/security/enhancing-cloud-security-in-ai-ml-the-little-pickle-story/
54. Why do we need Hugging Face's SafeTensor?, acessado em junho 15, 2025, https://franklee.xyz/blogs/2024-10-19-safetensor
55. AI security defenses potentially circumvented via picklescan flaws - SC Media, acessado em junho 15, 2025, https://www.scworld.com/brief/ai-security-defenses-potentially-circumvented-via-picklescan-flaws
56. Zero-Trust Artificial Intelligence Model Security Based on Moving Target Defense and Content Disarm and Reconstruction - arXiv, acessado em junho 15, 2025, https://arxiv.org/html/2503.01758v1
57. A new security threat to AI models - SRI, acessado em junho 15, 2025, https://www.sri.com/press/story/a-new-security-threat-to-ai-models/
58. Planting Undetectable Backdoors in Machine Learning Models : [Extended Abstract] | Request PDF - ResearchGate, acessado em junho 15, 2025, https://www.researchgate.net/publication/366665435_Planting_Undetectable_Backdoors_in_Machine_Learning_Models_Extended_Abstract
59. Backdoor Attack Against One-Class Sequential Anomaly Detection Models,

acessado em junho 15, 2025, https://par.nsf.gov/servlets/purl/10579798
60. Generating Adversarial Attacks in the Latent Space - CVF Open Access, acessado em junho 15, 2025, https://openaccess.thecvf.com/content/CVPR2023W/GCV/papers/Shukla_Generating_Adversarial_Attacks_in_the_Latent_Space_CVPRW_2023_paper.pdf
61. Graph Attacks with Latent Variable Noise Modeling, acessado em junho 15, 2025, https://grlearning.github.io/papers/37.pdf
62. Backdoor Defense in Diffusion Models via Spatial Attention Unlearning - arXiv, acessado em junho 15, 2025, https://arxiv.org/html/2504.18563v1
63. [2504.18563] Backdoor Defense in Diffusion Models via Spatial Attention Unlearning - arXiv, acessado em junho 15, 2025, https://arxiv.org/abs/2504.18563
64. Generative AI-driven Cross-layer Covert Communication: Fundamentals, Framework and Case Study - arXiv, acessado em junho 15, 2025, https://arxiv.org/html/2501.11068v1
65. Covert Channels Detection with Supported Vector Machine and Hyperbolic Hopfield Neural Network - SciSpace, acessado em junho 15, 2025, https://scispace.com/pdf/covert-channels-detection-with-supported-vector-machine-and-3jj64hr98e.pdf
66. Exploiting Deep Neural Networks as Covert Channels - IEEE Computer Society, acessado em junho 15, 2025, https://www.computer.org/csdl/journal/tq/2024/04/10197462/1Pezs6WNhbW
67. Generative modelling in latent space - Sander Dieleman, acessado em junho 15, 2025, https://sander.ai/2025/04/15/latents.html
68. [2503.20803] Leveraging VAE-Derived Latent Spaces for Enhanced Malware Detection with Machine Learning Classifiers - arXiv, acessado em junho 15, 2025, https://arxiv.org/abs/2503.20803
69. Leveraging VAE-Derived Latent Spaces for Enhanced Malware Detection with Machine Learning Classifiers - arXiv, acessado em junho 15, 2025, https://arxiv.org/html/2503.20803v1
70. Deploying Machine Learning model in production | CloudxLab Blog, acessado em junho 15, 2025, https://cloudxlab.com/blog/deploying-machine-learning-model-in-production/
71. Model Integrity Verification: The Essential Guide | Nightfall AI Security 101, acessado em junho 15, 2025, https://www.nightfall.ai/ai-security-101/model-integrity-verification
72. Common AI Model Formats - Hugging Face, acessado em junho 15, 2025, https://huggingface.co/blog/ngxson/common-ai-model-formats
73. Model Weights File Formats in Machine Learning - LearnOpenCV, acessado em junho 15, 2025, https://learnopencv.com/model-weights-file-formats-in-machine-learning/
74. 10. Model persistence — scikit-learn 1.7.0 documentation, acessado em junho 15, 2025, https://scikit-learn.org/stable/model_persistence.html
75. A Large-Scale Exploit Instrumentation Study of AI/ML Supply Chain Attacks in Hugging Face Models - arXiv, acessado em junho 15, 2025, https://arxiv.org/pdf/2410.04490

76. Joblib vs Pickle: Choosing the Right Serialization Tool in Python - Planmyleave, acessado em junho 15, 2025, https://www.planmyleave.com/BlogDetail/255/joblib-vs-pickle-choosing-the-right-serialization-tool-in-python/0/Blog
77. Storage: HDF, Pickle, Postgres | Machine Learning Podcast - OCDevel, acessado em junho 15, 2025, https://ocdevel.com/mlg/mla-3
78. Which Data Format to Use For Your Big Data Project? | Towards Data Science, acessado em junho 15, 2025, https://towardsdatascience.com/which-data-format-to-use-for-your-big-data-project-837a48d3661d/
79. Save and load Keras models - Colab - Google, acessado em junho 15, 2025, https://colab.research.google.com/github/tensorflow/docs/blob/snapshot-keras/site/en/guide/keras/save_and_serialize.ipynb?hl=zh-tw
80. Save, serialize, and export models | TensorFlow Core, acessado em junho 15, 2025, https://www.tensorflow.org/guide/keras/save_and_serialize
81. AI Model Evaluation In AI Security - Meegle, acessado em junho 15, 2025, https://www.meegle.com/en_us/topics/ai-model-evaluation/ai-model-evaluation-in-ai-security
82. Versioning & Reproducibility - LanceDB Enterprise, acessado em junho 15, 2025, https://docs.lancedb.com/core/versioning
83. ML Data Version Control and Reproducibility at Scale - lakeFS, acessado em junho 15, 2025, https://lakefs.io/blog/scalable-ml-data-version-control-and-reproducibility/
84. maximveksler/awesome-serialization: Data formats useful for API, Big Data, ML, Graph & co, acessado em junho 15, 2025, https://github.com/maximveksler/awesome-serialization