National Cyber Security Centre
a part of GCHQ

Principles for the security of machine learning

Version 1

Published August 2022

Contents

Introduction

Alongside 'traditional' cyber attacks, the use of artificial intelligence (AI) and machine learning (ML) leaves systems vulnerable to new types of attack that exploit underlying information processing algorithms and workflows. All aspects of an AI or ML system's security are potentially vulnerable, and compromises of system confidentiality, integrity and availability have all been previously observed. Without properly understanding and mitigating these vulnerabilities, system designers can't assure stakeholders, and ultimately users, that an AI system is safe and secure.

Who are they for and how were they developed?

These principles aim to be wide reaching and applicable to anyone developing, deploying or operating a system with a machine learning (ML) component. They are not a comprehensive assurance framework to grade a system or workflow, and do not provide a checklist. Instead, they provide context and structure to help scientists, engineers, decision makers and risk owners make educated decisions about system design and development processes, helping to assess the specific threats to a system.

There is considerable academic research on model strengthening and defences against adversaries' ML techniques, but in many cases they may be imperfect, or untested outside of the research domain. These principles therefore focus on pragmatic ways to address ML vulnerabilities.

Instead of focusing on a specific system or attack type, these principles have been developed by 'red teaming' a generic ML workflow, as the diagram on the following page shows. This allows the principles to cover a wide range of systems that are agnostic to data type, model algorithm or deployment environment. These principles therefore apply to a range of ML algorithms.

(Please note that we do not cover reinforcement learning (RL) because of its unique development workflow. When undertaking RL development, further security considerations may be necessary, although these principles may provide a foundation).

These principles should be considered alongside cyber security best practice for conventional software development.

Applying the principles

When applying the principles, you should assess the level of risk and consequence of compromise for your specific application, and then apply the principles and corresponding defences proportionately. The principles are designed to convey the underlying possible threat to the reader. Applying them to a specific context will help you consider risk and mitigation in line with your organisation's risk appetite, carried out by suitably qualified and experienced staff in your organisation.

A useful place to start is to familiarise yourself with attack techniques that exploit the inherent vulnerabilities in ML algorithms and development processes, and where they fit into the lifecycle outlined below. The security of machine learning is an active area of research which means an ever-changing threat landscape, however some attacks specific to ML processes are outlined Microsoft's blog on Failure Modes in Machine Learning and in the MITRE ATLAS knowledge base.

What do the principles look like?

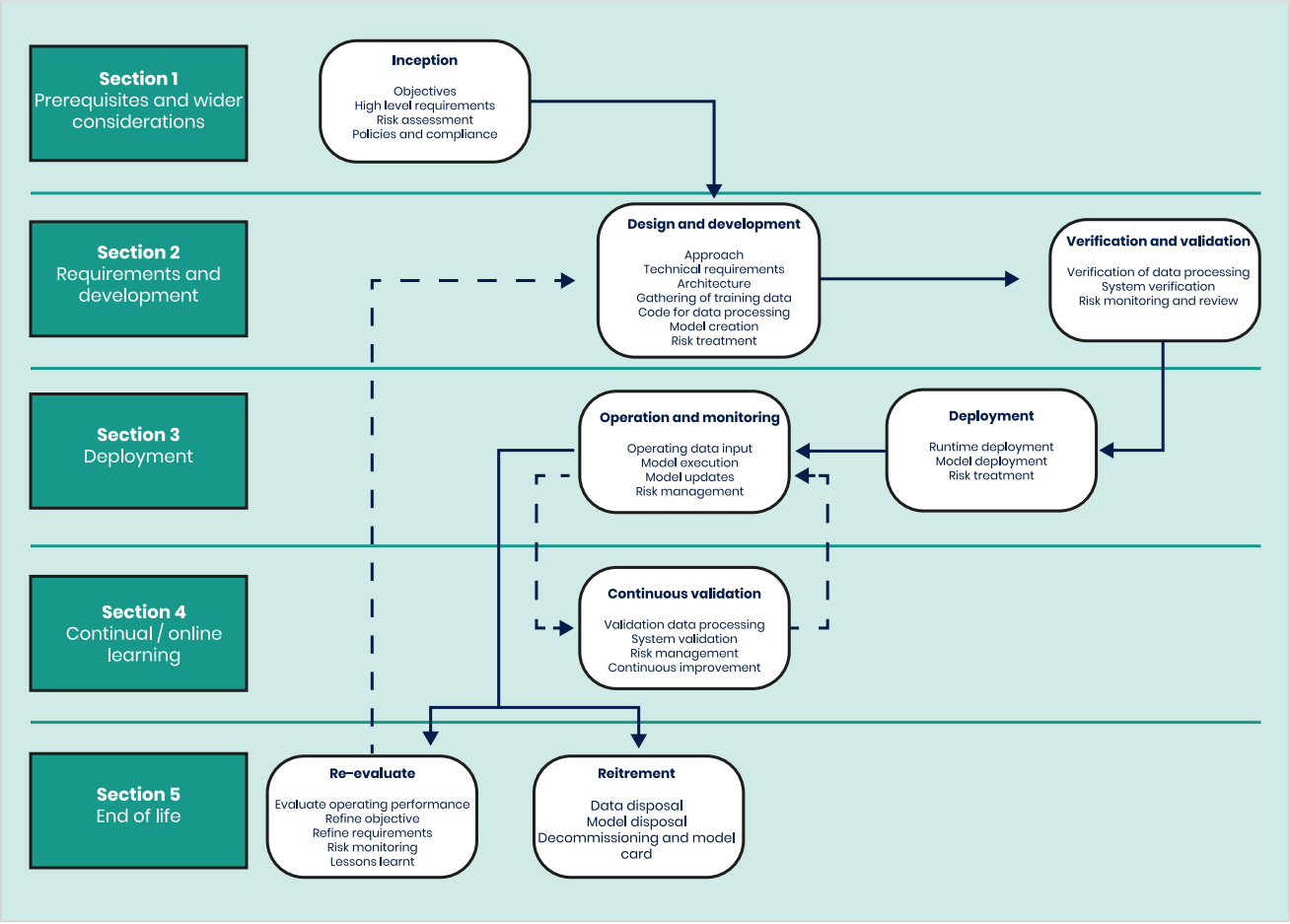
Each of the principles answers three questions:

1. **Why this principle?** A description of the context and motivation for this principle
2. **What are the goals of the principle?** Concrete objectives to consider when addressing the principle
3. **How could this principle be implemented?** Includes links to guidance and further information *(as the ML field is fast moving, this section is advisory rather than comprehensive)*

How are they structured?

To help users working on specific parts of the process and since many of the vulnerabilities addressed in the principles appear at specific points, the principles are structured around a typical ML lifecycle. But several themes run through the principles and are applicable throughout the lifecycle. They are:

- enable your developers
- design for security
- minimise an adversary's knowledge
- secure your supply chain
- secure your infrastructure
- track your assets
- enable your developers



Quick reference table

Although the principles are designed to be used as a single guidance document, the table below may be a useful quick reference to the most relevant sections for those in specific roles:

I am an: **ML practitioner** Refer to principles...	I am a: **Senior decision maker** Refer to principles...	I am an: **IT security professional** Refer to principles...
1.1 Be aware of ML security vulnerabilities and model failure modes 1.2 Consider failure modes 1.3 Understand what attackers can do with information about your system	1.1 Be aware of ML security vulnerabilities and model failure modes 1.3 Understand what attackers can do with information about your system	1.1 Be aware of ML security vulnerabilities and model failure modes 1.2 Consider failure modes 1.3 Understand what attackers can do with information about your system
2.1 Consider your system's vulnerability to known ML threats 2.2 Consider digital and physical supply chains risk 2.5 Document the creation, operation and lifecycle management of assets 2.6 Ensure your model capacity is proportionate to your requirements	2.2 and 2.3 Consider digital and physical supply chains risk 2.4 Secure your infrastructure	2.1 Consider the vulnerability of your system to known ML threats 2.2 and 2.3 Consider digital and physical supply chains risk 2.4 Secure your infrastructure 2.5 Document the creation, operation and lifecycle management of assets
3.2 Monitor and log your users' inference requests and queries 3.3 Understand the trade-off between security during operation and the output a model provides to users	3.2 Monitor and log your users' inference requests/queries	3.1 Limit access to your model 3.2 Monitor and log your users' inference requests and queries
4.1 Understand what continual learning is and the potential risks of using it 4.2 Track and test continual learning models		4.1 Understand what continual learning is and the potential risks of using it
5.1 Decommission your assets appropriately 5.2 Collate lessons learned and share	5.2 Collate lessons learned and share	5.1 Decommission your assets appropriately 5.2 Collate lessons learned and share

Development prerequisites and wider considerations

Once you've decided to develop an ML system, there are security questions you need to consider. Many decisions at this stage will be driven by scope: do you intend your model to be operational, or for research purposes only? If the system fails or comes under attack, what are the implications for the wider system in which the ML component will sit? The answers will give you an early indication of your appetite for security risk. This should be shared across your team, to make sure you have people with the right range of skills to develop the model and assess risks appropriately.

