



SCHOOL
FOR ADVANCED
STUDIES
LUCCA

Machine Learning for Software Analysis (MLSA)

Fabio Pinelli

Associate Professor
Sysma Research Unit
IMT School For Advanced Studies Lucca
fabio.pinelli@imtlucca.it
<https://sysma.imtlucca.it/pages/fabio-pinelli/>

- Myself introduction
- Course presentation
- Introduction on software analysis
- Introduction to Machine learning
- Motivations of Machine Learning for Software analysis
- Research activities on MLSA

I am Associate professor in Computer Science at IMT School for advanced studies Lucca

Machine learning/Data mining on spatio-temporal data

Machine learning applied on Blockchain and Binary code analysis

Federated Learning on spatio-temporal data

<https://scholar.google.it/citations?user=rfqxMS0AAAAJ&hl=it>

Phd held at University of Pisa



IBM Research in Dublin

IBM Research

Vodafone Italia





SCUOLA
ALTI STUDI
LUCCA

School for advanced
studies (PhD)

One of the 6 school
of Excellence in Italy



**Mirco
Tribastone**

Full Professor
SySMA Head
Modeling and
Simulation, Software
Performance
Engineering,
Computational Methods



**Alessandro
Armando**

Full Professor
(IMT and University
of Genova)

Cybersecurity,
Computer security



**Alessandro
Betti**

Assistant Professor

Machine Learning,
Computer Vision,
Lifelong Learning



**Lorenzo
Ceragioli**

Assistant Professor

Formal Methods,
Software Security,
Quantum
Communication and
Computing



**Gabriele
Costa**

Associate Professor

Cybersecurity,
Penetration Testing,
Formal Methods,
Software Verification,
Vulnerability
Assessment



**Letterio
Galletta**

Assistant Professor

Software Security,
Software Verification,
Formal Methods,
Programming
Languages



**Emilio
Incerto**

Assistant Professor

Software Performance
Modeling and Control,
Layered Queueing
Networks, Autoscaling,
Cloud Computing



**Cosimo
Perini Brogi**

Assistant Professor

Formal Methods,
Mathematical
Foundations of CS,
Proof Theory, Certified
Programming, Type
Theory, Non-classical
Logics



**Fabio
Pinelli**

Associate Professor

Data Science, Machine
Learning,
Spatio-temporal
Machine Learning



**Simone
Soderi**

Assistant Professor

Physical Layer
Security,
6G Security: Optical
communications,
Covert channels,
Security in critical
infrastructure systems



**Mirco
Tribastone**

Full Professor
SySMA Head
Modeling and
Simulation, Software
Performance
Engineering,
Computational Methods



**Alessandro
Armando**

Full Professor
(IMT and University
of Genova)

Cybersecurity,
Computer security



**Alessandro
Betti**

Assistant Professor

Machine Learning,
Computer Vision,
Lifelong Learning



**Lorenzo
Ceragioli**

Assistant Professor

Formal Methods,
Software Security,
Quantum
Communication and
Computing



**Gabriele
Costa**

Associate Professor

Cybersecurity,
Penetration Testing,
Formal Methods,
Software Verification,
Vulnerability
Assessment



**Letterio
Galletta**

Assistant Professor

Software Security,
Software Verification,
Formal Methods,
Programming
Languages



**Emilio
Incerto**

Assistant Professor

Software Performance
Modeling and Control,
Layered Queueing
Networks, Autoscaling,
Cloud Computing



**Cosimo
Perini Brogi**

Assistant Professor

Formal Methods,
Mathematical
Foundations of CS,
Proof Theory, Certified
Programming, Type
Theory, Non-classical
Logics



**Fabio
Pinelli**

Associate Professor

Data Science, Machine
Learning,
Spatio-temporal
Machine Learning



**Simone
Soderi**

Assistant Professor

Physical Layer
Security,
6G Security: Optical
communications,
Covert channels,
Security in critical
infrastructure systems

Different computer science topics from Formal methods to Machine Learning and CyberSecurity, performance, etc.

... and so why here at UNIFI?

1. What do you know about Machine Learning?
2. How do you use LLMs?
3. What do you know about Software Analysis?
4. What do you expect to learn in these class?

The course contains some **theoretical** and **practical** aspects

- Understand fundamental machine learning concepts and algorithms
- Explore the application of machine learning techniques to software analysis tasks
- Develop practical skills in collecting, preprocessing, and analyzing software data
- Apply machine learning models to detect bugs, assess code quality, and make recommendations for software improvement
- Evaluate the strengths and limitations of using machine learning in software analysis

We will **read** some papers related to this topic

We will **test** some practical examples on real use cases

Moodle: you can access the page without password

Exam: Most likely a project.

Lessons:

Tuesday: 10:30 - 13:30

Thursday: 11:30 - 13:30

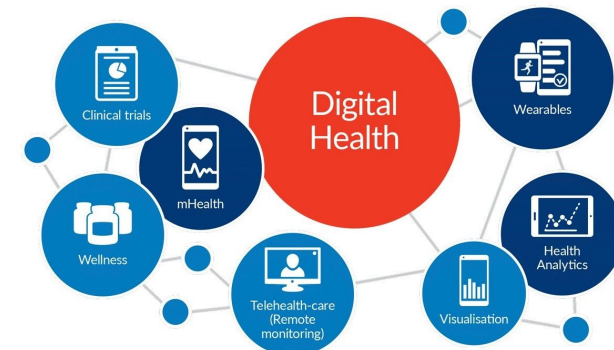
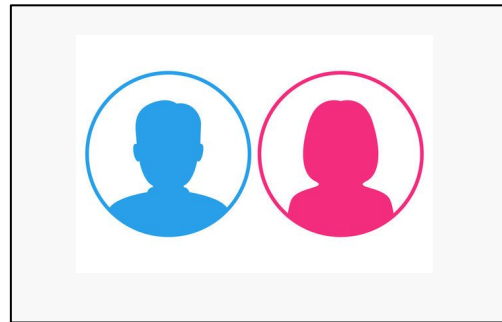
You can contact me anytime at: fabio.pinelli@imtlucca.it

<https://sysma.imtlucca.it/people/fabio-pinelli>

Slack:

https://join.slack.com/t/mlsaarea-di-lavoro/shared_invite/zt-3e19mvgwi-dcOjh6vXV98FGel4kTSukg

So... Let's start



You want the code to be...

Thus the software should be?

Correct

Correctness is important for good software. There shouldn't be faults with specification, design or implementation.

Usable

Users should be able to learn and use a system easily.

Efficient

The less resource a piece of software uses, the better. Processor, battery, bandwidth, memory and disk space usage should be minimized.

Reliable

A system that can perform the required functions stably is important. Failures should be as rare as possible.

Secure

Security should be taken into account. Our software shouldn't let attackers access unauthorized resources.

Adaptability

A system that can be used without modification for different situations it's good.

Accuracy

The accuracy of its outputs is good. This measures if the software outputs the right results for users.

Robustness

If a system is still working after getting invalid inputs and stressful environmental conditions, then it's good for our system.

Maintainability

The ease in which an existing system can be changed is important. The easier that we can make changes, the better.

Portability

A system that operates in different environments from which it's originally designed makes it good.

Reusability

The more reusable parts that a piece of software has, the better.

Using reusable parts means that we don't have to reuse them from scratch.

Readability

Easy to read code is easy to change code. If we understand them faster, then we can make changes faster and in a less error-prone way.

Testability

Making our software system testable is critical. If our code is easy to write unit tests for, then that's good.

Understandability

The ability for us to understand our system in a global view or at the detailed code level is important. Easier to understand systems are more coherent.

With good software, it's harder for us to create defects, and it's faster to make changes.

Today, the cost of **software development is less than 50% programming** with testing, debugging, security assessments, and similar tasks taking more resources than developing the software itself.

