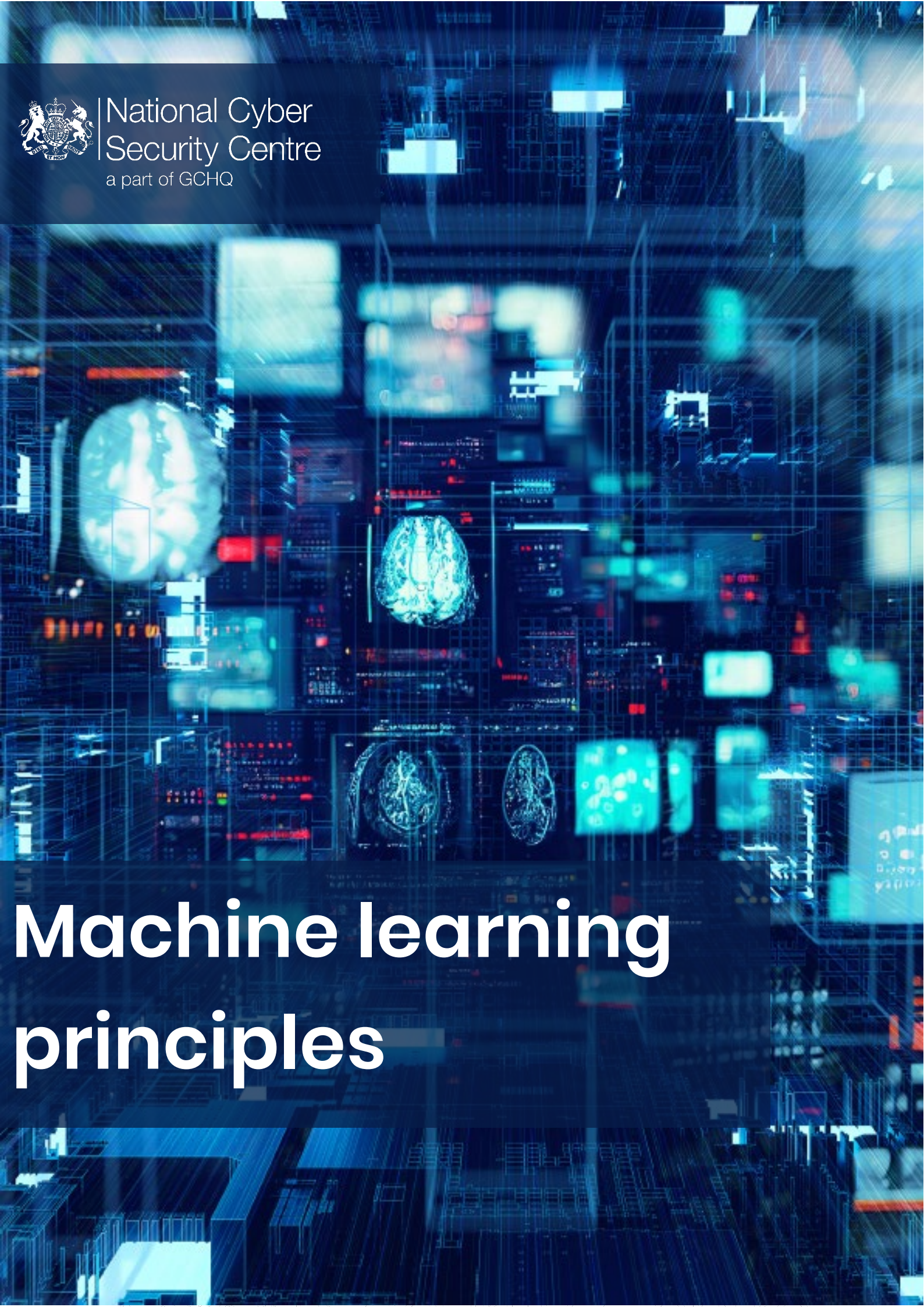


National Cyber Security Centre
a part of GCHQ

Machine learning principles

These principles help developers, engineers, decision makers and risk owners make informed decisions about the **design**, **development**, **deployment** and **operation** of their ML systems.

Contents

Introduction to the principles

The use of artificial intelligence (AI) and machine learning (ML) systems have the potential to bring many benefits to society. However, for the opportunities be fully realised, they must be developed, deployed and operated in a secure and responsible way.

AI and ML systems are subject to novel security vulnerabilities that need to be considered alongside standard cyber security threats. When the pace of development is high – as is the case with ML and AI – security can often be a secondary consideration. Designing security in from the offset is our best route to cyber resilience, which means security must be a core requirement, not just in the development phase, but throughout the life cycle of the ML system.

Without properly understanding and mitigating these vulnerabilities, system designers can't assure stakeholders, and ultimately users, that an ML system is safe and secure.

About these principles

These principles are aimed at 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 help developers, engineers, decision makers and risk owners make informed decisions about the **design**, **development**, **deployment** and **operation** of their ML systems.

The principles help underpin the NCSC's Guidelines for secure AI system development, which are aimed at providers of AI systems, whether created from scratch or built on top of tools and services provided by others.

These principles have been developed by considering a generic ML workflow, and focus on pragmatic ways to address ML vulnerabilities. This allows the principles to cover a wide range of systems that are agnostic to data type, model algorithm or deployment environment.

Note: The principles do **not** specifically cover reinforcement learning (RL) due to the fundamental differences in the way that RL applications are developed. The principles may provide a foundation when undertaking RL development, however additional security considerations (outside the scope of this document) are likely required.

Applying the principles

These principles should be considered in addition to established cyber security, risk management, and incident response best practice. In particular, we urge providers to follow the 'secure by design' principles developed by the US Cybersecurity and Infrastructure Security Agency (CISA), the UK National Cyber Security Centre (NCSC), and all our international partners. The principles prioritise:

- taking ownership of security outcomes for customers
- embracing radical transparency and accountability
- building organisational structure and leadership so 'secure by design' is a top business priority

Note that not all of the principles will be directly applicable to all organisations. The level of sophistication and the methods of attack will vary depending on the adversary targeting the ML system, so the principles should be considered alongside your organisation's use cases and threat profile.

Each principle addresses the following:

1. What are the goals of the principle?
2. Why is it important?
3. How could this principle be implemented?

The UK government's Code of Practice for Software Vendors, will ensure that security is fundamental to developing and distributing products and services, will also help in this respect.

Terminology

A few quick points on terminology before we start.

