

PRAISE: A Reproducible Workflow for Non-Programmer Scientists to Implement Published AI Models

Max Achterweust^{a,b}, Stefan Groot Nibbelink^{a,c}, Vincent J. de Beer^{a,d,e}, Stefan Schipper^b, Zeynep Yüksektepe^b, Pedro Andrade da Silva^b, Aurelia Peppelaar^b, Martijn J. van der Willigen^{a,e,f}, Christophe B. Minkenberg^{b,*}, Robert F. van der Willigen^{a,e,g}

^aHR Datalab EAS, School of Engineering and Applied Sciences, Rotterdam University of Applied Sciences, Rotterdam, the Netherlands

^bDepartment of Chemistry, School of Engineering and Applied Sciences, Rotterdam University of Applied Sciences, Rotterdam, the Netherlands

^cDepartment of Electrical Engineering, School of Engineering and Applied Sciences, Rotterdam University of Applied Sciences, Rotterdam, the Netherlands

^dDepartment of Biology and Medical Laboratory Research, School of Engineering and Applied Sciences, Rotterdam University of Applied Sciences, Rotterdam, the Netherlands

^eData Supported Healthcare: Data-Science Unit, Research Center Innovations in Care, Rotterdam University of Applied Sciences, Rotterdam, the Netherlands

^fSchool of Communication, Media and Information Technology, Rotterdam University of Applied Sciences, Rotterdam, the Netherlands

^gHR DataLab AI SusTech, Centre of Expertise HRTech, Rotterdam University of Applied Sciences, Rotterdam, the Netherlands

Abstract

Artificial Intelligence (AI) models are increasingly developed, published and used across scientific fields. Researchers, students and industry professionals who want to apply these models to their own research questions often lack the programming and data science expertise to do so. Existing guidelines cover single steps, but not the route from a published model to working predictions on one's own data. Here we present the hands-on workflow PRAISE, a Practical, Reproducible AI-model Implementation for Science and Education, that helps non-programmers implement published AI models step by step. Because implementations rarely work the first time, PRAISE anticipates common failures and provides a fallback route for each. As a proof of principle, three BSc chemistry students and their tutor applied PRAISE without prior programming or data science knowledge. They reproduced, validated and extended a state-of-the-art model that predicts enzyme-substrate interactions. This shows that with PRAISE, implementing an AI model from the literature becomes a learnable skill, one that turns non-programmer scientists from users into evaluators and improvers of AI models.

Keywords: Artificial intelligence, higher education, AI literacy, model deployment, low-barrier AI adaptation, AI accessibility

1. Introduction

Artificial intelligence (AI) has emerged as a rapidly advancing field, generating an increasing number of applications with societal relevance (Jumper et al., 2021; Sajadieh et al., 2026; Yang et al., 2025). Researchers, students and industry professionals, e.g. in STEM disciplines, who want to apply published AI models to a problem often encounter difficulties implementing and operationalizing these models. Implementing a published AI model in a new setting may be difficult for several reasons. Professionals may lack data-science expertise or suitable computational infrastructure. They may also be unfamiliar with the installation, training, and validation of AI pipelines. Insufficient AI literacy can further discourage professionals from initiating the implementation process (Biagini, 2025; Pinski & Benlian, 2024). Bridging the gap between the conventional scientific disciplines and data science is essential for useful and ethical AI model implementation (Pion-Tonachini et al., 2021).

This paper presents PRAISE, a Practical, Reproducible AI-model Implementation for Science and Education, a hands-on workflow that helps non-programmer scientists with a reproducible implementation of a published AI model to their own

practical application. Existing AI lifecycles and workflows describe the pipeline design, data preparation, training, validation, and deployment (Amershi et al., 2019; International Organization for Standardization, 2022b; Kreuzberger et al., 2023; National Institute of Standards and Technology, 2023; OECD, 2019), however these workflows stop short of the practical requirements for implementing an existing, literature-derived model. A different body of work addresses non-programmer scientists directly, through reporting guidelines covering data curation, representation, training, and reproducibility (Artrith et al., 2021; Walsh et al., 2021), and low-code environments for exploring machine learning (ML) (Magnano et al., 2022). These resources each address a particular part of the process. The route from a published model to a working deployment is left undescribed.

To avoid any possible confusion, we fix some terminology here that is used throughout this work: implementation is the umbrella term for the whole route of selecting, installing, running and validating an AI model. Reproduction indicates running the published model and obtaining metrics comparable to those reported. Adaptation means retraining the model on data the user adds, and/or with changes in the model pipeline. Deployment, the final stage of the AI life cycle (International Organization for Standardization, 2022b; OECD, 2019), denotes making the validated model available for new predictions in the use case, e.g. as a script or web interface.

*Corresponding author

Email address: c.b.minkenberg@hr.nl (Christophe B. Minkenberg)

The paper is structured as follows: Sections 2 and 3 supply background on how an AI model works, and the layered computing infrastructure it runs on, including the role of generative AI as a coding aid. Look them up whenever a step is unclear. The PRAISE workflow, from use case formulation to deployment, is set out in Section 4. Section 5 describes common issues and points to fallback loops in the workflow that solve them. Section 6 follows the workflow through a use case carried out by non-programmers.

Two strategies are suggested how this paper may be read: readers seeking a systematic understanding can follow the sections in sequence. Those who want to start implementing a model directly can begin in Section 4 and use the preceding two sections as reference material.

2. How an AI model learns from data, makes predictions and is validated

A predictive AI model learns relationships between input data and known outcomes during training, subsequently the model uses these learned relationships to make predictions for new data. The validity of this process depends on the quality of the input data and how the neural network is trained. Evaluation is essential to safeguard the validity. In this section the data, training process and validation of the outcomes are discussed.

2.1. Data as the basis for learning

AI models need to be trained on relevant, fair and valid data. For training to be successful, the dataset must fulfill several requirements.

2.1.1. Dataset size

The dataset should be of sufficient size to represent the underlying population. Small datasets may lead to overfitting, reducing the reliability of predictions (Rajput et al., 2023). Model performance generally improves with increasing dataset size until a saturation point is reached (Rajput et al., 2023). What constitutes a ‘small’ or ‘sufficiently large’ dataset depends on data complexity, model architecture, and task, since learning is more difficult when input data is highly variable or unstructured (Naser, 2026). As an empirical rule, a minimum number of samples per feature is often given, from 3 to 300 samples per feature, dependent on the intended use (Naser, 2026).

2.1.2. Non-ideal dataset properties

Datasets may suffer from imperfections that affect the applicability and reliability of the resulting AI models. Non-ideal data distributions take several forms:

Gaps in training data reduce the applicability and reliability of a model. For example, an AI model trained to predict chemical toxicity may perform poorly when certain categories of compounds are underrepresented or absent from the dataset. Such gaps reduce the applicability domain of the model and limit its practical relevance (Hartung & Kleinstreuer, 2025).

Biased datasets contain features or categories that are disproportionately represented relative to the true underlying population (Eid et al., 2021). In a toxicology context, a dataset dominated

by molecules with similar physicochemical properties may unintentionally specialize the model toward those compounds, causing performance to deteriorate for chemically distinct structures. J. Chen and Bao (2025) intentionally created biases in a dataset about strength of concrete beams and found that this compromised ML model accuracy and universality.

Skewness is an asymmetry in the data distribution (Monard & Batista, 2002). In the case of the toxicology example, a dataset containing 99% non-toxic chemicals and only 1% toxic compounds is skewed towards the majority class. Such imbalance is not necessarily unrealistic. Many real-world datasets, such as fraudulent transaction detection datasets, naturally contain rare minority classes (Monard & Batista, 2002). Increasing the dataset size will disproportionately benefit minority class representation (Weiss & Provost, 2001). Model performance increased when the minority classes were represented in the dataset for 50% instead of their true 19% presence (Weiss & Provost, 2001).

A related issue is the absence of confirmed negative data-points. In many datasets, negatives are constructed by random negative sampling, which can introduce false negatives into both training and validation sets (Eid et al., 2021).

2.1.3. Dataset splitting

Before usage, the data are generally split into separate subsets for testing, training and validation. This separation is important because the model should be evaluated on data that were not used to adjust its parameters. Conventional partitions are an 80:20 ratio for a two-way train-test split and 64:16:20 for a train-validation-test split, although no consensus exists on these proportions and an optimal split depends on the number of parameters in a linear regression model (Joseph, 2022). The partition itself is often generated randomly and is therefore controlled by a random seed. Reproducible splits can be made by recording and reporting the seed (Lyu et al., 2021; Wegmeth et al., 2023).

Data leakage occurs when a model is trained on information it should not have access to during training, often resulting in unrealistically high model performance (Apicella et al., 2025). The authors propose a data-leakage taxonomy, based on the place in the pipeline where the data leakage occurs. In Section 4.2 a leakage protocol checklist is proposed.

2.2. From input data to a prediction

Once a dataset has been assembled and split, the model itself determines how the input variables are converted into a prediction. This section describes the structure of a neural network, how it learns through minimizing the loss function, and which types of machine learning exist.

2.2.1. Structure of neural networks

A neural network consists of a chain of layers of nodes, sometimes referred to as neurons. Besides the input and output layers, the network may consist of several hidden layers in the case of deep learning. The length of the chain gives the network its depth.

Information is fed to the nodes of the input layer and then moves from one layer to the next, until it reaches the output layer, which gives the results of the network. Each layer receives the

previous layer's outputs as inputs and passes its own outputs to the subsequent layer. In this way, each consecutive hidden layer applies a further transformation to the representation produced by the preceding layers. This enables the network to transform input data into a representation that facilitates modelling, while optimizing its parameters to capture the underlying relationships accurately (Goodfellow et al., 2016a). Each node multiplies its inputs by weights, sums them, adds a bias, and passes the result through a nonlinear activation function to produce its output (Goodfellow et al., 2016a). This nonlinearity allows the network to capture complex relationships in the data.

2.2.2. Loss function minimization

The structure of a neural network, i.e. the number of layers, the number of nodes per layer and the activation functions, is typically fixed from the outset, while the weights and biases are optimized during training. In supervised learning, a loss function quantifies the discrepancy between a neural network's prediction and the corresponding target value, thereby measuring prediction error. A fully trained AI model has weights and biases such that the loss function is minimal on the training set. An optimal set of these parameters is obtained during the training of the network.

At the start of a training session, some typical values for these parameters are chosen (randomly or with some background knowledge). A single batch of the training dataset is then fed to the neural network and the output is determined.

During backpropagation, the prediction error is propagated backward through the network. Gradients are calculated for each weight and bias, and an optimization algorithm uses them to update these parameters and reduce the loss (Goodfellow et al., 2016a; Narihira et al., 2021). The updated parameters are then used in the next forward pass of the following batch. This process of calculating a prediction, calculating the loss, determining the gradients and updating the parameters is repeated batch by batch until the process has passed the whole training dataset. This completes one epoch of the training cycle. The number of batches per epoch depends on the selected batch size.

Full training of an AI model often involves many epochs. The loss should generally decrease over successive training epochs, although fluctuations may occur, particularly during the initial epochs (Fort et al., 2020).

2.2.3. Loss function monitoring and early stopping

To determine when an AI model is well trained, the loss function should be monitored on both the training set and a held-out validation set. When the training loss (loss function evaluated on the training dataset) continues to decrease while the validation loss (loss function evaluated on the validation dataset) starts to increase, this indicates that the model is overfitting the training data. The minimum validation loss can therefore be used to identify an appropriate point for stopping training, often using an early stopping procedure (Prechelt, 2012). The weights obtained at the selected point during training are subsequently used to calculate predictions for new, previously unseen input data.