As a result, there is an increasing focus in the software industry **on using tools to write better software.**

These tools can take the form of **testing tools that help find bugs**

They can also take the **form of analysis techniques** that have the goal of building a **stronger code foundation** with fewer areas where defects can emerge

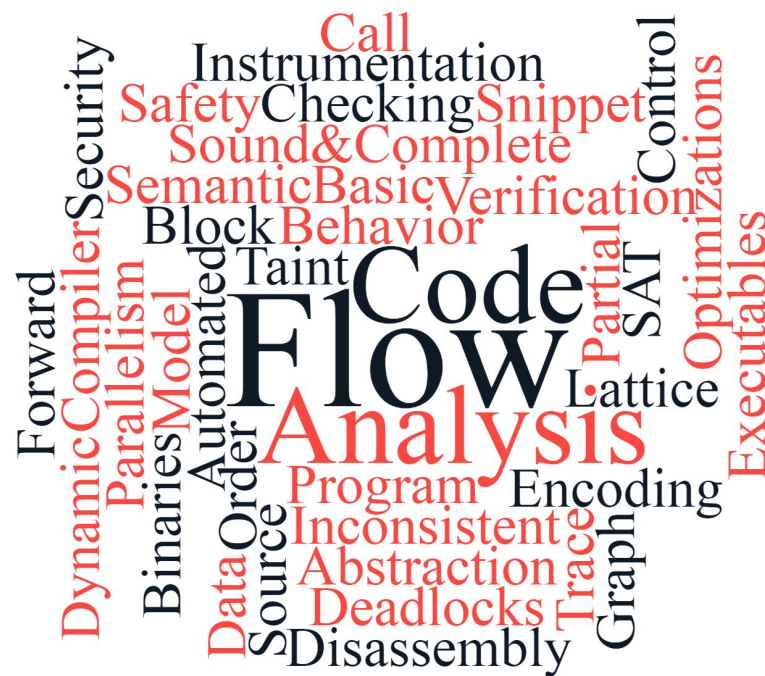
The end result being **less risk in the software development lifecycle.**

Software or Program analysis refers to the process of examining and evaluating software artifacts:

- source code,
- executables, and
- documentation

to gain insights into various aspects of software **quality, performance, security, and maintainability.**

The primary goal of program analysis is to improve software reliability, efficiency, and maintainability while reducing the risk of errors, vulnerabilities, and defects.

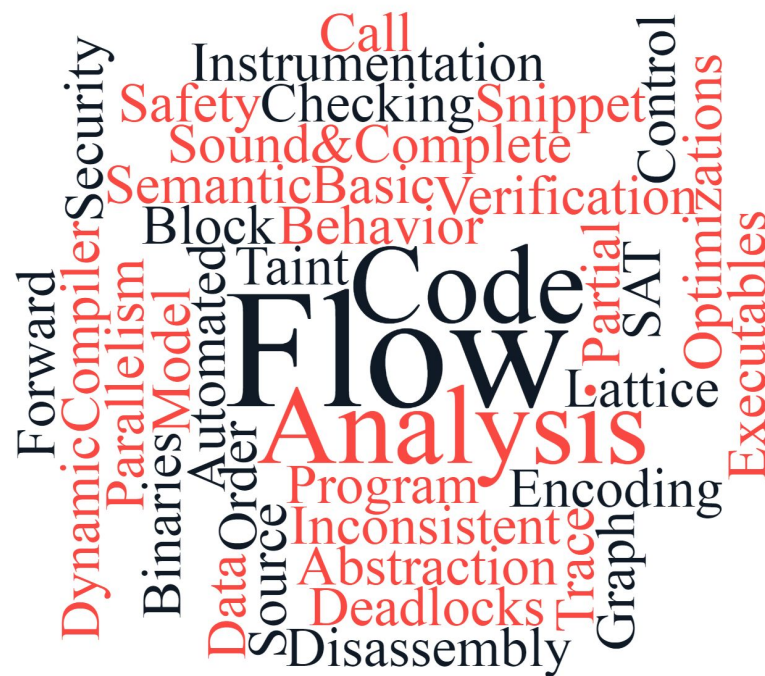


Software analysis encompasses a range of techniques, methods, and tools aimed at

- understanding,
- evaluating, and
- improving

software artifacts throughout the software development lifecycle.

It involves **static** and **dynamic** analysis approaches to identify software **defects**, **security vulnerabilities**, **performance bottlenecks**, and **compliance issues**.



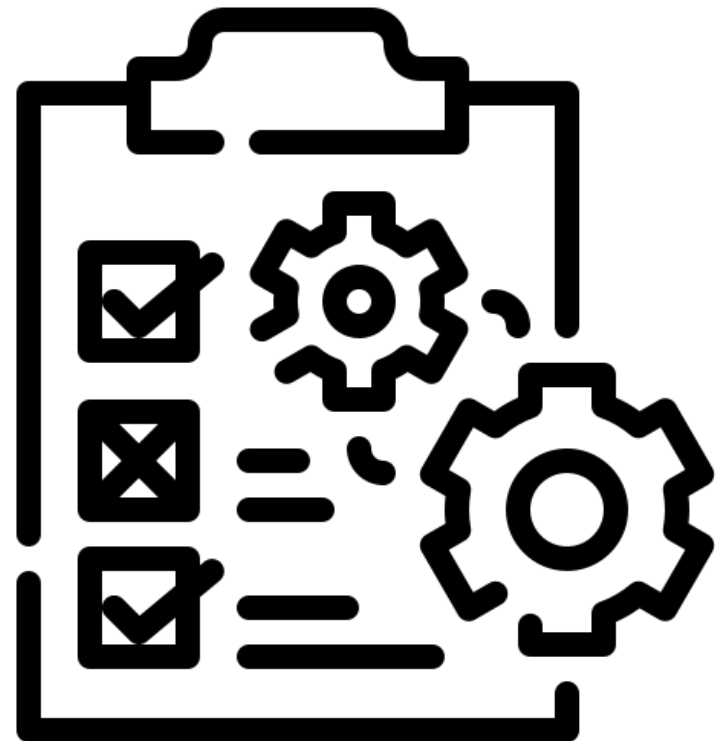
- Static analysis involves examining software artifacts **without executing the code**. It analyzes source code, configuration files, and documentation to identify potential defects, coding standards violations, and security vulnerabilities.
- **Techniques** include syntax checking, data flow analysis, control flow analysis, and abstract interpretation.
- **Tools:** Static analysis tools such as linters, static code analyzers, and code review platforms automate the process of identifying code issues and enforcing coding standards. Reverse engineering techniques are also used.



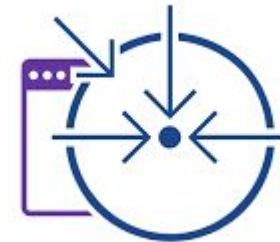
- Dynamic analysis involves analyzing software behavior **during execution**. It focuses on runtime properties, memory usage, performance characteristics, and error handling.
- **Techniques** include profiling, memory analysis, code coverage analysis, and runtime monitoring.
- **Tools:** Dynamic analysis tools such as profilers, debuggers, memory analyzers, and dynamic testing frameworks capture runtime information and diagnose performance issues, memory leaks, and runtime errors.



- Model checking is a **formal verification** technique used to systematically verify whether a software model **satisfies a set of desired properties or specifications**.
- It involves exhaustively exploring all possible states of a finite-state model to identify potential violations of safety and liveness properties.
- Tools: Model checking tools such as SPIN, NuSMV, and Alloy provide automated verification capabilities for concurrent and distributed systems.

