Friedrich-Alexander-Universität
Technische Fakultät

Department Computer Science

Computer Science 7 - Computer Networks and Communication Systems

Wladimir Urban

A Framework for Preparing and Assessing Cybersecurity Datasets

Bachelorarbeit im Fach Informatik

18. October 2025

Please cite as:
Wladimir Urban, “A Framework for Preparing and Assessing Cybersecurity Datasets,” Bachelor Thesis (Bachelorarbeit), Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Computer Science, Computer Science 7 - Computer Networks and Communication Systems, November 2025.

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Department Computer Science
Computer Science 7 - Computer Networks and Communication Systems

Martenstr. 3, 91058 Erlangen

https://www.cs7.tf.fau.de/

A Framework for Preparing and Assessing Cybersecurity Datasets

Bachelorarbeit im Fach Informatik

vorgelegt von

Wladimir Urban

geb. am 20. August 2001
in Ansbach

angefertigt am

Lehrstuhl Informatik 7
-
Rechnernetze und Kommunikationssysteme

Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

Betreuer: **Prof. Dr.-Ing. Reinhard German**
Dr.-Ing. Loui Al Sardy,
Mamdouh Muhammad, M. Sc.

Abgabe der Arbeit: **18. October 2025**

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.
Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Wladimir Urban)
Erlangen, 18. October 2025

Abstract

Cyber-physical security research relies on datasets that are trustworthy, well documented and temporally coherent. In practice, many public datasets often show weak documentation, inconsistent or missing labels, unstable feature definitions, class imbalance, temporal drift and other obstacles that can inflate reported performance and undermine external validity. Existing tools are fragmented, rarely support end to end reproducibility and are not tailored to cybersecurity datasets, particularly in smart-grid contexts. This thesis aims to develop a domain agnostic and all in one framework (A.R.C.) for standardized dataset preparation and assessment. Domain agnostic means the framework applies to heterogeneous sources without code changes (e.g. an industrial-control dataset such as DNP3), while all in one framework denotes that data loading, schema and label validation, leakage-aware splitting, preprocessing, model training, evaluation, and report generation are integrated into a single, reproducible run. Across the evaluated public datasets, A.R.C.'s standardized preprocessing reduced training time and model size while improving recall and F1 (with double-digit gains on noisier data), resulting in smaller, faster and more defensible models.

Kurzfassung

Cyber-physische Sicherheitsforschung ist auf Datensätze angewiesen, die vertrauenswürdig, gut dokumentiert und zeitlich stimmig sind. In der Praxis weisen viele frei verfügbaren Datensätze jedoch schwache Dokumentation, inkonsistente oder fehlende Labels, instabile Merkmalsdefinitionen, Klassenungleichgewichte, zeitliche Drift und weitere Hürden auf, die berichtete Leistungswerte künstlich erhöhen und die externe Validität untergraben können. Bestehende Werkzeuge sind fragmentiert, unterstützen selten End to End Reproduzierbarkeit und sind nicht auf Cybersecurity-Datensätze zugeschnitten, insbesondere nicht im Smart-Grid-Kontext. Diese Arbeit zielt darauf ab, ein domänenagnostisches All in One Framework (A.R.C.) für die standardisierte Aufbereitung und Bewertung von Datensätzen zu entwickeln. „Domänenagnostisch“ bedeutet, dass das Framework ohne Codeänderungen auf heterogene Quellen anwendbar ist (z. B. auf einen Industrial-Control-Datensatz wie DNP3), während „All in One Framework“ bezeichnet, dass Datenladen, Schema und Label Validierung, datenleckage sichere (leakage-aware) Aufteilung, Vorverarbeitung, Modelltraining, Evaluation und Berichtserstellung in einem einzigen, reproduzierbaren Lauf integriert sind. Über die untersuchten öffentlichen Datensätze hinweg senkte die standardisierte Vorverarbeitung von A.R.C. die Trainingszeit und die Modellgröße und verbesserte gleichzeitig Recall und F1 (mit zweistelligen Zugewinnen bei stärker verrauschten Daten); dadurch entstehen kleinere, schnellere und methodisch besser zu verteidigende Modelle.

Contents

Contents

Chapter 1

Introduction

1.1 Motivation

Cybersecurity has become a central concern over the past decade. The threat landscape now spans commodity malware and phishing to advanced persistent threats and direct attacks on critical infrastructure. These developments affect hospitals, energy systems, finance and virtually any sector exposed to networked environments. In response, researchers and practitioners increasingly employ machine learning, which can detect subtle regularities in high-volume, heterogeneous traffic and log streams that are difficult to discern by manual inspection.

The effectiveness of such models is inseparable from the quality of the underlying data. When datasets are realistic, consistent and well documented, models trained on them are more likely to retain performance in operational settings. Poor documentation, missing or unreliable annotations and synthetic or unrepresentative samples can bias evaluations and inflate expectations. Models that appear promising under laboratory conditions can fail when confronted with real world behavior.

These concerns are not new. Moulton et al. report that reproducibility in AI for cybersecurity remains fragile, due to inconsistent software versioning and insufficient methodological detail [1]. In a related domain, Sundararajan et al. show that even minor quality issues in synchrophasor datasets complicate security research for smart grids [2].

Data scarcity further exacerbates the problem in critical-infrastructure contexts. Authentic operational traces are rare, partly for security reasons and partly due to the nontrivial cost of collection, so many studies rely on a small set of public datasets. A survey by Alani and Baker concludes that available smart grid datasets are often narrow in scope, lack diversity or omit salient features [3]. When datasets are simultaneously scarce and imperfect, the imperative to improve quality assurance, standardization and transparent reporting becomes even more pronounced.

Theirfor A.R.C. was developed. It is a domain agnostic, all in one framework that unifies dataset preparation and assessment across heterogeneous sources. The same framework can ingest, validate, split, preprocess and evaluate without writing custom code, by integrating data loading, schema and label validation, leakage aware splits, standardized preprocessing ,model training and evaluation into a single reproducible run. A.R.C. reduces the need for custom code, prevents subtle evaluation errors and enables fair comparisons. The source code is available at the project's GitLab repository.

1.2 Problem statement

The existence of several well known public datasets can give the impression that foundational problems have been solved, but that is not the case. Each dataset introduces its own problems, e.g. schema heterogeneity (inconsistent feature names and semantics), shifting file formats and naming conventions across releases and misaligned data types. As a result, even basic cross dataset comparison requires substantial manual harmonization and custom code.

Documentation gaps further undermine scientific clarity. The dataset origin is frequently undocumented, collection windows are unspecified or ambiguous (days vs. weeks vs. months), tooling is not reported and attack taxonomies are only informally defined. Such omissions impede reproducibility and obscure the conditions under which results hold. Label quality is likewise uneven, encounters of misclassifications, ad hoc or conflicting nomenclature and even trivial orthographic errors are not rare. At scale, these defects compound and propagate training and evaluation.

Severe class imbalance, as seen in figure 1.1, is common. The very behaviors of interest, like rare attack types, often constitute a tiny fraction of the data. Models trained on such distributions tend to prioritize prevalent classes, suppressing recall on minority classes unless explicit countermeasures are adopted. Empirical benchmarks confirm that without targeted balancing strategies, downstream performance degrades substantially [4].

Temporal coherence represents an additional, less visible failure mode. Unreliable timestamps, unsorted merges or shuffled events break causal order and render time-dependent phenomena unobservable. A dataset may appear complete, yet be unsuitable for studying attack unfolding, escalation or response. Taken together, structural inconsistency, incomplete documentation, noisy labels, class imbalance and compromised time order, these issues form a pipeline of degradation that culminates in poor comparability and fragile reproducibility. A model trained on bad data will not give good results (Figure 1.2).

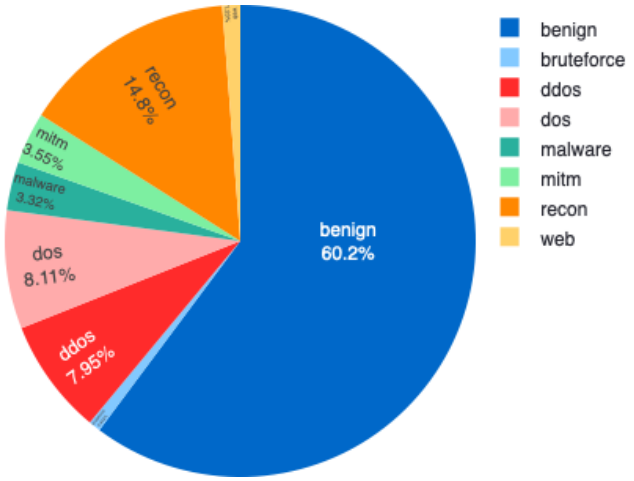


Figure 1.1 – Illustartion of an example for high class imbalance in CIC-IIoT[5]

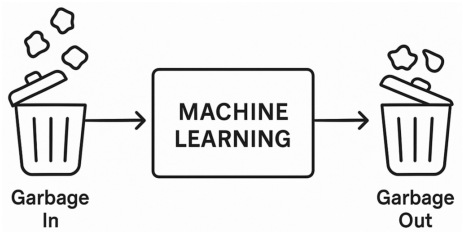


Figure 1.2 – Illustration of saying "Garbage in, garbage out" (AI generated)

Solutions exist, e.g. feature selection modules, cleaning scripts, normalization routines, but they are scattered and rarely interoperable. What is missing, is a unified and domain agnostic framework that validates, prepares and evaluates cybersecurity datasets end to end. The absence of such an integrating substrate impedes cumulative progress and motivates the need for a standardized, extensible and reproducible approach to qualify data readiness for machine-learning-based security research.

1.3 Research objectives and contributions

This thesis presents A.R.C. (Automated Research for Cybersecurity) a modular framework for the preparation and assessment of cybersecurity datasets. A.R.C. is designed to replace ad hoc preprocessing and unstandardized documentation with a transparent, repeatable, and extensible process, as seen in figure 1.3. Guided by reproducibility and comparability, A.R.C. specifies dataset with agnostic evaluation criteria, by covering class distribution, label consistency, temporal integrity and feature granularity, that can be applied uniformly across diverse datasets. It enables split aware experimentation by supporting random, stratified and time based partitioning,

so that results are reported under fair and explicitly stated regimes. It consolidates standardized preprocessing into a single and auditable catalog of cleaning, encoding, feature selection, normalization, outlier handling and class rebalancing with consistent defaults. It offers a baseline learning stack comprising Random Forests, Gradient Boosting, Support Vector Machines, Multi-Layer Perceptrons and Logistic Regression under a common interface and metric suite. It automates reproducibility by emitting structured reports as PDFs that capture preprocessing choices, split definitions, hyperparameters and evaluation metrics to permit exact reruns.

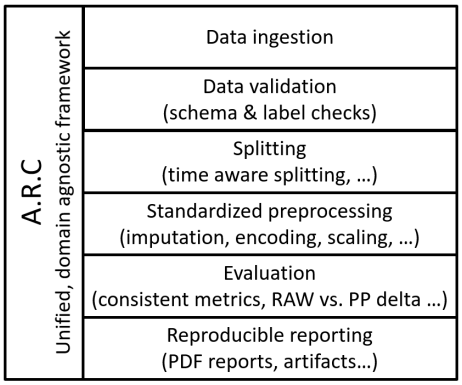


Figure 1.3 – Illustration of contributions and how they compose the A.R.C. framework

A.R.C. is realized as an interactive web application built with Streamlit and exercised on multiple public datasets to validate functionality and illustrate end to end use. It is available for public use in the project's GitLab repository

The thesis contributes a conceptual design for a modular and domain agnostic framework (A.R.C.) for dataset readiness in cybersecurity, a practical implementation that operationalizes this design in an accessible and interactive tool. And an empirical study showing that standardized preprocessing within A.R.C. improves data quality and downstream model performance while clarifying efficiency trade offs. By emphasizing reusability over one off scripting, A.R.C. raises the standard for dataset quality, documentation and experimental traceability, thereby enabling cleaner cross study comparisons and more trustworthy conclusions.

1.4 Thesis structure

The remainder of this thesis is organized into five main chapters, each building upon the previous to move from related work and design to implementation, evaluation, and results.

Chapter 2 - Related work

Related work surveys existing tools and frameworks relevant to machine learning and data pipeline management, including TensorFlow, Kubeflow, MLflow, and Apache Airflow. Their strengths and limitations are analyzed in the context of cybersecurity datasets, highlighting the need for a domain-specific and integrated approach such as A.R.C..

Chapter 3 - Methodology

Methodology presents the conceptual and architectural foundations of the framework, including schema and label validation, dataset splitting, preprocessing, model selection, and evaluation metrics. It details the design of each module and describes how reproducibility is achieved through configuration management and artifact logging.

Chapter 4 - Experiment

Experiment specifies the study used to evaluate A.R.C., it states the research questions, lists the datasets and compute environment and details the design comparing raw and preprocessed data preparation across supervised models. It defines the predictive and efficiency KPIs.

Chapter 5 - Case study and results

In case study and results the framework is applied to representative cybersecurity datasets (DNP3[6] and CIC-IIoT[5]), presenting empirical findings such as schema and label validation outputs, split analyses and model performance. The impact of standardized preprocessing is examined for Random Forest, Gradient Boosting, Support Vector Machines, Multi Layer Perceptrons, and Logistic Regression, followed by a summary of results and a discussion of challenges, limitations and mitigations.

Chapter 6 - Conclusion and future work

The chapter conclusion and future work summarizes the main findings and contributions of the thesis, concluding that standardized preparation significantly improves dataset quality, comparability and reproducibility. Future work includes testing A.R.C. on real anonymized field data, integrating CICFlowMeter for flow feature extraction and extending the framework to support unsupervised and continual learning modules for realistic cybersecurity scenarios.

Chapter 2

Related work

2.1 TensorFlow

Scope and positioning

TensorFlow Extended (TFX) is Google's production scale toolkit for defining, running and monitoring end to end machine learning pipelines. It provides a standardized sequence of components, implemented as portable pipeline elements, purpose built for scalable, high performance machine learning for any datasets. In practice, TFX acts as a reference architecture for operational machine learning frameworks. [7]

Architecture and components

TFX consists of libraries and pipeline components as shown in figure 2.1, where the white boxes represent libraries and the orange boxes represent pipeline components. TensorFlow Data Validation computes descriptive statistics, infers a data schema and detects anomalies, drift, or training–serving skew. TensorFlow Transform normalizes inputs and converts strings and floats to integer values. TensorFlow Model Analysis evaluates models at scale, reusing the metrics defined while training, and supports evaluation over slices of the data. ML Metadata records artifact lineage and execution metadata for auditability and reproducibility. Pipelines are portable across orchestrators such as Apache Airflow, Apache Beam, and Kubeflow Pipelines.[7]

Dataset preparation capabilities

Out of the box, TensorFlow Data Validation offers robust schema with centric checks (e.g. types, ranges, missingness, domain shifts) and can flag drift or skew between training and serving data. TFT provides deterministic, deployable transforms (e.g. scaling, vocabulary building) that ensure train/serve parity. However, class support

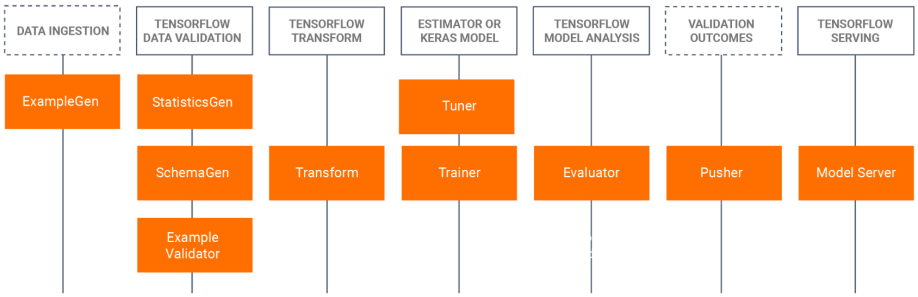


Figure 2.1 – Illustration of TensorFlow libraries [7]

for label taxonomy consistency (e.g. canonicalizing synonyms), temporal integrity checks tailored to security telemetry, and split quality auditing (e.g. unseen labels across train/validation/test) requires user defined logic or custom components. [7]

Reproducibility and metadata

TFX emphasizes provenance. TFX's ML meta data captures artifacts (e.g. datasets, models), executions (pipeline steps) and their relationships in a pluggable store (e.g. SQLite, MySQL), enabling experiment traceability and audit. TFMA reuses the training metric definitions for evaluation and supports data slicing, which facilitates consistent comparisons across runs. [7]

Domain alignment

For cybersecurity datasets, especially smart grid telemetry, TFX brings reliable generic capabilities (schema checks, reproducible transforms, sliced evaluation). But it lacks built in validators for domain specific issues such as time based splits with gap windows or label otologies common in IDS corpora. Those behaviors can be implemented, but they are not native features and would require additional engineering. [7]

Emulating A.R.C. features with TFX

Several A.R.C. functions map naturally, schema checks via TensorFlow Data Validation, standardized preprocessing via TensorFlow transform, evaluation via TesorFlow model analysis and provenance via ML meta data. A two branch raw vs. preprocessed design can be constructed by defining parallel pipeline paths, one with minimal transforms and one with full preprocessing and comparing TesorFlow model analysis outputs. Explicit split checks (e.g. unseen label detection, per split distributions), temporal coherence tests, and security specific reports would need custom components and reporting logic beyond core TFX. [7]

Strengths and limitations

The strengths are mature and modular components, strong statistical validation and reproducible transforms, scalable sliced evaluation, comprehensive lineage. The limitations are the TensorFlow general design choices, domain agnostic (no native label/temporal validators for IDS), higher engineering overhead to achieve the interactive, all in one experience and RAW vs. preprocessed comparisons provided by A.R.C. [7]

Suitability for this use case

TFX is a robust backbone for production pipelines and could underlie a cybersecurity workflow with additional domain components. For the specific A.R.C. objectives (security oriented schema, label and time validation, split quality auditing, standardized preprocessing with paired raw vs. preprocessed comparisons and consolidated researcher friendly reports), TFX provides many building blocks but is not a substitute. It is suitable as an infrastructure, achieving A.R.C.'s domain focus would still require nontrivial customization. [7]

2.2 Kubeflow

Scope and positioning

Kubeflow Pipelines is a platform for building and deploying portable, scalable machine learning workflows on Kubernetes. Pipelines and components are authored in Python, compiled to an intermediate representation (IR) YAML and then executed by a KF conformant backend. In short, KFP focuses on defining, running, and tracking ML workflows, not on any single domain.[8]

Architecture and components

A pipeline is a directed acyclic graph (DAG) that specifies the order of steps, their conditions and how parameters and data flow between them. At runtime, each step runs as a container. The Python SDK/DSL lets you define components and compose them into pipelines. KF also supports control flow constructs and local execution during development. KF distinguishes between small parameters and larger artifacts, such as datasets, models, and metrics. Components specify the types of their inputs and outputs. Larger results are treated as versioned objects that can be passed from one step to the next. The framework manages, tracks and visualizes pipeline definitions, runs, experiments and artifacts. It also enables efficient reuse via caching and parallel task execution.[8]

Dataset preparation capabilities

The documentation does not prescribe particular validation or preprocessing routines. Instead, users build custom components or reuse components from the wider ecosystem, to implement checks and transforms they need, then wire them into the DAG. This keeps KF flexible and portable across use cases.[8]

Reproducibility and tracking

Reproducibility follows from containerized execution, a portable IR YAML spec, and typed component interfaces. KF's UI and metadata capture make it possible to manage, track and visualize runs and artifacts across experiments, while caching avoids re-doing identical work.[8]

Domain alignment

For cybersecurity datasets, including smart grid, KF provides robust operational scaffolding, but no native domain semantics. Time based splitting, canonicalization of attack taxonomies, drift and leakage diagnostics and structure analyses are entirely feasible, yet only after they are encoded as components. KF supplies the foundation and the security specific logic must be built.[8]

Emulating A.R.C. features with KFP

An A.R.C.–style workflow can be rendered as a KF DAG, components for schema, label and temporal validation, split generation (random, stratified, time-based) with split-quality checks, standardized preprocessing, baseline training and evaluation and a parallel branch that trains the same learner on minimally processed RAW data to enable comparison. A final reporting component aggregates figures, tables, and manifests into a PDF. While this arrangement reproduces A.R.C.'s structure, the researcher experience provided by A.R.C.'s Streamlit interface would need to be recreated via notebooks, which is the KF's UI, or a custom front-end.[8]

Strengths and limitations

KF's strengths lie in cloud native orchestration, containerized reproducibility, component reuse, step caching, parameterized runs and convenient run comparison. Its limitations are meaningful dataset quality semantics, split auditing, and RAW vs. preprocessed comparisons are not included features. Operating KF at scale assumes Kubernetes expertise and the framework does not offer an all in one interactive research UI.[8]

Suitability for this use case

Kubeflow Pipelines is well suited as infrastructure to host and scale an A.R.C. like pipeline. It is not a substitute for A.R.C., since cybersecurity specific validation, split checks, standardized preprocessing defaults, paired RAW vs preprocessed training and consolidated researcher-oriented reporting must be implemented as custom components. For teams prioritizing large scale KF is an excellent backbone, but for fast and domain focused analysis and interactive reporting, A.R.C. remains the more appropriate tool.[8]

2.3 MLflow

Scope and positioning

MLflow is an open source platform that manages the machine learning life cycle with a focus on experiment tracking, model packaging and model governance. Its Tracking component provides an API and UI to log parameters, code versions, metrics and output files for later visualization and comparison. MLflow Models standardize how trained models are packaged and the Model Registry offers a centralized store and UI to manage model versions and stages. These facilities emphasize cross project comparability and model life cycle management.[9]

Architecture and components

Runs are recorded within experiments, each run can log parameters, metrics, tags, and artifacts. Run metadata is stored in a backend store, while artifacts (such as figures, reports, or model files) are saved to a configurable artifact store. The UI supports browsing and comparing runs. Packaged models use flavors, a convention that lets deployment tools understand how to load and score models from many machine learning libraries (e.g. the generic Python function flavor). The registry maintains model lineage, versioning, aliasing and annotations, exposing a UI and APIs for stage transitions (e.g. staging to production).[9]

Dataset preparation capabilities

The documentation centers on logging, packaging, evaluation and registry workflows. It does not define standard functions for data validation, preprocessing or dataset splitting. In practice, any such checks or transforms are produced by custom user code or external libraries and logged to MLflow as parameters, metrics, or artifacts for traceability.[9]