Enable your developers

Are your data scientists and developers aware of ML security threats and model failure modes?

Why this principle?

Security by design is a key principle in traditional software development. When developing systems with an ML component, it's important that threats and mitigations specific to ML systems are understood alongside 'traditional' software security standards.

Developers aren't always security or usability experts, but understanding where decisions affect security, and how to design and implement a solution that works for its intended users, are crucial parts of ensuring that security works in practice. It's therefore important to establish a positive security culture, supported by leaders, providing sufficient credibility so that security is considered at the start of a project.

Studies have shown that many ML practitioners aren't aware of the specific vulnerabilities of many ML algorithms, or don't have the right tools to assess these vulnerabilities in the first place. This can create a poor security culture and heightens the risk of vulnerabilities being introduced to a system. In addition, as the push to low code and automated ML continues, it's also important to think about the background and skills of people *using* these systems, who are less likely to be 'developers'. Although they may not consider themselves to be writing programs or systems, they are still using underlying ML techniques that contain security vulnerabilities. Understanding these will help them make the right design decisions, that are proportionate to a project's security requirements, as well as promoting security throughout the development.

What are the goals of this principle?

- Ensuring that your ML practitioners, data scientists and software developers are familiar with the inherent vulnerabilities and failure modes in ML workflows and algorithms. They should follow best practices to reasonably mitigate the threats associated with them.
- Security is considered a key part of your ML project and integrated into your workflows. Your team members at all levels understand this.

How could this principle be implemented?

Approach	Description	Guidance and links to further reading
Strengthen your people	Enable and encourage a positive security culture and accountability by ensuring team members have the relevant security knowledge. Support this by providing training on the threats that are unique to ML components, where knowledge gaps are identified.	A security-centric culture requires training and requires all stakeholders involved in the ML lifecycle, from requirements to operational use, to understand the threats to a system, including ones unique and inherent to ML. You should therefore be prepared to invest time and resources to promote this. The NCSC has guidance on growing a positive security culture and how people shape security. Understand that security is not always part of formal data science or ML course curricula and that expertise in these areas may not translate to knowledge of system security. Even when best security practice is understood in theory, its practical use should still be encouraged. Technical team members working on model development should be familiar with a range of attacks on ML systems. Some of these attacks are highlighted in this Microsoft blog on Failure Modes in Machine Learning and in the MITRE ATLAS framework. ATLAS is a knowledge base of adversary tactics, techniques and case studies for ML systems based on real-world observation, demonstrations from ML red teams and security groups, and the state of the possible from academic research. As research in this field is fast moving, you should make sure that practitioners receive the right support to keep up to date with the latest attacks and vulnerabilities for ML techniques, or any other common vulnerabilities and exposures on applicable assets (such as models and datasets) or relevant software. A method capturing ML-related objective is cyber exercising.
Strengthen your approach/ processes / development	Encourage best security practices by making sure security reviews are integrated into your system lifecycle processes. Make people accountable for these processes and use automation where appropriate.	Each application's development lifecycle will be different and it's important to integrate security reviews appropriately. For example, if you take an agile approach to development, consider integrating security reviews into each sprint. If teams are aware of and accountable for security-related responsibilities through the lifecycle, it encourages a positive security culture. Depending on your specific development process, automated security testing may be an option, although it's still important to have a security-centred culture. See below for guidance on incorporating security-focused activities throughout the development lifecycle. (Although it's important to tailor advice and implementation to your own requirements.) NCSC guidance on secure development and deployment OWASP Securing the SDLC

Design for security

What does your system look like if the ML component fails? Identify harm done virtually, physically and reputationally. How will you identify if your system has 'failed'?

Why this principle?

An ML model is often just one of many components in a system, and how it interacts and behaves with the wider system should be considered in the design. It is important to understand the impact to the wider system of a successful attack on the confidentiality, integrity or availability of your ML component, and use this understanding to inform design decisions up and downstream from the model. The impact on other assets, systems or processes (such as dataset confidentiality) should be considered here too.

It will never be possible to assure complete security against adversaries, whose techniques are always evolving. The field of adversarial ML and AI is immature: new attack techniques appear regularly and there are significant limitations to existing defence methods. It's therefore important to ensure that you understand what your ML model may look like in different scenarios and the potential impact on your wider system. Scenarios of interest should include:

- How your ML system is used in normal operation, and what would happen if its error rate spiked (following an attack, the introduction of out-of-sample observations, or data drift).
- How you would identify such an error rate (you can't rely on the ML model to do this itself).
- How your ML component fits into your larger system and what would happen if the component no longer operated as intended.

Exploring these scenarios will help system designers decide whether additional mitigations are required to prevent undesirable behaviour if the ML model is attacked.

What are the goals of this principle?

- You understand how your ML model should perform, and design processes to identify and/or correct if it fails, either unintentionally or because of an adversary.
- You understand the implications of attacks on your ML model and the effects on wider system behaviour.
- You have sufficient expertise for your whole system design and application, not just AI/ML knowledge.
- You understand the wider consequences if your system is compromised, such as lower user confidence and reputational damage to both your system and organisation.

How could this principle be implemented?

Approach	Description	Guidance and links to further reading
Strengthen your system (including pre-processing)	*Consider how the design of your wider system can be used to limit the impact if the confidentiality, integrity or availability of the ML component (eg, model) is compromised. This may be baked in natively to the wider system design (including humans in the loop), or may require you to make design decisions based on ML component's limitations and your risk appetite.*	*The limitations of ML components can often be mitigated with non-ML components to restrict the model's effect on the wider system. But this may restrict the effectiveness of the ML component, which may cap its advantages to the system. It's important that both ML expertise and wider system/domain expertise are used in the design, and that people understand the limits of your ML components.* *Where you have identified the requirements for logic/rule-based controls and constraints outside of the model itself to secure wider system behaviour, consider using technologies such as:* • *expert systems – systems that use explicitly defined 'if-then' rules* • *subjective logic – a type of probabilistic logic that factors in uncertainty and source into the decision* • *decision trees – a branching structure where each node represents a test, the outcome of which dictates the next test* *Another approach is to include diversity in your system. This can include:* • *model ensembles: using different architectures to make predictions* • *training on different subsets of training data* • *using augmentations in pre-processing to create input diversity*
	Create system models for worst-case scenarios and modelling the effects of integrity attacks.	Use the CIA (confidentiality, integrity, availability) triad and known ML vulnerabilities (as outlined in MITRE ATLAS) to explore further. An effective way of capturing and sharing the necessary ML model information may be to treat it as a component in the system. Documenting inputs and outputs and highlighting limitations (such as uncertainties and value bounds) can be an effective way to do this. The NCSC has guidance on understanding system-driven risk management.
Strengthen your people	Ensure your design team and operational end users have experience across the range of disciplines or domains required for the system design process.	Encourage a cross-discipline culture that recognises the need to work collaboratively during the design process. Put in place processes and procedures to share knowledge and experience where disciplines meet and overlap. This is particularly important when highlighting the limitations of the ML components with the wider system designers. Ensure ML experts work collaboratively with system designers and formalise methods of exchanging information.

Minimise an adversary's knowledge

Understand that attackers can gather information to strengthen attacks.

Why this principle?

Sharing information about system designs, development processes and research findings can be beneficial to the whole ML user community, while responsible disclosure if an attack occurs is an important part of good security practice and is likely to improve ML security throughout the industry. But reconnaissance is often the first stage in an attack, and publishing too much information could help attackers. Information about model performance, architectures and training data can help an attacker to develop attacks, as discussed in principle 3.3 – Deployment – Minimising an adversary's knowledge.

Let's consider how this may increase your system vulnerability. If the open source parent model that you are using for transfer learning is publicly known, you are helping an attacker with a transfer attack. Transfer attacks can be developed on a surrogate model with the same parent model as the victim, and 'transfer' across, with any model similarities helping the 'transferability' of the attack. A model trained using transfer learning is particularly vulnerable to this type of attack because the victim model has many of the same properties as the surrogate used for attack creation. (While this risk should be recognised, you should also consider the many positives of transfer learning which, when carried out securely (discussed in section 4 – In operation: continual / online learning), allows you to cut costs, training time and carbon emissions.)

When deciding whether to release information, consider the balance between the motivation for sharing (marketing, publication or improving security practices) while protecting core system, development and model details. This balance requires understanding the implications on your system vulnerability of releasing information.

As a part of this balance, you should also consider when a model or system may be used in future, rather than just in its current application or format. Many elements of the ML industry have roots in research and academia, where there is a culture of knowledge sharing. This means that details of many models in operation may already be public knowledge.

What are the goals of this principle?

- You understand the risks of disclosing information, and how releasing unnecessarily detailed information could advantage to an attacker.
- You understand that information on any of your publicly available assets may also be available to adversaries -- this could include code libraries, pre-trained models or open-source datasets.
- You can make a balanced assessment of the benefits and risks of sharing information about your systems.

How could this principle be implemented?