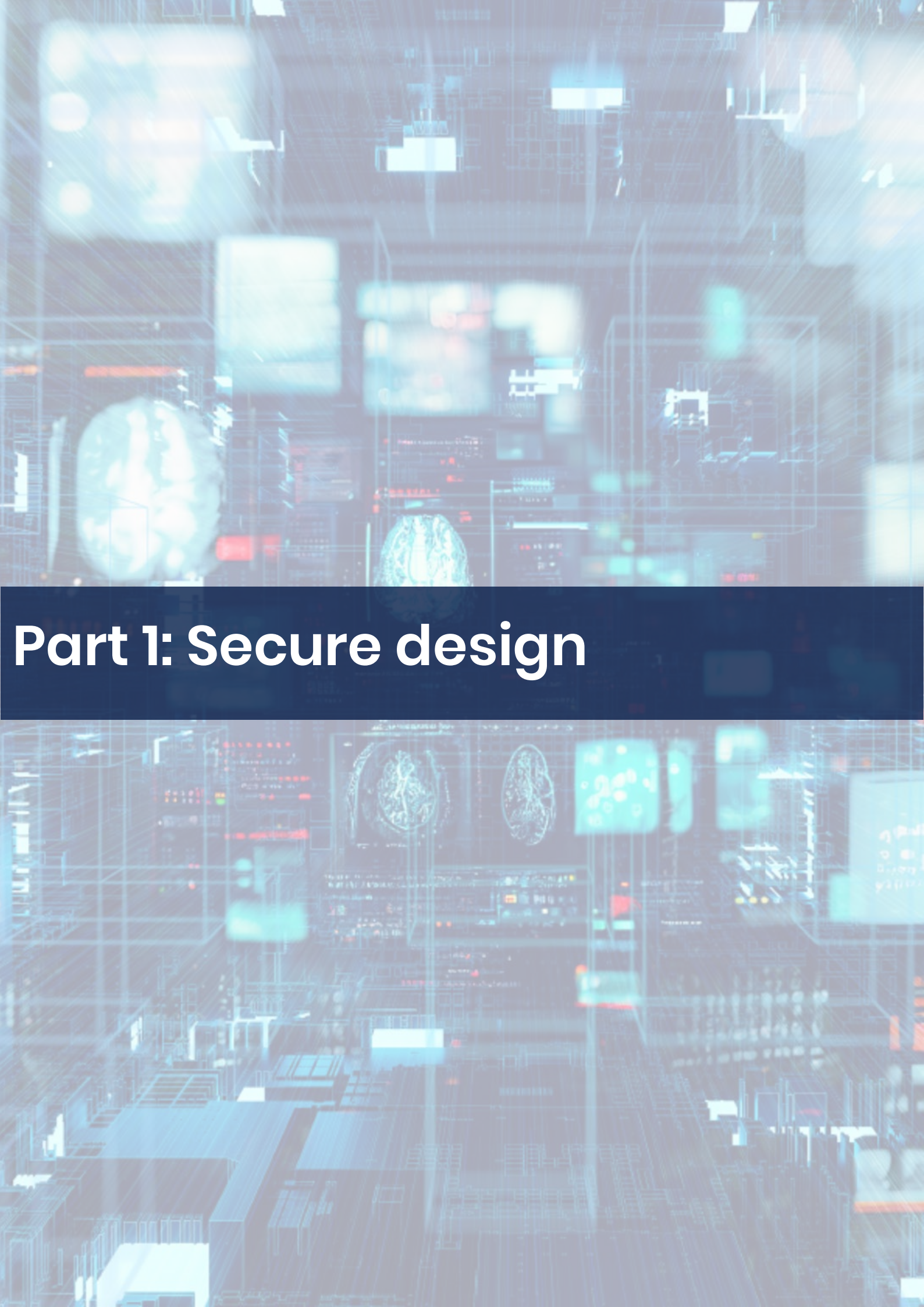
Artificial intelligence (AI) describes computer systems which can perform tasks usually requiring human intelligence. This could include visual perception, speech recognition or translation between languages.

Machine learning (ML) is a type of AI by which computers find patterns in data or solve problems automatically without having to be explicitly programmed. Almost all AI in current use is built using ML techniques.

Large language models (LLMs) use algorithms trained on a huge amount of data, turning relationships between pieces of data into probabilities to predict sequences of text (or increasingly other content) in response to user prompts.

Continual learning (CL) comprises methods of adjusting or continuing to train a model once it is in deployment, based on operational interactions and feedback.

Note: For ease of reference, we've collated all the external references used in these principles into the 'Further reading' section.



Part 1: Secure design

Part 1: Secure design

This section contains principles that apply to the design stage of the ML system development life cycle. It covers understanding risks and threat modelling, as well as specific topics and trade-offs to consider. Crucially, it also highlights the need to ensure that your staff are aware of the importance of 'secure by design', both across ML components and the wider system.

1.1 Raise awareness of ML threats and risks

Goals:

- Security is considered a key part of your ML project and integrated into your workflows at all stages of the system's life cycle.
- ML practitioners, data scientists and software developers are familiar with the inherent vulnerabilities and failure modes in ML workflows and algorithms, and follow best practices to mitigate known risks.

Why is this important?

'Security by design' is a key principle in traditional software development, and requires significant resources throughout a system's life cycle. However, adopting this approach from the start will prevent costly redesigns later. It means developers must invest in prioritising **features**, **mechanisms,** and **implementation** of tools that protect customers at each layer of the system design, and across all stages of the development life cycle.

In addition to ML-specific threats, your teams should have experience across the range of disciplines or domains required for secure system design more generally. This is particularly important when highlighting the limitations of ML components to wider system designers.

When developing systems with an ML component, it's important that threats and mitigations specific to ML systems are understood alongside non-AI software security standards. Studies - such as this one from Cornell University 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.

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.

How could this principle be implemented?

Provide guidance on the unique security risks facing AI systems

Research on ML security is fast-moving and you should make sure that practitioners receive the right support to keep their knowledge up to date, including training on the threats that are unique to ML components. For example, developers should be aware of the different types of threats their ML systems may be vulnerable to, which include:

- **Evasion attacks**: techniques to evade the correct behaviour of an ML system by providing a specific input designed to cause a failure. Examples include adversarial examples and prompt injection.
- **Poisoning attacks**: introducing malicious data into a model's training process to make the model malfunction given a specific input, degrade performance more generally, or introduce backdoors that can be exploited later.
- **Privacy attacks**: obtaining confidential information by repeated or targeted querying of a model. A model extraction attack attempts to estimate model parameters or build a copy of the model. A data extraction attack attempts to determine the data a model was trained on (membership inference) or extract samples of training data from a deployed model (model inversion).

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:

- Microsoft's blog on Failure Modes in Machine Learning
- The National Institute of Standards and Technology (NIST) Adversarial ML: A Taxonomy and Terminology of Attacks and Mitigations (NIST AI 100-2 E202 3)
- Germany's Federal Office for Information Security (BSI) AI Security Concerns in a Nutshell report
- The MITRE ATLAS framework (ATLAS is a knowledge base of adversary tactics, techniques, mitigations 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)

Enable and encourage a security-centred culture

Developers aren't necessarily 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 right from the start of a project.

In addition, security is not always part of formal data science or ML course curricula, and expertise in these areas may not translate to knowledge of system security. Encourage a cross-discipline culture that recognises the need to work collaboratively with security

specialists and subject matter experts, and put in place processes and procedures to share knowledge and experience where disciplines meet and overlap.

A security-centric culture recognises the importance of training and requires all stakeholders involved in the ML life cycle, 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 secure development and deployment. In addition, the European Union Agency for Cybersecurity (ENISA) has published a framework for good cybersecurity practices for AI.

Encourage best security practices by making sure security reviews are integrated into your system life cycle processes. Each application's development life cycle 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 you use a machine learning operations (MLOps) approach to automate workflows, ensure that this includes the security of the products you are developing

If teams are aware of and accountable for security-related responsibilities through the life cycle, 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.