Reproducibility and metadata

Reproducibility follows from immutable run records in Tracking (parameters, metrics, code versions, and artifacts), standardized model packaging via flavors (with environment specifications) and registry. These allow teams to trace which experiment and run produced a model, reevaluate models with the same metrics in the UI and promote specific versions with full provenance.[9]

Evaluation facilities

MLflow provides evaluation tooling that aggregates metrics, visualizations and validation outputs into a structured result, viewable programmatically and in the UI. The project also documents a separate evaluation framework for generative AI use cases.[9]

Domain alignment

MLflow's documentation presents it as general purpose life cycle tooling. as an API and UI to log parameters, code versions, metrics and artifacts. Standardized model packaging via flavors that work across libraries, a Model Registry for versioning and stage transitions and evaluation utilities that generate metrics and visualizations (e.g., confusion matrices) with support for custom metrics and plots are available. These capabilities are domain neutral, they can be applied to any application area, while domain specific checks and transformations are implemented by user code and logged to MLflow for traceability. In practice, this means teams can capture experiments, artifacts and model lineage for security oriented workloads using the same tracking, packaging, registry and evaluation interfaces described in the docs.[9]

Suitability for this use case

Within an A.R.C. style workflow, MLflow functions as the governance layer, it tracks each pipeline run (with its parameters and metrics), logs artifacts (figures, reports, models), packages models with flavors and register/promote versions. It does not replace a data preparation or orchestration pipeline, it records what was run and preserves the artifacts and lineage needed to compare experiments and manage models over time.[9]

2.4 Apache Airflow

Scope and positioning

Apache Airflow is an open source platform for programmatically authoring, scheduling and monitoring batch oriented workflows. It provides a web UI to visualize, manage and debug pipelines and can run from a single process setup on a laptop to distributed deployments capable of handling large workloads. Its remit is workflow orchestration rather than domain specific analytics.[10]

Architecture and components

Airflow represents a workflow as a Directed Acyclic Graph (DAG) composed of Tasks with explicit upstream and downstream dependencies. Tasks are typically instantiated from Operators (predefined task templates) or Sensors (tasks that wait for external conditions) and can use control flow constructs to express complex execution order. A persistent Scheduler evaluates DAGs and dispatches ready tasks to an Executor, while a metadata database records runs, state and lineage. Execution and observability are surfaced through a web UI with per task logs and run views.[10]

Dataset preparation capabilities

The project's documentation focuses on orchestration primitives—DAGs, tasks, operators, sensors, scheduling and execution. Validation, preprocessing, or dataset splitting are not prescribed abstractions, they are implemented as user tasks using operators or custom code and then scheduled within a DAG. This separation keeps Airflow flexible across use cases.[10]

Reproducibility and metadata

Reproducibility and operational visibility are supported via the metadata database (which persists DAG and task run history), structured logging with per task log which are accessible in the UI and built in metrics emission. Best-practice guidance covers database backends, maintenance (e.g. cleaning old metadata) and production deployment patterns, all of which contribute to stable, repeatable operation.[10]

Domain alignment

Airflow is positioned as a general workflow platform. It is designed to connect "virtually any technology" through an extensible Python framework and a large operator ecosystem, rather than to provide domain specific checks or data semantics. Consequently, pipelines for cybersecurity or smart-grid analytics are expressed by

composing domain logic into tasks and scheduling them with Airflow's orchestration features.[10]

Strengths and limitations

Airflow offers mature DAG based orchestration, rich operator and sensor abstractions, a modern UI for monitoring and scalable execution models (including Kubernetes-based executors). At the same time, it does not have built in domain level dataset validation or ML specific evaluation primitives, these must be implemented in tasks or paired with external libraries.[10]

Suitability for this use case

For teams that need reliable scheduling and dependency management around data and ML jobs, Airflow provides a solid backbone. It can host an end to end pipeline by orchestrating user code at each stage and preserving operational history in the metadata store. It complements, rather than replaces, frameworks that supply cybersecurity specific validation, split auditing, or standardized ML evaluation.[10]

2.5 Summary

Table 2.1 contrasts TensorFlow, Kubeflow, MLflow, and Airflow along four axes, primary role, key features, reproducibility and domain suitability for cybersecurity and ICS datasets. In short, these frameworks are powerful infrastructure primitives, for modeling, orchestration, tracking, and workflow control, but none provide dataset first evaluation safeguards (e.g., schema and label validation, leakage- and time-aware splitting, standardized quality reports). This motivates the A.R.C. framework, a complementary and dataset first layer that can run on top of or alongside these tools while enforcing defensible evaluation practices.

Remaining gaps

Despite mature support for modeling, orchestration and tracking, the reviewed frameworks do not treat cross dataset and time checks as primary concerns, practitioners must design these protocols themselves. Data quality diagnostics, such as detecting clock skew, quantifying class-imbalance impact or surfacing mislabeled samples, remain custom and uneven across tools. The ecosystem also lacks in standardized and reproducible benchmarks tailored to smart grid and ICS scenarios, which limits comparability across studies. The A.R.C. framework addresses much of this by providing built-in schema and label validators, leakage and time aware splitting, standardized preprocessing and FAIR reports that enable comparisons.

Table 2.1 – Overview of widely used ML infrastructure frameworks in comparison to the goals of this thesis

Framework	Primary role	Key features	Reproducibility	Domain suitability
TensorFlow	Modeling and training	Keras/TF APIs, TF Data input pipelines, SavedModel export, optional TFX pipelines.	Models and data pipelines can be versioned, environment via Conda/Docker (external), determinism requires care.	Generic, no native checks for dataset quality, leakage, or time-aware splits, must be implemented manually.
Kubeflow	Orchestration / ML pipelines	Kubernetes native pipelines and components, ML Metadata (MLMD), Katib for HPO.	Strong via containerized steps and pipeline specs; parameterized, repeatable runs with metadata lineage.	Generic, validators and leakage controls need custom components, operational overhead for small projects.
MLflow	Experiment tracking	Runs/params/metrics tracking, artifact store, Model Registry, simple serving hooks.	Good via run IDs, artifacts and captured environments (conda.yaml/requirements.txt), no enforcement of data and split correctness.	Agnostic, pairs well with training but does not provide ICS-specific validation or leakage and time split safeguards.
Airflow	Workflow orchestration	DAG scheduling, operators and sensors, retries, rich plugin ecosystem.	Reproducible via code defined DAGs, environments externalized (Docker/venv), no ML semantics.	Agnostic scheduler, all dataset validation and leakage logic must be coded as custom tasks.
A.R.C. (this work)	**Dataset preparation and assessment**	**Schema and label validation, leakage aware time splits with embargo, standardized preprocessing, comparable training/eval, FAIR artifacts, PDF report.**	**High, single config/run creates versioned artifacts and reports, deterministic for repeatability.**	**Domain agnostic but cyber and ICS aware via validators (timestamp checks, class balance) and time aware evaluation.**

Chapter 3

Methodology

3.1 Concepts

High-quality consistent datasets are not just a addition in cybersecurity research, they are the ground everything else stands on. Even the most advanced detection algorithm loses its value if it has been trained on unreliable or inconsistent data. In fact, without careful preparation and validation, results risk misleading.

While data preparation and validation often sound like technical meaningless, they are really closer to the foundations of trustworthy science. A thorough preparation process, as seen in figure 3.1, makes datasets reusable across studies, helps ensure that experiments can be compared fairly and gives machine learning models a chance of producing meaningful outcomes.

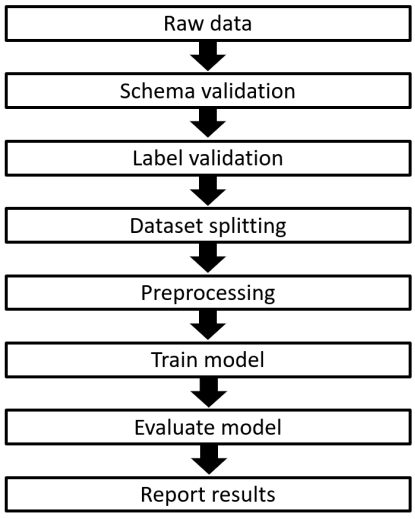


Figure 3.1 – Illustration of sequence through A.R.C.

3.1.1 Schema validation

Number of records

Apparent trivialities such as record counts are in fact methodological priors. Public intrusion detection datasets range from a few thousand to tens of millions of records. This variation directly conditions statistical precision, the expected support of rare classes and the feasibility of downstream methods. Exact counts are therefore required for reproducibility, power considerations and for interpreting class-imbalance metrics.

Duplicated records

Recorded duplication inflates apparent performance by enabling memorization and by leaking information across partitions. For CICIDS2017, Mondragón documents near duplicate flows repeated across scenario files, which distorted evaluation outcomes [11]. Deduplication should be a mandatory early step, performed prior to splitting to prevent cross-split contamination.

Missing values

Missingness comes from packet loss, sensor faults, anonymization or data integration artifacts. Its impact is not limited to information loss, if the missingness mechanism correlates with labels or features, estimates become biased. Croft and Babar demonstrate such correlations in vulnerability datasets, where missing values leaked label information [12]. Systematic detection of patterns over rows and columns or correlations with labels must be reported and context-aware handling must lead any imputation strategy. Standard imputation without analysis risks encoding spurious signals.

Granularity

The observational unit (packet, flow, session or higher level) aggregation determines the resolution at which behaviors are measured. Packet level data maximize data input, but is heavy in storage and computing costs. Flow or session abstractions reduce volume while discarding information. Wu's et al. systematic comparisons show that choosing between packet and flow representations can move both accuracy and generalization of intrusion detection models [13]. Granularity must therefore be treated as a design variable and documented accordingly.

Total number of features

High dimensionality is not correlating with high information content. Large feature sets often contain redundancy, duplications and lead to slow training. They can also degrade generalization. Reviews by Di Mauro highlight the necessity of aggressive selection to maintain interpret ability [14]. Mondragón's benchmarks report improved performance after reduction [11]. Reporting both raw and post preprocessing feature counts clarifies this trade off.

Duplicated features

Semantic duplication, columns that encode essentially the same signal under different names (e.g., bytes_sent vs. tx_bytes), increase the importance of feature families and complicates interpretation. Automated screening should go before model fitting, by identifying near duplicates using correlation thresholds, like distance measures on empirical distributions or mutual information diagnostics, then consolidating or removing them. Trimming these redundancies reduces estimator variance, stabilizes attribution (e.g. feature-importance rankings) and sharpens downstream explanations [14].

Mixed datatypes

Columns intended to be numeric frequently contain symbolic placeholders (e.g. Infinity, NaN, unknown) or locale generated artifacts, breaking parsers and type assumptions. In CICIDS2017, fields such as "FlowBytes/s" included such entries, which disrupt pipelines [15]. Robust schema validation should perform datatype checks and flag mixed typed columns.

Temporal integrity

Time underpins scenario structure and attack dynamics. Missing, duplicated or unsorted timestamps and unsynchronized clocks across sources destroy temporal coherence and enable models to exploit artifacts rather than learn behavior [11], [15]. Validation must therefore check monotonicity (within sources), detect gaps and bursts, quantify typical inter-arrival intervals, and verify synchronization when merging streams (including time-zone and offset handling). These diagnostics should be reported alongside split definitions, particularly when evaluating time aware models.

3.1.2 Label validation

In supervised ML learning, labels constitute the operational ground truth that anchors training and evaluation. When they are incorrect, inconsistent, or unstable, the modeling pipeline does not only degrade, it learns the error distribution and reproduces it.

Label consistency

Consistency covers spelling, granularity and missingness. Spelling variants (e.g. DoS vs. DOS) and near-synonyms (e.g. Brute Force vs. Password Guessing) introduce spurious classes, finer-grained splits of the same phenomenon create artificial boundaries. A rigorous pass should canonicalize case and spelling, map synonyms and aliases to a documented taxonomy and detect and resolve overlapping categories.

Consistency covers spelling, granularity and completeness. Spelling differences (e.g. DoS vs. DOS) and near-synonyms (e.g. Brute Force vs. Password Guessing) create spurious classes, while overly fine splits of the same phenomenon introduce artificial boundaries. Equally critical are missing labels (e.g. empty strings, NaNs, NULLs or placeholder tokens) which break supervised training and bias evaluation, if they are unevenly distributed across partitions. A check for case and spelling should detect possible problems and explicitly identify them. Unlabeled records must be handled. All remappings and exclusions should be logged, and the absolute and relative share of missing labels reported per split to preserve auditability.

Entropy and class distribution

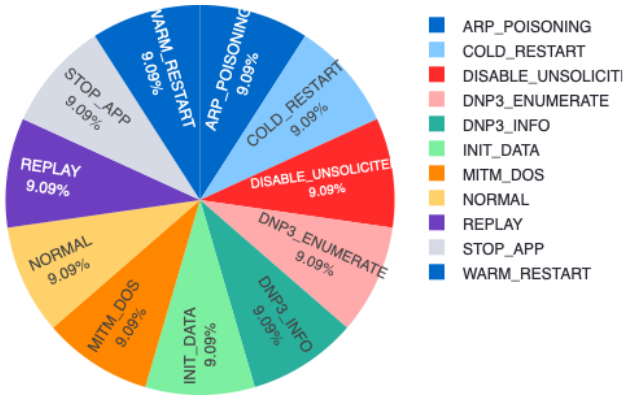
Class counts strongly shape performance. Accuracy, in particular, can be inflated by majority-class guessing. Entropy offers a compact characterization of distributional diversity,

$$H(y) = \sum_{k=1}^{K} p_k \log(p_k),$$

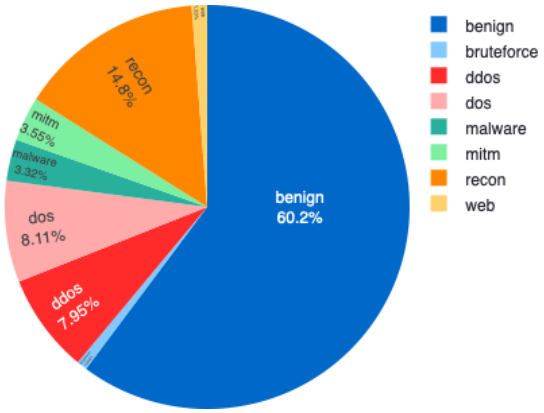
where p_k is the empirical frequency of class k; H is maximal for the uniform distribution ($p_k = 1/K$).

Figure 3.2a shows a datasets with a entropy of 3,46, which indicates a low imbalance. Figure 3.2b shows a datasets with a entropy of 1,90, which indicates a moderate imbalance.

In UNSW-NB15, for example, Normal and Generic exceed 60% of samples while Shellcode and Worms fall below 1%, yielding low entropy and rendering minorities difficult to learn [16]. Benchmarks show that models trained on such imbalanced data, under perform on the rarest and most consequential classes [4].



(a) Illustration of DNP3 class distribution [6]



(b) Illustration of CIC-IIOT class distribution [5]

Figure 3.2 – Illustration of class distribution comparison

Rare classes

Low represented classes can be noise or they can capture precisely the infrequent and high-impact behaviors of interest. Resampling and cost sensitive learning improve recall for such tails, but only when the rare class is well defined and consistently labeled [4]. In smart-grid contexts, rare labels often correspond to adversarial actions against critical assets, removing or collapsing them for convenience may simplify training while erasing the scenarios that matter most [3]. Label validation should therefore flag rarity explicitly and force a documented decision (retain, merge, or synthesize).

Dominant classes

Dominant classes can mask failure elsewhere. In CICIDS2017, early capture days are overwhelmingly BENIGN ($>80\%$), permitting superficially high accuracy via majority prediction [17]. Evaluations that ignore this skew routinely overestimate

performance and underestimate minority risk [18]. Reporting macro-averaged metrics (macro-F1, macro-Recall) and per-class confusion matrices mitigates this optimism.

Temporal drift

Traffic composition and attack types change over time. If records are randomly shuffled prior to splitting, identical or highly similar events can appear in both train and test, inflating scores. Strict chronological splits may yield unseen labels in validation/test as seen in figure 3.3. In the CICIDS2017 dataset benign dominated days precede DoS/DDoS-saturated days [15]. It is important to analyse the label distribution over time, to make the right choices later in the pipeline, e.g. split method.

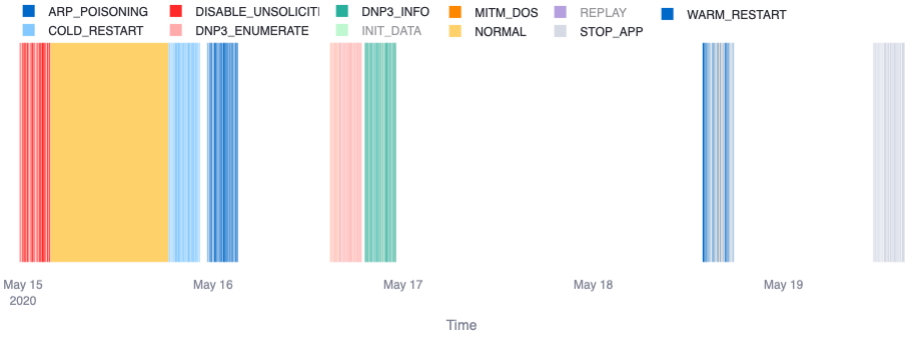


Figure 3.3 – Illustration of DNP3 label distribution over time [6]

Synthetic dataset recognition.

Synthetic datasets, derived from simulation or generative models, expand scale and controllability but often lack the irregularities of operational data as seen in figure 3.3. Identifiers are near perfect balance, low entropy of feature noise, and the absence of common problems (e.g. missing values, mislabels). Distinguishing synthetic from real traffic matters for reproducibility and threat modeling [19]. Label validation should give insights to identify such datasets.

3.1.3 Dataset splitting

Partitioning a dataset into training, validation and test subsets is often treated as a routine preprocessing step. In cybersecurity it is a design choice. The split strategy rules the fairness of algorithmic comparisons, the credibility of generalization claims and the degree to which evaluations are insulated from artefactual correlations. Because cybersecurity datasets are typically imbalanced, evolve over time and are

structured by scenario specific dependencies, no single strategy is universally appropriate. Instead, the splitting method must be aligned with both the dataset's properties and the study objective (exploration, model selection, deployment realism).

Random split

Random assignment draws records independent and identically distributed into training, validation, and test sets with the expectation that each subset mirrors the global distribution. This assumption is tenable in domains where samples are independent and stationary. Cybersecurity rarely fits that template. Campaigns (e.g. DoS bursts) generate highly correlated flows, randomization can scatter near-duplicates across partitions, enabling models to exploit campaign-specific artifacts rather than learning attack invariant structure. The resulting pattern leakage inflates performance estimates and undermines external validity [20]. Random splits remain useful for rapid exploration and sanity checks when independence can be justified, but they are insufficient as a sole basis for claims about deployment robustness.

Stratified split

Stratification preserves empirical class proportions in each partition, directly addressing the class imbalance which is common in intrusion detection datasets. Without stratification, minority classes can disappear from a subset, even from the training subset, thereby rendering undefined per class metrics and misleading micro averages. Datasets such as NSL-KDD or UNSW-NB15 exhibit dominant Normal and a few frequent attacks, with tails (e.g. Worms, Shellcode) below 1% [16]. Proportional representation is therefore a minimum standard for meaningful evaluation. Stratification does not address temporal dependencies and should be complemented with time aware analysis when scenario order matters.

Time based split

Chronological partitioning trains on earlier intervals and tests on later, unseen intervals. This arrangement matches operational deployment models, learn from the past and are evaluated on the future, while preventing information leakage backward in time. Its principal cost is that distribution shift (concept drift) is now exposed. Performance that appears strong under random/stratified regimes may degrade when attacks evolve over time. Such degradation is informative rather than undesirable, it quantifies robustness to change and highlights where adaptation or retraining is required.

Split quality checks

Split construction must be audited with two critical checks.

Unseen labels. If classes are absent from training but present in validation/test as seen in figure 3.4, supervised learners cannot be expected to recognize them. In this circumstance depressed recall reflects split design rather than model deficiency. This risk exists especially with time based splits when classes emerge late.

Per-partition class distributions. Even with complete label coverage, differences in class mix across partitions can skew results. Over representation of rare classes in training but not in testing reduces generalization, under representation inflates apparent difficulty. Reporting per-split distributions is therefore necessary and differences should be interpreted alongside metrics [15].

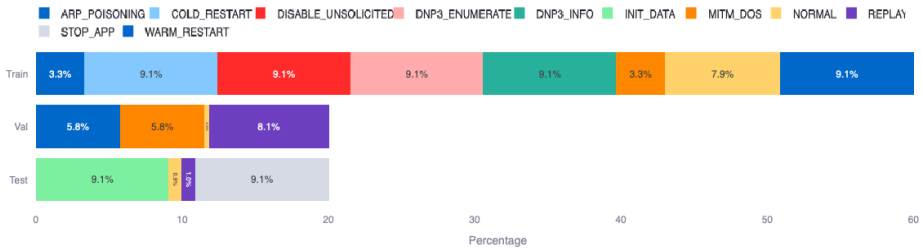


Figure 3.4 – Illustration of label distribution per split of DNP3 dataset using time based split with 0 gap [6]

3.1.4 Preprocessing techniques

Preprocessing converts heterogeneous and noisy traces into analyzable representations. In cybersecurity this stage is essential. Raw traffic logs, host telemetry and packet captures are sometimes incomplete, redundant and typically high dimensional. A sophisticated models may achieve attractive in sample scores, yet fail in operational settings due to brittleness, leakage or wrong correlations.

Cleaning: Removing duplicates

Duplicates arise from replayed traffic, merged captures and logging artifacts. Left in place, they distort class frequencies and enable cross partition leakage when identical rows appear in both train and test. The KDD'99 corpus famously contained >70% duplicate entries ,subsequently filtered in NSL KDD, artificially inflating accuracy metrics prior to filtering [21].

Cleaning: Imputing missing values

Missingness originates in packet loss, sensor faults, anonymization or extraction failure. It is rarely ignorable. Listwise deletion removes minority patterns and can increases imbalance. Imputations, like mean/median for simple settings or model-based and nearest neighbor schemes when structure is present, preserves sample size while limiting bias. Imputation must be fitted on training data only and applied to validation/test to prevent information leakage.