Approach	Description	Guidance and links to further reading
Strengthen your approach/processes /development	Develop a process or framework to review information intended for public release and assess the risk it could pose to your system. Seek views from a range of backgrounds (for example ML practitioners, security, software developers, non-technical subject matter experts) to build up a comprehensive picture of the potential impact.	Creating the right process starts with identifying what information is required to go through the process itself. Examples may include marketing material, privacy policies and an organisation's contributions to academic literature or open-source software. The process should be tailored for your application and risk appetite. But it's likely to include an assessment of: • the information to be released against known vulnerabilities such as those in MITRE ATLAS • the information to be released against the vulnerabilities in your system • how releasing the information will benefit your organisation, system or the wide community • whether the intended message of publication can reduce negative impact on security When releasing information that confirms you are using open source assets (especially models or datasets), this should align with your risk appetite. This can be done by carrying out a risk assessment for your model's usage situation and other details of your system.
Strengthen your people	Make sure that all staff (not just staff in technical roles) involved in releasing information to the public are trained on the potential impact of releasing the material. Make them aware of the required processes for releasing different types of material, for example marketing material, privacy policies or contributions to academic literature or open source software.	Brief the risks to non-technical staff. The aim of this briefing should be to make sure they understand what material should be reviewed and how to do so. Tailor this briefing to the level of technicality needed depending on roles.

Requirements and development

At the requirements stage, it's important to understand the threat landscape for your application's lifecycle: development, deployment and decommissioning. Alongside your risk appetite, this allows you to set up proportionate requirements. The development and training stages of the lifecycle can be the most complex, and it's important to recognise that security vulnerabilities found here will propagate throughout the life of your system. It's important to think about security of your data, people and processes at this stage, not just your system design.

Design for security (vulnerabilities)

When drafting system requirements, consider the vulnerability of your proposed system against the inherent AI/ML threats and continue to review throughout the lifecycle.

Why this principle?

Identifying specific vulnerabilities in your intended workflows or algorithms at the requirements stage helps you plan your defences at the start of a project. By supporting security by design, you prevent the need to mitigate vulnerabilities in an operational system and the work and disruption that comes with that.
Some of the vulnerabilities inherent to ML workflows and algorithms will be more relevant than others in your system, depending on variables such as:

- data source(s)
- data sensitivity
- deployment and development environment
- wider system application and consequences of failure

For example, if a system is to be deployed where the inference inputs are uncontrolled and the outcome of a drop in integrity is catastrophic, you could include a requirement to improve robustness and meet this by including adversarial training as a system component.

It's good practice to review decisions throughout the development process, with a formal security review before a model or system is released into production. The use of (automated) tools can make this process easier and more effective.

What are the goals of this principle?

- You understand the range of ML-specific attacks or accidents that could affect your system and the impact they would have.
- You understand potential defences and mitigations for the above, and have incorporated them into your system design.
- You understand the importance of reviewing the security impact of all design decisions throughout the development process.
- You review your security decisions throughout the development process, including an evaluation that all your security requirements are met before a model goes into production.

How could this principle be implemented?

Approach	Description	Guidance and links to further reading
Strengthen your processes	Establish the risk appetite or the security targets in your requirements and set up development processes to ensure you're on track to meet them.	Understanding your risk appetite and requirements requires collaboration across different disciplines. This will vary depending on your specific application, but it's important to make sure the team has a good understanding of the proposed use of the system, its architecture and knowledge of the vulnerabilities of ML techniques. In larger applications, a systems engineering skillset will be useful. Once you have established security requirements, consider using automated tools to quantify and automatically test your model's security performance against known vulnerabilities and attack techniques. Running automated tests throughout the model's development cycle will help practitioners meet the required security goals. There are many open source tools to assess a model's robustness, including but not limited to: • Tools created by the DARPA GARD program • Azure Counterfit • CleverHans It's likely that testing standards will emerge as the field matures, so it's helpful to keep up with the latest advice from the Office of AI and AI Standards Hub.
	In cyber security, red teaming (or 'exercising') a system consists of playing the role of adversary to try and compromise the confidentiality, integrity or availability of a system to provide useful feedback to a system's designers on its vulnerability. Red teaming can be an effective way to highlight vulnerabilities in your systems. Applying a red teaming or pen-testing mindset (without necessarily undergoing full red teaming or pen testing) can be useful to identify vulnerabilities in your system. This will be useful to inform security requirements and drive design decisions.	At this early stage in the lifecycle before artefacts are created, working through design proposals via visual or systematic means of flow diagrams may be useful. It's likely that you will get the best red teaming results when you have a range of skills and expertise relating to different aspects of your product. Data science, assurance, and domain-specific expertise where intended users and use cases are well understood will be important here. The ML-specific attacks described in Microsoft's blog on Failure Modes in Machine Learning and in the MITRE ATLAS knowledge base are a good foundation for thinking about ML vulnerabilities. You can establish detailed security requirements by exploring each vulnerability while considering your risk appetite. Continue to apply an 'ML red teaming' mindset at regular intervals throughout the system development cycle and whenever key design decisions are made. The NCSC has guidance on penetration testing. More information on red teaming can be found in the MOD red teaming handbook, although it's important to emphasise that the suggestion at this stage is to apply a red teaming mindset rather than a full red-team operation. But you may wish to carry out full red team activity for a product at the end of the development, before production begins.

Strengthen your people	Follow principle 1.1 and train your developers to understand the inherent vulnerabilities to ML algorithms and workflows. Ensure your team are familiar with appropriate defensive techniques and risk mitigation strategies.	Provide time and resources to allow your developers and decision makers to build awareness of the latest attacks and vulnerabilities in ML. This often comes from academic research, so encouraging a research culture in the team may be helpful. Specific resources and links are given in principle 1.1 - Development prerequisites and wider considerations: Enable your developers. The importance here depends on your specific application and the risk appetite outlined in your requirements. Consider cyber security scenario exercising to train staff – see the NCSC guidance on cyber exercising.

Secure your supply chain

Obtain your data and models from a trusted source.

Why this principle?

Once you've created your requirements, the next step is identifying the sources of your data (if you're training yourself) or pre-trained model(s). In doing this, it's important to understand the impact that model and dataset creators have on your system, and possibly the wider ecosystem. A model's logic and behaviour is determined by its training data: the instances and (where appropriate) their labels. Whoever creates and labels your data therefore has direct influence on your model's behaviour.

Data collection, cleaning and labelling is an expensive process, particularly for a large dataset. Practitioners often rely on externally created datasets (whether closed or open source), which introduces a threat vector. Data could be benignly but badly labelled, or an adversary could deliberately mislabel instances or insert triggers: so-called 'data poisoning'. When a model is trained on a poisoned dataset, the overall performance can degrade. Some poisoning attacks can introduce targeted backdoors which could harm the integrity or availability of a model's outputs.

It's important to be able to trust your data and its supply chain. Implementing good validation and verification (V&V) processes for creating and acquiring dataset won't just protect against deliberate poisoning, but also helps you understand and mitigate mistakes or biases in a dataset that can impact performance. This is the case even with synthetic datasets.

More complex vulnerabilities exist when using open source or publicly accessible models. Models can be used by adversaries to hide weaknesses and backdoors, which can then be exploited during operation to harm integrity. Transfer learning is a productive and powerful approach that allows users to quickly build accurate deep-learning models by 'learning' from (often open source) parent models that are pre-trained with large datasets. This can have substantial benefits, but it can also introduce vulnerabilities, if a model inherits characteristics from the parent, by allowing an adversary to gain knowledge of the parent model and use it to exploit the resulting model. In transfer learning, the threats from transfer learning attacks must be considered along with defences and mitigations.

Although it's important to consider the risk of transfer learning, you should also consider the benefits, which when practised securely (discussed in section 4 - In operation: continual / online learning) helps cut costs, training time and carbon emissions.

Once data is obtained, it must be stored securely and transmitted to and from your training environment securely – see principle 2.4 - Requirements and development: secure your infrastructure.

What are the goals of this principle?

- You understand that your data and its labels determine the logic of the model.
- You understand your data supply chain and have sufficient validation and verification (V&V) processes in place to ensure data can be trusted.
- You understand that the risk of supply chain contamination heightens when you purchase and use models that don't have a transparent creation method.
- You understand the risk posed by insider threats and human error, if labelling manually.
- You understand the key information about the model you are using.

How could this principle be implemented?