1.2 Model the threats to your system

Goals:

- 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 (including reputational damage for your organisation).

Why is this important?

An ML model is often just one of many components in a system, and how it interacts with the wider system should be considered when planning any ML deployment. It is important to understand how an attack on the confidentiality, integrity and/or availability of an individual ML component impacts the wider system, and to use this understanding to

inform design decisions up and down stream 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 guarantee complete security against attackers, whose are always developing new techniques. New attacks on ML systems are demonstrated regularly, and there are significant limitations to existing defence methods. So it's important to ensure that you understand the potential impact on your wider system, should an ML component fail. Scenarios to consider 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.

How could this principle be implemented?

Create a high-level threat model for the ML system

You can create a high-level threat model to gain an initial understanding of the wider system implications of any attack on your ML component. Assess the implications of ML security threats and model failure modes across the entire system. A baseline threat model such as that suggested by OWASP can then be refined during development.

Use the CIA (confidentiality, integrity, availability) triad and the high-level threat model to explore the system. 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.

An effective way of capturing and sharing the necessary ML model information may be to treat it as a component in the wider system. Documenting inputs and outputs and highlighting limitations (such as uncertainties and value bounds) can be an effective way to do this.

Model wider system behaviours

It's important that both ML expertise and wider system/domain expertise are used in system design, and that people understand the limits of your ML components.

A common design pattern is to implement layers of countermeasures that work in concert to determine genuine requests or identify dubious activity. This could include logic / rule-based controls outside of the model itself, for example:

- **expert systems**: use explicitly defined 'if-then' rules
- **subjective logic**: a type of probabilistic logic that factors uncertainty into decisions
- **decision trees**: a branching structure where each node represents a test, the outcome of which dictates the next test

Finally, consider **user access** and design your system accordingly. A web-hosted system whose source code is publicly available may need more defensive measures than a closed, proprietary system that can only be accessed through a controlled interface.

The NCSC has guidance on understanding system-driven risk management. Also, NIST provides risk management guidance with a focus on AI.

1.3 Minimise an adversary's knowledge

Goals:

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

Why is this important?

Sharing information about system designs, development processes and research findings can be beneficial to the whole ML user community. Similarly, responsible disclosure following an attack is good practice and is likely to improve ML security throughout the industry. However, reconnaissance is often the first stage in an attack, and publishing too much information about model performance, architectures and training data can help an adversary to develop attacks.

For example, If you make it known that you are developing a model by fine-tuning an open source base model (transfer learning), you may make yourself more vulnerable to an attack which has been developed on a surrogate model with the same parent model as the victim. A similar risk arises from the use of foundational models (for example large language models); if it is known that a variant of a certain open source LLM is being used to drive a particular service, then the knowledge about the parent model can be used to devise a transfer attack.

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 where a model or system may be used in the 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.

How could this principle be implemented?

Develop a process to review information for public release

Creating the right process starts with identifying what type of information needs to be considered. Examples may include marketing material, privacy policies and an organisation's contributions to academic literature or open source software. Assess the risk that publication 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.

Your process will reflect 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 attacks that were carried out using publicly available information (as discussed in the ProofPoint Evasion and GPT-2 Model Replication case studies)
- the information to be released against the vulnerabilities in your system
- how releasing the information will benefit your organisation, system or the wider community
- whether the intended message of publication is worth the risk of any potential negative security consequences

Brief the risks to non-technical staff

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 mentioned above. When briefing staff, the aim should be to make sure they understand what material should be reviewed, and how to do this.

1.4 Analyse vulnerabilities against inherent ML threats

Goals:

- You understand potential defences and mitigations for the ML-specific attacks, and have incorporated them into your system design.
- You use your new attack and mitigation-related knowledge to update the initial threat model created in Principle 1.2.
- You understand the importance of evaluating the model and all design decisions throughout the development process, and evaluate your security requirements before a model goes into production.

Why is this important?

Identifying specific vulnerabilities in your intended workflows or algorithms at the design stage helps you reduce 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, for a query-based chat assistant model, allowing unconstrained user inputs could enable a prompt injection attack, letting an attacker gain unintended access to system instructions or private data. A more secure approach would be to restrict or filter user inputs and limit the model's ability to access sensitive data.

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.

How could this principle be implemented?

Implement red teaming

In cyber security, red teaming means playing the role of an adversary to try and compromise the confidentiality, integrity or availability of a system and provide feedback on its vulnerability.

Some large technology companies (such as Google) now have 'AI Red Teams' dedicated to probing AI/ML systems for vulnerabilities. Though not all organisations will have the resources for that, applying a 'red teaming mindset' (without necessarily running a full red teaming exercise) can help inform security requirements and drive design decisions.

More information on red teaming can be found in the MOD red teaming handbook. Again, the emphasis is on applying a red teaming *mindset* rather than a full red team operation.

The attacks described in the resources mentioned previously (Microsoft's blog on Failure Modes in Machine Learning, NIST's AML Taxonomy, the MITRE ATLAS knowledge base and the BSI's AI Security Concerns in a Nutshell report) are a good foundation for thinking about ML vulnerabilities, while the Language model Vulnerabilities and Exposures (LVEs) project aims to raise awareness about vulnerabilities in state-of-the-art LLMs. You can establish detailed security requirements by considering these vulnerabilities in combination with your risk appetite.

Continue to apply a red teaming mindset at regular intervals throughout the system development cycle and whenever key design decisions are made. It's likely that you will

get the best results when your team has skills and expertise in different areas, for example data science, software assurance and knowledge of use cases for your product.

Consider automated testing

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. Commercial products to assess model robustness are beginning to come to market. There are also a number of open source tools that do this, which include:

- The DARPA GARD (**G**uaranteeing **A**I **R**obustness to **D**eception) Program: a selection of tools that 'seek to establish theoretical ML system foundations to identify system vulnerabilities, characterise properties that will enhance system robustness, and encourage the creation of effective defenses'.
- Azure Counterfit: a generic automation layer for assessing the security of machine learning systems. It brings several existing adversarial frameworks under one tool, or allows users to create their own.
- CleverHans: a Python library to benchmark machine learning systems' vulnerability to adversarial examples.
- AI Verify: an AI governance testing framework and software toolkit that validates the performance of AI systems against a set of internationally recognised principles through standardised tests.

Note: It's likely that testing standards will emerge as the field matures, so it's helpful to keep up with the latest advice from government's AI Standards Hub, an initiative dedicated to the evolving field of standardisation for AI technologies.

Part 2: Secure development

Part 2: Secure development

This section contains principles that apply to the development stage of the ML system life cycle. This stage can be the most complex, and it's important to recognise that security vulnerabilities introduced here will propagate throughout the life of your system. You will also think about security of your data, people and processes at this stage, not just your system design.

2.1 Secure your supply chain

Goals:

- You understand your supply chain and have sufficient transparency, traceability, validation and verification processes in place to ensure it can be trusted.
- You understand that your data and its labels determine the logic of your model.
- You understand that the risk of supply chain contamination heightens when you use models that don't have a transparent creation method.
- You understand the risk posed by insider threats.

Why is this important?