Encoding: Rare-category bucketing

Categorical features (e.g., IP ranges, ports, protocol flags) often exhibit long tails. Retaining each infrequent level leads to overfitting and impractically wide one hot vectors. Bucketing rare categories into an other class reduces dimensionality and stabilizes estimates, particularly when combined with frequency thresholds chosen on the training distribution.

Encoding: One hot encoding

One hot encoding is robust for low to moderate cardinality variables. However, for features with hundreds or thousands of levels, it induces sparsity, increases memory the footprint and complicates regularization. Alternatives, like target encoders, become preferable at scale.

Encoding: Gradient boosting categorical handling

Modern gradient boosting libraries incorporate specialized categorical encoders. CatBoost's permutation driven statistics, for example, reduce target leakage by computing category encodings without peeking at the current row's label. Such native handling is advantageous in cybersecurity, where categorical variables (e.g. application protocol) can correlate strongly with the target. Safer encodings yield more reliable generalization.

Encoding: Target mean encoding

Replacing categories with their average target value can be effective but is leak prone, if computed globally. Correct application requires training subset only estimation and careful regularization, otherwise, the feature inadvertently encodes the label and inflates apparent performance.

Feature filtering: Dropping constant or low-variance features

Constant or near-constant columns contribute no discriminative signal while increasing dimensionality and computation cost. Synthetic datasets sometimes include fixed ports or flags. Filtering these features clarifies the effective hypothesis space.

Feature filtering: Removing highly correlated features

Redundant predictors (e.g., bytes_sent and packets_sent) inflate variance and can distort importance rankings, especially in linear models. Correlation or mutual information based filtering reduces multicollinearity and improves interpretability. Prior work documents how unpruned redundancy misleads feature-importance analyses in IDS datasets [14].

Feature selection: RF importance

Tree ensembles provide embedded importance measures. As a rough but practical filter for mixed type features, they help prioritize variables and trim dimensionality. Empirical studies report both performance and interpretability gains after such selection in intrusion detection tasks [11].

Feature selection: Mutual information

Mutual information quantifies reductions in label uncertainty attributable to individual features and captures non-linear dependencies. On imbalanced datasets, mutual information based selection has been observed to improve minority class recall when applied judiciously and combined with rebalancing strategies [4].

Feature selection: ANOVA F-statistic

For numerical predictors, single variable ANOVA tests offer a fast initial filter by assessing mean differences across classes. In smart grid intrusion data, ANOVA based screening reduced noise while preserving discriminative temporal signals, providing an effective low-cost enhancement [22].

Dimensionality reduction: Principal component analysis (PCA)

Principal component analysis compresses high dimensional spaces into a smaller set of orthogonal components, mitigating collinearity and reducing runtime. The trade-off is interpretability, components are linear mixtures without a straightforward semantic mapping. PCA remains useful as an intermediate representation when downstream models are scale sensitive or when computational budgets are tight.

Scaling: Standardization and min max scaling

Algorithms such as support vector machines and neural networks are sensitive to feature scale. With features spanning seconds and bytes, unscaled inputs can misweight contributions. Standardization (zero mean, unit variance) or min–max scaling (fixed range) harmonizes magnitudes and improves optimization behavior. As with imputation, scalers must be fitted on training data only.

Normalization: Row wise L2 normalization

When inputs are counts or rates, longer sessions produce larger totals by construction. Row wise L2 normalization compares records proportionally rather than absolutely, reducing length effects and highlighting composition.

Outliers: Clipping

Extreme values arise from corruption, instrumentation limits, or genuine anomalies. A small number of extremes can dominate distances or gradients. Clipping at robust thresholds (e.g. percentile based caps) provides a conservative safeguard. In high-dimensional anomaly detection, this practice has been shown to stabilize estimators [23].

Transforms: Log1p transformation

Heavy tailed features (flow sizes, packet counts) benefit from variance stabilizing transforms. The log(1+x) function handles zeros, compresses dynamic range, and reduces skew, enabling models to focus on relative rather than absolute magnitudes.

Imbalance handling: Class weights

Most datasets are dominated by benign traffic. Cost sensitive learning reweights the loss to penalize minority misclassifications more heavily and consistently improves recall for rare attacks in IDS benchmarks [24]. Class weights should be derived from the training distribution and applied consistently across validation and test.

Imbalance handling: Resampling

Resampling complements reweighting. Oversampling synthesizes minority instances, undersampling trims the majority. Both alter the effective training distribution and should be evaluated with care to avoid introducing artifacts.

SVM kernel approximation

Nonlinear SVMs scale poorly with sample size. Kernel approximation schemes, like Nyström and random Fourier transformation, project data into a limited dimensional space that approximates the kernel, dramatically reducing training time with minimal loss in accuracy when dimensionality is chosen appropriately.

Time features

Temporal context is informative. Encodings such as hour of day, day of week and inter-arrival summaries capture diurnal cycles and campaign structure, improving robustness to drift. These features should be derived without leaking label or future information (e.g. computed strictly from past context in time-aware settings).

3.1.5 Machine learning models

This subsection summarizes the five supervised ML models employed as a baseline in A.R.C..They were selected to cover complementary inductive biases and practical trade offs for tabular intrusion-detection workloads. Tree ensembles for strong nonlinear performance with moderate interpretability, margin methods for high dimensional robustness, neural networks for flexible function approximation and a transparent linear-probability model as a reference.

Random forest (RF)

Random forests aggregate multiple decorrelated decision trees grown on bootstrap samples with random subspace selection at each split. This combination suppresses variance relative to single tree learners and delivers competitive performance on heterogeneous and nonlinear tabular data without extensive feature engineering. Because axis aligned partitions are averaged across the ensemble, moderate feature interactions are recovered and the method is comparatively robust to monotonic transformations and limited outlier contamination. Typical liabilities include larger memory footprints and increased prediction latency when many deep trees are employed. Extrapolation outside the convex body of the training data remains weak. Impurity-based importance measures can also be biased toward high variance or high cardinality predictors. Permutation based assessments are often more constant for attribution. [25]

Gradient boosted decision trees (GBDT)

Gradient boosting constructs an additive model by fitting shallow trees to the residuals of the current predictor, thereby reducing bias while controlling variance via

subsampling, and depth constraints. On structured, mixed type features it frequently attains high accuracy and inference is efficient, because prediction amounts to summing the outputs of a modest number of shallow trees. The method is sensitive to hyperparameter choices, label noise, and early stopping criteria, inadequate regularization, such as an overly large learning rate or excessive tree count, can lead to overfitting. Modern implementations, like LightGBM, introduce histogram based splitting, leaf wise growth and additional regularizer to improve throughput and scalability on large datasets [26], [27].

Support vector machines (SVM)

Support vector machines optimize a margin-based objective. The linear variant provides a strong high dimensional baseline, while kernelized SVMs map inputs into reproducing kernel Hilbert spaces to induce nonlinear decision boundaries, while retaining convex objectives under common kernels. Strengths include solid generalization under appropriate regularization and kernel choice, particularly when the number of informative features is large relative to sample size. Limitations are that kernel SVMs scale poorly with the number of samples, performance is sensitive to feature scaling and kernel selection. [28]

Multi layer perceptrons (MLP)

Multi-layer perceptrons are feed forward neural networks that compose affine transformations with nonlinear activations and are trained end to end via backpropagation. With sufficient capacity and appropriate regularization, MLPs approximate complex functions and capture interactions that elude linear models. In typical cybersecurity tabular settings, they can be data hungry and sensitive to optimization details. Overfitting risk is elevated for small or noisy samples, motivating explicit regularizer (e.g. dropout, weight decay) and close monitoring of validation dynamics. When well tuned performance can be competitive, when not convergence instability and degraded generalization are common [29].

Logistic regression (LR)

Logistic regression links predictors to log odds via a linear index and maps scores to probabilities through the logistic function. Coefficients admit direct interpretation as odds ratios, and associated standard errors enable interval estimation and Wald-type inference, making LR attractive for transparent risk modeling and effect estimation. Advantages include convex optimization, simplicity and mature diagnostics. Violations of logit linearity, multicollinearity, influential outliers or too few events per

variable can compromise estimation and inference. Extensions, like ordinal and penalized variants, mitigate several of these issues while preserving interpretability.

3.1.6 Evaluation metrics

Evaluating intrusion-detection models is inherently multi-objective. No single scalar summarizes utility across heterogeneous, imbalanced, and temporally structured datasets. Accuracy remains common but is often insufficient on its own; threshold-dependent measures, per-class diagnostics, and resource-centric indicators together provide a more reliable basis for comparison and deployment.

Training time

Training time is the wall clock duration to fit a model under a specified configuration. In fast-evolving threat landscapes retraining cadence is measured in days or even hours. Excessive fit times hinder adaptation. Classical ensembles (e.g. Random Forests) often offer a favorable accuracy time trade off relative to deep models on tabular data [11]. Training time is not only an efficiency KPI, it determines whether timely model updates are feasible.

Inference time

Inference time is the latency to classify new samples after training. It determines whether models can operate inline (e.g. smart-grid protection) or are constrained to batch analysis. High accuracy with prohibitive latency remains operationally inadequate.

Model size

Model size denotes the memory footprint of a trained artifact. While negligible on server class hardware, it constrains deployment on IoT devices and affects distribution, startup latency and memory contention. The size/accuracy trade off should be made explicit, especially for architectures intended for embedded or near real time use.

Accuracy

Overall accuracy measures the proportion of correct predictions,

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN},$$

where TP denotes true positives, TN true negatives, FP false positives, and FN false negatives.

On imbalanced corpora, high accuracy can be attained by overpredicting the majority class, masking poor minority detection. Accuracy is therefore reported, but not used in isolation.

Precision

Precision quantifies reliability of positive alerts, by calculating how many of the model's positive predictions were actually correct.

$$\text{Precision} = \frac{TP}{TP + FP}.$$

High precision reduces false-alarm load; in Security operation centers, excessive false positives erode trust and lead to alert fatigue [18]. Precision should be interpreted jointly with recall to avoid asymmetrical optimization.

Recall

Recall measures completeness of detection, by calculating how many of the model's positive predictions did the model catch?

$$\text{Recall} = \frac{TP}{TP + FN}.$$

In critical-infrastructure contexts, missed detections (false negatives) can be unacceptable; recall is often prioritized accordingly, with acceptable precision thresholds determined by triage capacity.

Precision recall curve

PR curves plot precision versus recall over thresholds and are particularly diagnostic under class imbalance, as they emphasize performance on the positive (attack) class (figure 3.5). Average precision (AP) summarizes the PR curve as the area under it—the weighted mean of precision at increasing recall levels:

$$\text{AP} = \sum_n \left(R_n - R_{n-1}\right) P_n,$$

with R_n and P_n the recall and precision at operating point n. AP approaches the positive-class prevalence for uninformative models and 1.0 for ideal detectors.

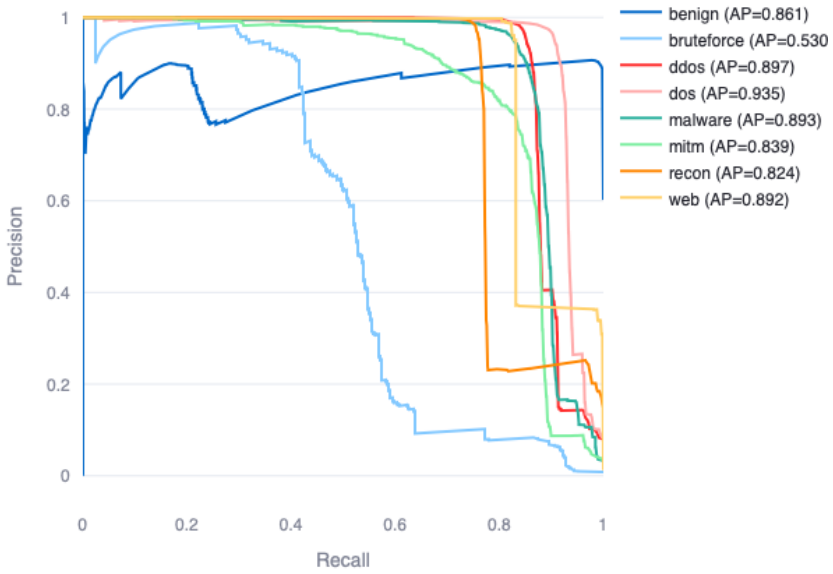


Figure 3.5 – Illustartion of a Percision-Recall curve for the raw CIC-IIoT test subset, splitted with startified splitt (60/20/20) and trained on a SVM [5]

F1 score

F1 aggregates precision and recall via the harmonic mean, it balances correctness with completeness.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

This is especially valuable in intrusion detection, where both false positives and false negatives matter. A system with great recall but terrible precision will overwhelm analysts. One with great precision but poor recall will let attacks slip by. The F1 score exposes these trade-offs in a simple and balanced way. On imbalanced datasets, F1 is typically more informative than accuracy, balancing over and underdetection [4].

ROC-AUC

Receiver Operating Characteristic curves plot true-positive rate against false-positive rate as the decision threshold varies. In the multiclass setting, one vs rest curves are computed per class and their areas averaged.

$$\text{ROC-AUC}_{\text{macro}} = \frac{1}{K} \sum_{k=1}^{K} \text{AUC}_k.$$

Macro-averaging assigns equal weight to each class and is less sensitive to prevalence than micro-averaging. Random ranking yields ≈ 0.5; perfect ranking yields 1.0.

ROC curve

Beyond its area, the ROC curve visualizes the TPR–FPR trade-off across thresholds (figure 3.6). The corresponding ROC–AUC equals the probability that a randomly chosen positive is ranked above a randomly chosen negative. Values range from 0.5 (uninformative) to 1.0 (perfect).

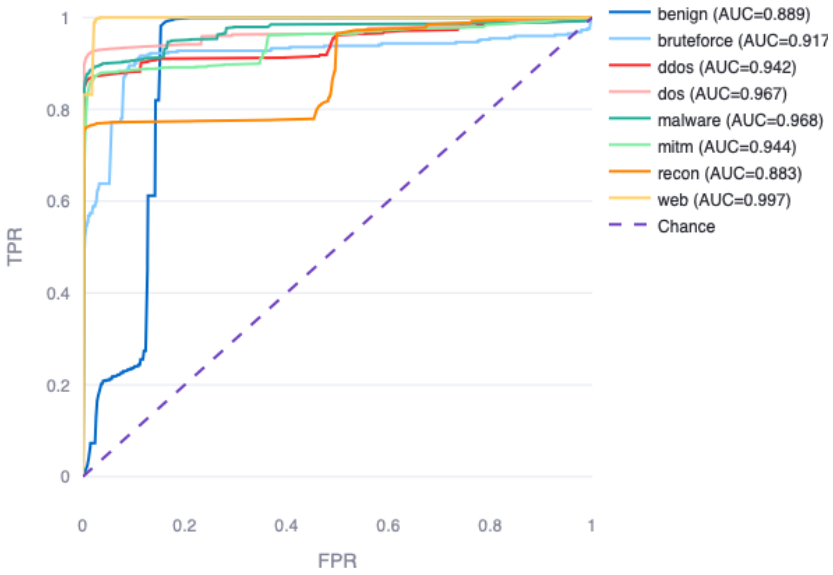


Figure 3.6 – Illustration of a ROC curve for the raw CIC-IIoT test subset, spitted with stratified split (60/20/20) and trained on a SVM [5]

Confusion matrix

The confusion matrix (figure 3.7) exposes per-class error structure beyond aggregates, revealing systematic confusions (e.g. specific attacks mislabeled as benign). In multi-class settings it is indispensable for diagnosing minority-class failure modes [11].

Feature importance (Top 25)

Feature importance estimates indicate which attributes contribute most to predictions. Tree models (Random Forests, Gradient Boosting) expose built in rankings, publishing the top-ranked features aids interpretability, highlights correlated families and can surface dataset artifacts when unstable across splits. Such reporting improves reproducibility and clarifies where models derive discriminative power [14].

Figure 3.7 – Illustration of a confusion matrix for the raw CIC-IIoT test subset, spitted with stratified split (60/20/20) and trained on a SVM [5]

3.2 Framework architecture

This thesis introduces A.R.C. (Automated Research for Cybersecurity), which is a framework that renders the preparation and validation of cybersecurity datasets systematic, modular, and reproducible. Prior studies often rely on improvised, dataset specific scripts that neither scale nor support replication, A.R.C. organizes the end to end workflow into well defined stages with explicit interfaces and auditable artefacts. The architecture is governed by four methodological principles, modularity, extensibility, reproducibility and transparency, which jointly counter recurrent weaknesses in existing practice, including undocumented preprocessing, inconsistent data preparation and leakage prone evaluation splits.

3.2.1 High-level architecture

A.R.C. comprises eight cooperating modules arranged into six functional groups that together realize the end-to-end workflow, as seen in figure 3.8. *Data ingestion* is handled by the **DataLoader**, which accepts heterogeneous inputs (e.g., CSV, PCAP) and harmonizes them into A.R.C.'s internal tabular schema with designated timestamp and label fields. *Validation* consists of two complementary components the **Schema Validator**, which inspects structural and temporal properties (feature inventory, datatypes, duplicates, ordering, coverage), and the **Label Validator**, which audits label completeness and consistency and summarizes class distributions. *Splitting* is

performed by the **Splitter**, which generates train/validation/test partitions (random, stratified, or time-based) with documented seeds and split manifests. *Preprocessing* is executed by the **Preprocessor**, a configurable pipeline that applies cleaning, encoding, feature filtering/selection, scaling or normalization, outlier handling, imbalance correction, and optional time-feature construction, all transformations are logged for replay. *Modeling and evaluation* comprises two modules. The **Trainer & Evaluator**, which fits baseline learners on the preprocessed splits and reports performance KPIs, efficiency KPIs and further graphs, like confusion matrices or ROC curves, under a common interface. And the **Compare module**, which trains the same model on not preprocessed ("RAW") data, with minimal preprocessing, and quantifies the delta to the preprocessed results produced by the Trainer & Evaluator. The *reporting* is provided by the **Reporter**, which compiles a PDF consolidating insights, parameter choices, figures, tables, and outcomes from each stage. All generated artifacts (datasets, split subsets, trained models, metadata) are available to download.

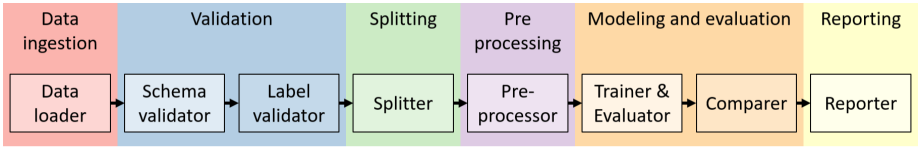


Figure 3.8 – Illustration of A.R.C.'s high level architecture

All modules are orchestrated by a pipeline controller exposed through a Streamlit interface. This controller enables interactive configuration while preserving a deterministic execution order. Decoupled back end components enable interactively use. The layered structure follows separation of concerns principles, improving maintainability and reliability across the backend.

3.2.2 Design principles

Modularity

Cybersecurity data preparation spans heterogeneous tasks, from missing value imputation and duplicate removal to temporal splitting and class rebalancing. Rather than entangling these steps in monolithic scripts, A.R.C. decomposes functionality into modules with stable, minimal interfaces. Components can be reordered, omitted or replaced without destabilizing the pipeline.

Extensibility

A.R.C. is designed to evolve. New methods can be registered behind existing interfaces without rewriting legacy code. An autoencoder based anomaly detector

can be inserted into the modeling stage, or a novel resampling strategy introduced during preprocessing, without adverse effects on other modules.

Reproducibility

Each step records parameters, random seeds and outputs as machine- and human-readable metadata. Reports capture results together with preprocessing lineage, such as which features were imputed or dropped and how splits were generated, so that experiments can be reconstructed exactly. Reproducibility is an architectural property rather than an afterthought.

Transparency

All components build on peer-reviewed, open-source libraries (e.g. pandas, scikit-learn, imbalanced-learn) and expose decisions through visual reports (e.g. class distributions, drift plots, confusion matrices). The Streamlit interface renders these steps auditable, accessible and understandable, even to researchers who are not machine-learning specialists.

3.2.3 Data flow through A.R.C.

Figure 3.9 describes the end-to-end data path in A.R.C., from raw inputs to exportable artifacts. The *Data Loader* admits one or more RAW datasets (e.g. CSV or PCAP-derived tables) and normalizes them into A.R.C.'s internal schema with designated timestamp and label fields. The resulting table flows to the *Schema Validator*, which inspects structural and temporal properties and to the *Label Validator*, which audits completeness and consistency. Both components surface issues before modeling and record their findings for later reporting.

Validated data proceed to the *Splitter*, where train, validation and test subsets are produced under random, stratified, or time-based regimes with documented parameters. The *Preprocessor* then applies the configured transformations (cleaning, encoding, feature filtering/selection, scaling or normalization, outlier handling, imbalance correction, and optional time-feature construction) yielding the PP (preprocessed) datasets used for training. The *Trainer & Evaluator* consumes these PP splits, fits the selected baseline learner and reports both discrimination and efficiency metrics under a common interface.

A second branch in Figure 3.9, labeled *Minimal preprocessing*, routes the same splits through a lightweight RAW pipeline. The *Compare* module trains the identical model family on these minimally processed data and contrasts the resulting "RAW" metrics with the PP metrics produced by the Trainer & Evaluator, thereby quantifying the contribution of standardized preprocessing while holding model choice constant.

Throughout the pipeline, modules emit artifacts that can be downloaded directly from the interface. The data loader can export the harmonized dataset as CSV. The Splitter saves per-split CSVs. The preprocessor writes PP datasets together with a JSON metadata manifest that captures parameter choices and transformation lineage. The Trainer & Evaluator persists the trained model on PP data and its metric bundle, the Compare module does the same for the RAW branch. Finally, the *Reporter* assembles a PDF that consolidates insights, parameter settings, figures, and results from each stage and it also exposes the complete log to support exact reruns.