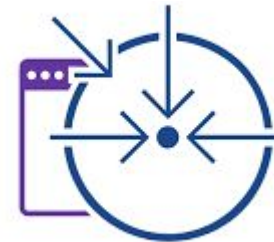
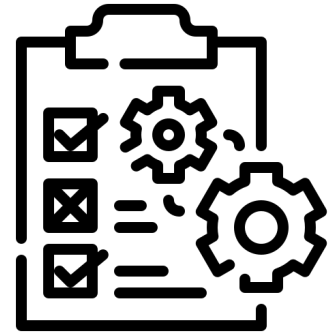


- Fuzz testing, also known as **fuzzing**, involves providing invalid, unexpected, or random inputs to a software system to uncover vulnerabilities, crashes, and unexpected behaviors.
- It helps identify security vulnerabilities, memory corruption issues, and boundary condition errors in software applications.
- Tools: Fuzz testing frameworks such as AFL, Peach, and Radamsa automate the process of generating and executing diverse input data to stress-test software systems.



Overall, program or software analysis plays a crucial role in ensuring software quality, reliability, and security throughout the software development lifecycle.

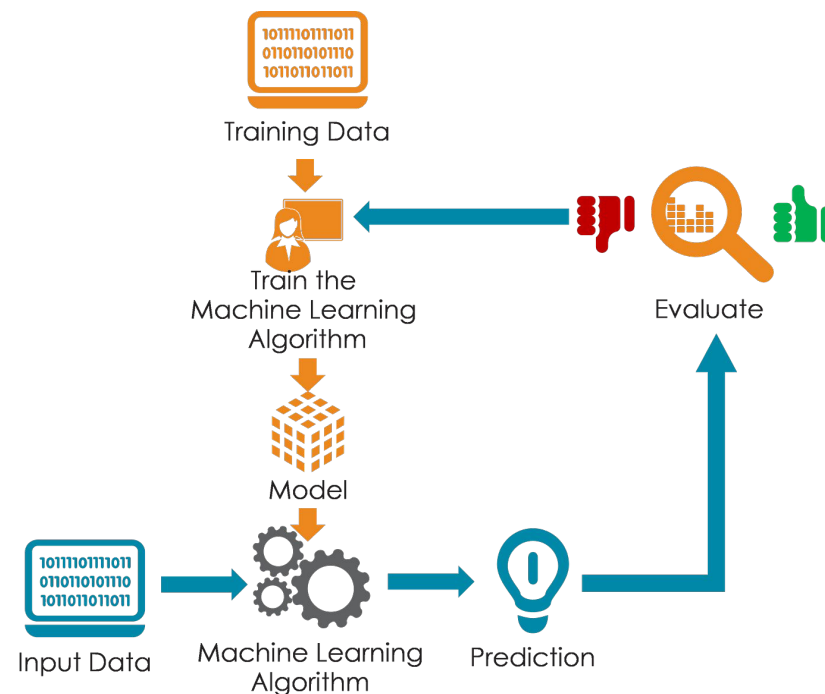
By employing a combination of static analysis, dynamic analysis, formal verification, and testing techniques, organizations can identify and mitigate software defects, vulnerabilities, and performance issues early in the development process.



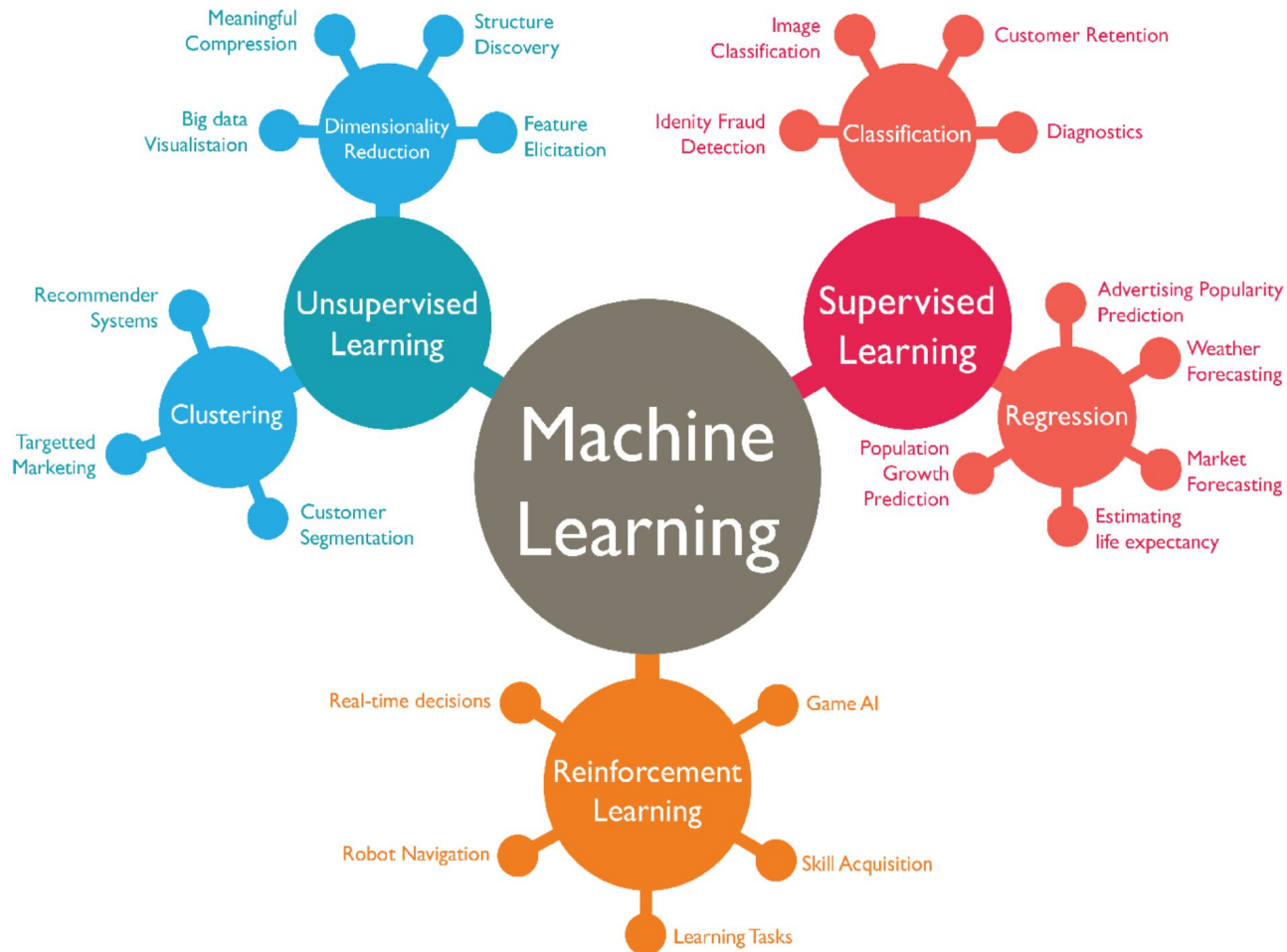
Machine Learning enables systems to learn from data and make predictions or decisions

Basic Components of Machine Learning:

- **Data:** Machine learning algorithms require data to learn patterns, relationships, and trends. Data can be structured or unstructured and may include **features** or **attributes** relevant to the problem.
- **Model:** A machine learning model is **a mathematical representation** of the patterns and relationships in the data. Models can be trained using various algorithms to make **predictions** or **classifications**.
- **Training:** The process of training a machine learning model involves feeding it with labeled data (supervised learning) or unlabeled data (unsupervised learning) and adjusting its parameters to minimize errors or maximize accuracy.
- **Inference:** Once trained, the machine learning model can make predictions or classifications on new, unseen data based on the patterns learned during training.



Machine learning tasks



Classification: Classifying data points into predefined categories or classes based on their features.

Features: Input variables or attributes that describe the characteristics of the data points.

Classes: Categories or labels that the data points belong to.

Classifier: Machine learning algorithm used to learn the relationship between input features and classes and make predictions on new data.

Applications of Classification:

Email Spam Detection: Classify incoming emails as spam or non-spam based on their content and features.

Medical Diagnosis: Predict the presence or absence of a disease based on patient symptoms, medical history, and diagnostic tests.

Sentiment Analysis: Determine the sentiment of text documents, such as reviews or social media posts, as positive, negative, or neutral.