2.2.4. Common types of machine learning

What exactly a network learns depends on the type of learning that is being employed. In supervised learning every input is accompanied by a known true value, and the network learns to produce a correct output for inputs it has not seen (Goodfellow et al., 2016b). The loss function in this case measures the error between the output scores and the desired scores, and learning adjusts the weights to reduce it (LeCun et al., 2015).

In unsupervised learning only the inputs are available, and the network learns properties of the data itself, for example to group entries by their distinct features (Goodfellow et al., 2016b). The loss function in unsupervised learning expresses how well the model captures the underlying data patterns (Goodfellow et al., 2016b).

In reinforcement learning, a neural network observes a connected environment that gives an input, after which the neural network chooses an action that changes the environment and results in a new input for the network (Kaelbling et al., 1996). In a guided trial and error effort the neural network can learn from its actions and their consequences on the environment (Kaelbling et al., 1996).

2.3. Verifying model training and predictive performance

Training diagnostics assess whether the model was trained as intended and whether the training process shows indications of failure. This includes checking whether the training was completed successfully, if the loss function decreased during the epochs in the training, and how the learned parameters behave during the training.

Weight distribution analysis indicates how weights behave during training. Adjusting the weights and assessing the outcome is how a model learns, and weights represent the strength and direction of relationships between neurons within the network, e.g. a negative number meaning a negative relationship (Goodfellow et al., 2016a). In weight distribution analysis, it is examined how abstract parameters relate to each other during or after training (Glorot & Bengio, 2010). These weights are visualized for one layer or for the whole network and plotted as a histogram (Glorot & Bengio, 2010; Li et al., 2024). The weights are plotted on the x-axis. The resulting distributions can be compared between training stages or between different training runs (Glorot & Bengio, 2010; Li et al., 2024).

Most measures of classification performance derive from the confusion matrix, which counts true positives, true negatives, false positives and false negatives. Accuracy expresses the proportion of all predictions that are correct, and precision expresses the proportion of predicted positives that are truly positive (Hicks et al., 2022). Accuracy can be misleading when classes are imbalanced, because a model can achieve high accuracy by predominantly predicting the majority class while performing poorly on the minority class. Precision should be interpreted together with recall (sensitivity), which measures how many of the actual positive cases are detected (Hicks et al., 2022).

After training, model performance should be validated against known outcomes using statistical metrics. For binary classifiers, the Receiver Operating Characteristic (ROC) curve plots the true

positive rate against the false positive rate across all decision thresholds. The area under this curve (ROC-AUC) provides a threshold-independent measure of predictive capability: values near 1 indicate strong performance, 0.5 reflects random classification, and values below 0.5 indicate performance worse than random ranking (Gneiting & Vogel, 2022).

ROC-AUC can be a useful measure, but does not directly reflect the precision of positive predictions and can therefore provide an incomplete picture when performance on the minority class is of particular interest (Davis & Goadrich, 2006; Powers, 2011). For example, in a model trained to detect a rare disease within a largely healthy population, poor detection of the minority class may have only limited influence on the overall ROC curve despite being practically important.

For imbalanced datasets, the Matthews Correlation Coefficient (MCC) is often more informative than the ROC-AUC. MCC incorporates all four outcomes of the confusion matrix (Chicco & Jurman, 2023). The metric ranges from -1 , representing complete disagreement between prediction and observation, to 1 , representing perfect classification; a value around 0 indicates performance no better than random prediction. Because MCC considers all classes simultaneously, it is regarded as a balanced performance metric for skewed datasets. Another validation method for imbalanced datasets is the precision-recall curve (PRC) (Saito & Rehmsmeier, 2015).

Data leakage should always be checked to ensure honest model performance. This can be assessed by checking whether the training and test datasets are clearly separated, whether duplicate or highly similar samples occur in both datasets, and whether preprocessing steps such as normalization, feature selection, or data augmentation were performed using the complete dataset before splitting (Kapoor et al., 2024; Kapoor & Narayanan, 2023). All are potential causes of data leakage.

3. Computational stack and development tools for AI models

An AI model runs on a layered stack of hardware and software. Each layer of software depends on the layer below it, so each needs to be present in a compatible version. In this stack, information of the running AI model flows from top to bottom and back. An environment is made, which is a project-specific collection of a programming language interpreter and installable packages. The stack is built and externally maintained, through development tools and an environment manager that the user operates (increasingly with generative AI as coding assistance). Figure 1 represents this structure schematically, with the execution path on the left and the developmental path on the right. Each layer is discussed in turn, from the bottom up.

3.1. Computing resources

Computing resources are hardware that comprise the physical components to perform computations. Given the computational demands of AI, access to a capable GPU is strongly recommended. GPUs are optimized for parallel processing, enabling efficient handling of large, high-dimensional datasets typical in machine learning workflows (Raina et al., 2009).

3.1.1. Local hardware

High-performance or gaming PCs are decent entry-level AI hardware, especially when the system contains a dedicated GPU. Memory is often a bottleneck in training complex AI models, and ways to decrease memory consumption during training can be found in Section 5.3 (T. Chen et al., 2016; Ojika et al., 2020; Rajbhandari et al., 2020).

3.1.2. Cloud computing resources

An alternative to local hardware is cloud computing. Cloud computing is the on-demand use of scalable off-site computing resources (International Organization for Standardization, 2023). Commercial cloud providers for AI applications include Microsoft Azure, Amazon Web Services (AWS), Google Cloud Platform (GCP) and Oracle Cloud Infrastructure (OCI). Costs depend on factors such as computing power, user agreement duration, and the use of spot instances. Spot instances are discounted computing capacity offered when the cloud service provider has overcapacity (Amazon, 2026). National research and education networks (NRENs) are government-funded academic cloud infrastructure, such as the Dutch IT cooperative Samenwerkende Universitaire RekenFaciliteiten (SURF), and may offer a cost-effective and safer alternative for eligible researchers (GÉANT, n.d.; SURF, 2026a).

A practical concern with commercial cloud computing is data security. Sensitive data uploaded to off-site infrastructure may be subject to cross-border legal frameworks. For example, U.S. legislation can compel disclosure of data held by U.S.-based providers, even when stored in Europe (U.S. Department of Justice, 2018). Cloud providers may also use uploaded data for service improvement or AI training purposes, although opt-out mechanisms are available. It is advised not to upload privacy-sensitive information to off-site infrastructure, such as a cloud provider, but to anonymize sensitive data or work with synthetic data to reduce privacy risks (European Union, 2016; van Velzen et al., 2025).

3.1.3. Hybrid computing infrastructure

A hybrid infrastructure combining local and cloud computing may provide a practical solution for larger AI workloads. Initial installation, troubleshooting, and exploratory runs can first be performed locally to familiarize users with the model and identify technical issues before migrating to cloud infrastructure. This approach reduces unnecessary cloud GPU usage and associated costs in the startup phase.

3.2. Operating system

The computing hardware is controlled by its operating system (OS), which is called system software. Examples are Linux, Windows and macOS (J. Wang et al., 2023). Compatibility between AI models and OS environments is not guaranteed; even minor version differences can introduce instability or errors (J. Wang et al., 2023). Reproducing the original development environment as closely as possible is therefore advisable. This can be done via a local container, such as a Linux container (Flinders & Smalley, 2025).

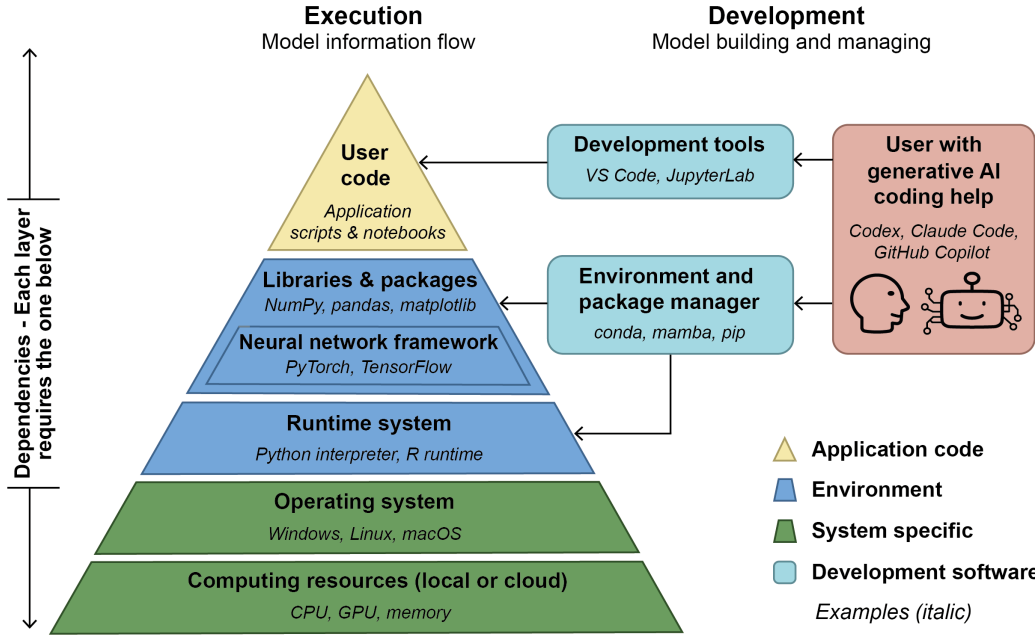


Figure 1: Visualization of the computational stack required to execute an AI model (left), and the tools that configure the stack (right). During a run, information flows from the top to the bottom of the stack and back. All layers and dependencies must be installed and compatible before the AI model can run. Users primarily work in the development tool and in the environment and package manager. Generative AI (GenAI) can help the user to develop the model and stack.

3.3. Runtime system

The operating system interacts with a user’s applications via a runtime system. The runtime system is software that allows code to execute, which is therefore programming language specific. Python, for example, is a computer language with its own interpreters, like CPython or PyPy (Python Software Foundation, n.d.). The interpreter’s task is to translate the programming language like Python in order for the operating system to understand it, perform tasks and allocate memory (Aho et al., 2006; Python Software Foundation, n.d.). For GPU-accelerated workloads, additional runtime components are often required. A common example is CUDA, which provides the necessary interface between software and NVIDIA GPUs, handling memory allocation and parallel task execution.

3.4. Libraries and packages

Above the runtime system sits a layer of libraries and packages. This is reusable code that the application code calls upon for tasks it does not perform itself. A library is a collection of functions written for a general purpose, and a package is the distributable unit in which a library is installed. In Python, pandas provides functions for reading and manipulating tabular data such as CSV files (McKinney, 2010), NumPy provides array operations (Harris et al., 2020), and matplotlib renders results as graphs (Hunter, 2007). Libraries of this kind support loading data, transforming it, and visualizing outcomes. The application code lists them as dependencies, and each one must be present in the environment in a compatible version.

3.4.1. Neural network frameworks

Neural network frameworks are libraries that form the ex-ecutional backbone of AI models. These frameworks consist

of three components: variables, functions, and parametric func-tions (Narihira et al., 2021). Variables represent the data used in the model and are stored in multidimensional arrays, such as vectors or tensors (Narihira et al., 2021). Functions perform mathematical operations on these variables, whereas parametric functions contain trainable parameters that are adjusted during training according to relationships identified in the training data. The model’s weights are stored within these parametric func-tions. The building of a model with parametric functions is more straightforward thanks to neural network frameworks like Ten-sorFlow (Abadi et al., 2016) and PyTorch (Paszke et al., 2019), because they automatically track operations in the forward pass and compute gradients in the backward pass (Ba Alawi, 2025).

3.5. User code: application scripts and notebooks

The top layer holds the code the user writes. In a single script or notebook, the user calls on the packages from the layers below and arranges them in the order the work requires: loading the data, preparing it with the appropriate libraries, and passing it to the neural network framework in small portions called batches. The same file specifies how the model is trained, which settings the optimization uses, and at which points the results are checked against validation data. Once the model has produced its predictions, supporting libraries turn these into figures and tables the user can interpret. The user code can be executed as a script in the used development tool.