An ML model is only as good as the data it is trained on. Whoever creates, processes and labels your data therefore has direct influence on your model's behaviour. Data collection, cleaning and labelling is an expensive process and in practice, people often rely on third party datasets. This has benefits, but it also introduces a threat vector. Data could be benignly but badly labelled, or an adversary could deliberately mislabel instances or insert triggers (a technique known as data poisoning).

When a model is trained on a poisoned dataset, its performance can degrade with serious consequences (as the poisoning attack on Microsoft's Tay chatbot illustrates). Other poisoning attacks can introduce targeted backdoors which could harm the integrity or availability of a model's outputs.

Supply chain security is even more important when you are using other people's models. In one example, researchers demonstrated how an LLM model could be made to produce erroneous output, then uploaded to a public model hub, disguised as a clean implementation of another popular model. Model files can also be exploited without affecting the model itself, for example by exploiting the model serialisation process or even the model compiler.

How could this principle be implemented?

Verify any third-party inputs are from sources you trust

You need to consider the security of your supply chain when acquiring assets, including models, data, labels and software components. Require suppliers to adhere to the same

standards your organisation applies to other 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. You can also follow advice provided by the Supply-chain Levels for Software Artifacts (SLSA) framework.

Understanding your external dependencies can help you identify vulnerabilities and risks. For example, by generating a Machine Learning Bill of Materials, you can assess all the open source and third-party components present in your ML solution.

Use untrusted data only as a last resort

If you must use untrusted data, you should be aware that techniques to find poisoned instances in a dataset (or to prevent them from having an impact) are mainly tested in academia. We therefore suggest their use only as a last resort when requirements prevent you gathering more trusted data. You should carefully evaluate their applicability and impact on your application and development process. There are also scanning tools that check code and data integrity, though these too remain imperfect. Hardening techniques can also be applied to models to detect and/or mitigate the effect of backdoors introduced by data poisoning.

Consider using synthetic data or limited data

There are a range of techniques to help you train ML systems on limited data, rather than data from untrusted or less secure sources. Some of these come with their own challenges, such as:

- generative adversarial networks (GANs) amplifying and recreating statistical distributions in the original data
- game engines introducing specific characteristics
- data augmentation being limited to manipulating only the original dataset

It's important to understand the limitations and the risks posed, as discussed in this explainer produced by the Alan Turing Institute and the Royal Society. The Defence Science and Technology Library (DSTL) publish a handbook ('Machine learning with limited data') which recommends approaches for small amounts of data, and for large amounts of unlabelled data.

Reduce the risk of insider attacks on datasets from intentional mislabelling

Ensure the level of vetting for your labellers is appropriate for the severity of impact that mislabelling could have. Refer to any industry specific guidance on personnel security and vetting, and in situations where it's not feasible to vet labellers, use processes that can limit a single labeller's influence (such as breaking a dataset into segments, ensuring that a single labeller never has access to an entire set).

Data labelling will be unique to your application, so it's important that you provide guidance and training to your labellers. Google offers useful advice on doing this. Consider whether you may benefit from the use of labelling software or a labelling service.

2.2 Secure your development infrastructure

Goals:

- You understand that datasets, models and other artefacts are crucial assets that need to be protected.
- You understand the potential impact of any theft or loss of these assets (including legal and reputational impacts).
- You know what data you hold, where it's stored and how it is being used.

Why this is this important?

Like your data and models, your development infrastructure needs to be sourced through a trusted supply chain, whether you choose to use local compute or public cloud services. Although infrastructure security is not unique to ML, it is particularly important in ML due to the possibility that a compromise at this stage could impact throughout a system's life cycle. Your digital assets represent a significant investment of resources and intellectual property, making them an attractive target for theft, while their manipulation could significantly affect your system's operation.

Securing your development infrastructure is likely to involve assessing multiple components and settings, applying fundamental cyber security best practice. It's important to ensure that software and operating systems are updated to the latest versions and that access to your environment is limited to those with a legitimate need. Access to and modification of components should be logged and monitored. Your specific use case will influence your security requirements, but you should understand that initiating development in a less secure R&D environment could result in vulnerabilities that are hard to remove if you want to move a product into production.

How could this principle be implemented?

Follow cyber security best practice and recognised IT security standards

The importance of following cyber security best practice is not unique to ML development. Sensible steps to take include protecting data at rest and protecting data in transit, patching software to the latest versions, implementing multi-factor authentication, security logging and, where necessary, using two-person controls. ISO/IEC 27001 is a globally used standard for information security management systems that promotes a holistic approach to managing cyber risks.

Follow technical advice on how to protect data on your specific infrastructure. Refer to the NCSC's guidance on platforms for advice about choosing, configuring, and using devices securely. If you are using a cloud platform then follow NCSC guidance on choosing, configuring and using cloud services securely. The Centre for Internet Security (CIS) provides configuration recommendations to help implement cyber security defences.

Access to your training environment should follow the principle of least privilege for user access. Implement access logging and monitoring in your development environment. The NCSC has guidance about logging for security purposes, and CISA now maintains Logging Made Easy software (originally developed by the NCSC).

Use secure software development practices

Secure software development practices provide steps to safeguard your software development life cycle and ways to improve the security of your ML model and/or system. See the NCSC's guidance on secure development and deployment and CISA's Secure by Design for further guidance.

Monitor common vulnerabilities associated with your development software and any code libraries you rely on. The common vulnerabilities and exposures (CVE) program provides a glossary (maintained by the MITRE corporation) that identifies, defines and catalogues publicly disclosed cyber security vulnerabilities.

Be aware of legal and regulatory requirements

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. The NCSC has guidance on approaches to securing personal data securely and the Information Commissioner's Office (ICO) publish information on the laws surrounding data protection.

2.3 Manage the full life cycle of models and datasets

Goals:

- You 'version-control' your dataset so you can (if required) roll back to a known good state.
- You have considered what metadata should be captured, and for what purpose.
- You know the intended use of the dataset, including where it can and can't be used.
- You know when your data needs to be deleted.
- You are aware of any biases in your data.
- You know what data you hold, where it is stored and who can access it.
- You recognise the need to manage technical debt as early as possible.

Why is this important?

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 life cycle of an ML solution. This 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.

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 life cycle. This can help implement the tracking of model behaviour, monitoring and logging user queries as outlined in Principle 2.2.

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.

How could this principle be implemented?

Use version-control tools to track and control changes to your software, dataset and resulting model

The size of your project assets (eg, datasets) may limit your choice of technologies. There are a range of tools to choose from, ranging from the general purpose (Git) to the more ML-specific (Data Version Control (DVC), MLflow, Comet, Aim). It will be for your development team to select the best for your use case, but it 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.

Use a standard solution for documenting and tracking your models and data

The data from a Model Card and a Data Card is complementary. Whilst the former contains information about the model, the latter contains a summary of the data's life cycle that the model was trained on. The ML-BOM (a variant of the software bill of materials) uses dataset and model-related metadata to form a single, comprehensive, and updatable document that is designed to support transparency and accountability.

Track *dataset* metadata in a format that is readable by humans and can be processed/parsed by a computer

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
- recommended retirement/destruction method of the data
- aggregated stats about the data (eg, if labelled, count of each label)

A popular method to provide transparent and human-centred dataset documentation is the use of 'Data Cards', which provide structured summaries of essential facts about ML datasets across a project's life cycle.