Credit Risk Assessment: Classify loan applicants as low, medium, or high risk based on their credit history, income, and financial data.

Regression: Predicting continuous values or numerical outcomes based on input features.

Features: Independent variables or attributes that influence the target variable.

Target Variable: Dependent variable or outcome that we want to predict.

Regression Model: Mathematical function that maps input features to the target variable.

Application fields:

Stock Price Prediction: Forecast future stock prices based on historical market data, economic indicators, and company performance metrics.

Demand Forecasting: Predict consumer demand for products or services based on historical sales data, marketing efforts, and external factors.

House Price Estimation: Estimate the market value of residential properties based on features such as location, size, and amenities.

Weather Forecasting: Predict future weather conditions such as temperature, precipitation, and wind speed based on historical climate data and atmospheric variables.

Clustering: Grouping similar data points together into clusters based on their inherent characteristics or patterns.

Data Points: Observations or instances in the dataset that are grouped based on similarity measures.

Centroids: Representative points within each cluster that summarize the characteristics of the cluster.

Distance Metric: Measure of similarity or dissimilarity between data points, often computed using Euclidean distance, Manhattan distance, or cosine similarity.

Applications of Clustering:

Customer Segmentation: Group customers with similar behaviors, preferences, or demographics for targeted marketing strategies.

Image Segmentation: Divide images into meaningful regions or segments for object recognition, image compression, and computer vision tasks.

Anomaly Detection: Identify outliers or unusual patterns in data that deviate from normal behavior.

Document Clustering: Organize text documents into clusters based on topics, themes, or content similarity for information retrieval and categorization.

Market Basket Analysis: Discover associations and patterns in transaction data to understand purchasing behavior and recommend related products.

Anomaly Detection: Identifying outliers or unusual patterns in data that deviate from normal behavior.

Normal Behavior: Baseline or typical patterns of behavior observed in the data under normal operating conditions.

Anomalies: Data points or events that deviate from normal behavior and may indicate potential errors, fraud, or unusual activity.

Detection Methods: Algorithms and techniques used to identify anomalies based on statistical analysis, machine learning, or domain-specific rules.

Applications of Anomaly Detection:

Fraud Detection: Identify fraudulent transactions, activities, or behavior in financial transactions, cybersecurity, insurance claims, and healthcare billing.

Network Intrusion Detection: Detect unauthorized access, malicious attacks, or suspicious activity in computer networks and systems.

Equipment Failure Prediction: Predict potential failures or malfunctions in machinery, equipment, or infrastructure based on abnormal sensor readings or operational parameters.

Health Monitoring: Monitor physiological data, patient vital signs, or medical imaging to detect anomalies indicative of disease, infection, or health risks.

Dimensionality Reduction: Reducing the number of input features while retaining relevant information to improve model efficiency and interpretability.

High-dimensional datasets often suffer from the **curse of dimensionality**, leading to increased computational complexity, overfitting, and reduced interpretability.

Dimensionality reduction helps alleviate these issues by extracting the **most relevant features** and reducing **noise** and **redundancy** in the data.

Feature Selection: Selecting a subset of the original features based on their relevance to the target variable or predictive power.

Feature Extraction: Creating new, lower-dimensional features that capture the most important information in the data.

Applications of Dimensionality Reduction:

Data Visualization: Visualize high-dimensional data in two or three dimensions for exploratory analysis and interpretation.

Feature Engineering: Create informative and concise features for downstream machine learning tasks, such as classification, regression, and clustering.

Image and Signal Processing: Reduce the dimensionality of image and signal data for compression, denoising, and feature extraction.

Text Mining and Natural Language Processing (NLP): Extract semantic meaning and structure from text documents by reducing the dimensionality of word embeddings and document representations.

Why machine learning for software analysis

Handling Complexity

Modern software systems are highly complex, often consisting of **millions of lines of code** with intricate **dependencies** and **interactions**.

Machine learning algorithms excel at processing **large volumes** of data and identifying **complex patterns**, making them well-suited for **analyzing software systems of varying scales and complexities**.



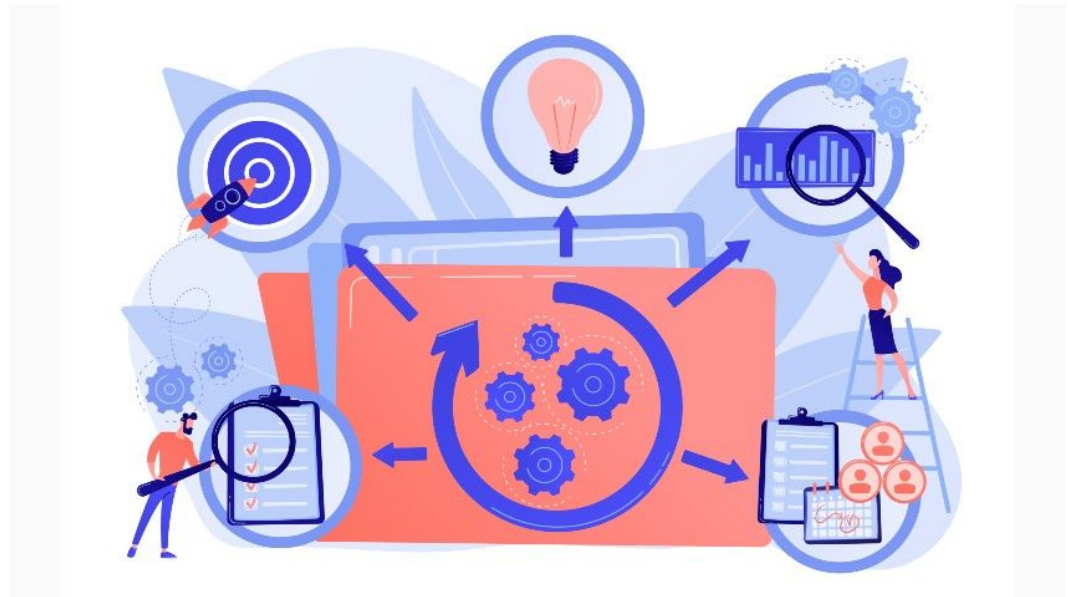
Automating Analysis Tasks

Traditional program analysis techniques often involve **manual inspection**, which can be **time-consuming** and **error-prone**, especially for large codebases.

Machine learning enables the **automation** of various program analysis tasks:

- bug detection,
- code optimization, and
- software testing,

allowing developers to focus on higher-level design and problem-solving.

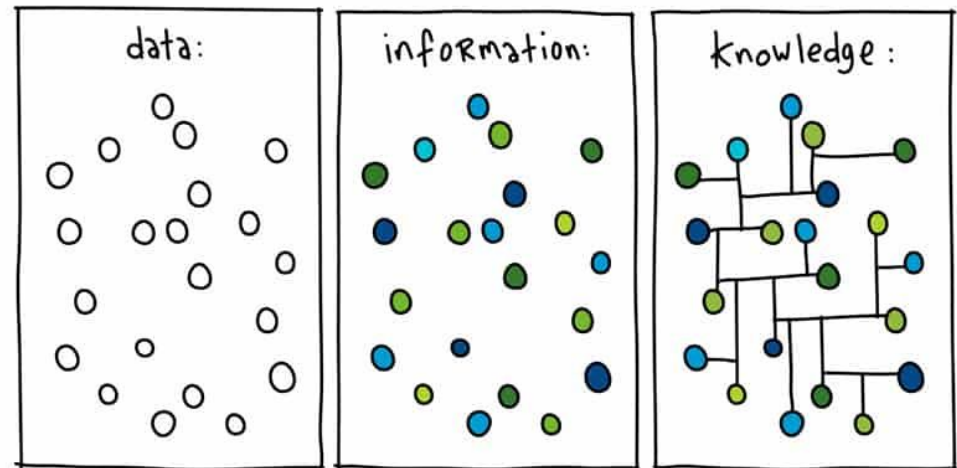


Why machine learning for software analysis

Learning from Data

Machine learning algorithms **learn from data**, which is abundant in software development projects.