3.6. Environment and package managers

AI models depend on a collection of software packages, often with strict version requirements. Environment and package managers (e.g., Conda, Mamba, Pip) facilitate the installation and isolation of these dependencies within dedicated environ-ments. This isolation is critical, as incompatibilities between

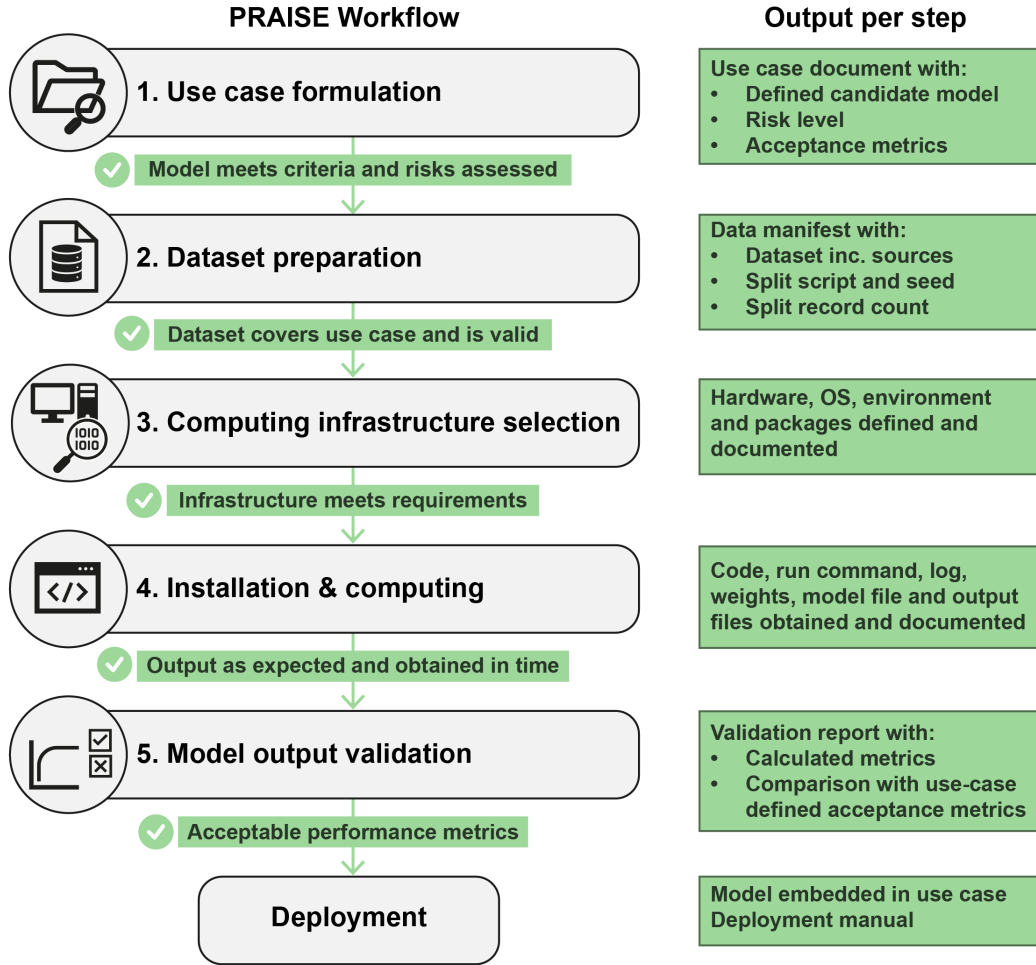


Figure 2: PRAISE (Practical, Reproducible AI-model Implementation for Science and Education) workflow for implementing scientific AI models to practical case studies, assuming no issues arise. Outputs and checks per step are given. Troubleshooting steps for common issues are shown in Section 5.

package versions or programming language versions are a common source of problems in AI workflows. Model developers frequently provide environment specifications (e.g., .yaml files), which should be followed where possible.

3.7. Development tools

Development tools offer an interface for writing, editing, and executing code. Examples of development tools are Visual Studio Code and JupyterLab. In this software, the application code can be opened and an environment selected. It is advisable to create the environment in an environment and package manager. Development tools are designed to be helpful to the human users, and often provide debugging features, file management and an integrated terminal.

3.8. Generative AI as a coding aid

Programming requires translating ideas into instructions a computer can execute, a skill that novices find difficult because of abstract concepts and entry barriers such as unfamiliar syntax, delimiters, indentation, and uninterpretable error messages (Figueiredo & García-Peñalvo, 2018). When implementing AI models from literature however, domain expertise remains the primary requirement, while the coding barrier can be lowered

through generative AI (GenAI) assistance. AI-assisted coding tools provide personalized help, on-demand feedback, and structured guidance in an educational setting (Barzanji & Loitsch, 2025), furthermore they have been shown to increase intrinsic motivation and reduce stress compared to unsupported approaches (Fan et al., 2025).

The underlying observations translate to non-programmer scientists. Access to AI coding support improves coding speed and quality without impairing the ability to understand or modify the resulting code (Kazemitabaar et al., 2023). This suggests that AI assistance can complement the development of practical programming skills. Across the whole programming workflow, these tools automate repetitive tasks, generate or explain code in natural language, and provide context-aware suggestions (e.g., relevant packages or functions) (Sergeyuk et al., 2025; Treude & Gerosa, 2025). Interaction modes range from conversational guidance to prompt-based code generation.

In practice, this support is accessed through tools such as GitHub Copilot, integrated into development tools like Visual Studio Code (Zhou et al., 2025), or assistants such as ChatGPT which can act as a collaborative programming partner (Hamza et al., 2024). These developments allow non-programmer scientists to engage with AI model implementation and rely less on

extensive coding expertise.

4. PRAISE: the workflow for the implementation of published AI models

The Practical, Reproducible AI-model Implementation for Science and Education (PRAISE) workflow suggested in Figure 2 aims to help non-programmer scientists to implement existing scientific AI models in practical case studies. The expected output and checks the workflow should pass before continuation to the next step are given for each step. When these checks are not met, the workflow should be extended with fallback routes to address these issues. This section will purely focus on the basic workflow, and Section 5 will go into the fallback routes for potential issues.

Complete and accurate documentation should be maintained throughout the workflow, with each action recorded as it is performed to support later troubleshooting and reproducibility on other computer systems. A private Git repository is recommended as an effective way to maintain this documentation. Its version-control system records each modification chronologically, creating a traceable log of the workflow and supporting subsequent review, troubleshooting, and reproduction.

4.1. Use case formulation

Arguably the most important step in the PRAISE workflow is the proper formulation of a use case. A clearly defined problem is essential for determining whether AI or ML offers an appropriate solution, or whether conventional IT or data-management approaches would be more suitable. In short: AI/ML must address a defined problem rather than serve as a default solution. Moreover, this aids the selection of what kind of AI or ML is required.

Hofmann et al. proposed a guide to formulate AI use cases, as this is not a trivial undertaking (Hofmann et al., 2020). Here the proposed guide is followed with modifications in the reformulation of the ‘problem and solution alignment’ and the inclusion of regulatory compliance (Figure 3).

4.1.1. Simultaneous problem and solution discovery

Central to the modification in problem and solution alignment from the original guide is that problem and solution discovery were performed consecutively, while in our view problem and solution discoveries can be performed simultaneously. This process should be repeated until the problem and the proposed solution are aligned. An initially vague problem can guide a search for potential AI solutions; studying these solutions clarifies which problems can be addressed, thereby enabling a more precise formulation of the original problem. This refined formulated problem is the starting point of the next iteration. The process naturally terminates when formulated problem and proposed AI solution become aligned.

When selecting potentially usable candidate models, the following should be considered:

- Recency: AI research advances fast, so a model of several years old is likely deprecated, although a more mature model often carries fewer bugs and more solved issues.

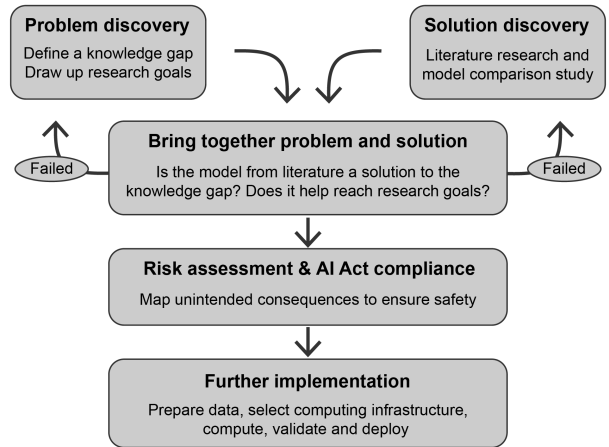


Figure 3: AI use case formulation guide. An AI use case can be constructed by bringing together both problem and solution discovery, until a candidate model is found that fits a defined knowledge gap. The regulatory compliance is performed afterwards.

- Developer support and community involvement: an active repository indicates that the model is kept working as dependencies and frameworks evolve, and that installation problems have been previously discussed.
- Available validation options: models that ship with ways to assess accuracy, robustness or feature attribution are preferred, as adding these afterwards requires programming expertise (Section 4.5). Acceptance values for the chosen validation metric (Section 2.3) should be set here to determine when model performance is acceptable.

4.1.2. Risk assessment and compliance

In addition, here the use case formulation guide by Hofmann et al. (2020) is extended by incorporating a risk assessment and compliance check, particularly in line with the EU AI Act (European Commission, 2023). This step is essential to incorporate data security and regulatory constraints from the start. This may include compliance with privacy regulations such as the General Data Protection Regulation (GDPR) (European Union, 2016), intellectual property (IP) rights, and non-disclosure agreements (NDA). Information protection is strengthened by aligning local data handling with established standards such as ISO/IEC 27001 (information security management system) (International Organization for Standardization, 2022a) and ISO/IEC 27701 (privacy information management system) (International Organization for Standardization, 2025). The safety of local large language model (LLM) use should be assessed against the Open Worldwide Application Security Project (OWASP) guidelines for LLM risks. These guidelines include (but are not limited to) prompt injection, sensitive information disclosure, and insecure output handling (OWASP, 2024). The use case reporting framework proposed by Hupont et al. (2024) can be used to systematically collect contextual information, in order to estimate the associated risk level of AI models. Particular caution is required when models process personal data or are applied in sensitive domains such as eligibility decisions, law, migration, or elections (Hupont et al., 2024; van Velzen et al., 2025).

4.1.3. Implementation and documentation

Hofmann et al. (2020) describe implementation in terms of development responsibilities, system integration, data handling, and ethical considerations. The workflow presented here adds practical steps for implementing the model itself: dataset preparation, infrastructure selection, model training, and validation (Sections 4.2 to 4.5), with troubleshooting in Section 5. The output of use case formulation is a document that reports the defined problem, the intended use (users, input and output), the candidate model, the possible risk level, and the acceptance criteria for model performance. These criteria are set before training. They state which performance metrics (Section 2.3) will be used to validate the model, for example on predictive performance, and which threshold values the model must reach to pass validation.

4.2. Dataset preparation

Data is the foundation of a predictive AI model and often the limiting factor in machine learning (Roh et al., 2021). Hence, the essential step of dataset preparation should be performed with care.

To be able to reproduce a model from literature, the dataset should be provided by the author of that model. Datasets should be Findable, Accessible, Interoperable and Reusable (FAIR) (Wilkinson et al., 2016). Often the used datasets are included with the source code at a repository maintained by the author.

Once access to the dataset has been obtained, its validity for the use case should be evaluated according to the criteria set out in Section 2.1 such as: dataset size, training set gaps, biases or the absence of negative datapoints. Assessment should address how well the dataset represents expected situations in the intended use case and how its inevitable imperfections may affect model-training performance.