Approach	Description	Guidance and links to further reading
Strengthen your model	When you are procuring assets, consider the security of the supply chain. This also applies when using auto labelling/labelling aid software.	Follow the NCSC's supply chain security guidance, which advises understanding your suppliers and their security posture, and ensuring they are aware of your security expectations. Follow advice in the ETSI AI Data Supply Chain Security paper.
	It may not always be possible to obtain data from secure and trusted sources. When this is the case, you should consider using synthetic data or limited data, rather than gathering from untrusted or less secure sources. There are a range of techniques to help train ML systems on limited data.	Synthetic data can be generated in a range of ways and the right method for you depends on your dataset requirements. Techniques for generating synthetic data can come with their own challenges, such as GANs amplifying and recreating statistical distributions in the original data, game engines introducing specific characteristics that exist only in the engine used, or augmentations being limited to manipulating only the original dataset. It's important to understand the limitations and the risks posed. DSTL provides a handbook for ML with limited data. This is useful when it's best for security not to use untrusted data.
	There may be times when you can't trust your supply chain but operational requirements mean you must use untrusted data anyway. From a security point of view, this isn't ideal, and you should certainly look to improve the trustworthiness of your asset supply rather than check a	Techniques to find poisoned instances in a dataset, or which prevent them from having an effect include: • clustering • model pruning • trigger reconstruction Many of these techniques are mainly tested in academia and not widely used by practitioners. We can't therefore confidently recommend them as a solution and so only suggest their use as a last resort when requirements prevent you gathering

	potentially untrustworthy one. But in those situations where you may be required to use it anyway, there are techniques that can help check its integrity. Hardening techniques can also be applied to models to detect and/or mitigate the effect of backdoors introduced by data poisoning.	more trusted data. It's for your development and security team to evaluate their applicability and impact on your application and development process. As hardening complex models against various types of attacks is a fast-changing area of research, make sure you keep up to date with the latest techniques.
Strengthen your processes	Implement and apply a robust validation and verification process to assess the quality and integrity of both internally created and externally acquired datasets.	Validation and verification for labels could include: When labelling internally to your organisation: • having multiple labellers look at each data input and generate notifications where labels differ For procured and externally created datasets: • follow the NCSC's supply chain security to help find a trusted vendor. • consider randomised audits of data labels to assess error rates
	Consider applying security-specific data augmentation alongside standard data augmentation. This is often referred to as 'adversarial training'.	Data augmentation techniques have become standard practice in ML (especially deep learning), as augmentation has been shown to greatly improve the generalisation abilities of models. It's therefore likely that your development process will involve some form of data augmentation, regardless of security benefits. But it's important to recognise that some types of augmentation may also be beneficial to improve robustness against adversarial inputs.
Strengthen your people	Create guidance and train labellers on their roles and responsibilities. This will help minimise unintentional mislabelling – especially important for nuanced labelling, such as spatial data (object detection bounding boxes).	Data labelling will be unique to your application, so it's important that you create custom guidance and training for your labellers. Google offers a useful guide to creating data labelling instructions. As well as guidance, consider whether your dataset creation may benefit from the use of labelling software. A wide range of products exist here. Having your labellers trained on the same software is likely to improve consistency and reduce errors in your resulting dataset.
	Reduce the risk of insider attacks on datasets by making sure the level of vetting for your labellers is appropriate for the severity of impact that mislabelling could have. This would be different, for example, for an autonomous vehicle and in a research environment.	Refer to any industry specific guidance on personnel security and vetting. In situations where it's not feasible to vet labellers, use processes that can limit a single labeller's power, for example: breaking a full data set into segments, ensuring that a single labeller never has access to an entire (or large portion) of the dataset. Validation and verification techniques, discussed above, are also important here.

Secure your infrastructure (development environment)

Baseline security by protecting your training and development environment, applying trust controls to anything and anyone that enters.

Why does this principle exist?

The security of the environment in which your model is developed and trained is critical. At the training stage of the lifecycle, valuable assets (eg, training data) come together with external inputs (eg, software or open-source pre-trained models). This creates a high-value target with multiple possible attack vectors. Damage or exploitation at this stage could enable attacks that propagate throughout your application's lifecycle. This principle concerns protecting this environment; related discussion of gathering and securing trusted training data is found in 2.2 and 2.4 respectively (Requirements and development: secure your supply chain and secure your infrastructure).

Like your data, the infrastructure that makes up your training environment needs to be sourced through a trusted supply chain, whether that involves locally owned servers, public cloud services or local training on an embedded device. Supply chain threats should be considered for each specific application and be evaluated against requirements. Developers should understand that initiating development in a less secure research and development (R&D) environment could create attack vectors that could be exploited in production. For example, an adversary could use information about model architecture following reconnaissance of an insecure environment to develop a surrogate model that could be used to develop attacks against a production version.

Securing development infrastructure is likely to involve assessing multiple components and settings. It's important to secure the operating system, ensuring you're updated to the latest version, and limiting access to the environment to only those with a legitimate need. Access to and modification of components should be logged and monitored. External components may include third-party software for building, training and deploying models. Exploitation of software dependencies is an attack vector for adversaries, hence software should only be obtained from trusted sources. It's important to note that securing the storage of training data is not in scope for this principle: this is covered in principle 2.4 - Requirements and development: secure your infrastructure. Advice on how to secure your deployment infrastructure is in 3.1 - Deployment: secure your infrastructure.

What are the goals of this principle?

- You have sufficient Quality Assurance (QA) or Quality Control (QC) processes for your supply chain, including your digital assets, following best practice.
- You understand the severity of the risk posed by the combination of valuable assets and supply chain inputs merging at this stage. These inputs may include:
- third party libraries and frameworks
- pre-trained models used directly or for transfer learning
- training data and labels

How could this principle be implemented

Approach	Description	Guidance and links to further reading
Strengthen your infrastructure	Source assets (including digital ones) appropriately from trusted supply chains This includes pre-trained parent models for transfer learning.	NCSC guidance on security of supply chains
	Source software to build and train models (eg, third-party Python libraries), data warehouses, MLOps software and experimentation or testing platforms from trusted supply chains.	Verify any third-party inputs (including pre-trained models) are from sources you trust.
	Use secure software development practices.	NCSC guidance on secure development and deployment
	Monitor CVEs associated with your development software and library dependencies. The level of monitoring should be proportionate to your security requirements.	CVE (common vulnerabilities and exposures)
	Ensure access to your training environment is appropriately secured while following the principle of least privilege for user access.	NCSC cloud principles
		Traditional cyber security practices NCSC Cyber Essentials
	Secure your firmware/operating system	NCSC guidance on platforms NCSC guidance on securing Windows OS NCSC guidance on securing Ubuntu OS CISecurity on CIS benchmarks
	Implement access logging and monitoring in your development environment.	NCSC guidance on security logging
		NCSC blog about what exactly you should be logging The NCSC has released premade software to support logging, called Logging Made Easy (LME). More information can be found about it on the LME Github page.
		NCSC guidance on introduction to security logging
Strengthen your people	Ensure your team have an appropriate level of awareness of cyber vulnerabilities and best practices to mitigate them.	NCSC guidance on advice for end users

Secure your infrastructure (digital assets)

Protect digital assets at rest and in transit

Why this principle?

Once you have your data and have acquired or generated your model, you need to protect them. These digital assets represent a significant investment of resources and intellectual property, making them an attractive target for theft, and their manipulation can significantly affect a system's operation. Knowledge of a model's training data or architecture can also enhance an adversary's ability to craft an effective attack. Although not a unique consideration of ML systems, the security of data and model artefacts, at rest and in transit, is therefore particularly important throughout an ML system's lifecycle.

There's actually a non-malicious angle to this one too. Data can be damaged either at rest or in transit by any number of innocent processes. Corruptions may not necessarily result in a run-time error, so it's possible for these changes to go unnoticed but still affect performance. Similarly, accidental but unnoticed corruption of a portion of your training data will likely propagate errors into the model.

What are the goals of this principle?

- You know what data you hold and where it's stored.
- You understand what data you are transmitting, between which devices or to whom.
- You understand the impact of theft or loss of integrity to this data, including legal and reputational impacts.
- You consider datasets, models and other artefacts to be crucial assets that need to be protected for security as well as other purposes (eg, data protection).
- You have followed appropriate advice and guidance on how to secure digital assets, including limiting access to those with a legitimate need, and logging and monitoring any access and modification.

How could this principle be implemented?

Although protecting data is especially important in ML workflows and systems, the ways to protect against and mitigate these vulnerabilities are not unique to AI or ML. Protecting data is a wider issue and the advice and regulations reflect this. Best practice is likely to include encrypting data at rest and in transit, keeping keys as secure as possible, implementing multi-factor authentication and, where necessary, in two person controls. Further guidance includes but isn't limited to:

Approach	Description	Guidance and links to further reading
Strengthen your infrastructure	Follow technical advice on how to protect data on your specific infrastructure (eg, local PC or cloud)	NCSC guidance on dealing with data NCSC guidance on cloud security principles
	Follow technical advice on how to protect data in transit (for when this is required) and at rest	NCSC guidance on data at rest NCSC guidance on data in transit
	Implement recognised IT standards	ISO – 27001
Strengthen your people	Ensure decision makers understand what data you hold, and the law and regulations around data collection Ensure your developers understand the impact and consequences of data breaches, their responsibility when processing data and the importance of developing secure software	NCSC guidance on protecting bulk personal data
		Information Commissioner's Office
		ICO guide to the UK General Data Protection Regulation (UK GDPR)

Track your assets

Document the creation, operation and lifecycle management of models and datasets.

Why this principle?