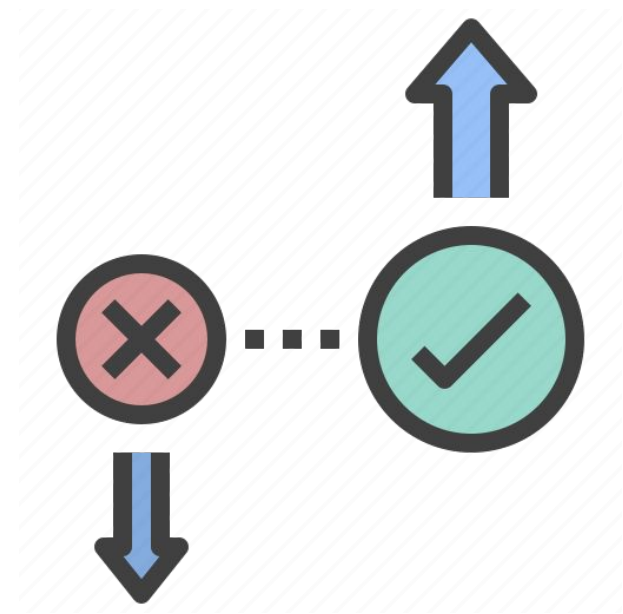
By training on historical **code repositories**, **bug reports**, **version control histories**, and **user feedback**, machine learning models can capture valuable insights and patterns that **may not be apparent** through **manual analysis** alone.



Improving Accuracy and Efficiency

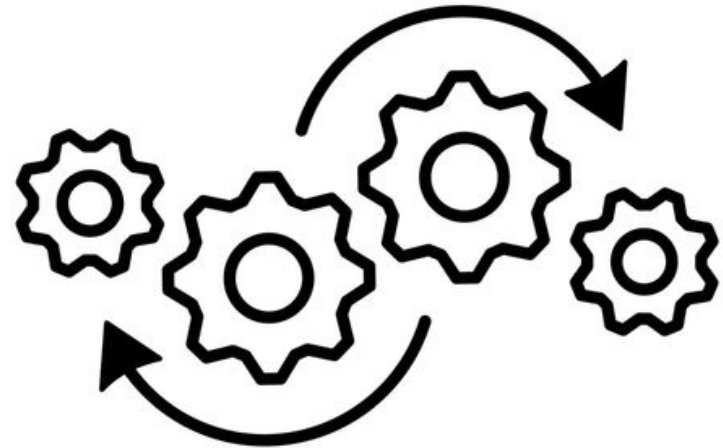
Machine learning techniques can enhance the accuracy and efficiency of program analysis by identifying **subtle patterns** and **anomalies** in software behavior that may **be difficult for humans to detect**.

Additionally, machine learning models can **prioritize analysis efforts** by focusing on the most critical areas of the codebase, thereby improving overall development productivity.



Adapting to Change

Software systems evolve over time in response to changing requirements, technologies, and user needs. Machine learning models can adapt and evolve alongside software systems, continuously learning from new data and feedback to improve their performance and relevance in dynamic environments.



Predictive Capabilities

Machine learning enables predictive analytics in program analysis, allowing developers to anticipate and mitigate potential issues before they arise. For example, machine learning models can forecast software defects, identify performance bottlenecks, and predict code maintainability based on historical data and patterns.



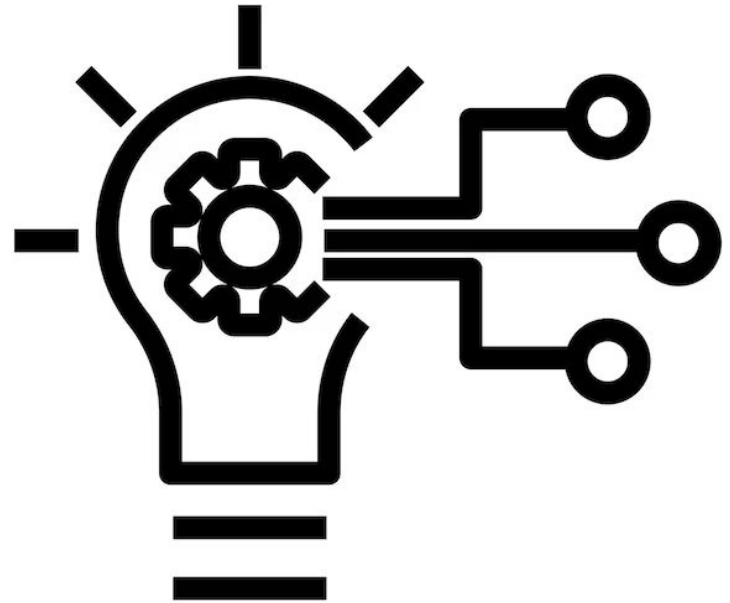
Supporting Decision-Making

Machine learning provides valuable insights and recommendations to support decision-making processes in software development. By analyzing software metrics, code patterns, and user behavior, machine learning models can inform decisions related to resource allocation, feature prioritization, and software architecture design.



Enabling Innovation

The application of machine learning in program analysis opens up new avenues for innovation in software development practices. From automated code generation to intelligent debugging tools, machine learning enables developers to explore novel approaches and solutions to complex software engineering challenges.



A survey on machine learning techniques applied to source code

Tushar Sharma, Maria Kechagia, Stefanos Georgiou, Rohit Tiwari, Indira Vats, Hadi Moazen, Federica Sarro
Journal of Systems and Software, 2024

Reports that:

- The use of ML techniques **is constantly increasing** for source code analysis.
- A wide range Software Engineering (SE) tasks involving source code analysis use ML.
- The study identifies challenges in the field and potential mitigations.
- They identify commonly used datasets and tools used in the field.

Search terms and corresponding relevant studies found in the second round of phase 1.

| Category | Search terms | #Studies |
|------------------------|---|----------|
| Vulnerability analysis | Feature learning in source code | 9 |
| | Vulnerability prediction in source code using machine learning | 70 |
| | Deep learning-based vulnerability detection | 8 |
| | Malicious code detection with machine learning | 45 |
| Testing | Word embedding in software testing | 2 |
| | Automated Software Testing with machine learning | 12 |
| | Optimal machine learning based random test generation | 1 |
| Refactoring | Source code refactoring prediction with machine learning | 39 |
| | Automatic clone recommendation with machine learning | 14 |
| | Machine learning based refactoring detection tools | 16 |
| | Search-based refactoring with machine learning | 6 |
| Quality assessment | Web service anti-pattern detection with machine learning | 25 |
| | Code smell prediction models | 34 |
| | Machine learning-based approach for code smells detection | 17 |
| | Software design flaw prediction | 37 |
| | Linguistic smell detection with machine learning | 2 |
| | Software defect prediction with machine learning | 66 |
| Program synthesis | Machine learning based software fault prediction | 35 |
| | Automated program repair methods with machine learning | 45 |
| | Program generation with machine learning | 2 |
| | Object-oriented program repair with machine learning | 15 |
| | Predicting patch correctness with machine learning | 3 |
| Program comprehension | Multihunk program repair with machine learning | 9 |
| | Autogenerated code with machine learning | 6 |
| | Commits analysis with machine learning | 34 |
| Code summarization | Supplementary bug fixes with machine learning | 9 |
| | Automatic source code summarization with machine learning | 43 |
| | Automatic commit message generation with machine learning | 19 |
| Code review | Comments generation with machine learning | 11 |
| | Security flaws detection in source code with machine learning | 20 |
| Code representation | Intelligent source code security review with machine learning | 2 |
| | Design pattern detection with machine learning | 10 |
| | Human-machine-comprehensible software representation | 1 |
| Code completion | Feature learning in source code | 6 |
| | Missing software architectural tactics prediction with machine learning | 1 |
| | Software system quality analysis with machine learning | 6 |
| | Package-level tactic recommendation generation in source code | 3 |
| | Identifier prediction in source code | 13 |
| | Token prediction in source code | 29 |