Depending on the outcome of these analyses, the dataset needs to be supplemented with additional data, to improve the coverage of the dataset. The required additional data may be gathered from public repositories, automated extraction of online sources (data scraping) or newly generated experimental or synthetic data. A manual literature search to build a dataset is time consuming. Data scraping models can speed up this process, although automatic data extraction from the internet must be performed responsibly (Ayuso et al., 2024; Kahlon & Singh, 2026).

Finally, as emphasized in Section 2.1 the (improved) dataset should be divided into subsets for testing, training and validation, while avoiding data leakage as much as possible. The split can be checked on four points, together forming the leakage protocol, condensed from the leakage items proposed by Kapoor et al. (Kapoor et al., 2024; Kapoor & Narayanan, 2023).

- *Split order.* The dataset is split into training, validation and test sets before any preprocessing, normalization, feature selection or augmentation. Preprocessing parameters are derived from the training set only.
- *Overlap.* After splitting, the subsets share no identical records. Where records are related or highly similar, they are assigned to the same subset according to a defined

similarity criterion. The similarity criterion and strictness of the split are reported (Cui et al., 2025; Elmi et al., 2026; Kroll et al., 2024).

- *Use of the subsets.* The validation set is used for early stopping and hyperparameter choices only. The test set is used once, for the final metrics in Section 4.5. Any part of the training pipeline that reads the validation or test set for another purpose fails this check (Goodfellow et al., 2016b).
- *Reproducibility.* The split is generated by a script with a fixed random seed (Wegmeth et al., 2023). Both are documented so the split can be regenerated.

The output of the dataset preparation is a data manifest, containing information of the sources of the data, record count per subset, and the script and seed for data splitting.

4.3. Computing infrastructure selection

The next step is to select the computing resources needed to run an AI model from literature. This involves choices regarding the properties of the computing resources, which may be local, cloud or hybrid (see Section 3.1). The selection of the computational resources is a delicate step in which prerequisites of the AI model and computing resources need to be balanced: if the computing facilities are too limited, the model might simply not run or take an inordinate timespan to execute, while if the resources are too large managing them becomes a daunting task and will possibly quickly exceed the available budget. The check for this step is to ensure the infrastructure meets the requirements. The output is a chosen infrastructure as detailed in Section 3.

4.3.1. Constraints on computational infrastructure

The computing memory required to train and run AI models is often the hardware bottleneck. Memory requirements of scientific AI inferences are notoriously difficult to estimate and compare since they are dependent on multiple factors like parameter size, how intermediate embeddings are stored, and how detailed the input is (Tian et al., 2025). Even when considering these complications, the disparity in memory requirements between AI models is striking (Tian et al., 2025). In an AI model replication study, seven of the 30 models were not replicated because the specific hardware required was practically unobtainable (Gundersen et al., 2025).

The specifications of the computer infrastructure used by the model developer could be used as a guideline for a required computing memory. However, using a trained model often requires much less memory than the training procedure itself (Sohoni et al., 2019). Thus, if retraining an AI model is not part of the AI use case, and the trained model is provided, the user would not need to match the developer's computational setup. If retraining is part of the AI use case, matching the computing infrastructure is fair for an honest comparison and full reproducibility. When this is not possible due to the earlier described balance in resources, the training and computing time will be longer. If this is not acceptable, the model can be downsized or the step to cloud computing can be taken (Section 5.3).

4.3.2. Local hardware computing

Dedicated local hardware for computing is ideal if the available or intended hardware is close to the specifications of the infrastructure used by the author of the used AI model. Local computation resources have the advantage that training, running and adjusting AI models can be done without additional token costs and access to the facilities is unconstrained. However, the investment might be substantial if the hardware needs to be purchased for this purpose only.

4.3.3. Cloud and hybrid computing

Cloud computing is an option when local hardware cannot support the model, even after optimization. Compared to on-site infrastructure, cloud computing has relatively low startup costs and can be deployed quickly, but comes with higher costs for prolonged GPU usage (Armbrust et al., 2010). For small-scale or exploratory work, local infrastructure remains preferable when suitable hardware is available. It is also worthwhile to consider a hybrid approach where initial setup optimizations are done with the local hardware so that heavy computational jobs can be efficiently performed in the cloud infrastructure.

4.3.4. OS selection

The basic rule of thumb for the choice of the operating system is to use the OS which was employed by the authors of the published AI model. Converting routines used on one platform to the next is in principle possible but is likely to fail given the complexity of using or training models and the many dependencies of the computational environment. If the required OS is not available, containers or full virtualizations can be used, though this often reduces performance. For local hardware, setting up a dual boot environment with different operating systems alongside each other might be an option.

4.4. Running an AI model: from installation to computing

Running a published AI model requires its code and a matching software environment on the local machine. Training the model additionally requires the training dataset and more computational resources than inference with pretrained models. Inference requires the pretrained model with its weights.

4.4.1. Published AI model repository

Requirements depend primarily on the architecture and implementation of the published AI model. Datasets, code and documentation of a published AI model are typically available on GitHub. The repository, i.e. all folders and files, can be cloned to the local machine through the git extension in a development tool like Visual Studio Code or JupyterLab.

4.4.2. Development tools

The choice of development tools is highly personal. JupyterLab is web-based and designed to be user-friendly, Visual Studio Code needs to be installed and is more complex due to its customizability (Kluyver et al., 2016). Beginners in programming are advised to try multiple tools to determine the ones they may prefer. Open-source availability does not imply security; the code, dependencies, and documentation should therefore be reviewed critically.

4.4.3. Computational environment

Before executing the code, an environment should be configured (Section 3.6). The documentation accompanying the AI model specifies the required programming language version and packages. In the development tool, the appropriate environment can be selected and the required packages installed via a command in the integrated terminal. A check should be performed for all required files, like datasets or trained model weights. These often need to be downloaded separately. File paths in the code should be checked and possibly adjusted to fit the repository structure. GenAI tools like Codex, Claude Code or GitHub Copilot can be used as personal programming assistants (Section 3.8).

4.4.4. Running the AI model

With the repository, environment and data locally available, the training or inference of the AI model can start. Ensure that the computational work is done by the GPU. Checkpoints can be saved in case the training stopped (TensorFlow, 2024). The output can be checked against expectation at each checkpoint. When the execution has been completed within an acceptable time, one can continue with validating the model (Section 4.5). If the hardware appears to be insufficient for model execution, the model might be downsized, or hardware expanded (Section 5.2 and 5.3).

Once training is complete, the model is saved in framework-specific formats, for PyTorch or TensorFlow. Pretrained models are shared in these same formats. Hugging Face developed Safetensors, which saves metadata in JSON format and is universally usable for different programming languages. Local inference might then be possible, since these calculations require less computing power than training. Both locally trained and downloaded pretrained models can be loaded for inference on the local machine with dedicated code.

4.4.5. Basic after-training checks

Whether a training run completed as intended can be judged from the computing time, the presence of the expected output files, and the values these files contain. A computing time far below the expected value or missing output files indicates that the training did not complete. Output files in which every prediction has the same value, for example 0.5 for a binary classifier, indicate that the model did not learn during training. Validation is only useful once none of these signs are observed.

4.5. Model validation

The basic question of the validation step is whether the results of the AI model warrant deployment. Reliable AI model predictions depend on the quality of the training data and on what the model has actually learned from it (Hartung & Kleinstreuer, 2025). Weight distributions (Section 2.3) can be compared between training runs to indicate whether each run produced a model of comparable quality, and a tool such as WeightWatcher automates this comparison by analyzing the weight matrix of every layer and reporting a quality value per layer, which helps to identify layers that remained undertrained or became overtrained (Martin et al., 2021).

The performance indicators introduced in Section 2.3 can be calculated on the validation set during training and on the test set once training is finished. Prediction scores and true labels are the input for ROC-AUC calculations. MCC operates on discrete class labels and therefore requires a threshold, commonly 0.5. Standard options (such as `roc_auc_score`, `average_precision_score`, `matthews_corrcoef`) are available in scikit-learn (Pedregosa et al., 2011). Repeating the calculation over several random seeds or data splits indicates how stable the obtained values are. Whether the obtained predictive performance metrics are acceptable depends on the use case. Calculating the same indicators for the part of the test set that lies furthest from the training data shows how well the model covers gaps in the dataset and how it behaves on unseen input.

Performance indicators do not need to match those reported for the original model exactly. Variations in outcomes might be caused by differences in the random seed used for weight initialization, the order in which data are processed in parallel, or the GPU libraries (cuDNN and CUDA) (Pham et al., 2021). Training runs across three datasets and six AI networks, with a fixed random seed, resulted in accuracy variations of up to 2.9% (Pham et al., 2021). Whether the validation was adequate is recorded in the use case formulation document (Section 4.1). If it does not meet these requirements, Section 5.4 helps locate the cause.

4.6. Deployment

When validation meets the acceptance criteria set in the use case document (Section 4.1.3), the model can be deployed in the use case. Predictions for new input data can be made, in the context described in the use case document: who uses the model, which input they provide, and which output they need. A short test with new, representative input confirms that the model is ready for this use. The model should accept the input format of the users, return output they can interpret, and run on the hardware available to them. A low-code web-based user interface gives users without programming experience access to the trained model (Kroll et al., 2023).

At this point the logbook with documentation of the implementation process can be turned into a wiki that others can follow. An LLM such as Claude or Copilot is a useful drafting aid here, if the result is checked against the original implementation to avoid propagating errors. The outputs presented in Figure 2 collectively form the reproducibility package: use case document, data manifest and split script, hardware and OS record, environment file, run commands, logs, model files and metrics. This package meets the bronze standard of Heil et al. (2021). Before publishing, the package can be checked against a reproducibility checklist such as REFORMS (Kapoor et al., 2024; Kwak & Kim, 2025). Code, data and documentation should be published in a public GitHub repository linked from the paper. Large datasets belong on Zenodo or DataverseNL, under their own DOI cited in the paper.

5. Fallback troubleshooting by workflow iteration

The PRAISE workflow as presented in the previous section indicates the ideal situation, in which all steps can be executed

without any difficulties. In practice, this will rarely be the case, thus the workflow needs to be adjusted accordingly at various stages when issues arise. The workflow diagram is extended with these troubleshooting routes (Figure 4).

5.1. Analyzing dataset mismatch

Dataset preparation fails when the use case and the available data do not match, because the dataset should cover the intended applicability domain of the model. The data needed for the use case might not be available or not of appropriate quality even after extension according to the procedures described in Section 4.2. The use case should then be reformulated to fit the available data while keeping the project relevant.

If changing the use case is undesirable and the mismatch comprises only minor gaps, dataset preparation and the remaining workflow can proceed, with the gaps documented as limitations on model validity. These limitations should be revisited during validation (Section 4.5), where performance can be assessed separately across poorly represented regions of the applicability domain.

5.2. Assessing computational infrastructure

Computational bottlenecks can sometimes be foreseen when the computing infrastructure is selected, but they can also become apparent during or after installation and test runs. In that case, the user code is revised first (Section 5.3). The infrastructure is reassessed when code changes are insufficient. Infrastructure here covers all layers of Figure 1 below the user code: libraries, runtime, operating system and computing hardware.

It is essential to first analyze what the main underlying reasons for the issues could be. Common issues are that the system runs out of memory, runs out of storage capacity, takes too long to complete or delivers unexpected outputs. Possible remedies essentially fall into a few categories:

- adjust the computational infrastructure
- adjust the datasets
- reformulate the use case

Often a combination of these remedies is needed to solve computational infrastructure issues in the workflow.