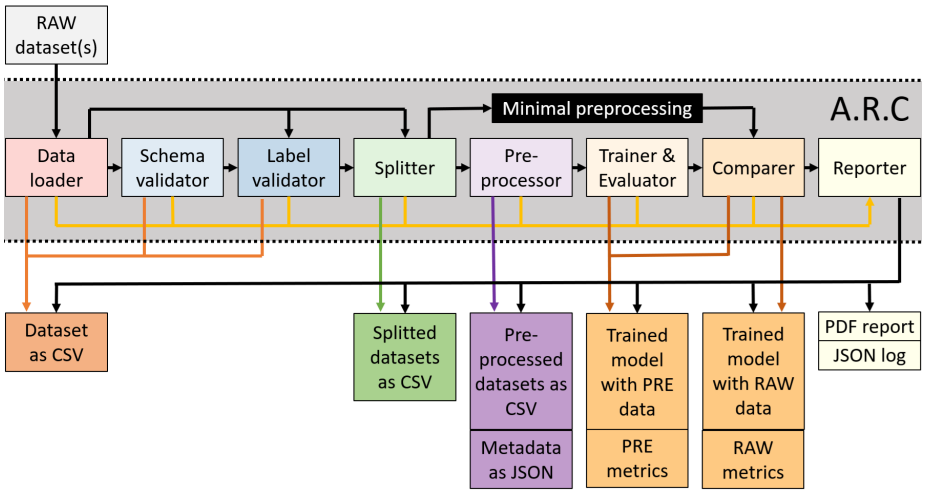


Figure 3.9 – Illustration of the data flow through A.R.C.

3.2.4 Software stack

A.R.C. was build with Python (3.13.6). User interaction and orchestration are provided by Streamlit (1.48.0). Tabular processing relies pandas for joins, aggregations, input and output. NumPy is supplying numerical kernels. scikit-learn contributes preprocessing utilities (e.g. scalers, encoders, PCA), models and evaluation tooling, while imbalanced-learn provides methods for class-imbalance mitigation. Interactive visualizations are implemented with Plotly, and ReportLab handles PDF export to produce citable, shareable reports.

Python as a foundation

Python offers readable and maintainable code with access to high performance kernels through vectorization and extensions. Its flexibility supports rapid experimentation without sacrificing execution speed where it matters.

Streamlit for accessibility

The Streamlit front-end lowers the barrier to adoption and encourages good experimental hygiene. Inputs are explicit, outputs are visible and runs can be shared as self-contained dashboards.

Pandas and NumPy for data handling

Cybersecurity datasets are predominantly tabular, large, and occasionally messy. pandas NumPy support the core transformations (e.g. joins, filters, resampling) that determine preprocessing quality within A.R.C.

Scikit-learn and imbalanced-learn for modeling

These libraries provide stable APIs, deterministic behavior when seeded, and well tested implementations across preprocessing, model training, cross-validation, metrics, and resampling. While reducing engineering overhead and enhancing reliability.

Plotly and ReportLab for reporting

Exploratory analysis benefits from interactive visualizations and reproducibility benefits from fixed artefacts. Plotly covers the former, while ReportLab consolidates decisions, metrics, and figures into PDFs that do not depend on a live session.

3.2.5 Repository structure

The repository separates user interface and scientific logic. Streamlit pages (e.g. pages/01_Data_Loader.py) define UI elements and visuals. Computational logic for schema checks, preprocessing, and training resides under functions/ (e.g. function/f01_Dataloader.py), enabling import for testing or notebook use independent of the UI. This arrangement preserves a lightweight, replaceable UI while keeping scientific logic reusable and extensible within A.R.C.

3.2.6 Integration strategy

Integration proceeds across three layers. The UI layer supplies controls, configuration, and visualization through Streamlit components. The processing layer implements validation, preprocessing, training, and evaluation as callable functions. The data layer manages shared datasets and parameters via streamlit's session_state. Because interfaces are stable, modules remain optional and can be bypassed or replaced without breaking the workflow that A.R.C. enforces.

3.2.7 Module interaction

Modules communicate primarily through session_state, which exposes shared data and configuration to all pages. A JSON log records key decisions and parameter values, producing an auditable trail from ingestion to reporting. Loose coupling allows skipping modules entirely or inserting domain-specific feature engineering without modifying downstream components. UI orchestration clarifies execution order while avoiding rigid, hard-wired dependencies.

3.2.8 State management

Long-running workflows require continuity of state. Datasets are stored under a key (e.g. _DF). Intermediate outputs such as schema statistics and preprocessing summaries persist as Python objects and user configurations follow the user across modules. Serialized snapshots of session state enable exact reconstruction of experiments, directly supporting the reproducibility goals of A.R.C..

3.2.9 Workflow orchestration

The default execution path is semi linear as seen in figure 3.9. datasets are loaded, schemas and labels are validated, splits are created and audited, preprocessing is applied, models are trained and evaluated, results are compared against a minimal "RAW" baseline, and reports and artifacts are generated. Individual steps can be re-run or skipped as required by the study design, while the controller maintains a coherent lineage for all outputs.

3.2.10 Export handling

Artifacts persist beyond the session. Datasets at each stage are exported as CSV, metadata and parameters as JSON, trained models as PKL and consolidated reports as PDF. The Streamlit interface provides download widgets for convenient access to all outputs produced by A.R.C..

3.2.11 Reproducibility mechanisms

Reproducibility is enforced at multiple levels. JSON logs capture parameters and actions, fixed random seeds are set as default to ensure determinism and generated reports embed dataset provenance together with complete preprocessing configurations. Collectively, these mechanisms provide end-to-end traceability and align A.R.C. with FAIR research practice.

3.3 Framework modules

Building on the conceptual architecture in section 3.2, this chapter details the modules that instantiate the framework as a functioning end to end pipeline for cybersecurity datasets, as seen in 3.9. Each component fulfills a narrowly defined role and exposes explicit contracts for inputs and outputs, it also records its own execution. This separation of concerns is deliberate. It reflects enduring principles of modular software engineering that focus on clarity, testability, and ease of evolution.

The design aligns with the current practice in reproducible machine learning. Orign of preprocessing steps, environment configuration and parameter choices is preserved so that experiments can be reconstructed exactly. Interfaces are intentionally minimal and pipeline like. Consistent with the FAIR guidelines, all modules return machine readable metadata, ensuring that artefacts are not only consumable once but reusable across studies and over time.

3.3.1 Data loader

Motivation and role in the framework

Ingestion is the first and often most consequential operation in A.R.C.. If the entry point is inconsistent or poorly documented, all subsequent stages inherit that fragility. The Data Loader addresses this risk by transforming heterogeneous inputs into a uniforma and auditable representation that anchors validation, preprocessing and modelling. File formats are abstracted away, outputs adopt a consistent schema, provenance is captured. As a result, every experiment begins from a stable and transparent dataset state.

Implementation design

The module comprises three cooperating layers an interactive UI, a flexible parsing engine and a standardization pass that unifies outputs. A Streamlit interface guides the user through single file ingestion (CSV, TXT, PCAP, PCAPNG) or multi-file composition (CSV collections). For packet captures, the UI exposes granular controls over decoded protocol layers and aggregation level (packet, flow, session), thereby aligning extraction with analytical intent (Figure 3.10). Packet captures are processed with scapy, with optional streaming via PcapReader for large files and toggles for Level 2 to Level 7 features, including lightweight payload peeks. For multi CSV workflows, the interface supports delimiter and encoding selection, header handling, optional source file tagging and schema reconciliation by union or intersection (Figure 3.11). These choices are explicit in the UI rather than hidden in code, which improves auditability. Tabular inputs are parsed with pandas. All parsing occurs in

memory to ensure consistent hand off to downstream components. Regardless of origin, outputs are then normalized to a pandas dataframe with explicit data types and designated fields for timestamp and label. In parallel, a JSON log records file origins, parsing parameters, selected protocol layers, aggregation settings, and any inferred temporal fields.

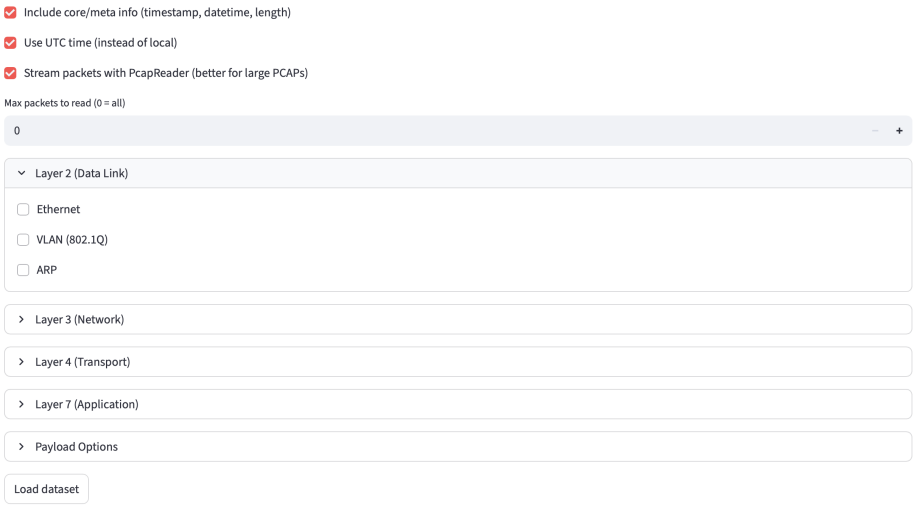


Figure 3.10 – Illustration of the configuration menu for PCAP and PCANPNG files in the A.R.C. UI

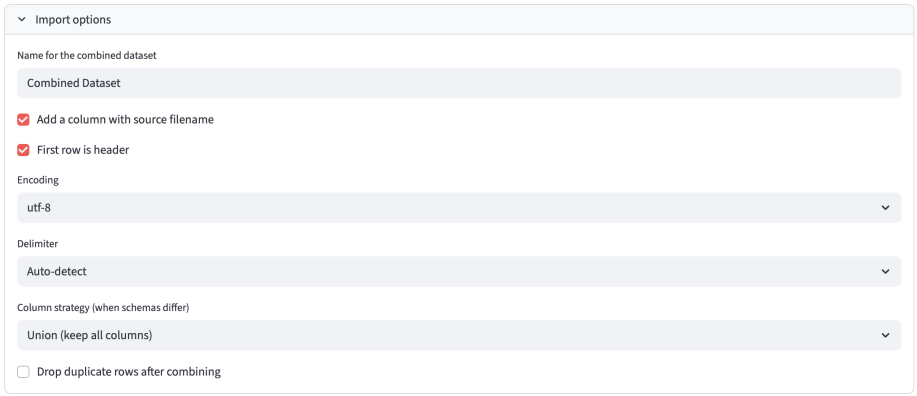


Figure 3.11 – Illustration of the configuration menu for multiple CSV files in the A.R.C. UI

Inputs and outputs

The module accepts single CSV files, TXT files, PCAP captures, PCAPNG captures or multiple CSV files as shown in figure 3.12. PCAPs and PCAPNGs with configurable

parsing options, multiple CSV files with user defined merge rules. It then reads each source and transforms it into a pandas dataFrame, logs all data-loading metadata, scans for temporal fields to establish time features and designates the label column. The outputs are a standardized pandas dataFrame ready for validation and a harmonized CSV dataset available for download.

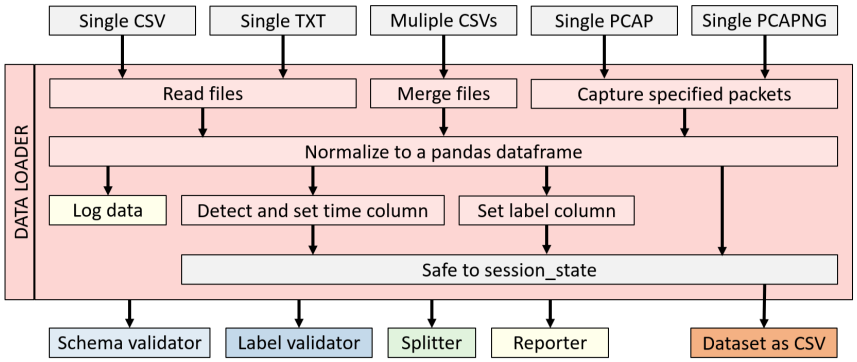


Figure 3.12 – Illustration of the data flow through the data loader

Automatic timestamp detection

Because temporal integrity is a central key for many cybersecurity analyses, the loader inspects columns for timestamp like content and proposes candidates. If several are plausible, a default is selected and can be overridden interactively. This simple safeguard prevents downstream errors, such as drift analyses based on an incorrect temporal key, while keeping the decision visible to the user.

User visualization and feedback

During ingestion the interface provides immediate feedback. A preview, with the first five records, and a summary, with indicators reporting on row and column counts, are displayed. Candidate timestamp fields are highlighted and the currently selected label is displayed, enabling rapid verification before proceeding to validation.

Reproducibility considerations

All ingestion decisions are persisted. The JSON log enumerates file names and sources, delimiter and encoding choices, protocol layer toggles for packet parsing, aggregation level and the selected timestamp and label columns. A concise dataset report can be embedded into the final PDF generated by the Reporter, ensuring that the ingestion lineage remains citable and reproducible across runs and studies.

3.3.2 Schema validator

Motivation and role in the framework

Schema validation in A.R.C. is the monitor for dataset readiness. It verifies structural integrity, datatype coherence, and temporal consistency prior to any modeling. This stage is critical in cybersecurity settings, where datasets are often aggregated from heterogeneous sensors and collection pipelines. If irregularities such as duplicate rows, missing or unsorted timestamps, mixed datatypes or inconsistent feature naming are not resolved early, downstream training and evaluation become brittle, irreproducible, and potentially misleading.

Implementation design

The module comprises two coordinated capabilities, insight extraction and schema modification. Insight extraction profiles the dataset and renders diagnostics, while remediation applies user approved fixes and records them for auditability. The insight extraction reports record counts and duplicate rates, quantifies missingness at both table and column level, infers granularity and summarizes statistics, it inspects declared dtypes and flags mixed-type columns. Checks for temporal integrity, like monotonicity, duplicates, gaps and overlaps, are concludet. And it verifies the presence of researcher defined required features. Where needed, the schema modification offers safe transformations, like normalization of column names to snake_case, interactive renaming of features, timestamp sorting, derivation of record time deltas and removal selective dropping of problematic features. All actions are deterministic, applied only on explicit user confirmation and captured in an audit recorded.

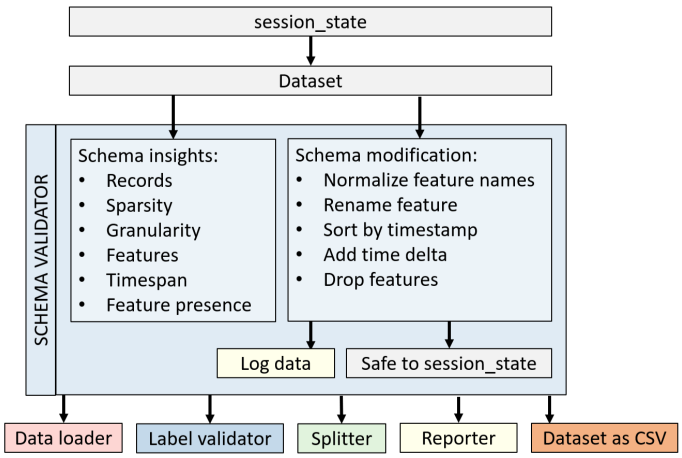


Figure 3.13 – Illustartion of the data flow through the schema validator

Inputs and outputs

The validator accepts any standardized pandas dataframe stored in a session_state, which emitted by the A.R.C. data loader (figure 3.13). Its outputs are a schema cleaned dataframe suitable for subsequent splitting and preprocessing, a JSON manifest that records diagnostics and every applied transformation with parameters, an exportable CSV of the cleaned table and a PDF report that summarizes findings and changes for human review.

User visualization and feedback

A.R.C. exposes schema diagnostics through an interactive dashboard that couples concise numeric summaries with visuals and modification controls.

The total record count is displayed as an absolute number, while duplicate rows are reported both as counts and as a percentage of the dataset. Overall sparsity is quantified as the count of missing cells, when present, per-feature missingness is highlighted and the proportion per column is shown. Granularity (e.g. packet, flow, session) is summarized in textual form. The feature inventory reports the number of columns, accompanied by flags for duplicated features and for mixed datatype columns. Datatype composition is visualized as a pie chart and complemented by grouped lists that enumerate counts per type class. Temporal diagnostics provide flags for duplicated and overlapping timestamps as well as order, they summarize start and end times, total duration, most common interval and average interval as text. Feature presence is validated against a user supplied list of "required" attributes, a text input captures the specification and the validator returns two lists, missing required features and unexpected/extra features, each annotated with counts.

Modification is initiated via explicit controls with a description and explanation of purpose. A single action normalizes feature names to snake_case. An interactive renaming control accepts plain text mappings of the form old_name = new_name (one per line) and translates them into an internal renaming matrix. A "sort by timestamp" control applies chronological ordering and optionally derives a time-delta column via a checkbox toggle. Feature removal is supported through a searchable dropdown that allows users to select features to drop. At the end of the workflow, a download control exports the cleaned dataset as a CSV file, enabling immediate reuse in subsequent A.R.C. stages or external tools.

Reproducibility considerations

Every modification is logged with sufficient detail to recreate the cleaned dataset, the pre post column names, the sort key and order, duplicate detection policy, and any feature drops are written to a machine readable manifest and embedded in the

PDF report. This guarantees that the same input can be transformed into the same output, enabling exact reruns and external audit.

3.3.3 Label validator

Motivation and role in the framework

Labels constitute the ground truth for supervised machine learning. Their quality directly bounds what a classifier can learn and how results should be interpreted. In cybersecurity datasets, often aggregated across tooling, teams, and campaigns, label noise manifests as missing targets, spelling and ontology drift, and severe imbalance. Left unchecked, these issues inflate headline metrics and undermine external validity. A.R.C.'s label validator therefore establishes label readiness before training by auditing consistency, quantifying distributional properties, and surfacing temporal behavior.

Implementation design

The module combines automated diagnostics with optional remediation under explicit user control. It quantifies missing labels at table and class level, detects inconsistent strings (e.g. case variants, stray whitespace, near duplicates), summarizes class prevalence with entropy and absolute and relative shares, flags rare and dominant classes using user configurable thresholds and profiles temporal dynamics by binning the timeline and tracing label proportions to reveal drift or scenario block effects. When enabled, remediation tools canonicalize case and whitespace, apply user defined synonym and alias mappings to merge classes. All actions are deterministic, require user confirmation and are recorded for auditability.

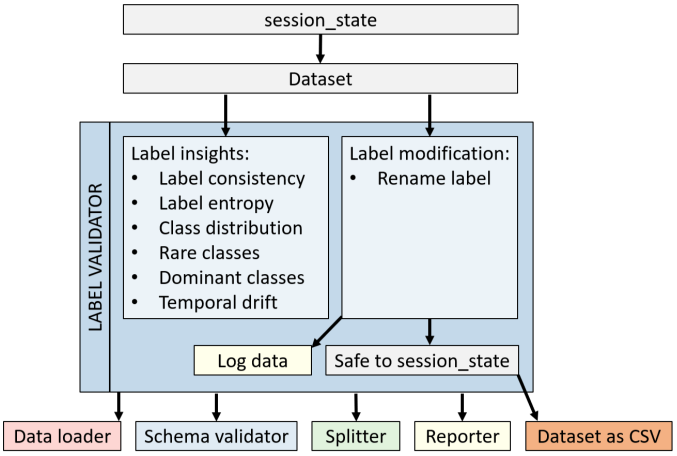


Figure 3.14 – Illustration of the data flow through the label validator

Inputs and outputs

Input is a standardized pandas dataframe from the data loader with a single, designated label column and if available, a timestamp (figure 3.14). Outputs include a label cleaned dataframe, a JSON manifest capturing diagnostics, parameters and any applied mappings, a exportable CSV containing the label validated data and a PDF report with tables and figures documenting label characteristics and changes.

User visualization and feedback

A.R.C. presents label diagnostics in a consolidated, interactive view. Missing labels and spelling consistency are surfaced as flags so that potential issues are immediately visible. Class entropy is reported as a numeric value accompanied by a plain language interpretation (e.g. “labels are mostly balanced”), helping readers relate the statistic to practical class skew. The class distribution itself is rendered three ways, as a pie chart, a bar chart and a table of absolute and relative frequencies, to support both quick inspection and precise reading. Imbalance is made explicit through two parameterized checks. “Rare classes” and “dominant classes” are each flagged according to user defined thresholds, changing these thresholds updates the flags and summaries in real time. Temporal behavior is configurable via check boxes, users can display a raw timeline (no bucketing), equal interval time bins (all bins share the same duration), or equal count time bins (all bins contain the same number of records). The number of bins is user specified, enabling coarse or fine temporal resolution as needed.

When label heterogeneity arises from inconsistent naming, labels can be renamed directly in the interface using the same plain text mapping syntax as the schema validator (old_name = new_name). The input is transformed into an internal renaming matrix and applied deterministically. After review, the validated dataset can be exported via a download button as a CSV file.

Configuration and defaults

Default settings include a rare class threshold of <1%, a dominant class threshold of >90% and equal interval binning with 10 bins for temporal summaries. All defaults are editable in the UI, any deviation is collected and embedded in the PDF report.

Reproducibility considerations

Every diagnostic and modification is logged, the original and canonical label strings, the alias mapping applied, thresholds used for rare/dominant flags, temporal binning parameters, software versions, and dataset hashes. The manifest is embedded in the

PDF report and saved as JSON, ensuring that the same input can be transformed into the same output and that results remain auditable across runs and environments.

3.3.4 Splitter

Motivation and role in the framework

Reliable assessment in supervised learning requires evaluation on samples that were not available to the learner during training. In cybersecurity datasets, where temporal autocorrelation and attack bursts,are common, partitioning can induce leakage and distort metrics. A.R.C.'s splitter institutionalizes dataset partitioning into training, validation, and test subsets so that class coverage is explicit, leakage risks are minimized, and the evaluation protocol mirrors realistic deployment. Standardizing this step is essential for fair model comparison and reproducible results.

Implementation design

The Splitter is implemented as an interactive Streamlit page organized into three sequential panes, information, splitting and quality checks, followed by a download area (figure 3.15).

The information pane presents a compact comparison of the three supported regimes (random, stratified, time based), summarizing advantages and typical use cases to ground method selection.

The splitting pane performs the partitioning. A drop down selects the regime. For random and stratified modes, a user settable random_state (set by default to 42) guarantees determinism. For the time based mode, a user settable gap ratio (set by default to 0) specifies an embargo between the training window and combined tail with validation and test. Two sliders define validation and test proportions, with the remainder assigned to training (set by default to 20% validation, 20% test and 60% training). A single action button triggers the run. Immediately after completion, subset sizes are reported as absolute counts and percentages to confirm that requested proportions have been realized.