Data drives the development of an ML model and impacts the model's resulting behaviour. Tampering with a dataset can therefore have a significant impact on a system's integrity. However, in many workflows, datasets and models are unlikely to remain static throughout the development and wider lifecycle of an ML solution. This regular mutation can make it difficult to distinguish nefarious changes from legitimate updates.

It's therefore crucial that system owners implement a framework for monitoring and recording changes to an asset and its metadata throughout its life. Good documentation and monitoring strengthens your ability to respond when a dataset or model has been compromised, for example, through poisoning. It's worth noting that although datasets are used to produce models, creating an inherent connection between model and dataset metadata, they should be treated as separate entities - a dataset is not necessarily exclusive to a single model and a model can be trained on multiple datasets.

Version control allows changes to be tracked and rolled back, and for metadata to be generated for confidence checking. Tracking changes between versions enables scrutiny between releases and updates and allows developers to understand and mitigate attacks or accidents.

This has an overlap with preventing 'data leakage', where data used to train a model can be accidentally reused in validation.

The most significant metadata will vary from asset to asset, and it's for the asset owner to decide what is important to record. The format of this metadata should however be standardised because information captured in a consistent way is easier to share and communicate, allowing users to be better informed and make better judgements, which ultimately improves security. Collecting data in a machine-readable format has several advantages. One is that a digital catalogue for datasets and models can be created, allowing users to filter and search depending on requirements. Machine-readable data can also be consumed by other programs that can be used as security or monitoring solutions throughout the operational stage of the lifecycle. This can help implement tracking model behaviour, monitoring and logging user queries as outlined in principle 3.2 - Deployment: design for security.

Good dataset documentation enables more trustworthy sharing of datasets. Hashing and digital signing of a dataset can ensure that recipients of a shared dataset can verify the authenticity of its contents.

What are the goals of this principle?

- You know the intended use of the dataset, including where it can and can't be used.
- You know who is authorised to access your data.
- You know when your data needs to be deleted.
- You are aware of any biases in your data.
- You have considered what metadata should be captured, and for what purpose.
- You 'version-control' your dataset throughout its lifecycle, and can roll back datasets or models easily to help with root cause analysis.
- If a model is compromised model, you can roll back to a known safe state.
- You know what data you hold and where it is stored.
- If you are distributing your dataset or model, the recipients can verify the authenticity of the contents using attached metadata and confirm it hasn't been tampered with.
- You are monitoring key metrics that can be tracked across the data lifecycle.

How could this principle be implemented?

Approach	Description	Guidance and links to further reading
Strengthen your processes	Consider using version-control software (such as Git) to track and control changes to software, datasets and your resulting model. The size of your project assets (eg, datasets) may limit your choice of technologies. Larger assets may require more specific, database-centric technologies.	There are a range of version-control paradigms and supporting software to choose from. It will be for your development team to select the best for your use case, though in general your version-control software should be able to: • track which users make changes to datasets and models with full details of any modification (including the author, time and date) • allow for reviews before changes to an asset are made • 'roll back' in case of a security incident where an older version of the asset is required The UK government has advice for maintaining version control in coding and advice specifically around using Git and Github.
	Track dataset information and updates in a consistent format that is readable by humans and that can be easily processed or parsed by a computer (eg, json, csv). This metadata should be paired with the dataset throughout its lifecycle and include updates to the dataset made throughout operation, via continual learning, as discussed in section 4 - In operation: continual/online learning.	Technical implementations for tracking datasets include a data catalogue or dictionary, or implementation as part of a bigger solution (for example, as a part of a 'data warehouse' or a version-controlled database). The metadata you track on your dataset should be decided based on your specific application. Metrics that can benefit security may include: • a description of how data was collected • the sensitivity level of data • key data metrics (eg, RGB values over a set of images) • the dataset creator or maintainer • intended use of the data and any known limitations • retention time of the data

		• retirement/destruction method of the data • aggregated stats about the data (eg, if labelled, count of each label)
	Track model information in a consistent format that is readable by humans and that can be easily processed or parsed by a computer (eg, json, csv). The model's metadata should be paired with it throughout its lifecycle and include updates to the model made throughout operation, via continual learning, as discussed in section 4 - In operation: continual/online learning. The 'Model Cards' format is becoming an increasing popular framework for tracking model metadata, with organisations such as Google using model cards.	Model cards can be used to track versions as ML models are updated or trained with different datasets throughout development or continual learning through operation. You may choose to create your own model card format. Useful metadata to track for security purposes may include: • the dataset on which the model was trained • the model creator/point of contact • intended scope of the model and its limitations • secure hashes of (or digitally signed) trained models • retention time of the dataset – as per the dataset metadata • retirement/destruction method of the model (Note: there are many other things you may choose to track on a model card for different reasons – here we have just outlined security related metrics.) Once model cards have been created, they should be stored alongside the model. Cataloguing the cards in a searchable (indexed) format is useful to users and developers, as this allows developers to find an appropriate model easily and to compare models. GCHQ have released a framework to enable this, called Bailo. More information on how to install and use this can be found on GCHQ Github.
Strengthen your people	Empower your people with accountability for assets. Accountability can be for the creation of an asset and/or the maintenance of it.	Ensure your development team follows good development operations management. Operations management is important for security because it covers policy compliance, configuration and management of development tools (eg, version-control software). This management should continue throughout the lifecycle.
	Ensure your development team follows good development operations management. Operations management is important for security because it covers policy compliance, configuration and management of development tools (eg, version-control software). This management should continue throughout the lifecycle.	Development teams should include a role that oversees management of development operations. These job titles are usually called DevOps or machine learning Operations (MLOps) engineers. Part of the specification for this role should be overlooking and managing digital assets, ensuring that organisational policies around the assets are adhered to. For smaller teams where there isn't the capacity or requirement for a single person to take this role, make sure that MLOps practices are still implemented and followed by the developers themselves. Training may be required when this is the case.

Design for security (model architecture)

Keep your model architecture and capacity proportionate to your training data and requirements.

Why does this principle exist?

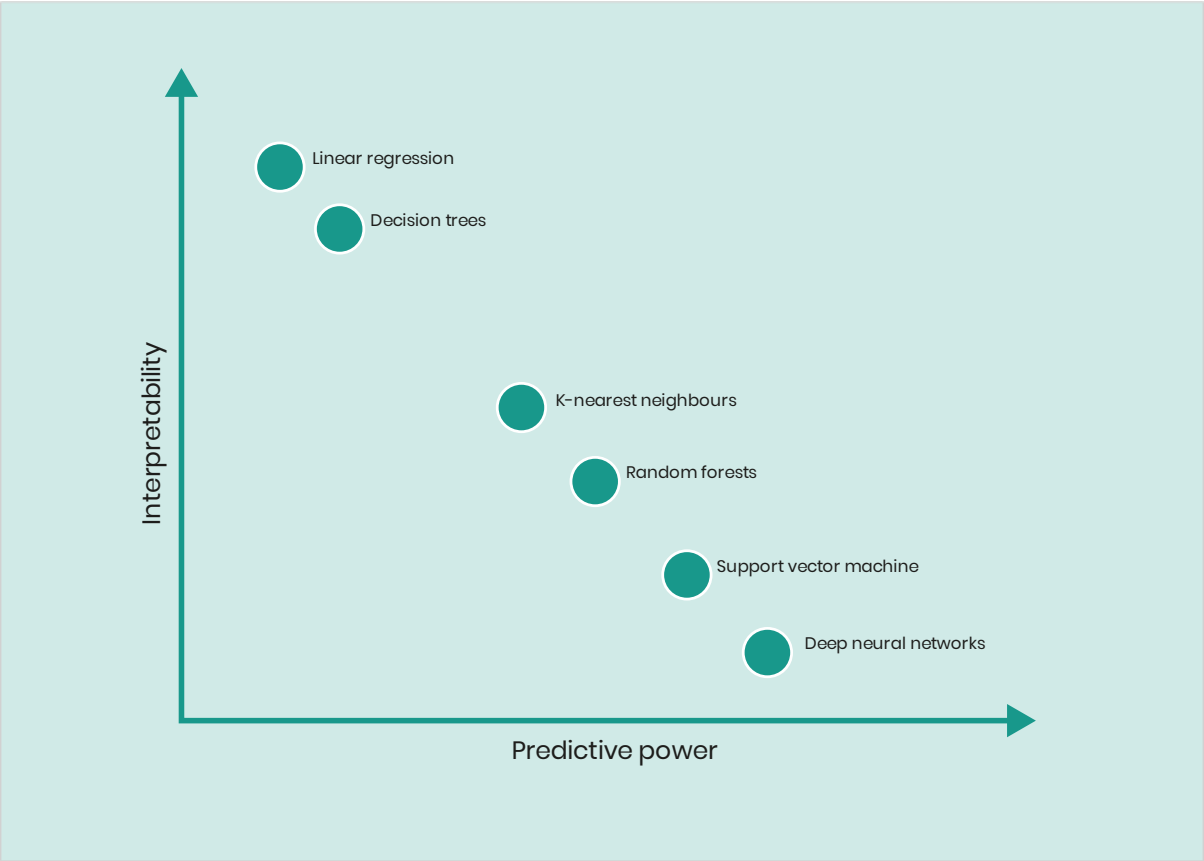
'Model capacity' describes how complicated a pattern or relationship is that a model can express, and is dictated by the architecture choice (for example, a neural network versus a decision tree) and, in some model types, the stipulated size of the model (i.e. its number of tuneable parameters).