5.2.1. Reselect infrastructure

The limits on computational performance can sometimes be raised with minor adjustments, such as extending the memory, acquiring more compute credits, or installing a different package in the environment. However, often this will entail a more major upgrade of the on-site hardware or move to cloud/hybrid computing (Section 3.1). To decide which adjustment is needed, the issues must be analyzed in detail to define the additional infrastructure requirements. A conclusion can be that the issues are not practically or financially solvable. In that case, the limitation must be accepted, and the solution shifts to reducing the dataset size or reformulating the use case (Section 4.1 and Section 4.2).

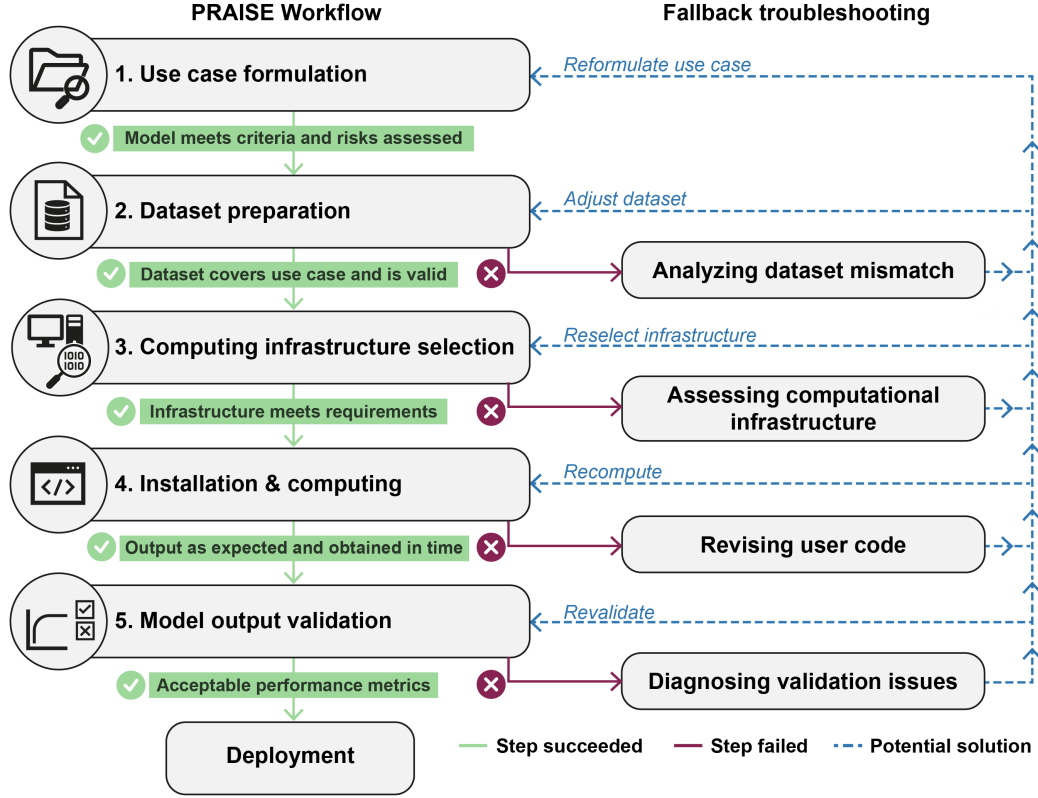


Figure 4: The PRAISE workflow for implementing scientific AI models to practical case studies, extended with troubleshooting routes. When a checkpoint is not met, the arrow marked with a cross leads to a decision point that helps decide which solution is required. These bring the user one or more steps back, depending on the issue. Such iterations are common in AI model implementations.

5.2.2. Adjust the dataset or reformulate the use case

With the experience gained to this point, a clearer picture of what questions and what kind of datasets the available computational infrastructure would be able to handle should be present. For example, the datasets might be reduced in size without losing the characteristics needed to use or train the model properly. Another option could be to modify the questions of the use case, such that they can be answered by the current setup or adjustments thereof.

5.3. Revising user code

Problems during installation and computing, such as a run that crashes, runs out of memory, takes too long or gives unexpected outputs, can often be solved by revising the user code and recomputing. When the cause lies in computing infrastructure, for example a failed installation or package conflict, the infrastructure is reassessed (Section 5.2).

5.3.1. Model downsizing

To make a model fit the computational requirements within the existing infrastructure, downsizing of the model can be considered. One downsizing method is reducing the batch size, which is configurable in frameworks like PyTorch or TensorFlow, though smaller batches produce noisier gradients (Keskar et al., 2017). Other common approaches include gradient checkpointing, precision scaling, and gradient accumulation.

5.3.2. Gradient checkpointing

Another way to reduce infrastructure requirements is gradient checkpointing. This adjustment reduces memory usage by storing less information during the forward pass and recalculating intermediate values when needed (T. Chen et al., 2016). This adjustment reduced the memory consumption of a thousand-layer AI model sixfold with only a 30% increase in computation time.

5.3.3. Precision scaling

Further lowering of memory requirements can be achieved by precision scaling. This method reduces the numerical precision used within the model, for example from 32-bit to 16-bit values, thus resulting in lower memory requirements while often maintaining comparable model performance (Dettmers et al., 2022).

5.3.4. Gradient accumulation

Gradient accumulation spreads gradient updates across multiple smaller batches before adjusting model parameters, allowing larger models to be trained on limited hardware (Rajbhandari et al., 2020). Regardless of the approach, the resulting model performance should be validated against outcomes reported in the literature to confirm that downsizing did not negatively affect results. If downsizing is insufficient, the use of pretrained models may be a more suitable alternative.

5.3.5. Alternative types of machine learning

As reviewed in Section 2.2, there are various types of machine learning with corresponding loss functions to be minimized. It is worth considering whether there are modifications in the applied training procedure to alleviate the computational bottlenecks (as different implementations might have impact on the required computational power, memory, etc.). If code changes do not resolve the issue, the next step is to reassess the infrastructure or to adjust the dataset or use case.

5.4. Diagnosing validation issues

If the validation of the trained model does not yield acceptable results, model diagnostics should be performed on the dataset, the scripts and/or notebooks. An unmodified reproduction of the published model is expected to reach performance metrics comparable to those reported by the authors, but variations in validation outcomes might occur (Section 4.5).

5.4.1. Dataset Composition and Validation Performance

Modifications to the dataset influence the validation outcome. In supervised learning, a different data split (Section 2.1) changes which samples the model is optimized on and which are held out, so the reported metrics shift as well (Cawley & Talbot, 2010). Extending the original dataset with additional records can improve predictive performance and can widen the coverage of the model (Rajput et al., 2023; Z. Wang et al., 2026).

5.4.2. Permutation Testing for Data Leakage

When metrics are higher than expected, a permutation test can be performed. Here, the training labels are shuffled at random, and the model is retrained to check whether predictive performance is attributable to relationships in the original data. In the permutation test, performance should approach the random baseline (ROC-AUC $\approx$ 0.5 and MCC $\approx$ 0). Substantially higher performance indicates that information remains available to the model despite removal of the original label-feature relationship and may point to data leakage or an error in the training or evaluation procedure (Ojala & Garriga, 2010).

5.4.3. Diagnosing Deviations from the Published Model

Finally, the training and evaluation scripts should be compared with the published implementation to identify unintended changes in data handling, preprocessing, model configuration, optimization, or evaluation. Changes to how data is preprocessed or separated before a training can introduce data leakage and result in misleading performance estimates (Kapoor et al., 2024). Other differences, such as the learning rate, batch size, number of training epochs, or model parameters, can alter the training outcome even when the dataset is unchanged. The diagnostics should therefore establish whether obtained unexpected validation performances originate from issues in the validation protocol, differences in the dataset or issues in the training process. If none of these solve the validation issues, the use case should be reformulated.

6. Case study of workflow application in research and education

The preceding sections present the PRAISE workflow as a sequence of generic steps. This section illustrates how these steps were executed in a worked example. The sections follow the actual course of the project: a first pass through the steps of Figure 4, then four iterations, each triggered by an issue that required to return to a previous step (Figure 5). These loops are an integral part of the PRAISE workflow.

The case concerns the prediction of enzyme-substrate interactions in a chemical recycling project. Three third-year BSc chemistry students and their tutor selected a published deep learning model, reproduced it, validated its performance and extended it to their own dataset within four months, using GenAI programming assistants. The tutor was a chemistry teacher and researcher, who helped formulate the use case and select computing infrastructure. None of them had a background in programming or data science.

6.1. First pass: reproducing ProSmith

The first pass started with the formulation of the use case (Section 4.1). The work is part of C-Recycle, an EU Just Transition Fund project on chemical recycling, where waste streams such as plastic pyrolysis oil are upcycled into high-value chemicals that serve as feedstock for industry (Stimulus, 2024). Oxidative enzymes can functionalize these streams, but finding suitable enzyme-substrate pairs experimentally is laborious, and public databases only cover characterized pairs. The aim was therefore to reproduce an existing prediction model and deploy it in this research to match enzymes to substrates, find candidate enzymes and prioritize mutants for testing. The model would be considered ready for deployment when it reproduced the performance reported by its authors on the held-out test set and could predict interactions for new enzyme-substrate pairs.

With the use case defined, a limited literature search at the start of the project in 2024 identified the Enzyme Substrate Predictor (ESP) and subsequently ProSmith as suitable models (Kroll et al., 2023; Kroll et al., 2024). Further models appeared during implementation and were also considered (Campbell, 2024; Cui et al., 2025; Du et al., 2025; Elmi et al., 2025, 2026). Publication in this field is rapid enough that a selected model can be superseded before its implementation is completed, which makes model choice a recurring decision throughout the project, and possibly an iteration loop back to this step.

The risk assessment and compliance check of Section 4.1 was carried out before implementation. No personal data were processed, and the application falls outside the sensitive domains identified there, placing the use case at a minimal risk level under the EU AI Act.

The project started with the ProSmith ESP dataset (Kroll et al., 2024). ProSmith was trained on a single NVIDIA A100 GPU with 40 GB memory, and training on a 24 GB NVIDIA RTX 6000 yielded similar performance at the cost of a longer run time. A 24 GB GPU would therefore suffice. This local training was done in the HR Datalab EAS, an in-house facility with high-performance workstations shared under clear usage guidelines. The workstations contained an NVIDIA GeForce RTX 4090

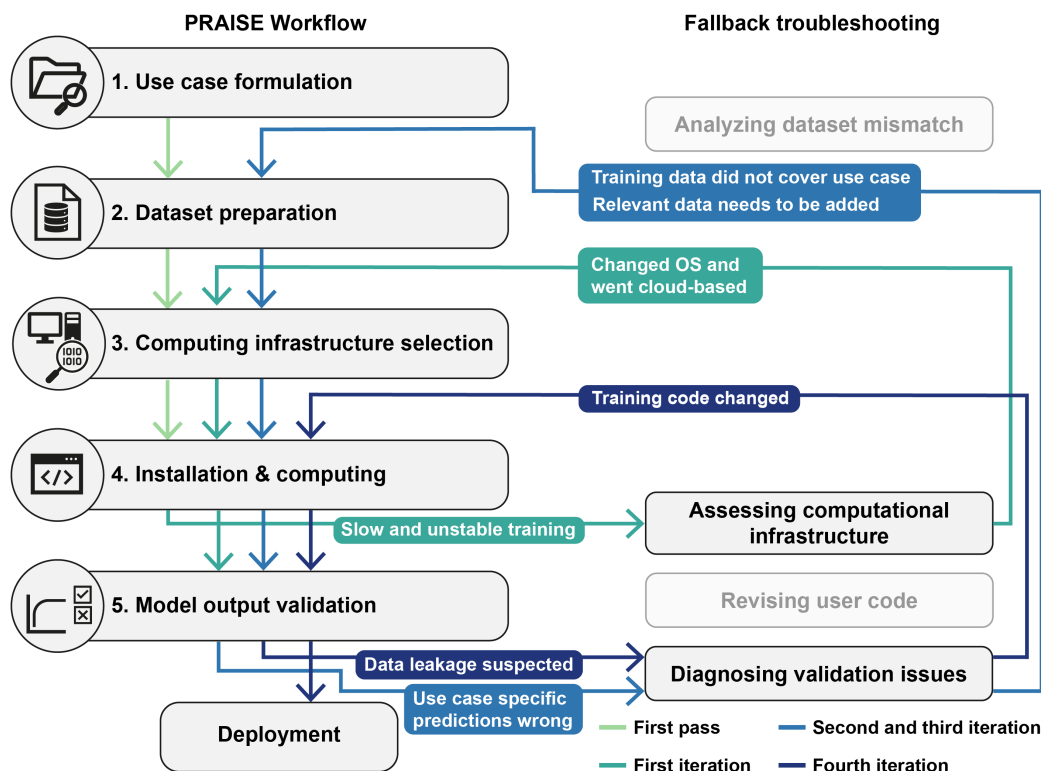


Figure 5: Implementation of an AI model to predict enzyme-substrate interactions following the PRAISE workflow. Colored arrows mark the first pass and the four iterations that followed, numbered in the order they occurred. First pass: the initial implementation of ProSmith, which stopped at the first issue. First iteration: the switch from Windows to a cloud-based Linux OS. Second iteration: the retraining on a dataset supplemented with data gathered manually from the literature. Third iteration: the retraining on an additional dataset containing true-negative data. Fourth iteration: the changes to the training pipeline to investigate suspected data leakage.