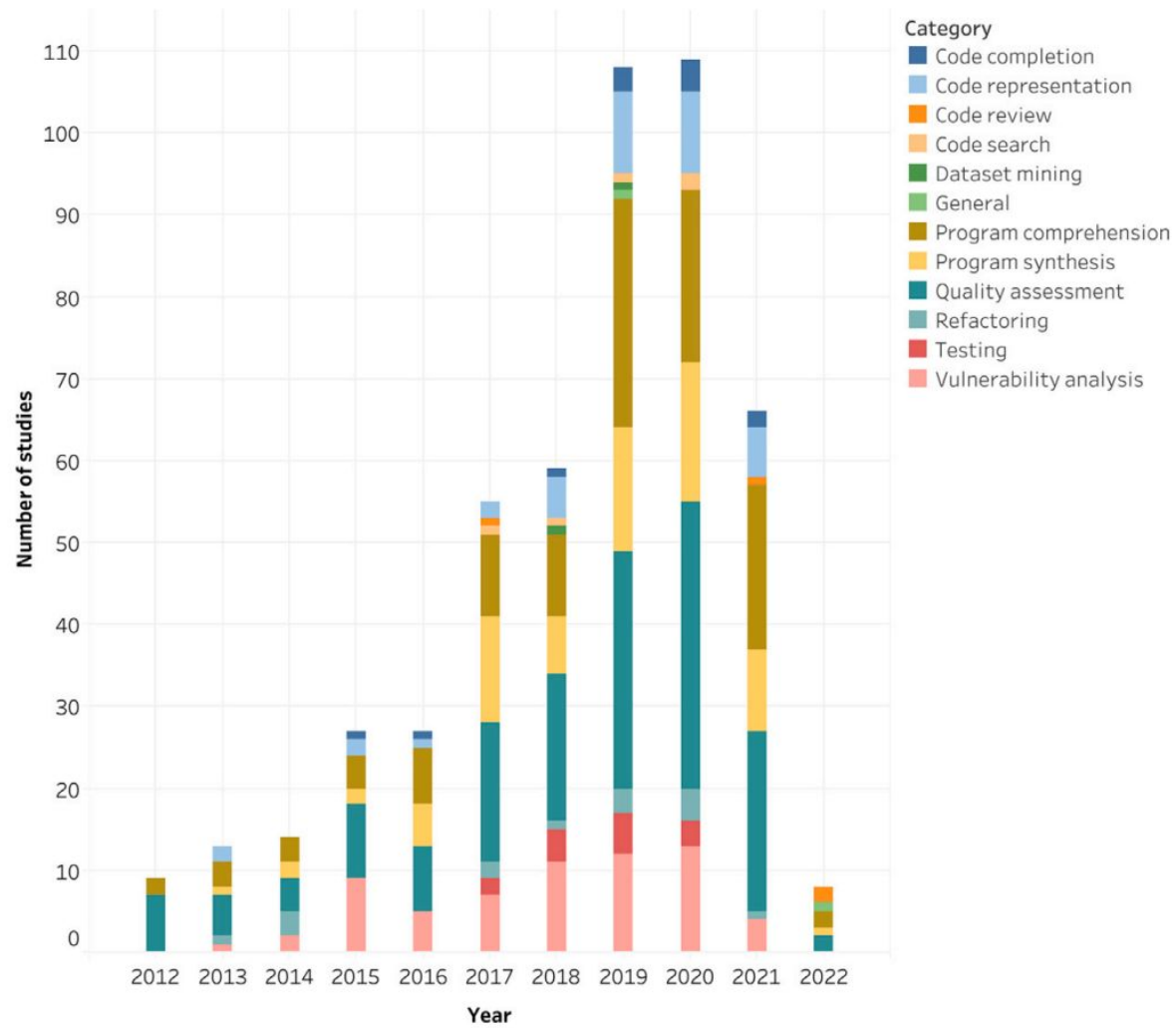


Fig. 2. Category-wise distribution of studies.

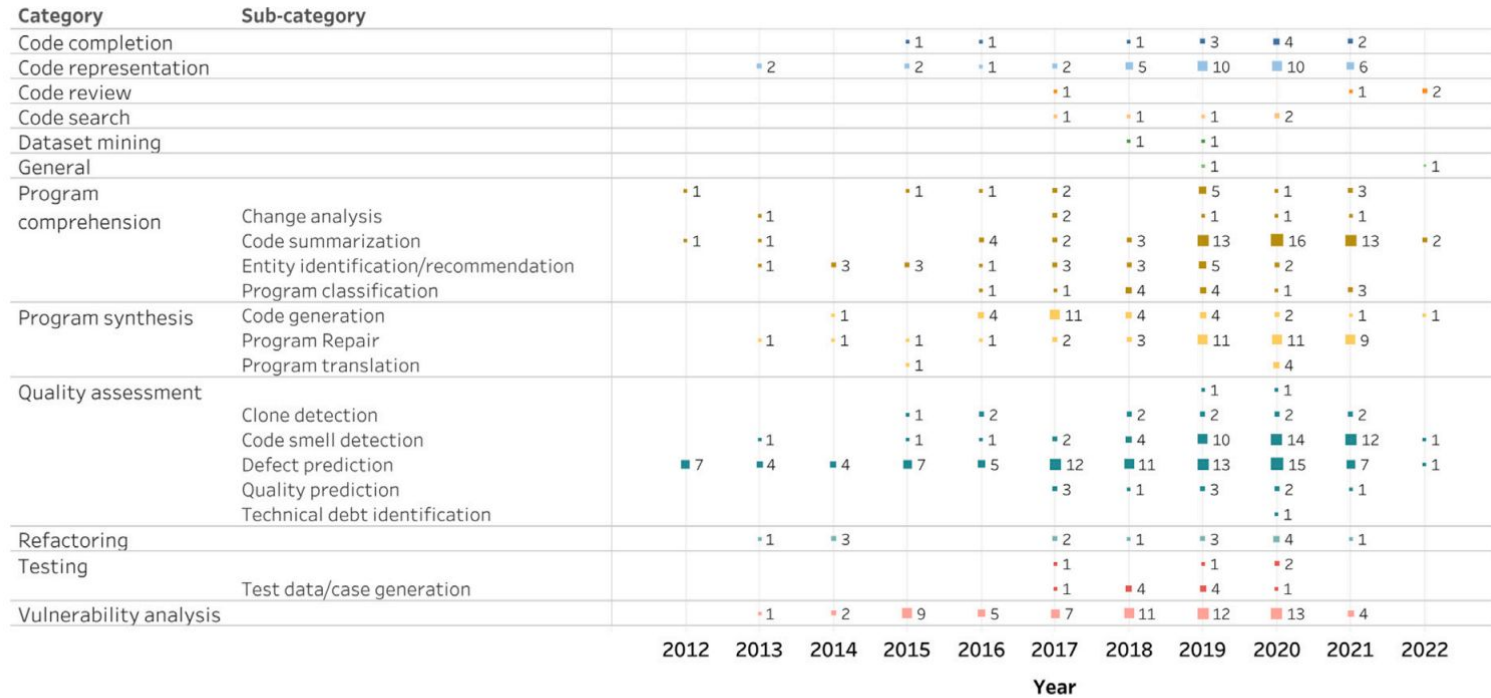


Fig. 3. Category- and sub-categories-wise distribution of studies.