Track *model* metadata in a format that is readable by humans and can be processed/parsed by a computer

The 'Model Cards' format is becoming an increasing popular framework to track versions as ML models are updated or trained with different datasets throughout development or continual learning through operation.

Regardless of the chosen tool, 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)
- recommended retirement/destruction method of the model

Model cards stored alongside the model in a searchable (indexed) format allows developers to find an appropriate model easily and to compare models. GCHQ have released a framework ('Bailo') on GitHub to allow this.

Ensure each dataset and model has an owner

Development teams should include roles responsible for overseeing and owning risk. Depending on the size of the team, it may be useful to establish a specific role with responsibility for managing digital assets, as part of an MLOps approach.

Whether or not a specific role is appropriate, every artifact created should have an owner responsible for ensuring its life cycle is managed securely. Ideally, this should be someone involved in creating the asset. Their contact details may be captured in the asset's metadata, although in doing this, consider the security implications of sharing personal information, especially if in the public domain.

Following these best practices can help minimise technical debt.

2.4 Choose a model that maximises security and performance

Goals:

- You understand that different models are suited to different tasks and the importance of benchmarking a model's suitability for a given task.
- You understand that different models have different vulnerabilities, and the risks associated with using MLaaS and foundation models.
- The complexity of your model 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 understand the potential security issues that overfitting and underfitting models can cause.

Why is this important?

Once you are confident that the task at hand is most appropriately addressed using AI, it is important to consider which model should be used. You should consider security as well as performance; choosing a sub-optimal model may result in poor performance and be open to exploitation by attackers.

It's important to properly evaluate whether to use externally pre-trained models or machine learning as a service (MLaaS) platforms. While their use eliminates many of the technical and economic costs of implementing large resource heavy models, they come with their own security characteristics which may leave you vulnerable to transfer attacks that have been designed for the base model.

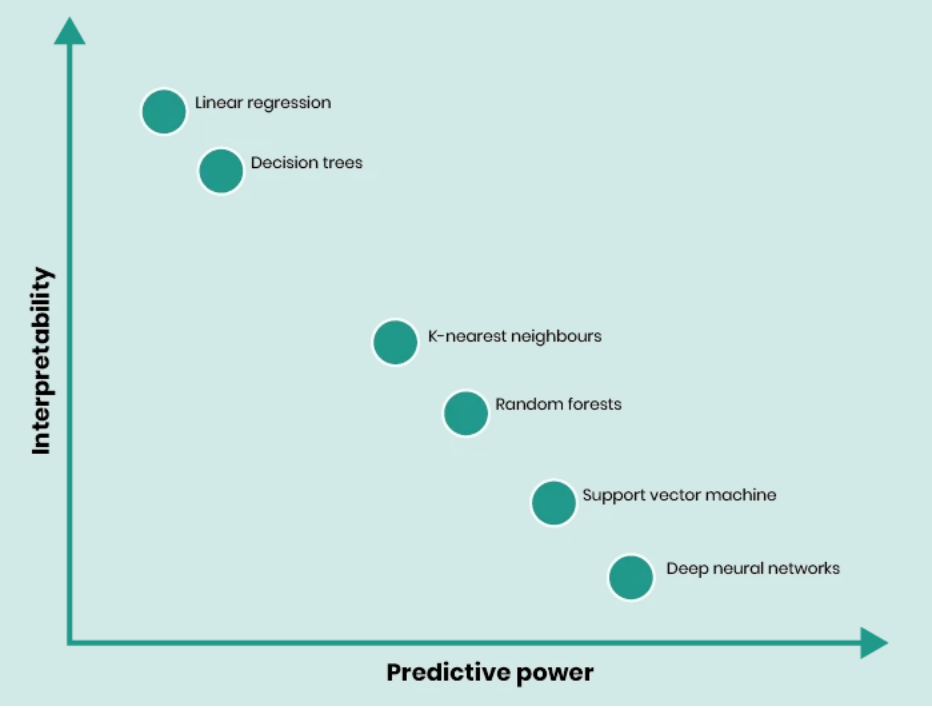
Model capacity describes the complexity that a model can express and is dictated by the architecture choice (such as a neural network versus a decision tree) and size of the model (ie the number of tuneable parameters). For a given model to perform well, it's important that its capacity fits the dataset size and variety. A model with capacity that is **too low** for a task won't generalise well and will perform poorly. In contrast, a model with a capacity that is **too high** for the dataset may perform and generalise well, but may be more vulnerable to exploitation by (for example):

- increasing the likelihood of training data memorisation
- giving attackers extra capacity in which to hide malware
- increasing the proportion of feature space in which the model behaves unreliably (if the training data does not adequately cover the feature space)

To validate your model for security or robustness, it's important to understand (and be able to explain) its behaviour. This allows you to:

- detect anomalous behaviour
- better predict how the model will react to Out of Distribution (OoD) inputs (that is, 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

Some model architectures are harder than others to interpret and understand, so your design choice must be justified (see the illustration below for details). The use of complex architectures such as neural networks should be a deliberate choice based on the requirements of your project.



The relationship between interpretability and predictive power of various machine learning architectures.

How could this principle be implemented?

Consider a range of model types on your data

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. Model selection is the data scientists/ML practitioner's job, and they must be focused on fulfilling requirements with no bias towards the latest algorithms.

Consider supply chain security when choosing whether to develop in house or use external components

For example, when assessing the suitability of MLaaS platforms or pre-trained models from model hub pages (such as Hugging Face), ensure that pre-trained models come from reputable sources. Use vulnerability scanning tools (such as Pickle scanning) to check for potential threats.

Review the size and quality of your dataset against your requirements

Rules of thumb for a evaluating the required **size** for a dataset will depend on your specific context and may 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 Dataquest tutorial

Common metrics for evaluating a dataset's **quality** include completeness, timeliness, validity, consistency, integrity and balance between classes. Where you have insufficient data to train a model from scratch, possible approaches include gathering more data, using transfer learning, augmenting existing data and synthetic data generation. Selecting an algorithm that runs well on limited data may also prove useful. The UK's Defence Science and Technology Laboratory have guidance on ML techniques for limited data problems.

Consider pruning and simplifying your model during the development process

Simple models may achieve sufficient performance in many applications, but if a complex neural network model is required, a range of techniques exist for reducing its size. Many fall under the umbrella of 'pruning' (as described in this PyTorch tutorial), which involves removing unnecessary neurons or weights after the model is trained. Be aware, however that pruning can impact performance and has mostly been explored in an academic research context.

Part 3: Secure deployment

Part 3: Secure deployment

This section contains principles that apply to the deployment stage of the ML system life cycle, which includes protecting the system from a range of attacks that include:

- evasion attacks, designed to make a model misclassify specific examples (such as adversarial data) or produce unintended outputs (LLM prompt injection attacks)
- model extraction attacks, when an attacker uses repeated querying to build a copy of a model
- model inversion attacks, when an attacker aims to identify or reconstruct data on which the model was trained
- data extraction attacks, when an attacker attempts to infer training data membership (such as a membership inference attack), or extract full training samples from the deployed model (such as model inversion or LLM prompt extraction attacks)
- availability attacks, when an attacker deliberately manipulates training or inference data to degrade the performance of the system to cause a denial of service

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

3.1 Protect information that could be used to attack your model

Goals

- You understand what information is available to users, and how attackers could exploit this.
- You understand the trade-off between transparency and security 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.
- You create system configuration options that provide defences against common threats.