GPU with 24 GB VRAM and ran Windows 11. Windows was the operating system the students already knew, which lowered the entry barrier.

Installation guides in the programming community are typically written for Linux, and the ProSmith bash commands were incompatible with Windows PowerShell. Coding assistance (Section 3.8) was used to convert these commands to work on PowerShell, after which the original model could be reproduced.

6.2. Iteration 1: unstable retraining on Windows

Retraining in step 4 of Figure 5 failed in multiple attempts, with errors that could not be traced to a specific cause. The operating system was identified as the likely cause, so implementation moved to a Linux environment on SURF Research Cloud, the national research computing infrastructure of SURF (SURF, 2026a, 2026b). The JupyterLab workspace ran Ubuntu 22.04 on a single NVIDIA A10 GPU. This was an iteration to step 3 of Figure 5. After a fresh installation, baseline training ran without the earlier errors. Shorter training time was an added benefit: all training steps took less than two days, compared with about four days on the in-house workstations.

6.3. Iteration 2: training data did not cover the use case

Validation of the predictions (step 5 in Figure 5) showed that the enzymes and substrates of interest were absent from the training data. The project looped to the earlier step 2 in Figure 5, an extended dataset was compiled from the literature. The ESP

dataset was supplemented with enzyme-substrate pairs from online databases, each traced back to the original publication to verify its label. The larger dataset required targeted modifications to the preprocessing and gradient boosting modules.

6.4. Iteration 3: simulated negatives in the training data

Subsequent validation (step 5 in Figure 5) showed that the training set contained simulated negatives, which can cause misclassification (Campbell, 2024). Negative interactions were scarce, because negative results are rarely published. The ProSmith ESP dataset compensates for this through random negative sampling. ProSmith was therefore trained on the VIPER dataset (Campbell, 2024), a high-throughput screening dataset with experimentally determined negative outcomes (step 2 in Figure 5). The VIPER data were converted to the ProSmith input format, shuffled and split into test, training and validation sets using the ratio of Kroll et al. (2024), with a fixed random seed for reproducibility.

6.5. Iteration 4: suspected data leakage

The leakage protocol of Section 4.2 indicated a potential issue since the validation set was used in training during the gradient boosting step. This came to light in step 5 in Figure 5. The suspected leakage was reported to the developer (Kroll, 2024). Code inspection showed that the test set was used once and the ensemble weights were chosen on the validation set only, so check 3 of the leakage protocol was met. However, in

Table 1: Outcomes of five different ProSmith model training sessions and the corresponding validation scores. Each training was performed once, and the values are only indicative. The suspected data leak fix was investigated and found not to be data leakage.

Model	Iteration	Accuracy	Precision	ROC-AUC	MCC
Standard ProSmith Windows	Original	0.93	n.d.	0.96	0.80
Standard ProSmith Linux	Install & computing → infrastructure selection	0.92	0.94	0.95	0.80
ProSmith manually supplemented database	Validation → data preparation	0.91	0.92	0.95	0.76
ProSmith supplemented with VIPER database	Validation → data preparation	0.93	0.83	0.95	0.75
Standard ProSmith suspected data leak fix	Validation → install & computing	0.90	0.92	0.95	0.74

the original pipeline the models were retrained on the training and validation data together before they were applied to the test set. Refitting the chosen model to all data used for selection is common practice (Hastie et al., 2009), but it means that the tested models were different from the validated models. For a reproducible pipeline, the students preferred a single set of models, trained on the training set only, which is validated, tested, and saved as one object. The cost of this choice is that the final models were trained on fewer records.

Two other adaptations were made. The trained gradient boosting models were discarded after training, and predictions were not automatically linked back to their original enzyme-substrate records. The pipeline was restructured to train the model on the training set only, to save models and ensemble weights automatically, and to map predictions to the original records (step 4 of Figure 5).

6.6. Comparison of performance metrics across training sessions

For each ProSmith training session, accuracy, precision, ROC-AUC and MCC (Section 2.3) were calculated on the test set (Table 1). All trainings were executed once, so the values are indicative only. ProSmith remains predictive for the supplemented datasets, as the positive MCC scores show. The lower MCC after iteration 4 was suspected to be a result of data leakage, but was eventually deemed consistent with the smaller training set of the restructured pipeline and falls within the run-to-run variation reported for deep learning pipelines (Pham et al., 2021). The restructured pipeline met the criteria of the use case (Section 6.1). Validation was therefore considered successful, deployment could commence.

6.7. Deployment

The retrained ProSmith model was deployed as a prediction script on SURF Research Cloud. The script loaded the saved model and ensemble weights, predicted interactions for new enzyme-substrate pairs, and mapped each prediction to its input record. The trained model could be used to screen enzymes for potential interactions with a substrate of interest, for example in functionalization reactions or in the degradation of waste streams. A second application was the prediction of how point mutations affected enzyme-substrate interaction. Both applications still require future experimental validation before the predictions can guide enzyme selection. The data, models and code of all iterations are available via the authors’ GitHub page.

6.8. Case study outcome

Three BSc chemistry students and their tutor, all without programming experience, were able to reproduce and extend the predictive model ProSmith within four months using the PRAISE workflow. The model and computing infrastructure were selected by the institution, and the students performed the installation, data management, model training and validation independently. They adapted the pipeline and applied it to additional datasets (Table 1). The implementation required four iterations. Each started where the expected result was not obtained and the students returned to an earlier step (Figure 5). The workflow allowed the students to understand each step and how to solve problems they encountered. A separate paper on the educational implementation of the PRAISE workflow into the BSc chemistry program is in preparation.

7. Discussion

We developed the structured, reproducible workflow PRAISE, which guides non-programmer scientists through implementing an AI model published in the literature. The workflow divides the process into explicit, verifiable steps and includes fallback loops for common issues.

By explicitly incorporating dataset and model validation, the PRAISE workflow positions reproducibility as a prerequisite in the implementation of AI models. The PRAISE workflow covers performance metrics that indicate whether predictions are accurate. The scientific value of the resulting predictions is a separate question that the user should assess themselves. Feature analysis can indicate which characteristics in the data are responsible for certain predictions (Li et al., 2020). LIME estimates feature importance for individual predictions by generating synthetic data around a single datapoint (Linardatos et al., 2021; Ribeiro et al., 2016). SHAP produces local explanations and global importance scores across the model (Linardatos et al., 2021; Lundberg & Lee, 2017). In models of the complexity of ProSmith, these methods are difficult to apply, because the model inputs are learned vector representations produced by several preprocessing steps, and individual dimensions carry no direct chemical interpretation. Feature analysis is therefore not included as a validation step in the proposed workflow.

Several other disclaimers for adopting the PRAISE workflow exist. One is that basic computer and data skills remain necessary, such as installing a program, understanding documentation

and managing large datasets. In addition, the availability of the published model and its dataset, and the quality of the documentation, determine whether the workflow can be started at all. This becomes clear in the first step of the PRAISE workflow, where the use case is matched with potential AI solutions. Even with good documentation, specialized AI models may still require more advanced expertise than the workflow and generative AI coding helpers can give. If large AI models require more extensive computing power than the institution can provide, these models stay out of reach for users as well. The most restrictive limitation is the absence of data covering the research question of the use case, which results in unreliable predictions regardless of the quality of the model.

In conclusion, implementing an AI model from literature is a skill that can be learned. With the basic techniques described here and an understanding of the PRAISE workflow steps and their iterations, non-programmer scientists can reproduce published models and implement them in their own practice. A non-programmer scientist who adopts the PRAISE workflow becomes an evaluator and improver of AI models.

Authorship contribution statement

Conceptualization: MA, SGN, VdB, SS, RvdW; Methodology: MA, SGN, RvdW; Software: MA, SS, ZY, PAS, AP, MvdW; Validation: MA, SS, ZY, PAS, AP; Investigation: MA, SS, ZY, PAS, AP, MvdW; Resources: SGN, VdB, SS, MvdW, RvdW; Data Curation: MA, SS, ZY, PAS, AP; Writing - Original Draft: MA, SGN; Writing - Review & Editing: SGN, VdB, CM, RvdW; Supervision: MA, SGN, SS, CM, RvdW; Project administration: CM; Funding acquisition: CM

Data availability statement

As the purpose of this paper is to present the PRAISE workflow and the specific use case, discussed in Section 6, was included only to provide a concrete example of this workflow in practice, the results obtained in that use case do not constitute a publishable data for the present paper. In accordance with the practical suggestions of section 4.6, the datasets, scripts, run commands, model files and validation results of the case study were stored in a private GitHub repository for internal use only. If the deployment of the use case leads to novel scientific insights, they will be made available with a future publication.

Declaration of competing interest

The authors declare to have no known competing financial interests or personal relationships that could have influenced the reported work.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this work the author(s) used Claude (Anthropic) to search for literature, interpret and write code, check grammar and shorten text. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the published article.

Funding

This research was carried out within the C-Recycle consortium and co-funded by the European Union via the Just Transition Fund (JTF), project JTFZ-00101, administered by Stimulus Programmamanagement. The funder had no role in study design, data collection, analysis, or the decision to publish.

Acknowledgement

The authors would like to thank the Rotterdam University of Applied Sciences Chemistry Department research group and the Engineering and Applied Sciences Datalab, as well as the C-Recycle Just Transition Fund consortium partners. This work used the Dutch national e-infrastructure with the support of the SURF Cooperative using grant no. EINF-17601.