Time based splitting enforces chronological order and an optional gap between training and the combined tail, containing validation and test. The module first checks for a valid timestamp column. A user defined gap ratio inserts a contiguous hold out interval immediately after training to reduce near boundary leakage. those records are excluded from all subsets. The residual tail is then divided, in time order, into validation and test according to the requested proportions. The splitter records the regime parameters, and the exact temporal boundaries for training, embargo, validation, and test to enable exact reconstruction.

The quality checks pane audits the resulting partitions. Unseen labels are flagged separately for validation and for test, each accompanied by an explicit list of classes absent from training. Class composition is then visualized in three complementary views that are also provided as tables for exact counts. A per subset view in which training, validation, and test each sum to 100% as horizontal stacked bars to assess within subset balance, a per label view in which each class sums to 100% as vertical stacked bars to show how records for a given label are apportioned across subsets and a global view in which all subsets together sum to 100% as horizontal stacked bars to summarize overall allocation.

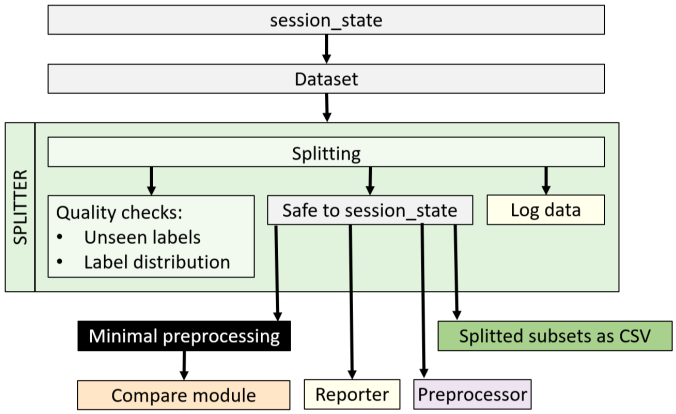


Figure 3.15 – Illustration of the data flow through the splitter

Inputs and outputs

The Splitter consumes the labeled dataframe produced before and for time based partitioning, the timestamp column identified during ingestion. It produces three subset dataframes (train, validation, test). A download control exports the the splitted partitions as X_train, y_train, X_validate, y_validate, X_test and y_test.

User visualization and feedback

All outcomes are rendered immediately in the Streamlit interface to support rapid iteration. Subset sizes appear as absolute counts and percentages; unseen label conditions are flagged with enumerated lists and the three distribution views, per subset, per label and overall, are available as both charts and tables for precise reading.

Reproducibility considerations

Determinism is enforced for stochastic regimes via the configured random_state, chronological regimes rely on stable sorting. The PDF report records all parameters

required for exact reconstruction (regime, proportions, seed, gap ratio, timestamp field). Charts in the PDF are regenerated from these parameters, ensuring that partitions can be reproduced and audited across runs and environments.

3.3.5 Preprocessor

Motivation and role in the framework

Preprocessing is the bridge between raw ingestion and effective learning. Cyber security datasets are heterogeneous, often mixing categorical and numerical attributes, exhibiting missing values, duplicates, heavy skew, and pronounced class imbalance. Left untreated, these issues degrade accuracy, inflate variance and impede comparability across studies. The A.R.C. preprocessing module consolidates cleaning, encoding, feature filtering and selection, scaling, outlier handling, imbalance mitigation, and optional time feature engineering into a transparent and leakage safe workflow.

Implementation design

A.R.C. separates guidance from execution. An information pane provides model aware recommendations with descriptions of each preprocessing step and a short statement of its purpose. The execution pane applies user selected transformations or a one click preset matched to the chosen algorithm (figure 3.16). All functionality relies on established open source libraries (pandas, scikit-learn, imbalanced-learn, NumPy) and every transformation is recorded with parameters in a machine readable log.

The module enforces a leakage safe order. Statistics are estimated only on the training split and then applied to validation and test. The pipeline proceeds as follows, first optional time feature extraction, then duplicate removal with label alignment, then missing value imputation (numeric and categorical), then bucketing of rare categories, then categorical encoding, either generic (one hot or ordinal) or library native for gradient boosting (CatBoost, LightGBM, XGBoost), then feature pruning and selection (low variance, high correlation, RF importance, mutual information, ANOVA F, optional PCA), then outlier clipping (IQR or percentile) and optional log1p transformations,then scaling and normalization (Z Score, Robust, Min Max, row wise L 2), then optional kernel feature maps for SVMs (Nyström or random Fourier features) and at the end class imbalance handling via class weights or resampling (SMOTE, random over under). The module guards common pitfalls e.g. it drops all NaN numeric columns after sanitization, converts infinity to NaN before imputing, refuses SMOTE prior to numerical encoding and records infeasible configurations with clear messages.

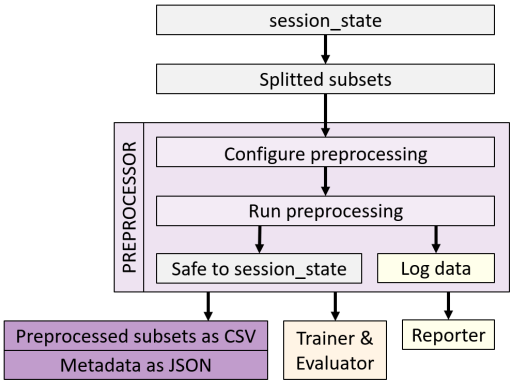


Figure 3.16 – Illustration of the data flow through the preprocessor

Model families and preprocessing guidance

A.R.C. supports five supervised baselines (Random Forests, Gradient Boosting, SVM, MLP, Logistic Regression) whose characteristics are summarized in section 3.1.5. The UI offers a one click preset per model that aligns preprocessing to the learner (no scaling for trees, Z Score scaling for SVM,MLP and LR, GBM native categorical handling where available, conservative clipping for gradient methods, class weights before resampling). Table 3.1 lists the initial suggestions; users then refine thresholds and choices in response to diagnostics from the schema and label validators.

Configuration variables and default settings

To make runs comparable, all variables have explicit defaults that are also encoded in the presets. Numerically imputed values use median by default, while categorical imputation uses the most frequent category. Rare category bucketing starts with a minimum frequency of 20. Generic encoding defaults to one hot unless a GBM native categorical mode is selected. Low variance filtering begins at 0.0 (to drop only constant features) and high correlation pruning uses an absolute correlation threshold of 0.98. RF importance selection keeps top 100 features unless changed, mutual information and ANOVA filters default to top 200 when enabled. PCA retains 95% explained variance. Outlier clipping uses IQR with whisker 3.0 by default, percentile clipping, when chosen, uses 0.5%/99.5% bounds. Scaling defaults to Z Score for SVM, MLP and LR and is off for tree models. Min max, if selected, maps to [0,1]. Row wise L2 normalization is disabled by default. Kernel feature maps for SVMs are off by default, when enabled, Nyström and Random Fourier use 2.000 components unless specified. Resampling is disabled by default and if enabled, SMOTE uses k_neighbors=5. Time feature extraction is enabled automatically when a timestamp column is present. All defaults are surfaced in the UI next to each

Table 3.1 – Overview of preprocessing steps applied per model

Category	Preprocessing step	RF	GBDT	SVM	MLP	LR
Cleaning	Drop duplicates	✓	✓	✓	✓	✓
	Impute missing values	✓	✓	✓	✓	✓
Encoding	Bucket rare categories	✓	✓	✓	✓	✓
	One Hot Encoding	✓	×	✓	✓	✓
	CatBoost encoding	×	✓	×	×	×
	Count encoing	×	×	×	×	×
	Target mean encoding	×	×	×	×	×
Feature selection	Drop constant / low variance	✓	✓	✓	✓	✓
	Drop highly correlated numerics	✓	✓	✓	×	✓
	Feature selection by RF importance	✓	×	×	×	×
	Feature selection by mutual information	×	×	✓	×	✓
	Feature selection by ANOVA F	×	×	✓	×	✓
	PCA reduction	×	×	×	×	×
Scaling & normalisation	Scaling	×	×	✓	✓	✓
	Row-wise L2 normalisation	×	×	×	×	×
Outlierer & transformation	Outlierer clipping	✓	✓	✓	✓	✓
	log1p transformation	×	×	×	×	×
Class imbalance	Class weigths	✓	✓	✓	✓	✓
	Resampeling	×	×	×	×	×
SVM	SVM Kernel Approcimation	×	×	✓	×	×
Time features	Add time features	✓	✓	✓	✓	✓

control, are saved to the JSON metadata log with exact values, and are echoed in the PDF report for auditability.

Inputs and outputs

Inputs are the three splits produced by the splitter: X_train, y_train, X_val, y_val, X_test, y_test. The module returns transformed splits and a comprehensive JSON metadata record. Metadata includes the ordered list of executed steps with parameters, label encoding maps, kept and dropped features, selector scores, scaler statistics, kernel map settings, imbalance configuration, column alignment performed, and final shapes. A PDF summary and CSV exports of preprocessed splits are generated for audit and reuse.

User visualization and feedback

Each preprocessing step is shown in the Streamlit interface with a concise description of the operation, a understandable purpose statement explaining why one would enable it, and the configuration variables required to control it. Toggles enable or disable a step; parameter widgets expose thresholds (e.g. variance or correlation cut offs), encoder choices, selector budgets, scaler modes and ranges, outlier whiskers or percentiles, kernel map components, and imbalance options.

Reproducibility considerations

All transformations are deterministic given the recorded configuration. Fit artifacts (imputers, encoders, scalers, PCA, kernel maps) are implicitly captured through their parameters and training statistics, random seeds are fixed where relevant. The metadata log serializes step order, hyperparameters and data dependent statistics computed on the training split, enabling exact reconstruction. The final report embeds these details.

3.3.6 Trainer and evaluator

Motivation and role in the framework

Training and evaluation turn validated, preprocessed data into deployable models. In A.R.C., these stages are transparent and reproducible by design, learners are trained through a uniform interface, results are computed consistently across splits, efficiency is profiled alongside accuracy and every choice is logged so a run can be explained and repeated.

Implementation design

The module is organized as two cooperating layers connected by the Streamlit configuration panel. The model training layer and the evaluation layer (figure 3.17), which computes statistical metrics and efficiency indicators while persisting full run metadata. The UI writes the selected hyperparameters to session state and the orchestrator composes the estimator accordingly.

The configuration panel exposes the important training variables for each learner with recommended defaults prefilled. Random Forest is setup by default with 100 trees, the depth is capped at 18, a minimum of two samples per leaf, square root feature sub sampling, bootstrap enabled, and a seventy percent per tree subsample. These settings are editable and persist between sessions. Gradient Boosting is setup by default with a selectable engine, 800 boosting rounds and a learning rate of 0.05. Support Vector Machines default to an RBF kernel with probability estimates enabled. The Multi layer perceptron view defaults to a two layer layout (128, 64) and a maximum of 300 epochs. Logistic Regression starts with the LBFGS solver, a generous iteration cap for convergence of 2000. Global controls include the random seed (default 42), the number of CPU threads where supported.

Evaluation spans predictive performance, computational efficiency, and interpretability. Accuracy, macro precision, recall, and F1 are reported for train, validation, and test splits, with ROC–AUC when calibrated scores are available. Training time, per split inference latency, model size and the number of features seen at fit quantify operational viability. Confusion matrices support error analysis, while ROC and

precision–recall curves provide threshold swept views. Where available, top 25 feature importance are extracted from tree based models.

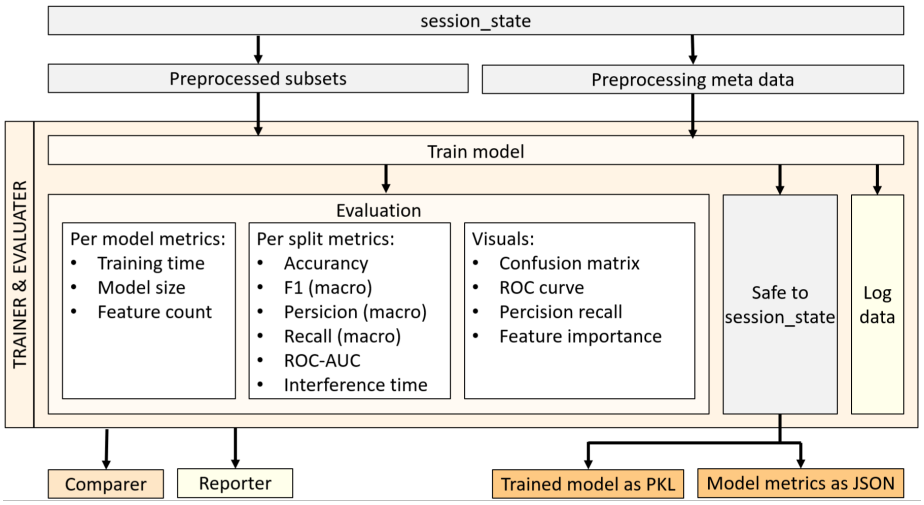


Figure 3.17 – Illustration of the data flow through the trainer & evaluator

Inputs and outputs

The module consumes the preprocessed train, validation, and test sets, together with optional sample weights and metadata describing categorical handling. It returns a trained model parameterized exactly as configured in the UI, per split metrics and timings, interactive plots for confusion, ROC, and precision–recall and a size summary for the serialized artifact. The model can be downloaded as a PKL and a PDF report can be generated for archival and review.

User visualization and feedback

The interface presents KPIs for training time, model size and feature count alongside a compact table of accuracy, macro F1, precision, recall, ROC–AUC, and per split inference time. Confusion matrices render with counts, ROC and precision–recall curves are plotted per class when probabilities are available, and feature-importance bars highlight the most influential inputs. The preprocessing panel remains visible during review, linking observed behavior to the transformations that produced the training matrices.

Reproducibility considerations

Every run fixes the random seed and persists the exact hyperparameters received from the UI.

3.3.7 Comparer

Motivation and role in the framework

Preprocessing is widely recommended, but its concrete effect on downstream models is rarely quantified. The Compare Module makes that effect explicit by training and evaluating two pipelines under identical conditions, a minimally processed route and the fully preprocessed route produced by A.R.C. It answers a single question with evidence rather than conjecture, how much does careful preprocessing change predictive quality and operational efficiency?

Implementation design

The module runs in two coordinated passes driven by the same splits, seeds and hyperparameter that the user selected in the Training and Evaluation module (figure 3.18). The first pass establishes a baseline with a minimal pipeline (table 3.2) that applies only what is strictly necessary to make the learner trainable and numerically stable. The second pass reuses the trained, fully preprocessed model already present in session state. Both passes are executed through a unified orchestrator, which composes the estimator with minimal preprocessing when comparing on raw data and leaves the estimator unchanged when comparing on the preprocessed matrices. Class labels are aligned using the label encoder retained from preprocessing so that confusion matrices, ROC curves and precision–recall curves are comparable across routes. The UI then renders the metrics and timing for each route and computes deltas consistently, reporting absolute percentage, point differences for rate metrics and relative percent change for latencies and resource measures. Colors indicate whether a difference is beneficial.

Table 3.2 – Overview of minimal preprocessing steps applied per model

Preprocessing step	RF	GB (CatBoost)	SVM	MLP	LR
Preclean (DataFrame pass-through)	✓	✓	✓	✓	✓
Replace ∞ with NaN	✓	×	✓	✓	✓
Impute numerics (median)	✓	×	✓	✓	✓
Impute categoricals (most frequent)	✓	×	✓	✓	✓
One-hot encode categoricals	✓	×	✓	✓	✓
Standardize numerics (mean/std)	×	×	✓	✓	✓
Cast numerics to `float32`	✓	×	×	×	×

Inputs and outputs

The module consumes the raw training, validation, and test splits and applies minimally preprocessing, alongside the fully preprocessed results already present in training and evaluation. It returns a paired result set comprising per split accuracy,

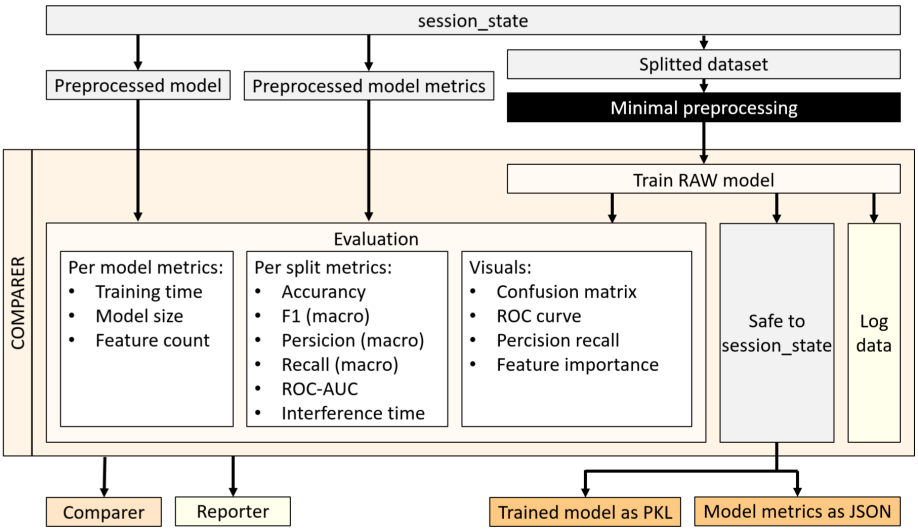


Figure 3.18 – Illustration of the data flow through the comparer

macro precision, recall, F1, and ROC–AUC together with training time, per split inference time, model size and the number of effective features seen at fit. Visual outputs include side by side confusion matrices, ROC and precision–recall curves and feature-importance charts when available. Both the raw and preprocessed models can be exported as PKL artifacts.

User visualization and feedback

The Streamlit view presents a compact KPI strip for training time, model size, and feature count for both routes with their corresponding deltas. A comparison table summarizes accuracy, macro F1, precision, recall, ROC–AUC, and inference time for train, validation, and test, and annotates each cell with the absolute change in percentage points or relative percent change as appropriate. Confusion matrices, ROC and precision–recall curves are rendered side by side for direct inspection and feature-importance bars appear in parallel to surface ranking changes that can inform subsequent feature engineering.

Reproducibility considerations

Both routes are executed with identical data splits, fixed random seeds, and the exact same hyperparameters chosen in the training UI.

3.3.8 Reporter

Motivation and role in the framework

Strong results only matter when they can be audited, reproduced and communicated without ambiguity. In cybersecurity machine learning, that means preserving not just scores and plots but the full context that produced them, like dataset lineage, validation outcomes, split logic, preprocessing decisions, training settings, and model artifacts. The Reporting and Export Module in A.R.C. does exactly that. It turns a complex experimental run into a durable and human readable record while keeping every machine readable artifact available for downstream reuse.

Implementation design

The module is implemented in two layers that mirror how A.R.C. itself is organized. The export layer serializes all intermediate and final artifacts in stable formats, data subsets are written to CSV, trained estimators are persisted as PKL. The report layer composes the assets from A.R.C. into a structured PDF built with ReportLab. The document opens with a compact dataset overview and proceeds through schema validation (duplicates, sparsity, granularity, feature types with vector pie charts), label validation (class balance, rare/dominant classes, spelling consistency, and temporal drift via timelines and time binned distributions), split diagnostics (method, proportions, unseen label checks and per-split label distributions with stacked bars) and model evaluation (general KPIs for train time, model size, and feature count, split wise metrics, confusion matrices, ROC and PR curves and feature importance rendered as crisp vector graphics).

Because reproducibility depends on surfacing the “why” behind preprocessing, the report explicitly documents the preprocessing pipeline used in training. It lists each step in order, describes its purpose in plain terms, and records all variables with the values actually applied together with their default settings at the time of the run. For example, imputation strategies, one hot encoding bounds, numeric scaling flags, categorical handling for GBMs, downcasting rules, and any ID like column removal are all captured next to a short rationale.

Inputs and outputs

The module ingests everything produced by previous stages through A.R.C.’s session state, the raw and preprocessed datasets, schema and label validation metadata, split products with their checks, preprocessing configuration and fitted transformers, trained models, and evaluation results for both the preprocessed and minimal preprocessing routes. It emits CSV files for train, validation, and test, JSON bundles for each stage’s parameters, defaults, and computed summaries, PKL files for trained

estimators and a single PDF that consolidates text, tables, and vector plots. Where available, LaTeX snippets used elsewhere in A.R.C. (for example, top 25 feature importance figures) are also included into the report.

User visualization and feedback

From the Streamlit UI, the researcher selects which sections to include and triggers report compilation with one action. The same page exposes direct download buttons for individual artifacts, like CSV datasets, JSON metadata, and PKL models so users can retrieve exactly what they need without regenerating the full document. Inside the PDF, KPIs and metric tables are formatted to mirror the application, deltas are expressed in percentage points for interpretability, significance annotations appear next to their corresponding metrics when computed, and all plots are embedded as vector drawings to remain legible at print resolution.

Reproducibility considerations

Every element that could influence results is recorded. Random seeds, library backends for GBMs, model hyperparameters, preprocessing variables and their defaults, class weights or sample weights, and timing or hardware notes are written into the JSON sidecars and summarized in the PDF. Data splits and preprocessed tables can be exported. Trained models are stored as PKL. With these provisions, reproducing a past experiment is a matter of restoring the artifacts and re executing the training script, not rediscovering the undocumented setup.

Chapter 4

Experiment

4.1 Scope and goals

This chapter describes and motivates a focused case study that exercises A.R.C. end to end on two representative datasets. The study has four goals. First, to characterize data readiness through structured insights produced by the Schema Validator, with an emphasis on duplicates, sparsity, feature typing, and temporal integrity. Second, to examine the label space with the Label Validator, quantifying class balance, rare or dominant classes, and spelling consistency, while probing for temporal drift in the target distribution. Third, to assess the behavior of the splitter under explicit quality checks by materializing all three partitioning regimes (random, stratified, and time based) and auditing each for preservation of class proportions, unseen labels, and temporal coverage. Fourth, to quantify the marginal effect of standardized preprocessing on downstream model quality and efficiency when training five baseline learners under a stratified split. The overarching goal is to show, in a traceable and reproducible manner, how A.R.C. converts heterogeneous cybersecurity datasets into auditable inputs for fair model evaluation, and to measure what is gained, if anything, by adopting its standardized preprocessing catalogue.