Why is this important?

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 'open box' (when an attacker has complete information about a model's architecture, weights and biases) through to 'closed 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 its data input 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 an open box attack.

An example of an information extraction attack is model stealing, which involves the creation of a substitute model to imitate a target, such as an ML as a Service (MLaaS) model. Data is collected by querying the target then inferring information about the model's architecture and training parameters.

During operation, providing users with visibility of a model's output can allow for quick diagnosis that a system is behaving unexpectedly. 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 'standard' user querying the system, an engineer servicing the system, and a system admin diagnosing errors. Each class 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.

How could this principle be implemented?

Ensure your model is 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. You 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.)

Due to the recent popularity of LLMs, prompt injection attacks have become a concern. There are a number of mitigations that can be put into place including limiting what can be entered as a prompt. However, it is recognised that such mitigations cannot eliminate the likelihood of an attack, so you should exercise caution when using LLMs. You may still want to consider access controls here (such as role-based access control), enabling different levels of insight depending on user authorisation.

Use access controls across 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? Two popular implementations of access controls are role-based access controls (RBAC) and attribute-based access control (ABAC).

Limit higher-fidelity access to model outputs to authorised users in particular roles, such as trusted maintainers and developers. 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 they be with lower accuracy, or provided at a slower rate (ie lower frequency of queries)?

Configuration settings (such as model output settings per role) should be assessed in conjunction with the benefits they derive, and any security risks they introduce. Ideally, the most secure configuration will be integrated into the system as the only option. When several configuration options are necessary (ie a configuration per role), the default option should be broadly secure against common threats (that is, secure by default).

3.2 Monitor and log user activity

Goals

- 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.
- You have a well-defined set of procedures for managing security incidents.

Why is this important?

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. With the rapid adoption of LLMs prompt injection attacks have been developed, where specific queries to online APIs are able to bypass set restrictions and allow interactions with models in unintended ways.

How could this principle be implemented?

Consider automated monitoring of user activity and logs

Understanding how suspicious behaviour might look different to expected behaviour is key. 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. For example, there are known suspicious behaviours related to prompt injection attacks, so defensive measures can be designed accordingly.

Flagged behaviour should be retained for audit and reviewed regularly for evidence of suspicious behaviour. The NCSC has guidance about logging for security purposes, and

CISA now maintains Logging Made Easy software. Note that human reviewers can be used as part of routine audits or to examine behaviour identified as suspicious. When the review contains personal or sensitive data, appropriate guidelines and security procedures must be followed for handling and viewing data.

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 a closed box attack.

Part 4: Secure operation

Part 4: Secure operation

This section contains guidelines that apply to your ML component once in operation. Continual learning (also known as online learning) comprises methods of adjusting or continuing to train a model after deployment, based on operational interactions and feedback. This adds an extra level of complexity to a system an can bring security benefits as well as new opportunities for attackers.

4.1 Understand and mitigate the risks of using continual learning (CL)

Goals:

- You understand whether CL is appropriate for your specification and system.
- You understand that CL can be exploited, and the potential impact of attackers influencing your model.
- Updates to your models and datasets throughout operation are tracked in line with requirements established during development.
- You have mechanisms in place to identify new data that may have an adverse effect on your model. Updates from CL are identified and rectified if they negatively affect your model.
- You understand that CL carried out on a local system will only affect one instance of the model, and have considered if you want each specific instance of your system behaving slightly differently.

Why is this principle important?

The use of CL can be a crucial part of 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.

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, or 'forgotten', in certain circumstances. 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 life cycle 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.

CL complicates the tracking of dataset and model updates as outlined in principle 2.3 and makes supply chains more difficult to secure. You should be aware of these risks and have systems or processes in place to mitigate them. Updates to your model and dataset should be validated, tracked and documented in the same way as they are during your development process. This allows you to monitor for anomalous updates from CL manipulation. You should 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.

How could this principle be implemented?

Develop effective MLOps so performance targets are achieved before an updated model goes into production

Creating up-to-date CL models should be an automated process that is driven by MLOps best practice and includes suitable security monitoring. This should allow the tracking of data throughout the CL process including the ability to trace the source of an error or attack (such as poisoning). It's also important to be able to identify and manage 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.

There are lots of sources that describe how to conduct effective MLOps and several tools that can be used to create an effective MLOps environment. These include Continuous Machine Learning (CML), Data Version Control (DVC) and MLflow.

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. 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.

Consider using a pipeline architecture with checkpoints for testing

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 be to use sandboxed environments to run updated models for 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.

Capture updates to datasets and models in their associated metadata

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. 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? Follow guidance for creating metadata for an asset discussed in principle 2.3.

Consider processing data and training locally

If you don't intend to update a central/global model, CL can be carried out locally on a user's device without the updated model ever leaving the device. When you intend a local instance to contribute to updating a central model, your solution may include techniques such as federated learning. This blog by the government's Responsible Technology Adoption Unit highlights some considerations related to privacy, security and federated learning.

4.2 Appropriately sanitise inputs to your model in use

Goals:

- You handle and process inputs to your model in ways that help it operate securely.
- You protect sensitive data in line with appropriate legal, regulatory and ethical guidance.

Why is this principle important?

Appropriate sanitisation or pre-processing can help protect your model from attacks that rely on specially crafted inputs to create an adversarial effect. An example of this would be an evasion attack that used tiny changes to an input image to fool an object detection model. In some cases, sanitising the image using a simple jpeg compression could prevent this type of attack from working successfully.

If your model is trained using sensitive data (either prior to release or by CL once in use), then an attacker could try and extract this information. Properly sanitising or anonymising data could reduce the impact of a data extraction attack, and potentially make an attacker less likely to conduct one in the first place.

CL can cause particular complications around protecting privacy, as it requires collecting data directly from users.

Implement tracking and filtering of data

As a part of your data collection and MLOps processes, filters can be used for sanitising and cleaning incoming data. This can prevent your model being exposed to data that is unwanted or deliberately malicious. Consider introducing a standard set of transformations and augmentations that will standardise an input for model retraining when in use.

The filters that you require will depend on the application type, where you expect data to come from, and how much you can trust that source (in the case of CL, are they an authorised, vetted user?).

Examples could include:

- using hateful word detection in an NLP or LLM-based application
- taking pixel value averages for images
- identifying outlying data points

What happens to filtered data will depend on the application and it will likely need regular review by humans. Filters will also need regular review and may need updates if adversaries find ways to bypass them.

Implement out of distribution detection on model inputs

Your development team should choose the best OoD detection for your application. Options include using maximum softmax probability and temperature scaling. If you have a complex model with large feature spaces that are OoD, it may be worth exploring whether you can factor the OoD distance into the model's confidence scores. In some cases, this could act as detection mechanism for adversarial attacks. Further information on OoD detection can be found in this article from Encord.