References

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., ... Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (pp. 265–283). USENIX Association. <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>
- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, techniques, and tools* (2nd ed.). Addison-Wesley.
- Amazon. (2026). *Amazon EC2 Spot Instances*. Retrieved February 17, 2026, from <https://aws.amazon.com/ec2/spot/>
- Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., & Zimmermann, T. (2019). Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* (pp. 291–300). <https://doi.org/10.1109/ICSE-SEIP.2019.00042>
- Apicella, A., Isgrò, F., & Prevete, R. (2025). Don't push the button! Exploring data leakage risks in machine learning and transfer learning. *Artificial Intelligence Review*, 58(11), 339. <https://doi.org/10.1007/s10462-025-11326-3>
- Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, 53(4), 50–58. <https://doi.org/10.1145/1721654.1721672>
- Artrith, N., Butler, K. T., Coudert, F.-X., Han, S., Isayev, O., Jain, A., & Walsh, A. (2021). Best practices in machine learning for chemistry. *Nature Chemistry*, 13(6), 505–508. <https://doi.org/10.1038/s41557-021-00716-z>
- Ayuso, E., Brogya, M. S. D., Ahlawat, V. K., & Sain, M. (2024). From manual to machine: How AI is redefining web scraping for superior efficiency: A literature

- review. In *2024 International Conference on Communication, Control, and Intelligent Systems (CCIS)* (pp. 1–9). <https://doi.org/10.1109/CCIS63231.2024.10931912>
- Ba Alawi, Z. (2025). *A comparative survey of PyTorch vs TensorFlow for deep learning: Usability, performance, and deployment trade-offs*. arXiv. <https://doi.org/10.48550/arXiv.2508.04035>
- Barzanji, C., & Loitsch, C. (2025). Exploring conversational agents for novice programmers: A scoping review. *Discover Artificial Intelligence*, 5(1), 271. <https://doi.org/10.1007/s44163-025-00521-4>
- Biagini, G. (2025). Towards an AI-literate future: A systematic literature review exploring education, ethics, and applications. *International Journal of Artificial Intelligence in Education*, 35(4), 2616–2666. <https://doi.org/10.1007/s40593-025-00466-w>
- Campbell, M. J. (2024). *VIPER: A general model for prediction of enzyme substrates*. bioRxiv. <https://doi.org/10.1101/2024.06.21.599972>
- Cawley, G. C., & Talbot, N. L. C. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11(70), 2079–2107. <http://jmlr.org/papers/v11/cawley10a.html>
- Chen, J., & Bao, Y. (2025). Effect of dataset representation bias on generalizability of machine learning models in predicting flexural properties of ultra-high-performance concrete (UHPC) beams. *Engineering Structures*, 326, 119508. <https://doi.org/10.1016/j.engstruct.2024.119508>
- Chen, T., Xu, B., Zhang, C., & Guestrin, C. (2016). *Training deep nets with sublinear memory cost*. arXiv. <https://doi.org/10.48550/arXiv.1604.06174>
- Chicco, D., & Jurman, G. (2023). The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification. *BioData Mining*, 16(1), 4. <https://doi.org/10.1186/s13040-023-00322-4>
- Cui, H., Su, Y., Dean, T. J., Yu, T., Zhang, Z., Peng, J., Shukla, D., & Zhao, H. (2025). Enzyme specificity prediction using cross-attention graph neural networks. *Nature*, 647(8090), 639–647. <https://doi.org/10.1038/s41586-025-09697-2>
- Davis, J., & Goadrich, M. (2006). The relationship between precision-recall and ROC curves. In *Proceedings of the 23rd International Conference on Machine Learning* (pp. 233–240). Association for Computing Machinery. <https://doi.org/10.1145/1143844.1143874>
- Dettmers, T., Lewis, M., Belkada, Y., & Zettlemoyer, L. (2022). GPT3.int8(): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35, 30318–30332. <https://doi.org/10.52202/068431-2198>
- Du, Z., Fu, W., Guo, X., Caragea, D., & Li, Y. (2025). FusionESP: Improved enzyme–substrate pair prediction by fusing protein and chemical knowledge. *Journal of Chemical Information and Modeling*, 65(6), 2806–2817. <https://doi.org/10.1021/acs.jcim.4c02357>
- Eid, F.-E., Elmarakeby, H. A., Chan, Y. A., Fornelos, N., El-Hefnawi, M., Van Allen, E. M., Heath, L. S., & Lage, K. (2021). Systematic auditing is essential to debiasing machine learning in biology. *Communications Biology*, 4(1), 183. <https://doi.org/10.1038/s42003-021-01674-5>
- Elmi, V. A., Joeres, R., & Kalinina, O. V. (2025). *A general transformer-based multi-task learning framework for predicting interaction types between enzyme and small molecule*. bioRxiv. <https://doi.org/10.1101/2025.10.09.681419>
- Elmi, V. A., Joeres, R., & Kalinina, O. V. (2026). *Information leakage in enzyme substrate prediction*. bioRxiv. <https://doi.org/10.64898/2026.02.26.708291>
- European Commission. (2023). *EU AI Act or Artificial Intelligence Act*. <https://www.europarl.europa.eu/topics/en/article/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence>
- European Union. (2016). *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)* (CELEX No. 32016R0679). <https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng>
- Fan, G., Liu, D., Zhang, R., & Pan, L. (2025). The impact of AI-assisted pair programming on student motivation, programming anxiety, collaborative learning, and programming performance: A comparative study with traditional pair programming and individual approaches. *International Journal of STEM Education*, 12(1), 16. <https://doi.org/10.1186/s40594-025-00537-3>
- Figueiredo, J., & García-Peñalvo, F. J. (2018). Building skills in introductory programming. In *Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality* (pp. 46–50). Association for Computing Machinery. <https://doi.org/10.1145/3284179.3284190>
- Flinders, M., & Smalley, I. (2025). *What are Linux containers?* IBM Think. Retrieved July 30, 2026, from <https://www.ibm.com/think/topics/linux-containers>
- Fort, S., Dziugaite, G. K., Paul, M., Kharaghani, S., Roy, D. M., & Ganguli, S. (2020). Deep learning versus kernel learning: An empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Article 491). Curran Associates Inc.
- GÉANT. (n.d.). *GÉANT project partners*. Retrieved February 17, 2026, from <https://geant.org/projects/partners/>
- Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Vol. 9, pp. 249–256). <https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf>
- Gneiting, T., & Vogel, P. (2022). Receiver operating characteristic (ROC) curves: Equivalences, beta model, and minimum distance estimation. *Machine Learning*, 111(6), 2147–2159. <https://doi.org/10.1007/s10994-021-06115-2>
- Goodfellow, I., Bengio, Y., & Courville, A. (2016a). Deep feedforward networks. In *Deep learning* (Chapter 6). MIT Press.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016b). Machine learning basics. In *Deep learning* (Chapter 5). MIT Press.
- Gundersen, O. E., Cappelen, O., Mølne, M., & Nilsen, N. G.

- (2025). The unreasonable effectiveness of open science in AI: A replication study. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(25), 26211–26219. <https://doi.org/10.1609/aaai.v39i25.34818>
- Hamza, M., Siemon, D., Akbar, M. A., & Rahman, T. (2024). Human-AI collaboration in software engineering: Lessons learned from a hands-on workshop. In *Proceedings of the 7th ACM/IEEE International Workshop on Software-intensive Business* (pp. 7–14). Association for Computing Machinery. <https://doi.org/10.1145/3643690.3648236>
- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., . . . Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- Hartung, T., & Kleinsteuener, N. (2025). Challenges and opportunities for validation of AI-based new approach methods. *ALTEX*, 42(1), 3–21. <https://doi.org/10.14573/altex.2412291>
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). 7.10 Cross-validation. In *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer.
- Heil, B. J., Hoffman, M. M., Markowitz, F., Lee, S. I., Greene, C. S., & Hicks, S. C. (2021). Reproducibility standards for machine learning in the life sciences. *Nature Methods*, 18(10), 1132–1135. <https://doi.org/10.1038/s41592-021-01256-7>
- Hicks, S. A., Strümke, I., Thambawita, V., Hammou, M., Riegler, M. A., Halvorsen, P., & Parasa, S. (2022). On evaluation metrics for medical applications of artificial intelligence. *Scientific Reports*, 12(1), Article 5979. <https://doi.org/10.1038/s41598-022-09954-8>
- Hofmann, P., Jöhnk, J., Protschky, D., & Urbach, N. (2020). Developing purposeful AI use cases – A structured method and its application in project management. In *15th International Conference on Wirtschaftsinformatik*. https://doi.org/10.30844/wi_2020_a3-hofmann
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- Hupont, I., Fernández-Llorca, D., Baldassarri, S., & Gómez, E. (2024). Use case cards: A use case reporting framework inspired by the European AI Act. *Ethics and Information Technology*, 26(2), 19. <https://doi.org/10.1007/s10676-024-09757-7>
- International Organization for Standardization. (2022a). *Information security, cybersecurity and privacy protection — Information security management systems — Requirements* (ISO/IEC Standard No. 27001:2022). <https://www.iso.org/standard/27001>
- International Organization for Standardization. (2022b). *Information technology — Artificial intelligence — Artificial intelligence concepts and terminology* (ISO/IEC Standard No. 22989:2022). <https://www.iso.org/standard/74296.html>
- International Organization for Standardization. (2023). *Information technology — Cloud computing — Part 1: Vocabulary* (ISO/IEC Standard No. 22123-1:2023). <https://www.iso.org/standard/82758.html>
- International Organization for Standardization. (2025). *Information security, cybersecurity and privacy protection — Privacy information management systems — Requirements and guidance* (ISO/IEC Standard No. 27701:2025). <https://www.iso.org/standard/27701>
- Joseph, V. R. (2022). Optimal ratio for data splitting. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 15(4), 531–538. <https://doi.org/10.1002/sam.11583>
- Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., . . . Hassabis, D. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873), 583–589. <https://doi.org/10.1038/s41586-021-03819-2>
- Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 237–285. <https://doi.org/10.1613/jair.301>
- Kahlon, N. K., & Singh, W. (2026). A systematic review of web scraping: Techniques, LLM-enhanced approaches, performance metrics, and legal–ethical issues. *Data & Knowledge Engineering*, 164, 102598. <https://doi.org/10.1016/j.datak.2026.102598>
- Kapoor, S., Cantrell, E. M., Peng, K., Pham, T. H., Bail, C. A., Gundersen, O. E., Hofman, J. M., Hullman, J., Lones, M. A., Malik, M. M., Nanayakkara, P., Poldrack, R. A., Raji, I. D., Roberts, M., Salganik, M. J., Serra-Garcia, M., Stewart, B. M., Vandewiele, G., & Narayanan, A. (2024). REFORMS: Consensus-based recommendations for machine-learning-based science. *Science Advances*, 10(18), eadk3452. <https://doi.org/10.1126/sciadv.adk3452>
- Kapoor, S., & Narayanan, A. (2023). Leakage and the reproducibility crisis in machine-learning-based science. *Patterns*, 4(9), Article 100804. <https://doi.org/10.1016/j.patter.2023.100804>
- Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D., & Grossman, T. (2023). Studying the effect of AI code generators on supporting novice learners in introductory programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Article 455). Association for Computing Machinery. <https://doi.org/10.1145/3544548.3580919>
- Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2017). *On large-batch training for deep learning: Generalization gap and sharp minima*. arXiv. <https://doi.org/10.48550/arXiv.1609.04836>
- Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., & Willing, C. (2016). Jupyter Notebooks – A publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas. Proceedings of the 20th International Confer-*