On the methodological side, the objective is to validate that A.R.C.'s instrumentation provides an adequate and repeatable basis for empirical study. On the empirical side, the objective is to estimate the effect of standardized preprocessing on predictive metrics and computational cost across datasets that differ in cleanliness and balance.

4.2 Research questions and hypotheses

Three research questions guide the study. The first asks whether standardized preprocessing, as realized by A.R.C., improves detection quality relative to a minimal preprocessing baseline that is just sufficient to make each learner trainable. The second asks whether standardized preprocessing reduces computational cost as measured by train time, per sample inference latency and serialized model size. The third asks whether the size and direction of any gains depend on inherent dataset quality. From these questions, five hypotheses are stated a priori.

Four research questions guide the study. The first asks what the Schema Validator reveals about the inner readiness of each dataset and whether the reputed cleanliness of DNP3 is sustained relative to CIC-IIoT. The second asks what the Label Validator uncovers about class balance, rare and dominant classes, spelling consistency, and temporal drift, and how these properties motivate macro-averaged metrics over accuracy. The third asks how split choice (random, stratified, or time based) changes the audited data regime, when the same dataset is partitioned under different rules, how do per class proportions, unseen labels in validation and test, and temporal coverage differ, and which regime yields the most reliable conditions for supervised learning. The fourth asks whether standardized preprocessing improves predictive quality and operational footprint for the baseline learners and whether the magnitude of gains depends on dataset characteristics.

4.3 Datasets

Two datasets are used to cover distinct regimes of data quality, as seen in table 4.1. The DNP3 dataset (2022) is deliberately balanced and nearly free of schema defects (figure 3.2a), it contains 7.326 records, ten attack types, 86 features, and a small footprint of about 4.4 MB in the harmonized CSV export. The CIC-IIoT dataset (2025) is representative of a normal industrial IoT intrusion dataset, it is moderatly imbalanced (figure 3.2b), contains 227.191 records, seven attack types, eighty-seven features, and a size of roughly 820.5 MB. Both are synthetic in origin and are treated as such for external validity.

Table 4.1 – Overview of datasets used in the experiments

Datasetname	Year	Records	Attacktypes	Disk size	Features	Synthetic
DNP3 [6]	2022	7.326	10	4,4 MB	86	Yes
CIC-IIoT [5]	2025	227.191	7	820,5 MB	87	Yes

4.4 Compute environment and reproducibility

All experiments run on a MacBook Air (2024) equipped with an Apple M3 (eight CPU cores, ten GPU cores, sixteen-core neural engine) and 16,GB of unified memory, backed by a 512,GB SSD and macOS Tahoe 26.0. The framework fixes random seeds in all stochastic components and exports fitted models, configuration JSON files and consolidated PDF reports.

4.5 Implementation design

The experiment is orchestrated entirely inside A.R.C.'s Streamlit application. Each study is an interaction pattern over existing modules. The schema validator is invoked to produce dataset level diagnostics, which are embedded in the report. The label validator is run next, computing class counts, identifying inconsistent spelling, and generating temporal analyses either as a timeline of label segments or as time binned distributions by timestamp or by record count, these are rendered as charts in the report. The Split Module partitions data into train, validation, and test using a 60/20/20 ratio and stratification by label; it runs quality checks for unseen labels and summarizes per split label distributions. The Training and Evaluation Module trains one model per learner family on the preprocessed dataset and for comparison a matched model on minimally preprocessed data as defined in the comparer. Metrics are computed on all three splits and recorded along with general KPIs such as train time, model size, and feature count. The Compare Module aligns RAW (minimal preprocessed) and PP (preprocessed trough A.R.C) results and computes paired deltas. Finally, the Reporting Module assembles the artifacts into a single PDF and exposes download buttons for every intermediate product.

4.6 Experimental design and procedures

The study proceeds in four stages that mirror the research questions. The first stage, schema insights, applies the Schema Validator to each dataset and records total records, duplicate rates, percentage of missing values overall and by column, feature inventory by type, detection of mixed type columns and temporal integrity indicators such as sort order, coverage, and typical interval. The second stage, label insights, runs the Label Validator to obtain class counts and proportions, identifies rare and dominant classes, detects labeling inconsistencies and renders timelines and time binned distributions where timestamps are present to reveal distributional shift. The third stage, splitter insights with quality checks, constructs random, stratified, and time based 60/20/20 partitions for each dataset, verifies preservation of class

proportions, searches for unseen labels in validation and test relative to the training split and summarizes temporal coverage; because DNP3 is already balanced, the expectation is that stratification makes little difference to proportions, while for CIC-IIoT the stratifier stabilizes minority representation and reduces unseen label risk and the time-based split surfaces the strongest drift. The fourth stage, impact of standardized preprocessing under stratification, trains the five baseline learners (Random Forest, Gradient Boosting with CatBoost, Support Vector Machine, Multi Layer Perceptron and Logistic Regression) on both RAW and PP versions of each dataset using the stratified partition. Hyperparameters follow A.R.C.'s documented defaults to isolate the marginal effect of preprocessing. For each learner–dataset combination, A.R.C. measures accuracy, macro–F1, macro–Precision, macro–Recall, and ROC–AUC on train, validation, and test; measures per-sample inference latency; and records training time, serialized model size, and the effective feature count after preprocessing. The Compare Module reports the absolute change $\Delta = \mathrm{PP} - \mathrm{RAW}$ for metrics and signed percentage changes for latency and size, and renders side-by-side confusion matrices, ROC curves, precision–recall curves, and top-25 feature-importance charts to make differences interpretable.

4.7 Operational definitions and measurements

Dependent variables comprise the predictive metrics and efficiency indicators just listed. Accuracy is defined as the fraction of correct predictions on the split in question. F1 macro, precision macro, and recall macro are computed as unweighted arithmetic means over classes. ROC–AUC is computed as a one vs. rest macro average for multiclass problems. Inference latency is measured as median per sample time on the CPU. Training time is measured with a monotonic clock. Model size is the byte size of the serialized estimator written with the highest available pickle protocol. Effective feature count is the number of columns entering the learner after preprocessing. All metrics are computed on train, validation, and test. Summary tables in the report present triplets of the form Raw|Pre|Δ for each KPI so that the marginal value of preprocessing is visible at a glance.

4.8 Link to standardized preprocessing

Preprocessing adheres to the catalogue defined in A.R.C. and recorded in the report. Numeric columns are downcasted and stabilized, infinities are mapped to missing, missing numerics are imputed by median and categoricals by the most frequent value, categorical explosion is bounded during one hot encoding, scale-sensitive learners receive standardized numerics, tree learners operate on compacted numeric

and sparsely encoded categorical views and when applicable, GBMs honor native categorical handling. Every variable governing these steps has a name, a default, and a realized value stored in the run metadata. The DNP3 dataset, which is already balanced and schema clean, is expected to benefit little beyond small efficiency wins. The IIoT corpus, which is more typical, is expected to show non trivial macro metric gains and clearer error profiles after standardization.

4.9 Threats to validity and limitations

Two limitations are acknowledged up front. First, both datasets are synthetic; external validity to live industrial networks must therefore be argued cautiously. Second, to isolate the marginal effect of standardized preprocessing, learners are kept at documented defaults and no hyperparameter search is performed; absolute scores should not be read as limits of model families but as a controlled comparison between preparation regimes. These constraints are deliberate and are mitigated by complete logging, consistent splits, and paired evaluation.

4.10 Expected outcomes

If the hypotheses hold, DNP3 will pass schema and label checks with near-zero findings and will show small or negligible Δ on macro metrics under stratified training, with minor reductions in model size and inference time. CIC-IIoT will surface substantive issues in the validators that map directly to the standardized preprocessing steps, and the corresponding PER runs will improve macro–F1 and related metrics with supported deltas, while also producing faster or comparably fast models. Across split regimes, stratification will preserve class proportions and prevent unseen labels in validation and test; random splits will show higher variance in minority coverage; time based splits will depress absolute scores due to drift but still reflect the same qualitative PP advantages. Regardless of effect size, the experiment will have demonstrated that A.R.C. delivers a transparent, repeatable path from raw cybersecurity data to auditable model evaluation, with an explicit paper trail that supports both scientific comparison and operational deployment.

Chapter 5

Case Study and Results

5.1 Data diagnostics

5.1.1 Schema validator insights

The schema validator module characterizes each dataset structurally and temporally before any modeling. Table 5.1 reports record counts, duplicate rates, feature inventories, and timestamp properties for DNP3 and CIC-IIoT. This overview clarifies how inherent traits, such as duplication, feature composition and time coverage, may surface later in both results and their interpretation.

Table 5.1 – Overview insights after schema validation

Category	Schema insight	DNP3	CIC-IIoT
Records	Total records	7.326	227.191
	Duplicate records	540 (7,37%)	36 (0,02%)
Missing Values	Percentage of total	0 (0,00%)	0 (0,00%)
	Percentage of features	None	None
Features	Number of features	86	87
	Duplicated features	Yes	Yes
	Features with mixed type	None	None
	Feature type	Numerical: 81	Numerical: 71
		Object: 4	Object: 14
		Datetime: 1	Datetime: 1
Time features	Duplicated timestamps	Yes	Yes
	Sorted timestamps	Yes	Yes
	Start time	14.05.2020 23:00:00	15.01.2025 13:04:54
	End time	19.05.2020 15:18:54	09.09.2025 15:09:39
	Duration	4 days 15:18:54	237 days 02:04:45
	Most common interval	0 days 00:00:00	0 days 00:00:00
	Avarage interval	0 days 00:00:55	0 days 00:01:30

Key observations

Table 5.1 shows that both dataasets are clean in terms of missingness but differ materially in duplication, feature composition, and temporal coverage. DNP3 contains a non trivial share of duplicate records (540, 7.37%), whereas CIC-IIoT is virtually free of duplicates (36, 0.02%). Neither dataset exhibits missing values or mixed type columns, which simplifies downstream imputation. Both include duplicated features that should be collapsed before modeling. DNP3 is predominantly numeric (81 of 86 features) with few object typed columns, while CIC-IIoT carries a larger categorical footprint (14 object features of 87), foreshadowing heavier encoding. Timestamp series are present, sorted and dense in both datasets, with the most frequent interval equal to zero, which means many records share identical timestamps. Yet, their durations differ strongly, DNP3 spans roughly five days, while CIC-IIoT spans about eight months. The average interval is shorter for DNP3 (55,s) than for CIC-IIoT (90,s).

Discussion

These structural traits have direct implications for the rest of the pipeline and for interpreting later results. The duplicate rate in DNP3 is large enough to bias estimates, if duplicates straddle train, validation and test boundaries, inflating apparent generalization through near repeat observations. Removing or consolidating duplicates is therefore not a cosmetic step but a prerequisite for fair evaluation. In contrast, the negligible duplicate rate in CIC-IIoT reduces the risk of such leakage and means that any improvements observed after preprocessing are less likely to be confounded by de duplication alone.

The absence of missing values in both datasets suggests that imputation will not drive improvements, any performance or efficiency gains should instead be attributed to stabilization of numeric scales, categorical handling, and removal of redundant features. The presence of duplicated features in both datasets signals potential multicollinearity and unnecessary dimensionality. Pruning these redundancies can reduce model size and training time and for linear or margin based learners, mitigate coefficient instability. Because CIC-IIoT carries substantially more object typed columns than DNP3, its preprocessing will devote more effort to encoding. Without careful bounds on category cardinality, this would expand the feature space and slow inference, with standardized one hot encoding and capped category growth, the catalog keeps that expansion controlled. This asymmetry predicts a larger marginal benefit of the standardized preprocessing on CIC-IIoT than on DNP3, not because DNP3 is “too small,” but because CIC-IIoT presents more opportunities for consistent treatment of heterogeneous types.

The temporal diagnostics point to two distinct regimes. DNP3's short, dense horizon increases the chance that nearly identical events occur close together in time. If a random or stratified splitter is used without a temporal gap, these bursts can spill across splits and again overstate generalization. CIC-IIoT's long horizon makes it a better substrate for studying temporal drift, but the fact that the most common interval is zero in both datasets indicates event driven logging with concurrent records, which complicates time based partitioning and can introduce unseen labels or distributional shifts late in the timeline. In practice, these findings justify the study's emphasis on stratified splits for modeling, while using the splitter diagnostics to compare how each regime would behave with respect to class preservation and leakage risk.

Taken together, the schema validator's report explains why DNP3 is expected to show small, efficiency oriented gains after preprocessing, primarily from deduplication and feature deduplication, while CIC-IIoT is poised for more noticeable improvements due to its richer categorical structure and longer, more variable temporal footprint. Most importantly, the analysis provides a defensible link between dataset anatomy and downstream outcomes, ensuring that any observed improvements can be traced back to concrete, auditable properties rather than to opaque tuning or chance.

5.1.2 Label validator insights

The label validator summarizes class quality and temporal behavior before modeling. Table 5.2 reports missingness, spelling consistency, entropy, imbalance level, per-class proportions, and whether rare or dominant classes are present for DNP3 and CIC-IIoT. The time-binned distributions in figures 5.1 and 5.2 visualize how class mix evolves over the respective capture periods.

Table 5.2 – Overview of insights after label validation

Label insight	**DNP3**	**CIC-IIoT**
Missing labels	None	None
Consistent spelling	Yes	Yes
Entropy	3,46	1,90
Imbalance	Low	Moderate
Distribution	APP_POISONING: 9,09% COLD_RESTART: 9,09% DISABLE_UNSOLICITED: 9,09% DNP3_ENUMERATE: 9,09% DNP3_INFO: 9,09% INT_DATA: 9,09% MITM_DOS: 9,09% NORMAL: 9,9% REPLAY: 9,9% STOP_APP: 9,09% WARM_RESTART: 9,09%	benign: 60,20% bruteforce: 0,82% ddos: 8,11% maleware: 3,32% mitm: 3,55% recon: 14,80% web: 1,23%
Rare classes	None	brutforce (0,82%)
Dominant classed	None	None
Temporal drift	Figure 5.1	Figure 5.2

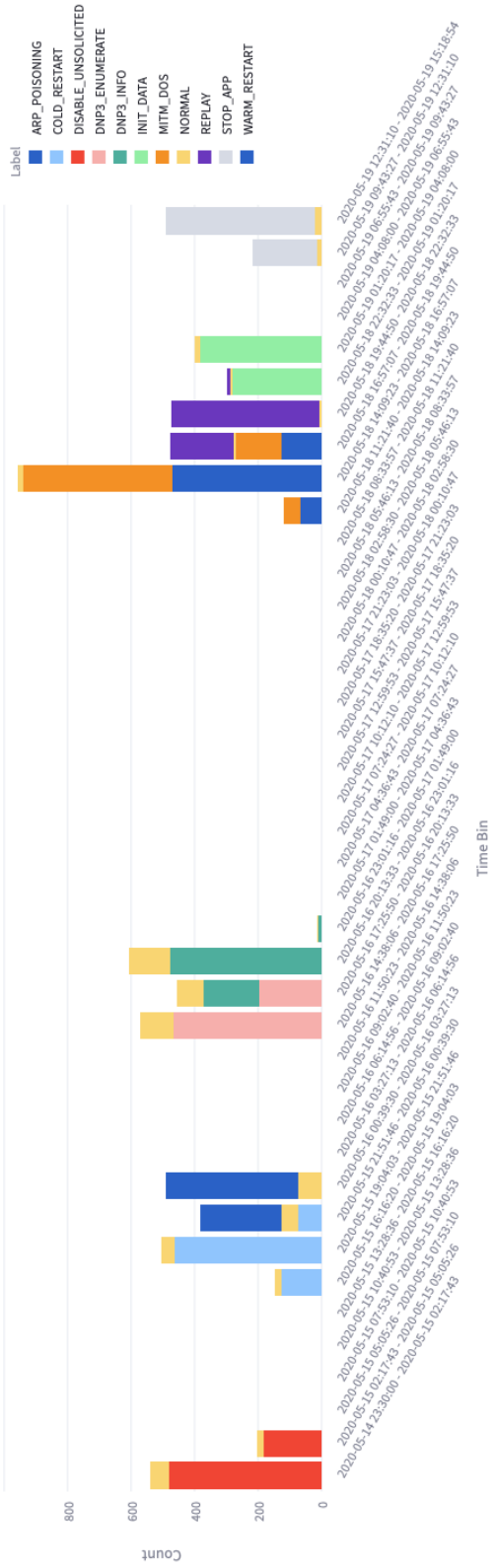


Figure 5.1 – Illustration of the DNP3 label distributions across 40 time bins

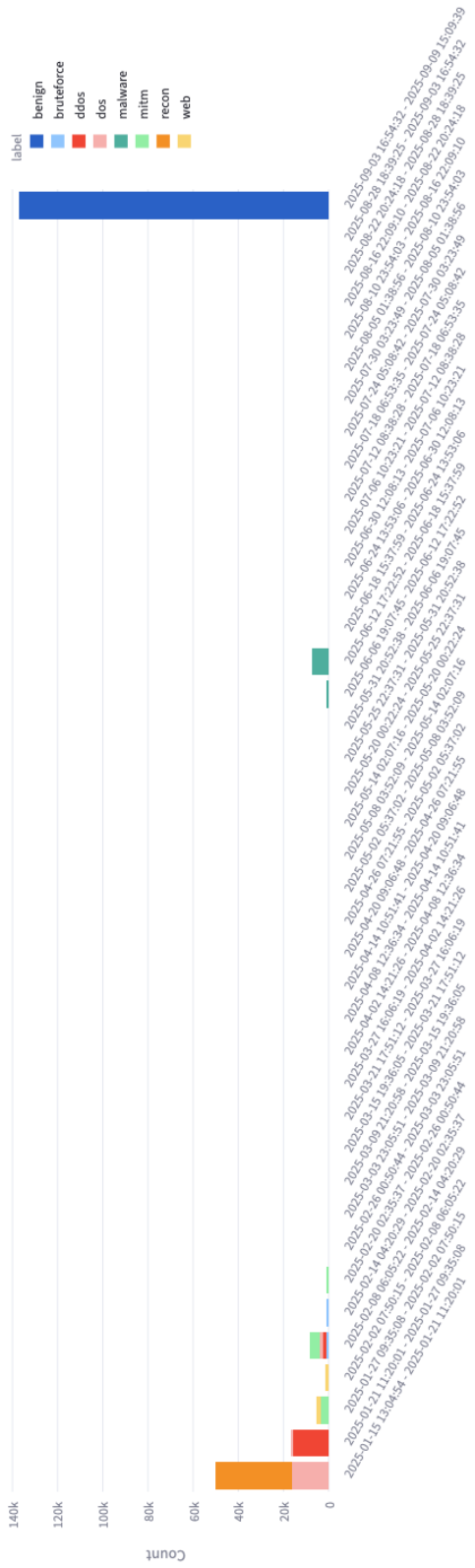


Figure 5.2 – Illustration of the CIC-IIoT label distributions across 40 time bins

Key observations from the Label Validator

Both datasets pass the basic integrity checks, no missing labels are detected and label tokens are used consistently throughout. Beyond this, their label spaces differ strongly. DNP3 exhibits a near uniform class mix over its eleven categories, with an entropy of 3.46 that is essentially at the theoretical maximum for eleven classes. Each class contributes roughly 9% of the observations and neither rare nor dominant classes are present. CIC-IIoT is markedly more concentrated, with a entropy of 1.90. A majority of the samples are benign (60,20%), two attack families (recon and ddos) account for most of the remaining mass, and one class (bruteforce) falls below one percent and is therefore flagged as rare. The time binned views mirror this contrast (figure 5.1 and 5.2). DNP3's short capture shows contiguous and scenario style blocks per label, including a long NORMAL segment, interspersed with short bursts of specific attack classes. CIC-IIoT unfolds over many months in a non stationary manner, early windows are attack heavy across several classes, there are long intervals with little activity, and the last window is dominated by a very large benign tranche.

Discussion

The even label distribution in DNP3 implies that macro averaged metrics will be stable and that the evaluation will be insensitive to the particular random seed used for stratification. Because labels occur in contiguous temporal blocks, however, purely random partitioning could still leak near duplicate situations across splits. In this dataset a time based split would be the more conservative choice if the goal were to test generalization across scenarios rather than within them. The absence of rare or dominant classes also means that class rebalancing steps are unnecessary and that any improvement observed later comes primarily from standardized cleaning, encoding, and scaling rather than from redistribution of training examples.

CIC-IIoT presents a different set of risks and expectations. Moderate imbalance and a pronounced majority class will inflate accuracy and micro averaged measures, which motivates the use of F1 macro, precision macro, and recall macro as primary scores. The presence of a rare class at 0.82% creates vulnerability to under representation in naively sampled validation and test partitions. Stratification is therefore essential simply to obtain stable estimates. The temporal profile suggests substantial drift in class priors and potentially in feature distributions, most attacks cluster early, while the late period is overwhelmingly benign. Under a time based split, this will depress absolute scores relative to random or stratified splits because the model is asked to extrapolate across different operating regimes, it also makes false negative risk particularly salient if late period attacks are sparse. These characteristics explain, in advance, why efficiency quality trade offs and error modes will differ between

the datasets. DNP3 should perform well in uniformly distributed confusion matrices with few systematic confusions, whereas CIC-IIoT should show class specific recall challenges, especially for the rare bruteforce class and for attacks that appear mainly in early windows.

Taken together, the validator's findings justify the design choices in the remainder of the study. They support the emphasis on macro averaged metrics, the use of stratified splitting for the performance experiments and a careful interpretation of any time based comparisons as measures of robustness under drift rather than of raw discriminative power. They also set realistic expectations, minimal gains on DNP3, where labels are already well balanced and larger, class specific improvements on CIC-IIoT, where imbalance and temporal non stationarity are the dominant challenges.

5.1.3 Split insights

The split module has three regimes (random, stratified, and time based split) and records set sizes together with label coverage. Table 5.3 summarizes the partitions for DNP3 and CIC-IIoT. Interpreting these splits in light of the earlier label analysis is essential. DNP3 has a near-uniform class distribution, so random and stratified splits behave similarly. CIC-IIoT is dominated by benign traffic with minority attacks. Stratification preserves proportions that random sampling might distort, while the time-based split reflects the dataset's bursty campaign windows and long benign stretches. The labels mix for the time-based regime, as shown in figure 5.5 and 5.6. For CIC-IIoT, confirms the stratified plot, seen in figure 5.3 and 5.3, the proportion preservation.