Use appropriate techniques to anonymise user data

This allows you to collect data while protecting privacy, reducing security concerns. Data anonymisation techniques include:

- Differential privacy: using statistical techniques to ensure privacy without affecting the dataset's overall statistical integrity.
- Data masking: a range of techniques that represent a point in a different way that's usually unreadable to a human, such as 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.

4.3 Develop incident and vulnerability management processes

Goals:

- You have a process in place for securely sharing information about incidents with the appropriate authorities.
- Following security considerations, you participate in information-sharing communities, collaborating with industry, academia and governments to share best practice.
- You have mechanisms in place so that anyone discovering a vulnerability in your model is able to report it back to you.
- You provide consent to security researchers to publish and report vulnerabilities.

Why is this principle important?

Like any other software, ML systems will contain vulnerabilities, and having proper vulnerability management processes will help address the risks associated with using them. The fast-developing nature of ML, and the increasing integration of ML components into critical systems, mean that sharing knowledge about vulnerabilities and incidents is particularly important.

Your ability to receive information from users about possible vulnerabilities in your own ML products will allow you to address security issues. In some sectors this may also help you comply with requirements or legislation linked to secure data handling.

How could this principle be implemented?

Develop a vulnerability disclosure process for your system and organisation

Vulnerabilities are discovered all the time, and security specialists want to be able to report them directly to the organisation responsible. These reports can provide you with valuable information that you can use to improve the security of your systems.

By providing a clear process, organisations can receive the information directly so the vulnerability can be addressed, and the risk of compromise reduced. This process also reduces the reputational damage of public disclosure by providing a way to report, and a defined policy of how the organisation will respond. The NCSC's Vulnerability Disclosure Toolkit contains the main components required to set up your own vulnerability disclosure process.

Develop a process for responsibly sharing relevant threat intelligence

The NIST Guide to Threat Information Sharing provides guidance on establishing and participating in cyber threat information sharing relationships. Identify and share your knowledge at appropriate forums such as industry specific conferences, blogs and journals.

Note that a real-life adversarial attack can be used to create a case study through MITRE ATLAS - Contribute. Principle 1.3 discusses benefits and risks of publishing specific detail about your systems. Nevertheless, there are likely to be various ways to share enough information to support others in developing secure ML, even if key sensitivities are removed.

Report incidents to relevant stakeholders and authorities

Follow the NCSC's advice for reporting and managing incidents and if appropriate report the incident to the NCSC. Where possible, share details about the incident to help build knowledge across the ML development community. One way to do this would be to create a case study in MITRE ATLAS, as discussed above.

Part 5: End of life

Part 5: End of life

How an asset is decommissioned at the end of life will depend on several factors. However it will likely mean archiving (for as long as is needed) or destruction. If an asset is being destroyed, this should be carried out via a process appropriate to the data the asset contained, or on which it was trained. At the end of the life cycle, it's also important to collate lesson's learned.

5.1 Decommission your assets appropriately

Goals:

- Your model's expected lifespan is captured in its metadata as laid out in principle 2.3.
- You understand that your model is a representation of the data on which it was trained and should be decommissioned through an appropriate process.
- Data associated with the use of the model (such as logs) is retained in an appropriate location and for an appropriate length of time for your system or organisational auditing requirements.

Why is this principle important?

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.

How could this principle be implemented?

Decommission assets using destruction or archiving methods

Review how your asset's metadata states to decommission data and the underlying model. Evaluate if this is still appropriate and compliant.

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. This is especially important when models are deployed at the edge of a network, outside of controlled environments. The NCSC has guidance on secure of sanitisation storage media.

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 will need to explain something, consider whether you will need access to security logs, model weights or training data. The National Archives have guidance on archiving personal data.

5.2 Collate lessons learned and share with the community

Goals:

- You reassess design/development security decisions, and consider how they could be improved based on the operational phase.
- Information about security incidents that occurred during the operational phase of the life cycle are shared appropriately with others.

Why is this principle important?

ML technology is developing extremely quickly and knowledge transfer is particularly important to ensuring that products, teams and processes continue to improve. At the end of a life cycle it's therefore important to collate lessons learned and document them for others. This process is likely to consider security design choices and review any security events observed while the model was in development or operation.

How could this principle be implemented?

Responsibly share relevant threat intelligence and lessons learned

Sharing lessons learned is particularly important at the end of a model's life. As discussed in principle 4.3 above, there may be various suitable ways to do this including industry conferences, written publications and MITRE ATLAS case studies.

Use a knowledge management system to identify, organise, store and share information relating to the lessons learned

Encourage all stakeholders to contribute to the knowledge management system with their lessons learned and ensure that they have visibility of the knowledge related to their role. Provide the relevant reviews and training for each stakeholder. Ensure all stakeholders have visibility of this knowledge relevant to their role.

Further reading

Further reading

This section collates the references included in these principles. Note that:

- References that occur more than once are only included where they first appear.
- The MITRE ATLAS techniques represent how an attacker achieves a tactical objective by performing an action.

Part 1: Secure design

https://www.ncsc.gov.uk/collection/guidelines-secure-ai-system-development
This document recommends guidelines for providers of any systems that use artificial intelligence (AI)

https://www.cisa.gov/resources-tools/resources/secure-by-design
This joint guidance urges software manufacturers to take urgent steps necessary to ship products that are secure by design.

https://arxiv.org/abs/2207.05164
A report that sheds light on real-world attacks on deployed machine learning.

https://atlas.mitre.org/techniques/AML.T0043/

https://atlas.mitre.org/techniques/AML.T0051/

https://atlas.mitre.org/techniques/AML.T0018.000/

https://atlas.mitre.org/techniques/AML.T0024.000

https://atlas.mitre.org/techniques/AML.T0024.001

https://learn.microsoft.com/en-us/security/engineering/failure-modes-in-machine-learning
A blog from Microsoft.

https://csrc.nist.gov/pubs/ai/100/2/e2023/final
This NIST Trustworthy and Responsible AI report develops a taxonomy of concepts and defines terminology in the field of adversarial machine learning (AML).

https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/KI/Practical_AI-Security_Guide_2023.pdf?__blob=publicationFile&v=5
This guideline from Federal Office for Information Security introduces developers to the most relevant attacks on machine learning systems and potential complementary defences.

https://www.ncsc.gov.uk/collection/board-toolkit/developing-a-positive-cyber-security-culture
A chapter from the NCSC's Board Toolkit, which helps board members to govern cyber risk more effectively.

https://www.ncsc.gov.uk/collection/developers-collection
8 Principles to help you improve and evaluate your development practices, and those of your suppliers

https://www.enisa.europa.eu/publications/multilayer-framework-for-good-cybersecurity-practices-for-ai
ENISA's scalable framework to guide NCAs and AI stakeholders on the steps they need to follow to secure their AI systems.

https://owasp.org/www-community/Threat_Modeling
The OWASP® Foundation works to improve the security of software through its community-led open source software projects.

https://www.cisecurity.org/insights/spotlight/ei-isac-cybersecurity-spotlight-cia-triad
The CIA Triad is a benchmark model in information security designed to govern and evaluate how an organization handles data when it is stored, transmitted, or processed.

https://hdsr.mitpress.mit.edu/pub/812vijgg/release/3
Editorial from Xiao-Li Meng, Whipple V. N. Jones Professor of Statistics, Harvard University.