- ence on Electronic Publishing (pp. 87–90). IOS Press. <https://doi.org/10.3233/978-1-61499-649-1-87>
- Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine learning operations (MLOps): Overview, definition, and architecture. *IEEE Access*, 11, 31866–31879. <https://doi.org/10.1109/ACCESS.2023.3262138>
- Kroll, A. (2024). *Issue page ProSmith GitHub page, issue #10*. Retrieved May 22, 2025, from <https://github.com/AlexanderKroll/ProSmith/issues/10>
- Kroll, A., Ranjan, S., Engqvist, M. K. M., & Lercher, M. J. (2023). A general model to predict small molecule substrates of enzymes based on machine and deep learning. *Nature Communications*, 14(1), 2787. <https://doi.org/10.1038/s41467-023-38347-2>
- Kroll, A., Ranjan, S., & Lercher, M. J. (2024). A multi-modal transformer network for protein-small molecule interactions enhances predictions of kinase inhibition and enzyme-substrate relationships. *PLOS Computational Biology*, 20(5), e1012100. <https://doi.org/10.1371/journal.pcbi.1012100>
- Kwak, S. G., & Kim, J. (2025). Comprehensive reporting guidelines and checklist for studies developing and utilizing artificial intelligence models. *Korean Journal of Anesthesiology*, 78(3), 199–214. <https://doi.org/10.4097/kja.25075>
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. <https://doi.org/10.1038/nature14539>
- Li, J., Liu, L., Le, T. D., & Liu, J. (2020). Accurate data-driven prediction does not mean high reproducibility. *Nature Machine Intelligence*, 2(1), 13–15. <https://doi.org/10.1038/s42256-019-0140-2>
- Li, J., Yuan, X., & Kuruoglu, E. E. (2024). Exploring weight distributions and dependence in neural networks with alpha-stable distributions. *IEEE Transactions on Artificial Intelligence*, 5(11), 5519–5529. <https://doi.org/10.1109/TAI.2024.3409673>
- Linardatos, P., Papastefanopoulos, V., & Kotsiantis, S. (2021). Explainable AI: A review of machine learning interpretability methods. *Entropy*, 23(1), 18. <https://doi.org/10.3390/e23010018>
- Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (pp. 4768–4777). Curran Associates Inc.
- Lyu, Y., Li, H., Sayagh, M., Jiang, Z. M., & Hassan, A. E. (2021). An empirical study of the impact of data splitting decisions on the performance of AIops solutions. *ACM Transactions on Software Engineering and Methodology*, 30(4), Article 54. <https://doi.org/10.1145/3447876>
- Magnano, C. S., Mu, F., Russ, R. S., Cvetkovic, M., Treu, D., & Gitter, A. (2022). An approachable, flexible and practical machine learning workshop for biologists. *Bioinformatics*, 38(Supplement_1), i10–i18. <https://doi.org/10.1093/bioinformatics/btac233>
- Martin, C. H., Peng, T., & Mahoney, M. W. (2021). Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data. *Nature Communications*, 12(1), 4122. <https://doi.org/10.1038/s41467-021-24025-8>
- McKinney, W. (2010). Data structures for statistical computing in Python. In *Proceedings of the 9th Python in Science Conference (SciPy 2010)* (pp. 56–61). <https://doi.org/10.25080/Majora-92bf1922-00a>
- Monard, M. C., & Batista, G. E. A. P. A. (2002). Learning with skewed class distributions. *Advances in Logic, Artificial Intelligence and Robotics*, 85, 173–180.
- Narihira, T., Alonsogarcia, J., Cardinaux, F., Hayakawa, A., Ishii, M., Iwaki, K., Kemp, T., Kobayashi, Y., Mauch, L., Nakamura, A., Obuchi, Y., Shin, A., Suzuki, K., Tiedmann, S., Uhlich, S., Yashima, T., & Yoshiyama, K. (2021). *Neural network libraries: A deep learning framework designed from engineers' perspectives*. arXiv. <https://doi.org/10.48550/arXiv.2102.06725>
- Naser, M. Z. (2026). A review of machine learning with small and limited data. *Journal of Big Data*, 13(1), 18. <https://doi.org/10.1186/s40537-025-01346-9>
- National Institute of Standards and Technology. (2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)* (NIST AI 100-1). <https://doi.org/10.6028/NIST.AI.100-1>
- OECD. (2019). The technical landscape. In *Artificial intelligence in society* (pp. 19–43). OECD Publishing. <https://doi.org/10.1787/eedfee77-en>
- Ojala, M., & Garriga, G. C. (2010). Permutation tests for studying classifier performance. *Journal of Machine Learning Research*, 11, 1833–1863.
- Ojika, D., Patel, B., Reina, G. A., Boyer, T., Martin, C., & Shah, P. (2020). *Addressing the memory bottleneck in AI model training*. arXiv. <https://doi.org/10.48550/arXiv.2003.08732>
- OWASP. (2024). *OWASP Top 10 for LLM Applications 2025*. <https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/>
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., . . . Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.
- Pham, H. V., Qian, S., Wang, J., Lutellier, T., Rosenthal, J., Tan, L., Yu, Y., & Nagappan, N. (2021). Problems and opportunities in training deep learning software systems: An analysis of variance. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering* (pp. 771–783). Association for Computing Machinery. <https://doi.org/10.1145/3324884.3416545>
- Pinski, M., & Benlian, A. (2024). AI literacy for users – A comprehensive review and future research directions of

- learning methods, components, and effects. *Computers in Human Behavior: Artificial Humans*, 2(1), 100062. <https://doi.org/10.1016/j.chbah.2024.100062>
- Pion-Tonachini, L., Bouchard, K., Martin, H., Peisert, S., Holtz, B., Aswani, A., Dwivedi, D., Wainwright, H., Pilińska, G., Nachman, B., Marrone, B., Falco, N., Prabhat, Arnold, D., Wolf-Yadlin, A., Powers, S., Climer, S., Jackson, Q., Carlson, T., & Brown, J. (2021). *Learning from learning machines: A new generation of AI technology to meet the needs of science*. arXiv. <https://doi.org/10.48550/arXiv.2111.13786>
- Powers, D. (2011). Evaluation: From precision, recall and F-measure to ROC, informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1), 37–63.
- Prechelt, L. (2012). Early stopping — But when? In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade* (2nd ed., pp. 53–67). Springer. https://doi.org/10.1007/978-3-642-35289-8_5
- Python Software Foundation. (n.d.). *The Python language reference: 1.1. Alternate implementations*. <https://docs.python.org/3.14/reference/introduction.html#alternate-implementations>
- Raina, R., Madhavan, A., & Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning* (pp. 873–880). Association for Computing Machinery. <https://doi.org/10.1145/1553374.1553486>
- Rajbhandari, S., Rasley, J., Ruwase, O., & He, Y. (2020). ZeRO: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1–16). <https://doi.org/10.1109/SC41405.2020.00024>
- Rajput, D., Wang, W.-J., & Chen, C.-C. (2023). Evaluation of a decided sample size in machine learning applications. *BMC Bioinformatics*, 24(1), 48. <https://doi.org/10.1186/s12859-023-05156-9>
- Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). “Why should I trust you?”: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1135–1144). Association for Computing Machinery. <https://doi.org/10.1145/2939672.2939778>
- Roh, Y., Heo, G., & Whang, S. E. (2021). A survey on data collection for machine learning: A big data – AI integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, 33, 1328–1347.
- Saito, T., & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, 10(3), e0118432. <https://doi.org/10.1371/journal.pone.0118432>
- Sajadieh, S., Fattorini, L., Perrault, R., Gil, Y., Parli, V., Santarlasci, L., Pava, J., Maslej, N., Altman, R., Brynjolfsson, E., Brodley, C., Clark, J., Dignum, V., Kumar, V., Landay, J., Lyons, T., Manyika, J., Niebles, J. C., Shoham, Y., ... Weld, D. (2026). *The AI Index 2026 annual report*. AI Index Steering Committee, Institute for Human-Centered AI, Stanford University. <https://doi.org/10.48550/arXiv.2606.15708>
- Sergeyuk, A., Golubev, Y., Bryksin, T., & Ahmed, I. (2025). Using AI-based coding assistants in practice: State of affairs, perceptions, and ways forward. *Information and Software Technology*, 178, 107610. <https://doi.org/10.1016/j.infsof.2024.107610>
- Sohoni, N. S., Aberger, C. R., Leszczynski, M., Zhang, J., & Ré, C. (2019). *Low-memory neural network training: A technical report*. arXiv. <https://doi.org/10.48550/arXiv.1904.10631>
- Stimulus. (2024). *Just Transition Fund C-Recycle*. Retrieved September 28, 2026, from https://www.stimulus.nl/just-transition-fund/avada_portfolio/c-recycle/
- SURF. (2026a). *History of SURF*. Retrieved February 17, 2026, from <https://www.surf.nl/en/about/the-history-of-surf>
- SURF. (2026b). *SURF Research Access Management*. Retrieved February 17, 2026, from <https://www.surf.nl/en/services/identity-access-management/surf-research-access-management>
- TensorFlow. (2024). *Save and load models*. Retrieved July 31, 2026, from https://www.tensorflow.org/tutorials/keras/save_and_load
- Tian, K., Qiao, L., Liu, B., Jiang, G., & Li, D. (2025). *A survey on memory-efficient large-scale model training in AI for science*. arXiv. <https://doi.org/10.48550/arXiv.2501.11847>
- Treude, C., & Gerosa, M. A. (2025). How developers interact with AI: A taxonomy of human-AI collaboration in software engineering. In *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)* (pp. 236–240). <https://doi.org/10.1109/Forge66646.2025.00033>
- U.S. Department of Justice. (2018). *CLOUD Act resources*. <https://www.justice.gov/criminal/cloud-act-resources>
- van Velzen, M., van der Willigen, R. F., de Beer, V. J., de Graaf-Waar, H. I., Janssen, E. R. C., van Leeuwen, S., van der Willigen, M. F., van der Willigen, M. J., Renardus, G., El Maaroufi, R., Satimin, S. J., Hartog, L. M., Hulsens, T., van Meeteren, N. L. U., & Scheper, M. C. (2025). Privacy-, linguistic-, and information-preserving synthesis of clinical documentation through generative agents. *Frontiers in Artificial Intelligence*, 8, Article 1644084. <https://doi.org/10.3389/frai.2025.1644084>
- Walsh, I., Fishman, D., Garcia-Gasulla, D., Titma, T., Pollastri, G., Capriotti, E., Casadio, R., Capella-Gutierrez, S., Cirillo, D., Del Conte, A., Dimopoulos, A. C., Del Angel, V. D., Dopazo, J., Fariselli, P., Fernández, J. M., Huber, F., Kreshuk, A., Lenaerts, T., Martelli, P. L., ... ELIXIR Machine Learning Focus Group. (2021). DOME: Recommendations for supervised machine learning validation in biology. *Nature Methods*, 18(10), 1122–1127. <https://doi.org/10.1038/s41592-021-01205-4>
- Wang, J., Xiao, G., Zhang, S., Lei, H., Liu, Y., & Sui, Y. (2023). Compatibility issues in deep learning systems: Problems and opportunities. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and*

- Symposium on the Foundations of Software Engineering* (pp. 476–488). Association for Computing Machinery. <https://doi.org/10.1145/3611643.3616321>
- Wang, Z., Wang, P., Liu, K., Wang, P., Fu, Y., Lu, C., Aggarwal, C. C., Pei, J., & Zhou, Y. (2026). A comprehensive survey on data augmentation. *IEEE Transactions on Knowledge and Data Engineering*, 38(1), 47–66. <https://doi.org/10.1109/TKDE.2025.3622600>
- Wegmeth, L., Vente, T., Purucker, L., & Beel, J. (2023). The effect of random seeds for data splitting on recommendation accuracy. In *Proceedings of the 3rd Workshop on Perspectives on the Evaluation of Recommender Systems (PERSPECTIVES 2023), co-located with the 17th ACM Conference on Recommender Systems (RecSys 2023)* (Vol. 3476). CEUR-WS.org. <https://ceur-ws.org/Vol-3476/paper4.pdf>
- Weiss, G., & Provost, F. (2001). *The effect of class distribution on classifier learning: An empirical study* (Technical Report No. ML-TR-44). Rutgers University. <https://doi.org/10.7282/t3-vpfw-sf95>
- Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., . . . Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1), 160018. <https://doi.org/10.1038/sdata.2016.18>
- Yang, J., Yonack, N., Zyskowski, K., Yarats, D., Ho, J., & Ma, J. (2025). *The adoption and usage of AI agents: Early evidence from Perplexity*. SSRN. <https://doi.org/10.2139/ssrn.5887343>
- Zhou, X., Liang, P., Zhang, B., Li, Z., Ahmad, A., Shahin, M., & Waseem, M. (2025). Exploring the problems, their causes and solutions of AI pair programming: A study on GitHub and Stack Overflow. *Journal of Systems and Software*, 219, 112204. <https://doi.org/10.1016/j.jss.2024.112204>