For a given model to perform well, it's important that its capacity fits the dataset size and variety. A model with capacity that is too low for a task won't generalise well and perform poorly. This can create security vulnerabilities especially around edge cases, lowering the entry barrier to to 'trick' your system. In contrast, a model with a capacity that is too high for the dataset also creates security vulnerabilities, such as increasing the proportion of feature space in which the model behaves unreliably (if the training data does not adequately cover the feature space) or even giving adversaries extra capacity in which to hide malware.

To dictate a model's capacity there are two design decisions: its architecture and its size. Some model architectures, such as neural networks, are harder to interpret and understand than simple linear models, so the design choice must be justified. To validate your model for security or robustness, it's important to understand (and be able to explain) your model's behaviour. This allows you to:

- detect anomalous behaviour
- better predict how the model will react to out of distribution (OoD) inputs (ie inputs that don't appear in its training data)
- understand when you may require downstream rules or controls to constrain the output or effects of your model
- build trust in the user base

The size of the model may be a hyperparameter that a designer must select. There is no hard-and-fast rule when making this decision, but your design process should aim to align the model's capacity to the size and quality of your dataset, and reduce the vulnerabilities associated with overcapacity.



What are the goals of this principle?

- The complexity of your model and size/quality of your dataset is justified for meeting your performance requirements.
- You understand the potential trade-off between interpretability and predictive power and the effects this can have on security.
- You can assess that your model capacity will be appropriate for the size and quality of your dataset.

How could this principle be implemented?

Approach	Description	Guidance and links to further reading
Strengthen your approach	Test a range of model types on your data and assess their performance. This will help you make an informed decision about which model is best for your application, but also which performs best on security aspects.	Assess the performance of a range of architecture types (where performance is likely to include robustness), using a bottom-up approach to selecting your model architecture. Start with classical ML and highly interpretable techniques where possible, rather than diving into the latest deep learning models. *(Although an experienced practitioner could reduce development time by suggesting a model architecture type to start with – see point below.)* When you understand the performance of a range of architecture types, assess the trade-off between meeting performance requirements and the security and how explainable each architecture type is. Once you have narrowed down your model architecture type selection to the best performing ones, the next phase is to tune hyperparameters (and iterate). The aim of this stage is to ensure your model's capacity matches your requirements and dataset.
Strengthen your people	While there are no hard-and-fast rules for selecting a model type and hyperparameters, experienced practitioners and data scientists understand what type of models are right to consider for a given problem. This can be useful for narrowing down the range of models you need to test and provide a starting point for your bottom-up testing processes as outlined below.	Discuss with experienced practitioners in your team or research area where others have overcome similar problems to yours. This knowledge will speed up development time, allowing more time for fine tuning the model parameters to ensure a good fit between your model and training data. If your team has little direct knowledge or experience of the problem you're working on, a useful place to start may be to consider which models are **not** appropriate for this solution.
	Model selection is the data scientists/ML practitioner's job and there is no formal process for it. It's important to ensure your developers and practitioners are familiar with the advantages and disadvantages of a range of ML and probabilistic techniques, and are focused on fulfilling requirements, with no bias towards the latest algorithms.	When selecting a team of developers, put in place processes to ensure their knowledge of algorithms is wide, and invest in the appropriate training and guidance to fill gaps in knowledge. Training should be continual, as new trends and developments come out of academia.

	Review the size and quality of your dataset against your requirements. The metrics you use to evaluate it will depend on your specific application, although there are many rules of thumb that can be applied here. The data scientists and practitioners should select which are most appropriate for your application.	Rules of thumb for a evaluating the required size for a dataset include: For regression analysis: • the 1-in-10 rule, 10 cases per prediction For computer vision: • 1,000 images per class For classification: the learning curve equation, as explained in this tutorial This list is not exhaustive and there are many techniques for calculating required sample size in traditional statistics. It's worth searching around your specific use case, as there may be rules of thumb or guidance from others' experiences. Common metrics for evaluating a dataset's quality include completeness, timeliness, validity, consistency, integrity and balance between classes. Where data is insufficient, several approaches may help. They include gathering more data, using transfer learning, augmentation of existing data and synthetic data generation. Selecting an algorithm that runs well on limited data may also prove useful. Dstl have guidance on ML with limited data.
Strengthen your data	Attackers may exploit out of distribution (OoD) areas of the feature space for attacks. Where complex models are required, consider implementing OoD detection on inference inputs. Depending how your developer implements this, it could double up as detection mechanism for some adversarial attacks.	Your development team should choose the best OoD detection for your application. Options here include: • maximum softmax probability (MaxProb) • temperature scaling If you have a complex model with large feature spaces that are OoD, it may be worth exploring whether you can factor in the OoD distance in the confidence score.
Strengthen your model	Firstly, consider whether the use of more complex model (eg, a neural network) is justified for your use case. In many applications, particularly those using structured data, classical ML methods may achieve sufficient performance whilst being more interpretable and	Where you have identified a requirement for your model to be pruned, how to best implement pruning for your model and application. Many techniques show promising signs in a laboratory environment but are relatively untested in deployment. Each defensive method has its own advantages and disadvantages (often a drop in performance). Some key metrics to evaluate when pruning are: • a model's accuracy before and after • a model's robustness before and after • a model's size before and after

	potentially subject to a smaller range of attacks. If a complex model is required, there are a range of techniques highlighted in research and academia for reducing the size of a complex model (often a neural network). Many fall under the umbrella technique of 'pruning'. Pruning occurs after you have a trained model and involves removing unnecessary neurons or weights. Pruning can have a detrimental impact on performance and should be used when other more pragmatic techniques have fallen short. When required, consider regularly pruning and simplifying your model during the development and testing process, rather than doing a final test pre-deployment.	• computation time

Deployment

Once your model is in operation, it may be vulnerable to a range of different attacks. These include:

- evasion attacks, designed to make a model misclassify specific examples
- extraction attacks, when an attacker uses repeated querying to build a copy of the model
- inversion attacks, when an attacker aims to identify or reconstruct data on which the model was trained

Protecting information about your model will help strengthen the barrier to entry against an attacker.

Secure your infrastructure

Limit access to your model and its processing pipeline in deployment.

Why this principle?

Knowledge of your model can enable prospective attackers to create better performing attacks against it. The range of this knowledge can be described on a scale between 'transparent box', when an attacker has complete information about a model's architecture, weights and biases, through to 'opaque box' when an attacker has no prior knowledge, except for the ability to query the model and view its decision. This knowledge may be derived directly via access to the model, or inferred via knowledge of the pipeline. For example, reconnaissance of pre-processing steps, such as the shape of input data, gives an attacker information that can be used to help move closer to a transparent box attack.

If a model is used in more than one place (such as over a range of products), an attacker who gains access to one instance could then create an attack that is effective against all instances of that model. This is especially important when deploying a model in uncontrolled environments, when attackers are likely to have free access to hardware, for example, in a product that is mass produced and sold directly to consumers.

What are the goals of this principle?

- You understand what attacks are possible against an ML model once it is in operation.
- You understand what information is available to users (directly or through querying the model) about your model and its pre-processing, and the implications this could have in enabling an attack.

How could this principle be implemented?

Approach	Description	Guidance and links to further reading
Strengthen your infrastructure	It's important to assess what aspects of your system could leak information about your model's architecture (eg, its required input shape) and weights to an adversary. An example of this is using a web-based front end to resize images to the input size of the model. This is likely to include pre-processing or model output processing steps. Ensure these and your model are appropriately protected for your security requirements. Unless further detail is needed by another process or the end user, the model should be limited to providing the required output or prediction. This limits the potential of adversarial attacks by obscuring accuracy scores and weights, etc. (Teams should be aware that certain adversarial techniques allow forms of model inference or evasion based only on the model output, although this is technically more challenging for an adversary.) You may still want to consider access controls here (eg, role-based access control), enabling different levels of insight depending on user authorisation.	How you protect access to hardware and software will depend on your deployment infrastructure. It's important to follow the guidance relevant to your specific infrastructure. Principle 2.3 - Requirements and development: secure your supply chain and infrastructure has a list of security guidance. Use this as a starting point by selecting the advice relevant to your deployment method, then tailor this guidance to the requirements of your deployment system to meet your security requirements. Where your deployment method is not covered, seek specialist knowledge for your deployment type and how to protect that platform specifically.
Strengthen your people	Make sure you have the right expertise for protecting the parts of your deployment infrastructure and your team understands which parts of the system need to be protected or hidden from adversaries.	Model deployment infrastructures vary vastly and there is no one skillset or list of skillsets required. It's important that your team has expertise for the specific deployment infrastructure you plan to use. They should have sufficient knowledge of the security of that infrastructure and work closely with the development team to understand what parts of the system are valuable to an adversary (eg pre-processing) and need protecting.