Table 3
Usage of ML techniques in the selected studies (Part-1).

| | | | Code representation | Code completion | Code review | Code search | Dataset mining | Program comprehension | Program synthesis | Quality assessment | Refactoring | Testing | Vulnerability analysis |
|--|--|--|---------------------|-----------------|-------------|-------------|----------------|-----------------------|-------------------|--------------------|-------------|---------|------------------------|
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 2 |
| | | | 0 | 0 | 0 | 0 | 0 | 8 | 2 | 41 | 4 | 3 | 31 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | | 0 | 1 | 0 | 0 | 1 | 2 | 22 | 4 | 1 | 8 | 41 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 10 | 1 | 1 | 7 |
| | | | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | 0 | 1 | 1 | 0 | 0 | 8 | 3 | 52 | 2 | 1 | 19 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 3 |
| | | | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 2 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 2 |
| | | | 1 | 1 | 1 | 0 | 0 | 12 | 3 | 45 | 3 | 1 | 21 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 5 |
| | | | 0 | 0 | 0 | 0 | 3 | 0 | 13 | 0 | 1 | 9 | 26 |
| | | | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 8 | 1 | 0 | 6 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 2 | 5 | 5 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 1 | 6 |
| | | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 7 | 1 | 40 | 2 | 2 | 16 | 68 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 1 | 0 | 1 | 5 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 1 | 0 | 10 | 0 | 0 | 4 | 15 |
| | | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 2 |
| | | | 0 | 0 | 0 | 0 | 1 | 0 | 3 | 0 | 0 | 0 | 4 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 2 | 1 | 8 | 0 | 0 | 3 | 14 |
| | | | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 3 |
| | | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| | | | 1 | 1 | 0 | 0 | 0 | 5 | 1 | 0 | 0 | 0 | 8 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 1 | 0 | 1 | 6 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

Table 4
Usage of ML techniques in the selected studies (Part-2).

| | | | | Code representation | Code completion | Code review | Code search | Dataset mining | Program comprehension | Program synthesis | Quality assessment | Refactoring | Testing | Vulnerability analysis | Total |
|------------------------|-------------------------|---|-----------------|---------------------|-----------------|-------------|-------------|----------------|-----------------------|-------------------|--------------------|-------------|---------|------------------------|-------|
| Deep Learning | RNN | Bidirectional GRU | DL-RNN-BI-GRU | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 |
| | | Bidirectional RNN | DL-RNN-BI-RNN | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | Bidirectional LSTM | DL-RNN-BI-LSTM | 0 | 0 | 0 | 0 | 0 | 5 | 2 | 2 | 0 | 0 | 3 | 12 |
| | | Gated Recurrent Unit | DL-RNN-GRU | 1 | 1 | 0 | 0 | 0 | 9 | 0 | 1 | 0 | 0 | 3 | 15 |
| | | Hierarchical Attention Network | DL-RNN-HAN | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| | | Recurrent Neural Network | DL-RNN-RNN | 3 | 3 | 0 | 1 | 0 | 9 | 5 | 0 | 0 | 0 | 2 | 23 |
| | | Pointer Network | DL-RNN-PN | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | Modular Tree Structured RNN | DL-RNN-MTN | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| | | Long Short Term Memory | DL-RNN-LSTM | 3 | 4 | 0 | 1 | 0 | 21 | 10 | 6 | 1 | 1 | 5 | 52 |
| | Graph | Gated Graph Neural Network | DL-GRA-GGNN | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 3 |
| | | Graph Convolutional Networks | DL-GRA-GCN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | | Graph Interval Neural Network | DL-GRA-GINN | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | Graph Neural Network | DL-GRA-GNN | 2 | 0 | 0 | 0 | 0 | 3 | 0 | 1 | 0 | 0 | 0 | 6 |
| | CNN | Convolutional Neural Network | DL-CNN-CNN | 3 | 0 | 0 | 1 | 0 | 4 | 2 | 8 | 0 | 0 | 5 | 23 |
| | | Faster R-CNN | DL-CNN-FR-CNN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| | | Text-CNN | DL-CNN-TCNN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | Vanilla | Artificial Neural Network | DL-ANN | 0 | 1 | 0 | 0 | 0 | 2 | 1 | 21 | 3 | 1 | 3 | 32 |
| | | Autoencoder | DL-AE | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 1 | 4 |
| | | Deep Neural Network | DL-DNN | 2 | 0 | 0 | 1 | 0 | 6 | 2 | 5 | 1 | 0 | 4 | 21 |
| | | Regression Neural Network | DL-RGNN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | Multi Level Perceptron | DL-MLP | 0 | 0 | 0 | 0 | 0 | 2 | 3 | 14 | 1 | 1 | 5 | 26 |
| | Transformers | Bidirectional Encoder Representation from T | DL-XR-BERT | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 2 |
| | | CodeBERT | DL-XR-CodeBERT | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 2 |
| | | Generative Pretraining Transformer for Code | DL-XR-GPT-C | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| | | Transformer | DL-XR-TF | 2 | 1 | 2 | 0 | 0 | 4 | 3 | 1 | 0 | 0 | 0 | 13 |
| | Other | Bilateral Neural Network | DL-OTH-BINN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | Cascade Correlation Network | DL-OTH-CCN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | Code2Vec | DL-OTH-Code2Vec | 5 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 6 |
| | | Deep Belief Network | DL-OTH-DBN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 2 | 4 |
| | | Doc2Vec | DL-OTH-Doc2Vec | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 |
| | | Encoder-Decoder | DL-OTH-EN-DE | 3 | 1 | 0 | 0 | 0 | 17 | 10 | 0 | 0 | 0 | 0 | 31 |
| | | FastText | DL-OTH-FT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | | Functional Link ANN | DL-OTH-FLANN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | Gaussian Encoder-Decoder | DL-OTH-GED | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| | | Global Vectors for Word Representation | DL-OTH-Glove | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | Word2Vec | DL-OTH-Word2Vec | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | Sequence-to-Sequence | DL-OTH-Seq2Seq | 1 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 1 | 0 | 0 | 6 |
| | | Reverse NN | DL-OTH-ReNN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | Residual Neural Network | DL-OTH-ResNet | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 2 |
| | | Radial Basis Function Network | DL-OTH-RBFN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | Probabilistic Neural Network | DL-OTH-PNN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 2 |
| | | Node2Vec | DL-OTH-Node2Vec | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | Neural Network for Discrete Goal | DL-OTH-NND | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 2 |
| Reinforcement Learning | Hybrid | Double Deep Q-Networks | RL-DDQN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| | | Reinforcement Learning | RL-RL | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 3 |
| Others | Optimization Techniques | Adaptive neuro fuzzy inference system | OTH-HYB-ANFIS | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | Expectation Minimization | OTH-OPT-EM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | Gradient Descent | OTH-OPT-GD | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| | | Stochastic Gradient Descent | OTH-OPT-SGD | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 2 |
| | | Sequential Minimal Optimization | OTH-OPT-SMO | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 1 | 6 |
| | | Particle Swarm Optimization | OTH-OPT-PSO | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