https://www.ncsc.gov.uk/collection/risk-management
NCSC's guidance to help you better understand and manage the cyber security risks affecting your organisation.

https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf
Designed to equip organizations and individuals with approaches that increase the trustworthiness of AI systems.

https://www.ncsc.gov.uk/blog-post/exercise-caution-building-off-llms
Blog outlining the risks of building services that use LLMs.

https://atlas.mitre.org/studies/AML.CS0008

https://atlas.mitre.org/studies/AML.CS0007

https://blog.google/technology/safety-security/googles-ai-red-team-the-ethical-hackers-making-ai-safer/
Blog introducing Google's AI Red Team.

https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/1027158/20210625-Red_Teaming_Handbook.pdf
A practical guide for supporting individuals and teams who are faced with different problems and challenges in defence.

https://lve-project.org/
Documents and track vulnerabilities and exposures of large language models (LVEs).

https://www.gardproject.org/
A selection of tools that 'seek to establish theoretical ML system foundations to identify system vulnerabilities.

https://github.com/Azure/counterfit/
A generic automation layer for assessing the security of machine learning systems.

https://github.com/cleverhans-lab/cleverhans
a Python library to benchmark machine learning systems' vulnerability to adversarial examples.

https://github.com/IMDA-BTG/aiverify
An AI governance testing framework and software toolkit.

https://aistandardshub.org/
An initiative dedicated to the evolving field of standardisation for AI technologies.

Part 2: Secure development

https://atlas.mitre.org/techniques/AML.T0020/

https://atlas.mitre.org/studies/AML.CS0019

https://blog.trailofbits.com/2021/03/15/never-a-dill-moment-exploiting-machine-learning-pickle-files/
A blog discussing the underhanded antics that can occur simply from loading an untrusted pickle file or ML model.

https://openreview.net/pdf?id=v01xUvzem4
Paper exploring how backdoors can be added during compilation, circumventing any safeguards in the data-preparation and model-training stages.

https://www.ncsc.gov.uk/collection/supply-chain/guidance
Essential information, guidance and advice on securing your supply chain.

https://slsa.dev/
A specification for describing and incrementally improving supply chain security, established by industry consensus.

https://cyclonedx.org/capabilities/mlbom/
Model and dataset transparency for security, privacy, safety and ethical considerations.

https://huggingface.co/docs/hub/security
The Hugging Face Hub offers several security features to ensure that your code and data are secure.

https://www.turing.ac.uk/blog/what-synthetic-data-and-how-can-it-advance-research-and-development
Research carried out by The Alan Turing Institute.

https://www.gov.uk/government/publications/machine-learning-with-limited-data
A handbook from the Defence Science and Technology Library (DSTL).

https://cloud.google.com/ai-platform/data-labeling/docs/instructions
Data labelling guidance from Google.

https://www.ncsc.gov.uk/collection/cross-domain-solutions/using-the-principles/data-at-rest-protection
NCSC guidance for protecting data-at-rest.

https://www.ncsc.gov.uk/collection/cross-domain-solutions/using-the-principles/data-in-transit-protection
NCSC guidance for protecting data-in-transit.

https://www.iso.org/standard/27001
The globally used standard for information security management systems.

https://www.ncsc.gov.uk/collection/device-security-guidance
NCSC guidance for organisations on how to choose, configure and use devices securely.

https://www.cisecurity.org/cis-benchmarks
Configuration recommendations to help you protect your systems against threats.

https://github.com/cisagov/lme
Logging software maintained by CISA.

https://www.ncsc.gov.uk/guidance/introduction-logging-security-purposes
NCSC guidance to help you devise an approach to logging.

https://www.ncsc.gov.uk/collection/protecting-bulk-personal-data/what-are-you-protecting
the NCSC's 15 good practice measures for the protection of bulk data held by digital services.

https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/
GDPR guidance and resources from the Information Commissioner's Office.

https://git-scm.com/
A free and open source distributed version control system

https://technology.blog.gov.uk/2014/01/27/how-we-use-github/
A blog explaining how the UK government uses GitHub.

https://datatonic.com/insights/responsible-ai-data-model-cards/
How to use data cards and model cards to bring greater transparency between stakeholders and model development teams.

https://www.meti.go.jp/english/press/2023/0728_001.html
A guide mainly targeting software suppliers as a compilation of the advantages of introducing SBOM to companies.

https://www.ai.mil/blog_09_03_21_ai_enabling_ai_with_data_cards.html
Guidance from US Chief Digital And AI Office
https://github.com/gchq/Bailo
Software maintained by GCHQ.

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43146.pdf
A paper explaining several machine learning specific risk factors and design patterns to be avoided or refactored where possible.

https://huggingface.co/models
A platform where the machine learning community collaborates on models, datasets, and applications.

https://huggingface.co/docs/hub/security-pickle
A widely used serialization format in ML.

https://www.dataquest.io/blog/learning-curves-machine-learning/
Reducing bias and variance to build more accurate models.

https://pytorch.org/tutorials/intermediate/pruning_tutorial.html#:~:text=To%20implement%20your%20own%20pruning,apply%20%2C%20prune%20%2C%20and%20remove%20.
How to use torch.nn.utils.prune to sparsify your neural networks.

Part 3: Secure deployment

https://atlas.mitre.org/tactics/AML.TA0007/

https://atlas.mitre.org/techniques/AML.T0024.002/

https://atlas.mitre.org/techniques/AML.T0056

https://research.nccgroup.com/2022/12/05/exploring-prompt-injection-attacks/
A blog from NCC group discussing what a prompt is in the machine learning context.

Part 4: Secure operation

https://atlas.mitre.org/studies/AML.CS0009

https://ml-ops.org/content/mlops-principles#:~:text=The%20complete%20MLOps%20process%20includes,designing%20the%20ML%2Dpowered%20software.
MLOps we strive to avoid "*technical debt*" in machine learning applications.

https://cml.dev/
Continuous Machine Learning (CML) is CI/CD for Machine Learning Projects.

https://mlflow.org/
Build better models and generative AI apps on a unified, end-to-end, open source MLOps platform.

https://rtau.blog.gov.uk/2024/02/22/privacy-preserving-federated-learning-understanding-the-costs-and-benefits/
Examples of the costs and benefits associated with applying PETs to enable privacy-preserving federated learning.

https://encord.com/blog/what-is-out-of-distribution-ood-detection/#h3
Approaches to Detect OOD Instances from Encord.

https://www.ncsc.gov.uk/information/vulnerability-disclosure-toolkit
A toolkit making it easier for organisations to create a vulnerability disclosure process.

https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-150.pdf
Provides guidelines for establishing and participating in cyber threat information sharing relationships.

https://www.ncsc.gov.uk/section/about-ncsc/incident-management
Resources from the NCSC helping to reduce the harm from cyber security incidents in the UK.

Part 5: End of life

https://www.ncsc.gov.uk/guidance/secure-sanitisation-storage-media
Why sanitisation is necessary, the risks involved, and how to sanitise affordably.

https://cdn.nationalarchives.gov.uk/documents/information-management/guide-to-archiving-personal-data.pdf
Guidance from the National Archives.