Design for security

Monitor and log your users' inference requests/queries, implementing an alert system for anomalous behaviour if required.

Why this principle?

Understanding how users are querying your model and flagging unusual behaviour for investigation can help you identify and prevent attacks.

Many attacks, including model inversion, membership inference and denial of service, are conducted by querying a model repeatedly, often at a much faster rate than a typical user. Repeated querying is also used in the creation of adversarial examples to be used in model evasion attacks.

Collecting and monitoring information representing both the content of queries and their metadata (for example, frequency and patterns querying) can help identify anomalous activity. You should understand what expected inputs to your model look like and use that information to spot unusual inputs that may be worth investigating further. In the event of an attack or accidental system failure, accurate logs of user activity provide a basis for a post-attack investigation.

What are the goals of this principle?

- You know what expected inputs to your model look like and have criteria to identify anomalous inputs.
- You can audit use of the system and its inputs and outputs.
- You have appropriate log data to allow you to investigate a compromise of your system, even if not identified immediately.

How could this principle be implemented?

For further guidance on how to log effectively:

Approach	Description	Guidance and links to further reading
Strengthen your system	Follow appropriate guidance when applying logging and auditing of logs.	NCSC guidance on introduction to security logging
		NCSC blog on what exactly you should be logging The NCSC has released pre-made software to support logging, called Logging Made Easy (LME). More information can be found about it on the LME Github page.
	Consider automated monitoring of user activity and logs. Understanding how suspicious behaviour might look different to expected behaviour in event logs is the foundation here.	The criteria for 'suspicious' behaviour will be different for each system and you should use your understanding of your system and its intended use, to develop the criteria. This should be clear from the start of the development process and include an understanding of the expected input space. Flagged behaviour should be retained for audit and reviewed regularly for evidence of suspicious behaviour. The actions your system takes in response to detecting unusual behaviour will vary. Common responses may include: limiting (throttling) the rate of queries that can be sent to your model and implementing appropriate pre-processing, to heighten the threshold for an adversarial user developing an opaque box attack. (Open API safety best practices)
Strengthen your approach /processes /development	When an alert has been raised or an investigation has concluded that a cyber security incident has occurred, report to the relevant stakeholders and authorities.	Follow the NCSC's advice for reporting and managing incidents and if appropriate report the incident to the NCSC.
Strengthen your people	Use humans in the loop to investigate what automated processes flag as unusual.	If user behaviour has been flagged as suspicious, or during routine audits, humans should be reviewing user behaviour and the system response to this behaviour. As part of this process, the reviewer should decide whether the user interaction was malicious or not and whether the system/model response was expected and/or correct. To be able to draw conclusions, the reviewer must have a good understanding of the system requirements and architecture. When the review contains personal or sensitive data, the correct guidelines and security procedures must be followed for handling and viewing data. Compliance and security when handling data is discussed in principle 2.4 - Requirements and development: secure your infrastructure (digital assets).

Minimise an adversary's knowledge

Understand the trade-off between security during operation and the output a model provides to users.

Why this principle?

During operation, providing users with visibility of a model's output can allow for quick diagnosis that a system is behaving unexpectedly. This won't always be required or even possible, but in many situations it can be beneficial, especially when the negative impact is significant.

The creation of many attacks on ML models relies however on the same information: a model's output for a given input. If an attacker receives a more detailed output, it can make a successful attack more likely. It's therefore desirable to find a balance between protecting key information, while still allowing for 'sanity checking'.

A suitable balance between transparency and security will depend on the specific system application. It's likely that your system will have different classes of user – such as a normal user querying the system, an engineer servicing the system and a system admin diagnosing errors. Each user requires a different level of detail about the model's behaviour and model outputs should be tailored appropriately. This can be seen as a type of role-based access control.

Consider, for example, whether an end-user needs to know the confidence level of your model's predictions to multiple decimal places, or whether providing a low-fidelity summarising output (eg, simple classification) would be sufficient, retaining more detailed information for internal audit only. You may also consider the way that information is provided at each level of user access. For example, displaying model outputs to a user via a dashboard whist restricting programmatic access could still provide benefits to the user while greatly increasing the difficulty of an attacker's task.

Transparency of a model's decision is not the same as system explainability…

What are the goals of this principle?

- You understand the trade-off between transparency and security in the context of your system and use that understanding to make informed judgements about the potential consequences of an attack.
- Your model provides users with useful answers but doesn't reveal an unnecessary level of detail.

How could this principle be implemented?

Approach	Description	Guidance and links to further reading
Strengthen your model	Use access controls to vary users' access to different levels of detail from model outputs. Consider what level of detail each user requires. Can you represent data in a way that allows for a user to quickly check the system's behaviour without giving easy access to detailed output information? Disguising model outputs can be beneficial when it is done in a way that converts the information to be more easily digestible by a reader. Transforming data in this way can also be beneficial for security as it presents it in a way that is easier for 'sanity checking', therefore a user is able to detect adverse behaviour more quickly.	Two popular implementations of access controls are role-based access controls (RBAC) and Attribute-Based Access Control (ABAC). The most appropriate implementation of access control will depend on your security requirements. For user roles that require transparency but aren't necessarily fully trusted, consider displaying information in summary or graphical format rather than allowing raw values to be accessed. If access to the values is required, can these be with lower accuracy, or provided at a slower rate (ie lower frequency of queries)? Limit higher fidelity access to model outputs to authorised users in limited roles, such as trusted maintainers and developers.

In operation: Continual / online learning

Continual or online learning is the process of adjusting a model's behaviour or performance based on operational interactions and feedback. This adds an extra level of complexity to a system. Allowing external sources to influence your model's behaviour in operation is a double-edged sword: if the influence is positive, it continues to improve performance and keeps a model secure throughout its life. However, an opportunity is also created for external inputs to be used to cause damage.

Design for security

Understand if and when you're using continual learning, and the risks associated with doing so.

Why this principle?

The use of continual learning (CL) can be a crucial component in helping keep a system secure as well as performant: it allows you to dynamically tackle issues like model drift and erroneous predictions, while also creating a 'moving target' for attackers. CL can come in many forms and is not always obvious. From a security point of view, you can consider CL to be any time that your model's behaviour is affected by data collected during operation (an example would be adding a new data point to a k-nearest-neighbour algorithm during its deployment).

Although CL can bring important security benefits, the process of taking user input and feedback to retrain a model during operation also opens a new attack vector, as you are allowing the logic and behaviour of your model to be changed by possibly untrusted sources. Retraining on an updated dataset may cause the model's decision surface to change, causing old (potentially correct) behaviours to be lost under certain circumstances. This can lead to the model 'forgetting' certain behaviours, causing a drop in performance. An attacker could harness this to deliberately induce bias or undesirable behaviour in your model, meaning you should make sure that a retrained model still behaves how you intended.

CL also brings complications around protecting user privacy, as it requires collecting data directly from users. It also indirectly complicates the tracking of dataset and model updates as outlined in principle 2.5 - Requirements and development: track your assets, and makes supply chains more difficult to secure.

You should be aware of these risks, and have systems or processes in place to ensure users can't maliciously affect your model's behaviour. This will help mitigate the risk of adversarial manipulation from continual learning and improve confidentiality for users, as you won't be collecting data that doesn't meet your requirements. How to prevent malicious manipulation and continual forgetting is discussed in principle 4.2 - In operation: track your assets.

What are the goals of this principle?

- You understand whether use of CL is justified for your specification and system.
- You understand that using CL opens up your system to possible exploitation by users.
- You understand that CL carried out on a local system will only affect one instance of the model, and have considered whether it's desirable to have each specific instance of your system behaving slightly differently.
- You understand the potential impact of adversaries influencing your model's logic (as happened in the example of Microsoft Tay).

How could this principle be implemented?