ML methods

Programming languages

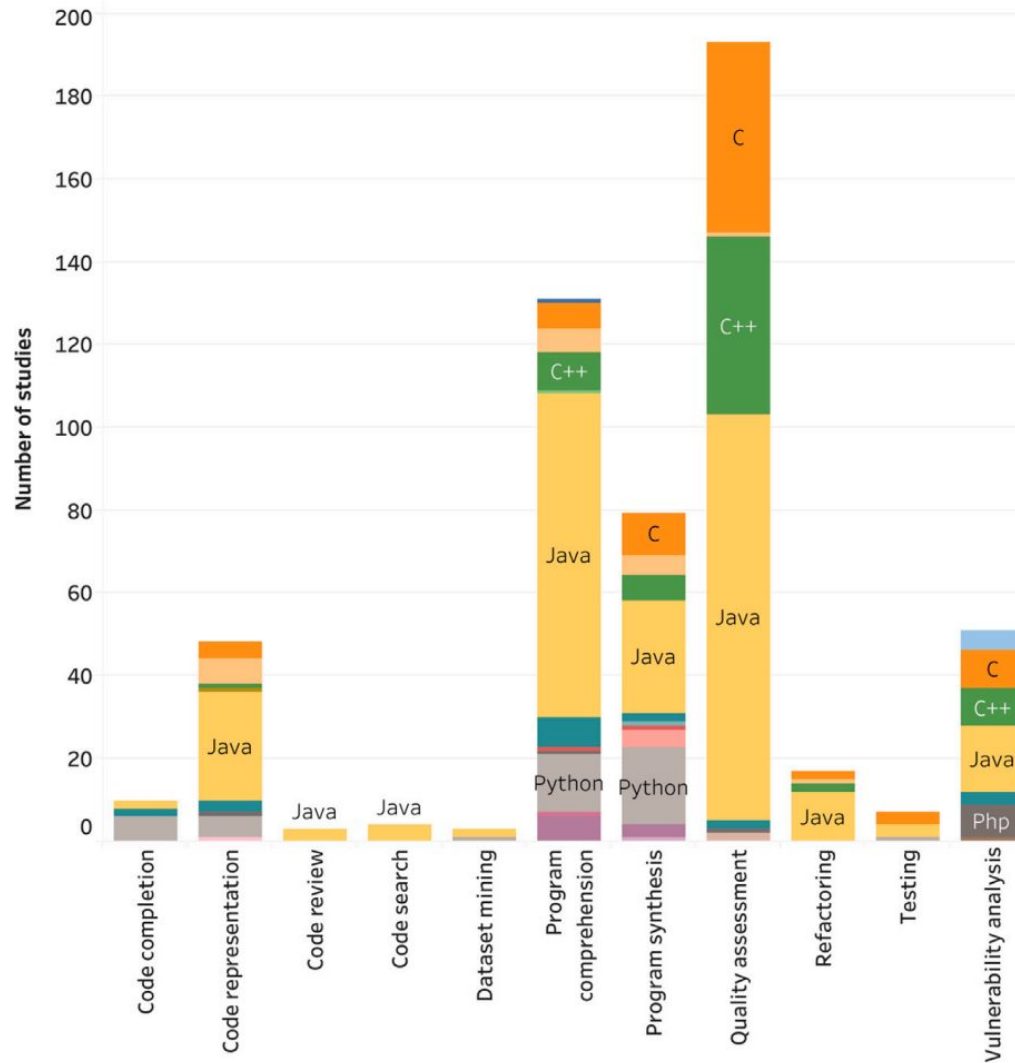


Fig. 6. Target programming languages for each considered category.

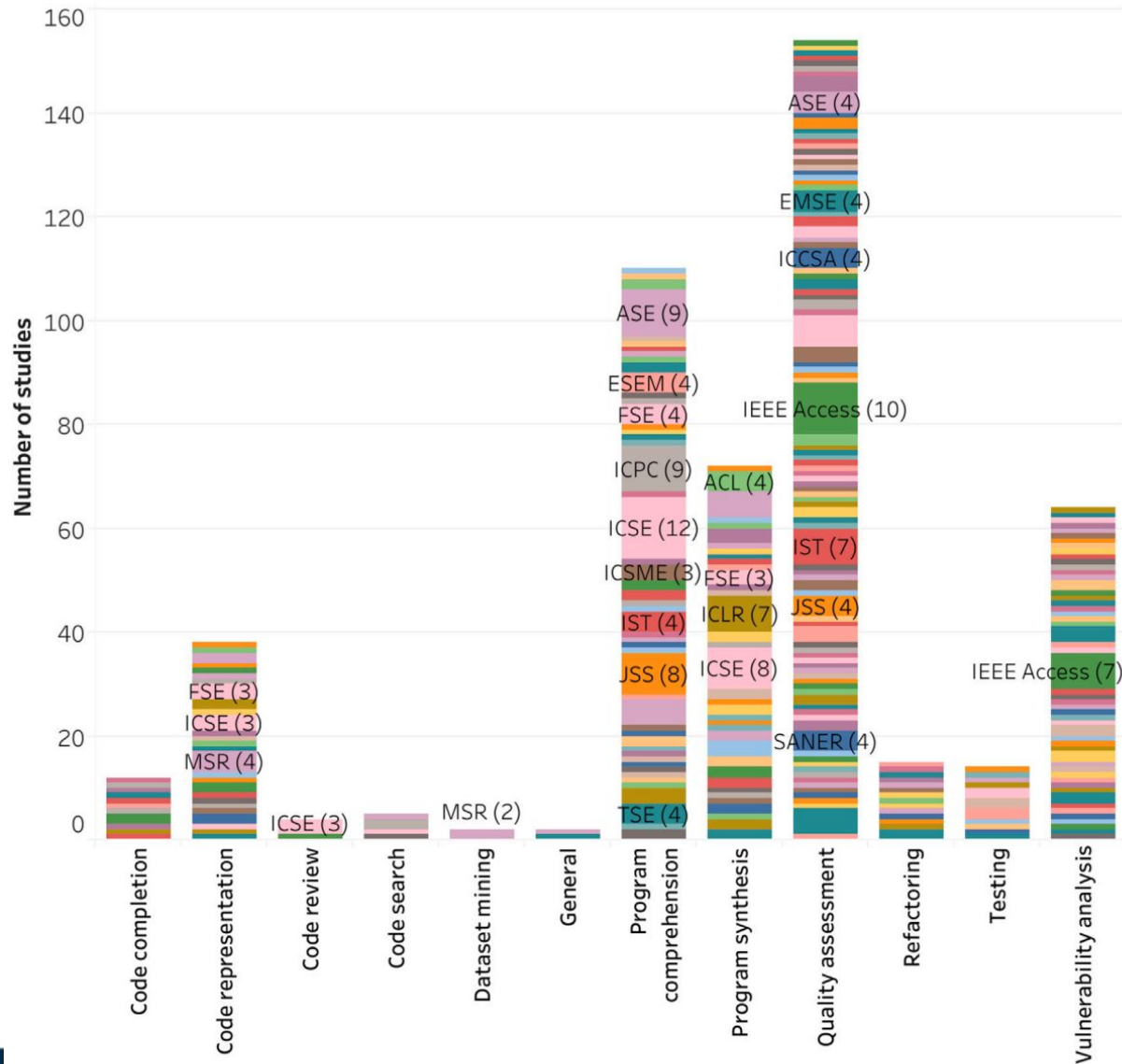


Fig. 5. Top venues for each considered category.

Table 5
Popular models proposed in the selected studies.

| Model | #Citations | Model | #Citations |
|--|------------|--|------------|
| Transfer Naive Bayes (Ma et al., 2012) | 513 | Code Generation Model (Yin and Neubig, 2017) | 651 |
| Path-based code representation (Alon et al., 2018) | 230 | Multi-headed pointer network (Vasic et al., 2019) | 128 |
| Inst2Vec (Ben-Nun et al., 2018) | 234 | Code-NN (Iyer et al., 2016) | 681 |
| DeepCoder (Balog et al., 2016) | 612 | ASTNN (Zhang et al., 2019) | 498 |
| Code2Seq (Alon et al., 2019a) | 643 | Code2Vec (Alon et al., 2019b) | 1093 |
| TBCNN (Mou et al., 2016) | 695 | Program as graph model (Brockschmidt et al., 2019) | 159 |
| SLAMC (Nguyen et al., 2013) | 130 | Coding criterion (Peng et al., 2015) | 128 |
| TransCoder (Roziere et al., 2020) | 115 | TreeGen (Sun et al., 2020) | 124 |
| Codex (Chen et al., 2021b) | 897 | AlphaCode (Li et al., 2022) | 317 |

Software solutions are all around us, therefore we need good software.

Software analysis allows us to improve our software but with some limitations

Some of these limitations can be overcome adopting Machine Learning solutions since it allows handling complex data, in a more efficient way, with higher accuracy.

In addition, Machine Learning for Software Analysis is an active research stream where various techniques have been applied on different software analytics tasks, e.g., vulnerability and defect detection, code representation, etc.