Table 5.3 – Overview of splits and label coverage across strategies

Category	Label insight	DNP3	CIC-IIoT
Random split	Training set	4.395	136.314
	Validation set	1465	45438
	Test set	1466	45439
	Unseen labels	None	None
Stratified split	Training set	4395	136314
	Validation set	1465	45438
	Test set	1466	45439
	Unseen labels	None	None
Time-based split	Gap ratio	0,0	0,0
	Training set	4395	136314
	Validation set	1465	45438
	Test set	1466	45439
	Unseen in train	INIT_DATA	
		REPLAY	
		STOP_APP	
	Unseen in validation	COLD_RESTART	bruteforce
		DISABLE_UNSOLICITED	ddos
		DNP3_ENUMERATE	dos
		DNP3_INFO	malware
		STOP_APP	mitm
		WARM_RESTART	recon
			web
	Unseen in test	APP_POISOMING	bruteforce
		COLD_RESTART	ddos
		DISABLE_UNSOLICITED	dos
		DNP3_ENUMERATE	malware
		DNP3_INFO	mitm
		MITM_DOS	web
		WARM_RESTART	

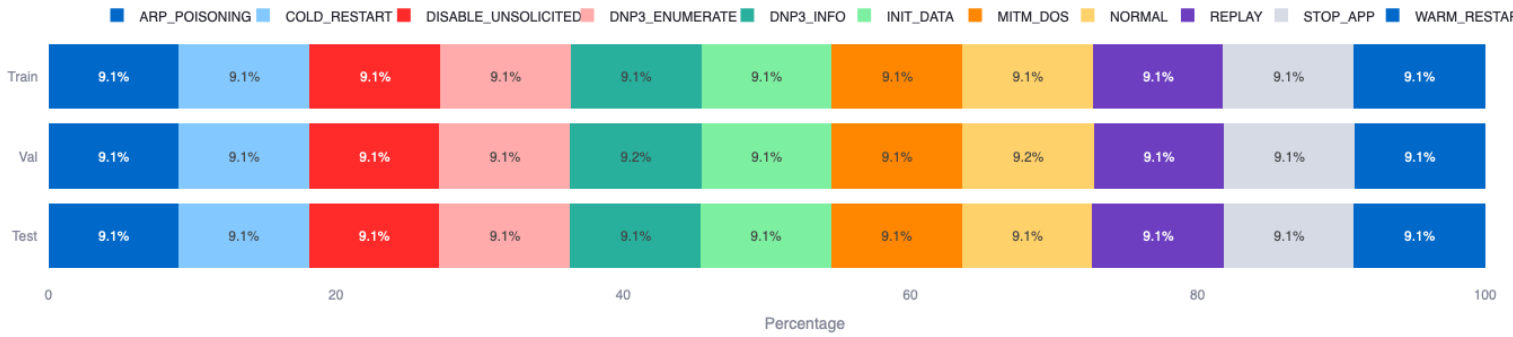


Figure 5.3 – Illustration of the DNP3 stratified split distribution (60/20/20)

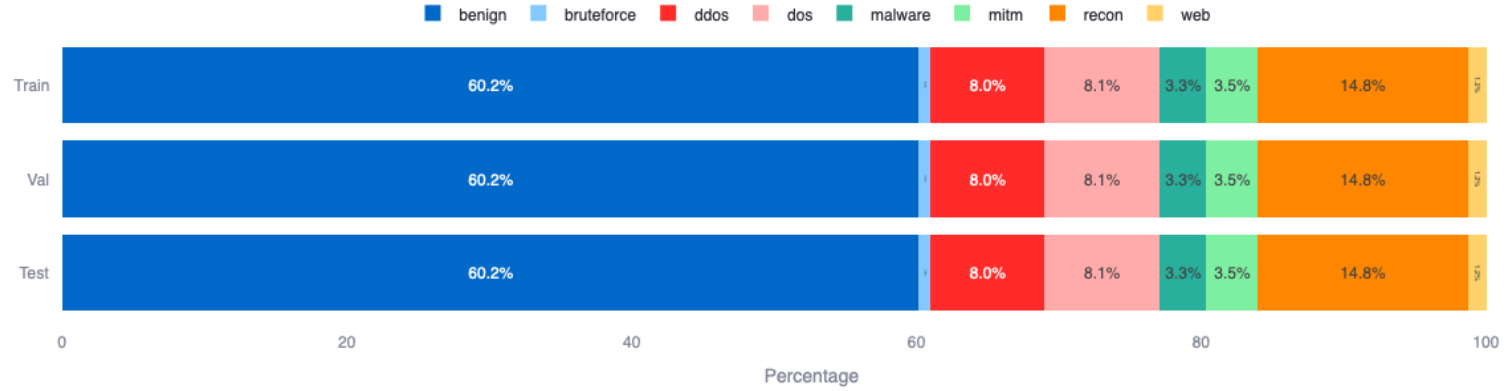


Figure 5.4 – Illustration of the CIC-IIoT stratified split distribution (60/20/20)

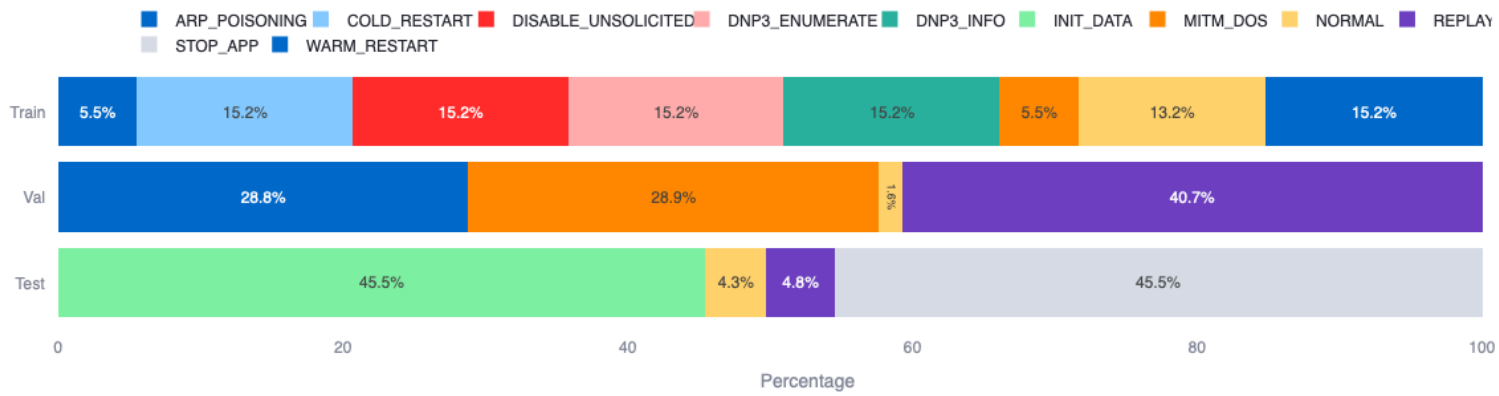


Figure 5.5 – Illustration of the DNP3 time based split distribution (60/20/20) with 0 gap

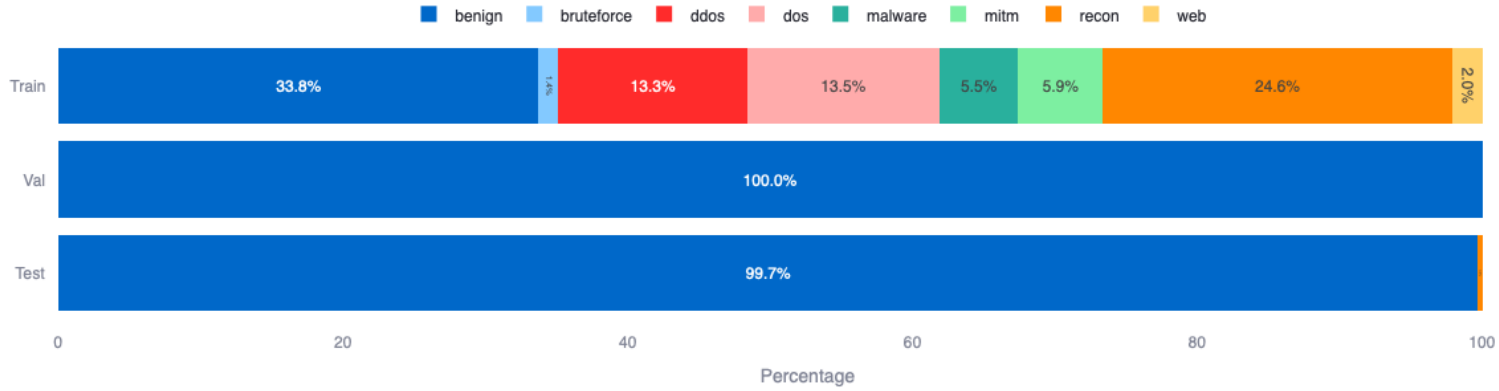


Figure 5.6 – Illustration of the CIC-IIoT time based split distribution (60/20/20) with 0 gap

Key observations

Running the three partitioning regimes confirms that the splitter diagnostics are doing exactly what they are supposed to do, surfacing class coverage problems before any modeling. Under the stratified regime, both datasets preserve per class proportions across train, validation, and test and no unseen labels appear. This is visible in the DNP3 plot, where every class contributes 9.1% on each split and in CIC-IIoT, where the splits mirror the global mix (60% benign with the remaining classes in the 0.8–14.8% range). Random splits also show no unseen labels in this run, because DNP3 is uniformly balanced and its random partitions remain close to equal, whereas IIoT exhibits the expected small fluctuations in minority coverage.

Time based splitting presents the problems recognized in the label validator. In DNP3, classes are temporally clustered: INIT_DATA, REPLAY, and STOP_APP are absent from the training window, while several other classes vanish from validation or test. The distribution bars confirm that different days are dominated by different attack types. In CIC-IIoT the effect is more extreme, the late window is almost entirely benign, producing a validation set that is 100% benign and a test set that is 99.7% benign, with most attack classes absent from these subsets. The table also records a gap ratio of 0.0, so there is no temporal buffer.

Discussion

For model selection and headline reporting on these datasets, stratified splitting is the only regime that results evaluable multi class metrics with stable variance. It guarantees class presence in every split, keeps macro averaged scores meaningful, and avoids accidental optimism or pessimism from class drop outs. Random splitting is acceptable for already balanced datasets like DNP3 but remains fragile on imbalanced data such as CIC-IIoT, where minority representation can drift noticeably between folds.

The time based results are informative but cautionary. Operationally, blocking by time is attractive because it mimics deployment, yet in both datasets the capture protocol concentrates classes into narrow windows. When whole classes are missing from train or evaluation, multi class learning and scoring become impossible, macro metrics collapse (undefined AP/AUC for missing positives, trivial accuracy inflated by a single class) and any comparison of preprocessing or models is confounded by label absence rather than algorithmic merit. Introducing a non zero temporal gap would also reduce leakage from near duplicates without materially worsening coverage when classes are sufficiently spread out.

Overall, the splitter diagnostics validate the study design, A.R.C. surfaces when a split is statistically usable and when it is not. For DNP3 and CIC-IIoT as provided, stratification produces fair, reproducible estimates, random is tolerable only on

the balanced datasets, and time based splitting, absent additional constraints, is unsuitable for multi class evaluation because it systematically removes classes from one or more partitions.

5.2 Impact analyses

5.2.1 Random forest (RF)

The random forest study quantifies the effect of A.R.C.'s standardized preprocessing under stratified splitting. Table 5.4 reports efficiency footprint (training time, serialized model size and effective feature count) for RAW and PP with percentage deltas. Table 5.5 reports accuracy, F1, precision, recall, ROC–AUC, and per split inference time for Train/Validation/Test, with paired differences (Δ). The accompanying figures visualize the test split behavior, precision–recall curves by class (figure 5.9), normalized confusion matrices (figure 5.8), and top-25 feature-importance rankings (figures 5.7 and 5.10) for RAW and PP, making the tabulated deltas and remaining error modes directly interpretable.

Table 5.4 – Overview of resource footprint of RF before (RAW) and after A.R.C.'s standardized preprocessing (PP) under a 60/20/20 stratified split

KPI	DNP3			CIC-IIoT		
	Raw	PP	Δ	Raw	PP	Δ
Training time	0,335 s	0.284 s	-15,50 %	14.606 s	5,137 s	-64,83 %
Model size	4,22 MB	2,44 MB	-42,08 %	411,90 MB s	18,66 MB	-94,47 %
Feature count	85	55	-35,29 %	86	57	-33,72 %

Table 5.5 – Overview of predictive performance of RF with (PP) and without (RAW) preprocessing on train, validation and test under a 60/20/20 stratified split

KPI	Split	DNP3			CIC-IIoT		
		RAW	PP	Δ	RAW	PP	Δ
Accuracy	Train	99,82 %	99,68 %	-0,13 pp	95,22 %	99,72 %	+4,50 pp
	Validation	99,18 %	99,36 %	+0,18 pp	94,57 %	99,56 %	+4,99 pp
	Test	99,59 %	99,79 %	+0,20 pp	94,70 %	99,59 %	+4,90 pp
F1	Train	99,82 %	99,54 %	-0,27 pp	93,00 %	99,33 %	+6,34 pp
	Validation	99,18 %	99,30 %	+0,12 pp	90,83 %	98,98 %	+8,15 pp
	Test	99,59 %	99,80 %	+0,20 pp	91,08 %	99,10 %	+8,02 pp
Precision	Train	99,82 %	99,40 %	-0,42 pp	98,39 %	99,15 %	+0,76 pp
	Validation	99,21 %	99,24 %	+0,04 pp	97,29 %	98,72 %	+1,42 pp
	Test	99,60 %	99,80 %	+0,21 pp	97,11 %	98,87 %	+1,76 pp
Recall	Train	99,82 %	99,70 %	-0,11 pp	88,61 %	99,53 %	+10,92 pp
	Validation	99,18 %	99,39 %	+0,21 pp	86,13 %	99,26 %	+13,13 pp
	Test	99,59 %	99,79 %	+0,20 pp	86,56 %	99,34 %	+12,77 pp
ROC-AUC	Train	100,00 %	100,00 %	0,00 pp	99,98 %	100,00 %	+0,02 pp
	Validation	100,00 %	100,00 %	0,00 pp	99,92 %	99,99 %	+0,07 pp
	Test	99,99 %	99,99 %	0,00 pp	99,91 %	99,99 %	+0,08 pp
Inference time	Train	0,0326 s	0,0159 s	-51,4 %	1,2016 s	0,4572 s	-62,0 %
	Validation	0,0175 s	0,0066 s	-62,0 %	0,5169 s	0,1555 s	-69,9 %
	Test	0,0171 s	0,0063 s	-62,9 %	0,5278 s	0,1551 s	-70,6 %

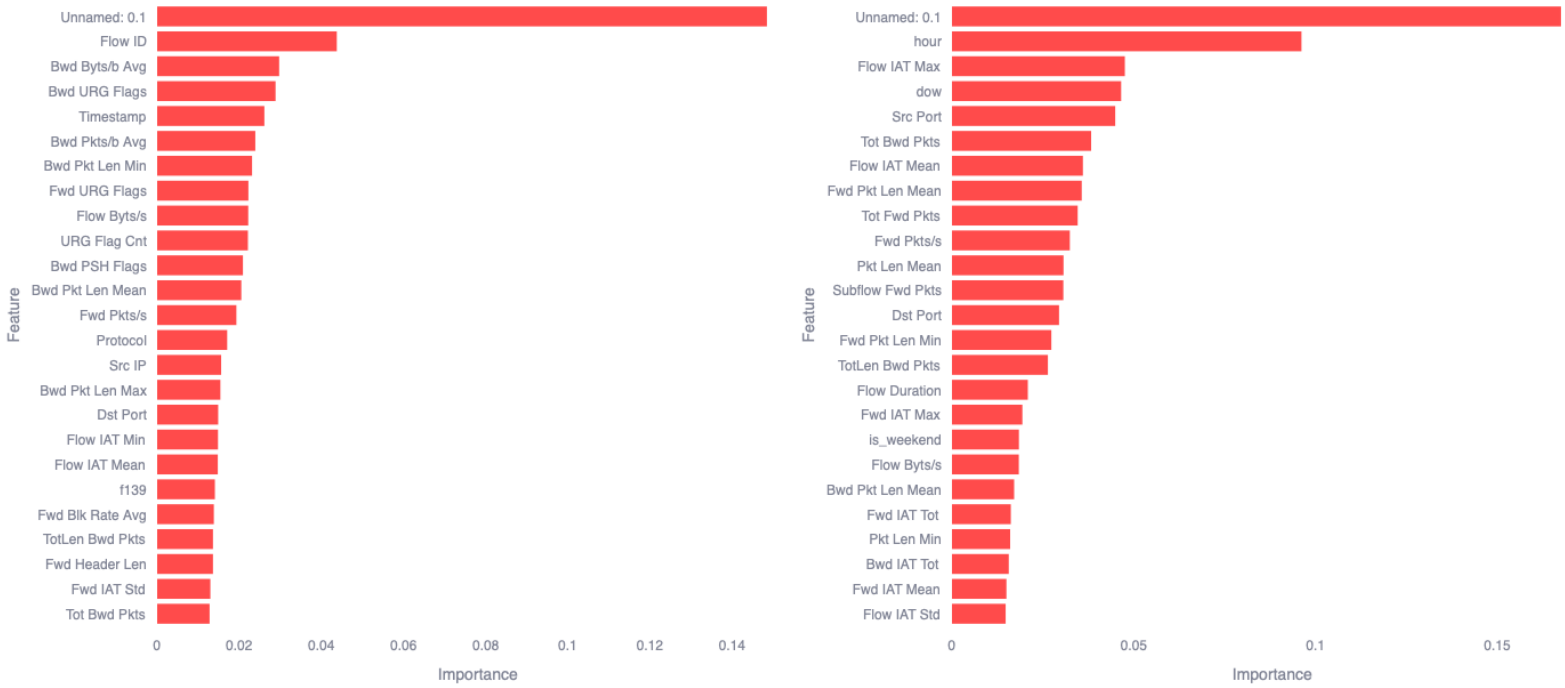


Figure 5.7 – Illustration of the top 25 most important features comparison between RAW (left) and PP (right) for RF on DNP3 under stratified splitting (60/20/20)

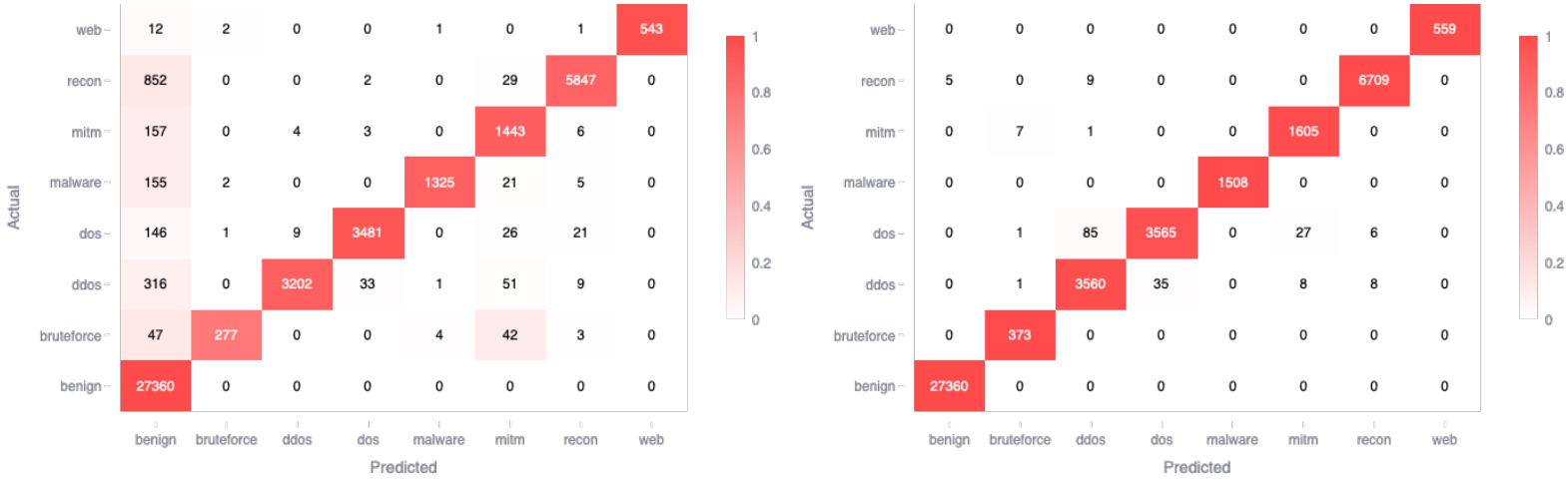


Figure 5.8 – Illustration of the confusion matrix comparison between RAW (left) and PP (right) for RF on CIC-IIoT under stratified splitting (60/20/20)

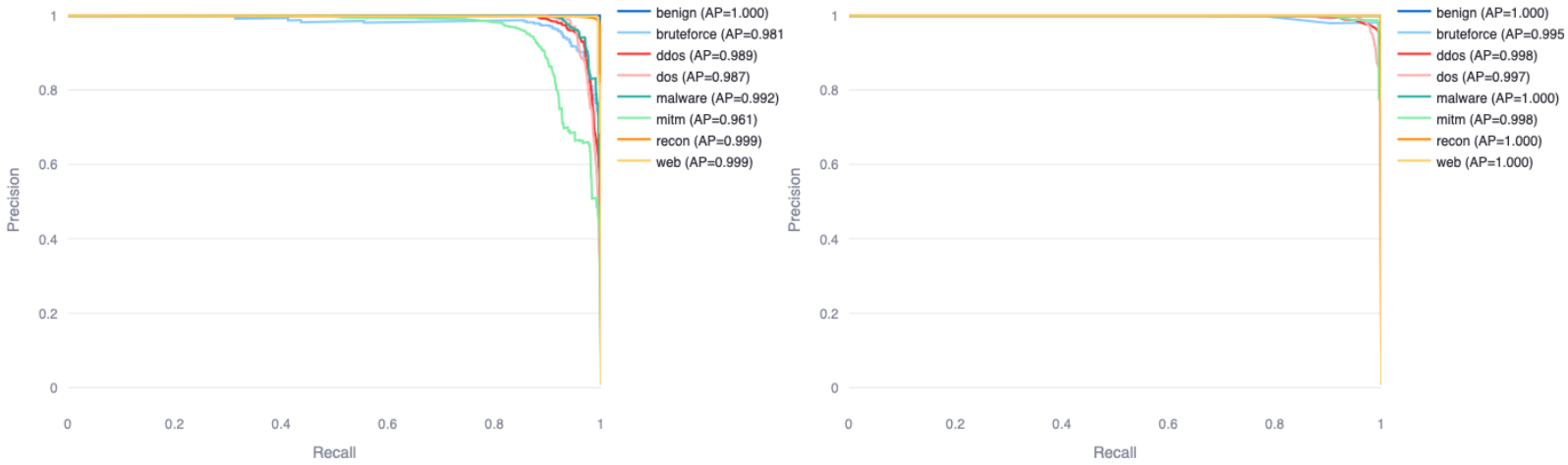


Figure 5.9 – Illustration of the PR curve comparison between RAW (left) and PP (right) for FR on CIC-IIoT under stratified splitting (60/20/20)