Approach	Description	Guidance and links to further reading
Strengthen your infrastructure (including pre-processing?)	When collecting data from users, use techniques to ensure the data is anonymised. This allows you to gain the advantage of collecting data while protecting users' privacy, reducing security concerns.	Data anonymisation techniques include: • Differential privacy – using statistical techniques to ensure privacy without affecting the datasets overall statistical integrity. • Data masking – a range of techniques that represent a point in a different (usually unreadable to a human) way, eg, encryption, hashing or data obfuscation. • Generalisation/aggregation – 'zooming out' of the data to anonymise individual contributions while keeping overall trends. • Data swapping – swapping attributes/data points between entries while maintaining the underlying statistics of the dataset. • Pseudonymisation – removing (and keeping separate) attributes of the data such that a data point can no longer be attributed to a specific entry without the removed information.
	Where possible, consider processing data and training locally (on the user's device) without removing the data from the device. Updates to a model can then be shared with a central version.	If you don't intend to update a central/global model the training can be done locally on a user's device without the updated model ever leaving the device. When you intend the updated model on a local instance to contribute to updating a central model, your solution may include techniques such as federated learning.
Strengthen your approach / processes / development	Continual learning data collection and model retraining is an important part of your machine learning Operations (MLOps) process. An effective MLOps process should be automatic using a robust, consistent and understandable framework. This will allow	There are lots of sources and texts that describe how to do effective MLOps. Consider using a pipeline architecture with checkpoints for testing and review processes as discussed in principle 4.2 - In operation: track your assets. As a part of this approach, you may consider running multiple models in parallel, enabling models to be updated and tested before being released for operation. A solution for this may use sandboxed environments to run updated models for

	the tracking of data throughout the continual learning process and give the ability to follow in the case of an error or attack such as poisoning. It's important to be able to identify and manage model/concept drift, where a model may start to adjust its predictions based on new training data that may move outside of the original operating criteria, or a model may begin to degrade as the data changes and exhibits new features or concepts.	comparison, or if models require user interaction, experimentation deployments can be set up that target smaller groups of users that interact with an updated model.
Strengthen your people	Ensure your engineers understand what continual learning is and the associated risks. Having developers who understand this will enable them to make sensible decisions throughout system design and development.	Provide the necessary time and resources to ensure technical staff have an understanding of the range of ways CL can present itself and the possible security vulnerabilities that are associated.

Track your assets

When carrying out continual learning, validate updates as if they are new models or datasets. Does your model still work in the way you would expect?

Why this principle?

Continual learning introduces an attack vector for manipulating a model's behaviour. In the past, systems that learn via unmonitored user interaction have been affected by malicious training data injection (eg Microsoft Tay).

Once a model has been retrained, it is *not* the same as the old model and may well give quite different results, particularly on the edges of the old model's competence. It should therefore be treated as a new model. The principles in the 'development' part of the lifecycle (section 2) highlight some key points that need to be considered when creating or developing a new model, including when gathering data. Testing processes should be in place to prevent external interactions having a negative effect on your model's behaviour, and updates may need to be rigorously tested in the same way as the original model release. The extent of this testing should be chosen depending on the specific system, and may be lower for local updates that would affect only a single instance of a model.

Updates to your model and dataset should be validated, tracked and documented in the same way as they are during your development process outlined in principles 2.2 and 2.5 (Requirements and development: secure your supply chain and track your assets). This allows you to monitor for anomalous updates from continual learning manipulation. You will also be able to react to attacks, for example, by having the ability to roll back in the case of collecting poisoned or mislabelled data. This will help protect against the risks of malicious manipulation and continual forgetting highlighted in principle 4.1 - In operation: design for security.

What are the goals of this principle?

- Updates made to your model and dataset throughout the operational life of an ML component are tracked in line with requirements established during their development.
- Updates to your model made from continual learning processes are identified and rectified if they are negatively affecting your model or compromising the integrity of your model's performance.
- You have mechanisms in place to identify new data that may have an adverse effect on your model.

How could this principle be implemented?

Approach	Description	Guidance and links to further reading
Strengthen your system	Design your architecture to ensure that certain performance targets are achieved before a newly updated continual learning model goes into production.	The process of creating CL models is likely to be automatic. It's important that this process only allows new models to enter operation when given parameters/performance is met.
	As a part of your CL data collection and model retraining process, it's important to implement tracking and filtering of the data. Filters can be placed at check points throughout the CL data collection process for sanitising and cleaning incoming data. This can prevent your model being trained on data that is unwanted or deliberately malicious. Within the pipeline, consider introducing a standard set of transformations and augmentations that will standardise an input for model retraining. Specific transformations, augmentations or dictionaries can be introduced over time to minimise new adversarial risks.	The filters that you require for sanitising your CL data will depend on the application type, where you expect CL data to come from, and how much you can trust that source (eg, are they an authorised, vetted user.) Example filters could include: • using hateful word detection in an NLP application • taking pixel values averages for images What happens to collected data that is filtered out depends on the application and will likely need regular review by humans. Filters will also need regular review and may need updates if adversaries find ways to bypass them.
Strengthen your approach / processes / development	Updates made to datasets and models during continual learning should be captured in their associated metadata created during the development stage of the lifecycle.	Creating associated metadata for an asset is discussed in principle 2.5 - Requirements and development: track your assets.
	Encourage manual/human testing of models and filtering of dataset updates.	Your application and risk appetite will determine how often your CL updates require manual testing. This may be at each release of an updated/retrained model. Where automated processes are used, consider using the tracked model metadata and parameters as an alert system. Significant divergence from the original or previous update of an asset may indicate anomalous or malicious behaviour. When a metric diverges by a given amount, a process can be triggered to request human verification.
	During CL you're collecting new data and this should be treated as such. The same security concerns that are present during the gathering of data in the development stage are also present here.	The risks when gathering data are discussed in principle 2.2 - Requirements and development: secure your supply chain - apply the same principles during CL. It's especially important to think about how much you can trust your data source. Are they trusted users of a system, or can anyone provide new data points?

End of life

The anticipated end of your system's life should be defined during initial requirements creation and recorded in its associated metadata (for example, by using a model card). How an asset is decommissioned will depend on several factors, however it will likely end in archiving or destruction. A key decision to make at this stage is on the retention of data, ensuring that data is archived for as long as needed. The other key decision is to ensure that when an asset is being destroyed, this is carried out via a process appropriate to the data the asset contained, or on which it was trained.

Minimise an adversary's knowledge

Decommission your assets appropriately.

Why this principle?

An ML system is uniquely reliant on the data on which it was trained. Assets generated in this way can be used to extrapolate information about the original data on which they were trained, whether through traditional cyber attacks or ML specific attacks such as membership inference or model inversion. It's therefore important that these assets are treated and decommissioned appropriately, using destruction or archiving methods and conditions suitable for the type of data involved and its sensitivity. This is especially important when models are deployed at the edge of a network, outside of controlled environments.

Retention, via archiving, should also be considered. If you have taken actions based on the output of an ML system, consider how long after those actions you reasonably need to be able to explain or defend them. If you need to explain something, consider whether it will be enough to have the logs, the model weights or the training data.

What are the goals of this principle?

- You understand that your model is a representation of the data on which it was trained and should be decommissioned through the same, appropriate, process.
- You know that any data collected during the operational life of the model needs to be destroyed or archived appropriately, and recognise this is particularly important if models are deployed in untrusted or controlled environments.
- Your model's expected lifespan is captured in its metadata (eg, model card) as laid out in principle 2.5 - Requirements and development: track your assets.
- Data associated with the use of the model (eg, logs) is retained in an appropriate location and for an appropriate length of time for your system or organisational auditing requirements.

How could this principle be implemented?

Approach	Description	Guidance and links to further reading
Strengthen your approach / processes / development	Review how your asset's metadata states to decommission data and the underlying model. Evaluate if this is still appropriate and compliant.	Review your asset's metadata (eg, model card). Review guidance from relevant professional bodies for the sector in which the system is deployed.
	Where it's appropriate to dispose of material, ensure this is done securely and an appropriate method is used for the nature of the asset's sensitivity.	NCSC guidance on secure sanitisation storage media
	Where it is appropriate to archive data (eg, for legal and/or compliance purposes), ensure this is done securely and an appropriate method is used for the nature of the asset's sensitivity.	The National Archives guide to archiving personal data

Enable your developers

Collate lessons learned and share with the community.

Why this principle?

The process of knowledge transfer is particularly important while ML development and technology is still in its infancy. Successful knowledge transfer will ensure each generation of products or teams continues to improve, and mature design processes and the ways of working for this technology, improving security of systems in the future. This is beneficial to all stakeholders, teams, organisations and the whole industry.

At the end of a lifecycle it's therefore important to collate lessons learned and document them for others. This review is likely to include a range of aspects and stakeholders. It's important to review security design, events and how they were dealt with. This process will differ for each application and depend on the length of the lifecycle itself. The appropriate information to document and the appropriate audience to share it with should be determined by applying principle 1.3 - Development prerequisites and wider considerations: minimise an adversary's knowledge.

What are the goals of this principle?

- You reassess security decisions taken during the design and development process, and consider how you would improved them, based on experienced throughout the operational phase
- Security incidents that occur during the operational phase of the lifecycle are documented
- Any specific information shared is done so, following appropriate security considerations

How could this principle be implemented?

Approach	Description	Guidance and links to further reading
Strengthen your approach / processes / development	Develop a vulnerability disclosure process for your system and organisation. This will allow users to report vulnerabilities to you in a responsible way.	NCSC Vulnerability Disclosure Toolkit
	Review best practice and develop a process for responsibly sharing relevant threat intelligence.	FCDO Cyber threat intelligence information sharing guide Identify and share your knowledge at appropriate forums such as industry specific conferences, blogs and journals. When deciding what information to share, ensure you consider the discussion in principle 1.3 - Development prerequisites and wider considerations: minimise an adversary's knowledge.
Strengthen your people	Implement a knowledge management system. Encourage all stakeholders to contribute to this with their lessons learnt. Ensure all stakeholders have visibility of this knowledge relevant to their role.	Provide the relevant reviews and training for each stakeholder.