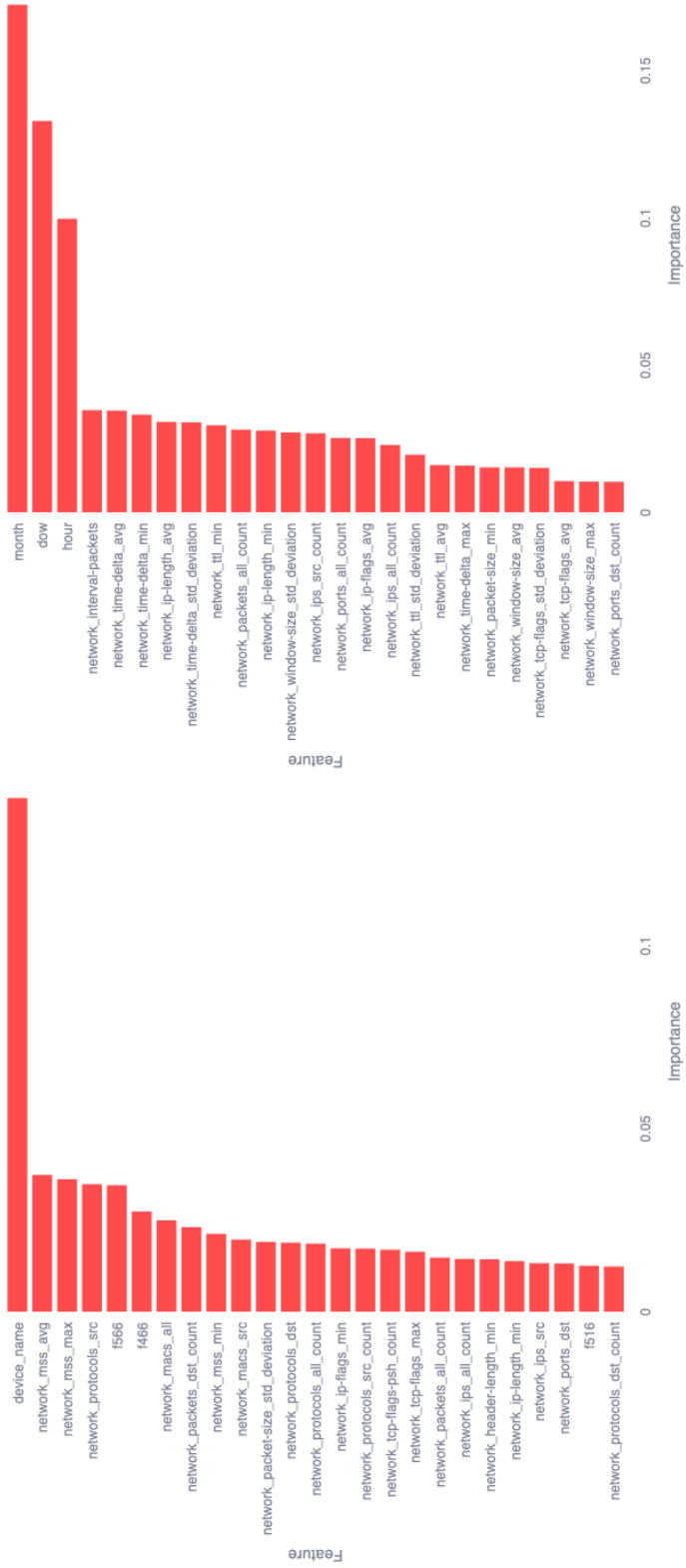


Figure 5.10 – Illustration of the top 25 most important features comparison between RAW (left) and PP (right) for RF on CIC-IIoT under stratified splitting (60/20/20)

Key observations

Across both datasets, standardized preprocessing produces large efficiency gains and on the noisier IIoT dataset, dramatic improvements in accuracy. Training time, model size and feature count are reduced in both cases, the reduction is moderate on DNP3 and high on IIoT. Predictive metrics on DNP3 change only slightly, while CIC-IIoT shows double digit percentage point gains in recall and F1 on every split. PR curves and confusion matrices corroborate these deltas, for CIC-IIoT the PP model's curves align with the upper right and off diagonal confusion counts nearly vanish, whereas the RAW model still confuses minority attacks, especially mitm and recon with benign. Feature importance explains the pattern. On DNP3, the RAW model is dominated by an identifier like field (device_name) and ad hoc features (f566, f466), while the PP model suppresses these and instead leans on engineered features such as month, dow, hour and stable network aggregates. On CIC-IIoT, the RAW model ranks indexing and identifier fields very highly (e.g. Unnamed: 0.1, Flow ID, timestamp adjacent counts), whereas the PP model shifts weight toward more generalizable temporal and traffic statistics. The one residual indexing column (Unnamed: 0.1) surfacing near the top signals a cleanup opportunity but does not undermine the overall PP gains observed elsewhere.

Discussion

The efficiency table shows that preprocessing reduces DNP3's training time from 0.335 s to 0.284 s and shrinks the model by approx. 42%, which is in line with a one third reduction in effective features (RAW:85, PP:55). On CIC-IIoT the effect is far stronger, fit time drops by approx. 65% (RAW: 14.6 s, PP: 5.14 s), the serialized forest contracts by approx. 94% (RAW: 412 MB PP: 18.7 MB) and the feature set is reduced by approx. 34%. Random Forest size scales with both the number and depth of trees and with the dimensionality of split candidates, by removing redundant and high cardinality features, bounding categorical encodings and down casting numerics, A.R.C. narrows the search space and results in shallower and more compact trees. These engineering changes directly account for the speedups and the order of magnitude reduction in the IIoT artifact footprint.

DNP3 behaves as anticipated in accuracy, F1, precision, and recall, for a clean and balanced dataset. Validation and test improve by only a few tenths of a percentage point and ROC–AUC is already saturated at 100.00%. The slight reduction on DNP3's training split after preprocessing is a healthy sign of less overfitting. A.R.C. removes id like and unstable features the RAW forest could memorize, while preserving generalization. The corresponding feature rankings make this visible. In the RAW model device_name dominates importance together with anonymous features (f566, f466), which shows classic leakage. After preprocessing these disappear from the

top ranks and the forest relies on calendar and traffic statistics (month, dow, hour, packet, length, interval aggregates). That the test metrics remain effectively identical, while the model is simpler, supports the claim that PP increases robustness without materially changing performance on an already easy datasets.

The IIoT results are different. Recall on the test split jumps by approx. 13 percentage points and F1 by approx. 8, accuracy climbs approx. 5 points into the high 99s. The precision recall plots explain where these gains come from (figure **??**). In the RAW curves, mitm bows downward (AP=0.96) and bruteforce, ddos, and dos fall earlier than desired, after preprocessing all curves concentrate at the top right with AP=0.99 to 1.00, indicating that minority classes previously getting blurred into benign now have distinctive signal. This is also visible in the confusion matrix (figure **??**), the RAW model shows substantial spillover from recon and the DoS variants into benign, whereas the PP model's diagonal is nearly clean with only a handful of residual mistakes. In practice this means the standardized pipeline is not just polishing an already good classifier, it is fixing the minority recall that matters operationally.

Why does IIoT benefit so much? Two mechanisms are visible in the feature profiles. First, the RAW forest places disproportionate weight on indexing or identifier features such as Flow ID, Timestamp, and the unnamed index (Unnamed: 0.1). Such fields are high cardinality and easy for trees to split on, but they tend to encode session and order effects rather than attack semantics, leading to brittle decision paths and poor minority recall. Second, many traffic features are heavy tailed or on incomparable scales in the RAW dataset, which encourages deep and fragmentary trees. A.R.C. standardizes and regularizes these inputs, caps one hot explosion and removes or down weights id like features. In the resulting model, calendar features and consolidated flow statistics rise in importance. This shift aligns the forest with physically meaningful structure in network traces, improving separability of rare attacks without overfitting superficial identifiers. The one cautionary note is the prominence of Unnamed: 0.1 in both CIC-IIoT ranking plots, this looks like an artifact feature and should be explicitly dropped. Its presence does not appear to inflate performance, RAW still underperforms PP, but removing it would further strengthen interpretability and guard against accidental leakage.

Finally, the ROC–AUC rows are flat at approx. 100.00% on both datasets for both regimes. This is consistent with forests being strong rankers even in the RAW setting. The improvements therefore surface primarily at fixed operating points, like F1 or recall, where cleaner preprocessing produces better calibrated partitions of the score space. Inference latency falls by roughly half on DNP3 and by about two thirds on CIC-IIoT across all splits, mirroring the smaller and shallower forest and reduced feature vector width.

In sum, the combined tables and figures show a coherent picture. On a balanced and clean dataset preprocessing mainly improves efficiency while preserving already excellent accuracy and removing reliance on dubious identifiers. On a noisier, moderately imbalanced dataset, the same standardized preprocessing substantially increases recall and F1. IT converts ambiguous classes into well separated ones as seen in PR curves and confusion matrices and results in lighter and faster models. The feature importance transitions diagnose the mechanism of change and flag a minor clean up task, by dropping unnamed and index columns, that would make the pipeline even more defensible.

5.2.2 Gradient boosted decision trees (GBDT) - CatBoost

The gradient boosted decision trees - CatBoost study quantifies the effect of A.R.C.'s standardized preprocessing under stratified splitting. Table 5.6 reports efficiency footprint (training time, serialized model size and effective feature count) for RAW and PP with percentage deltas. Table 5.7 reports accuracy, F1, precision, recall, ROC–AUC, and per split inference time for Train/Validation/Test, with paired differences (Δ). The accompanying figures visualize the test split behavior, precision–recall curves by class (figure 5.13), normalized confusion matrices (figure 5.12), and top-25 feature-importance rankings (figures 5.11 and 5.14) for RAW and PP, making the tabulated deltas and remaining error modes directly interpretable.

Table 5.6 – Overview of resource footprint of GBDT-CatBoost before (RAW) and after A.R.C.'s standardized preprocessing (PP) under a 60/20/20 stratified split

KPI	DNP3			CIC-IIoT		
	Raw	PP	Δ	Raw	PP	Δ
Training time	34,672 s	6,039 s	-82,58 %	1146,239 s	36,680 s	-96,80 %
Model size	5,22 MB	4,76 MB	-8,76 %	55,80 MB	3,58 MB	-93,58 %
Feature count	85	55	-35,29 %	86	57	-33,72 %

Table 5.7 – Overview of predictive performance of GBDT-CatBoost with (PP) and without (RAW) preprocessing on train, validation and test under a 60/20/20 stratified split

KPI	Split	DNP3			CIC-IIoT		
		RAW	PP	Δ	RAW	PP	Δ
Accuracy	Train	100,00 %	100,00 %	0,00 pp	99,99 %	99,80 %	-0,19 pp
	Validation	99,93 %	100,00 %	+0,07 pp	99,98 %	99,76 %	-0,22 pp
	Test	99,93 %	99,93 %	0,00 pp	99,97 %	99,74 %	-0,23 pp
F1	Train	100,00 %	100,00 %	0,00 pp	99,97 %	99,48 %	-0,49 pp
	Validation	99,93 %	100,00 %	+0,07 pp	99,95 %	99,41 %	-0,54 pp
	Test	99,93 %	99,93 %	0,00 pp	99,95 %	99,40 %	-0,55 pp
Precision	Train	100,00 %	100,00 %	0,00 pp	99,96 %	99,32 %	-0,64 pp
	Validation	99,93 %	100,00 %	+0,07 pp	99,94 %	99,23 %	-0,71 pp
	Test	99,93 %	99,93 %	0,00 pp	99,96 %	99,24 %	-0,72 pp
Recall	Train	100,00 %	100,00 %	0,00 pp	99,98 %	99,65 %	-0,32 pp
	Validation	99,93 %	100,00 %	+0,07 pp	99,97 %	99,61 %	-0,36 pp
	Test	99,93 %	99,93 %	0,00 pp	99,94 %	99,57 %	-0,37 pp
ROC-AUC	Train	100,00 %	100,00 %	0,00 pp	100,00 %	100,00 %	0,00 pp
	Validation	100,00 %	100,00 %	0,00 pp	100,00 %	100,00 %	0,00 pp
	Test	100,00 %	100,00 %	0,00 pp	100,00 %	100,00 %	0,00 pp
Inference time	Train	0,0139 s	0,0032 s	-76,7 %	0,5117 s	0,0723 s	-85,9 %
	Validation	0,0090 s	0,0016 s	-82,7 %	0,1641 s	0,0239 s	-85,4 %
	Test	0,0088 s	0,0015 s	-82,5 %	0,1756 s	0,0238 s	-86,5 %

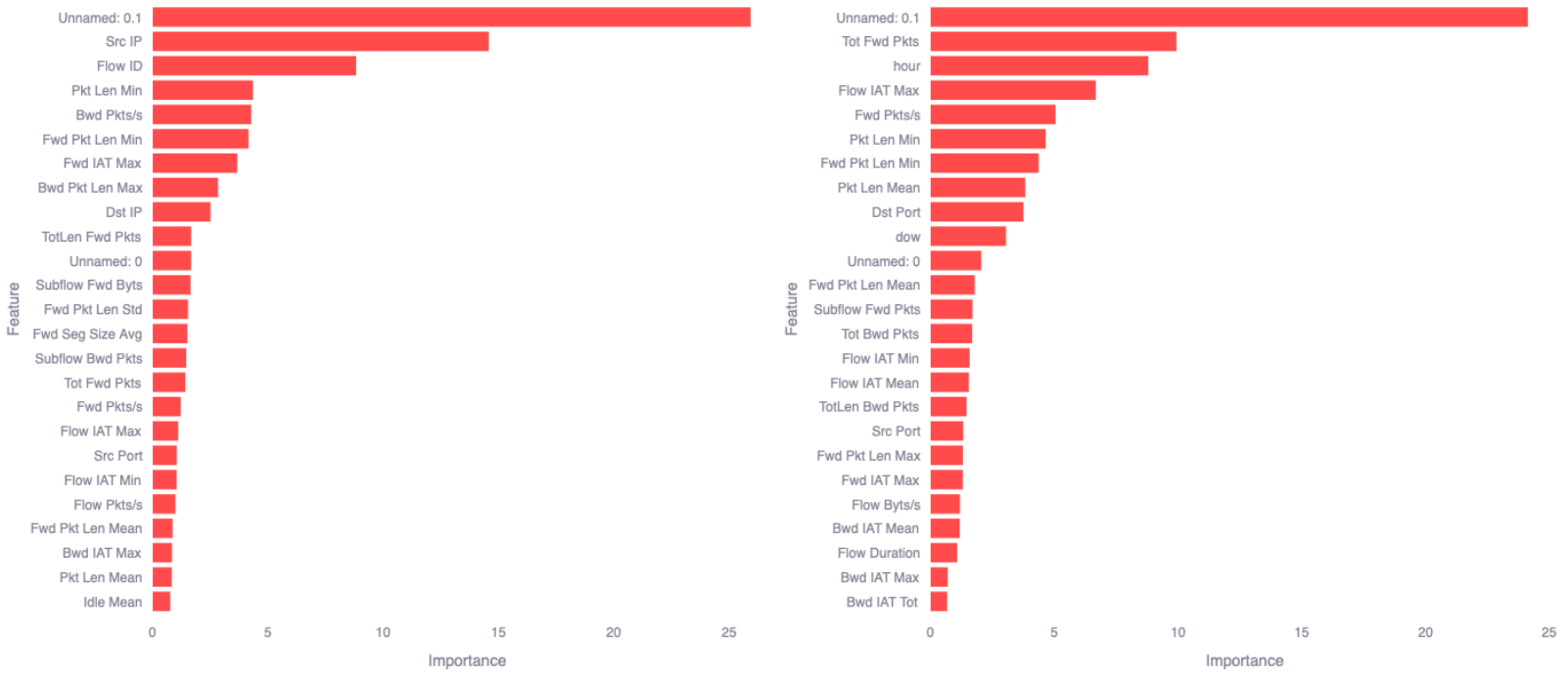


Figure 5.11 – Illustration of the top 25 most important features comparison between RAW (left) and PP (right) for GBDT-CatBoost on DNP3 under stratified splitting (60/20/20)

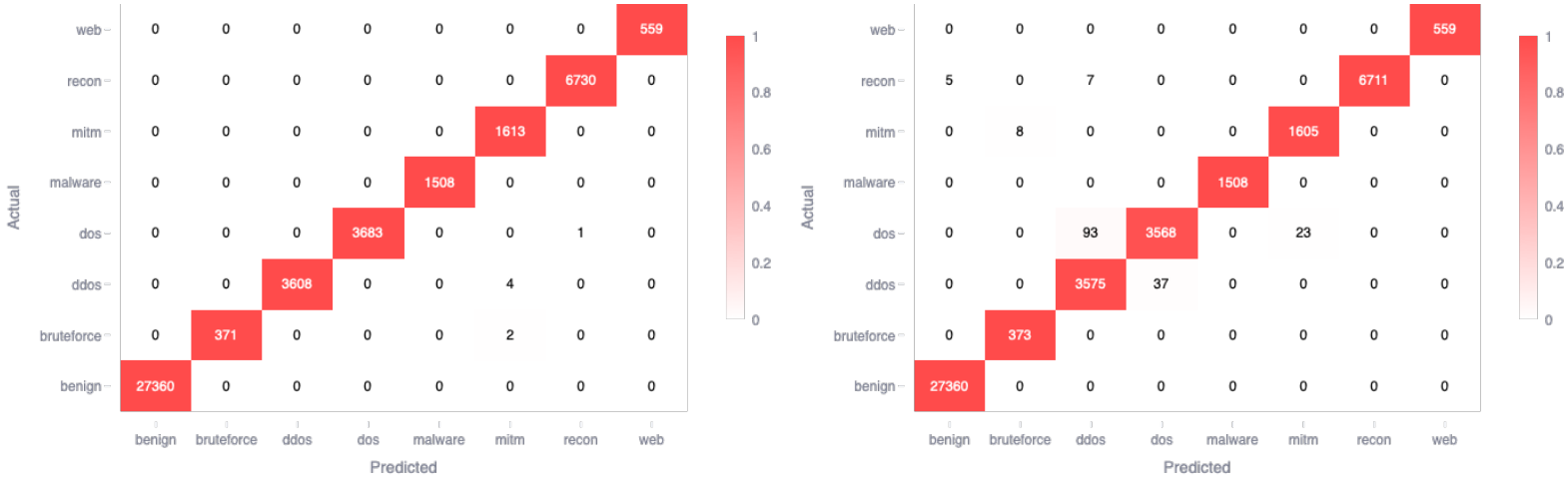


Figure 5.12 – Illustration of the confusion matrix comparison between RAW (left) and PP (right) for GBDT-CatBoost on CIC-IIoT under stratified splitting (60/20/20)

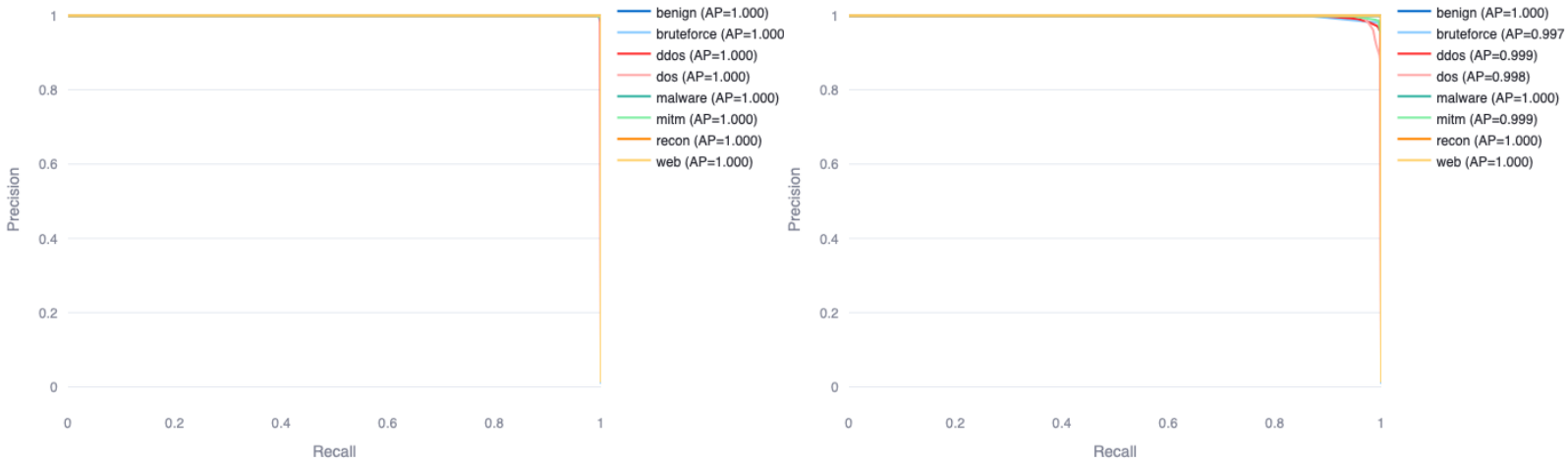


Figure 5.13 – Illustration of the PR curve comparison between RAW (left) and PP (right) for GBDT-CatBoost on CIC-IIoT under stratified splitting (60/20/20)

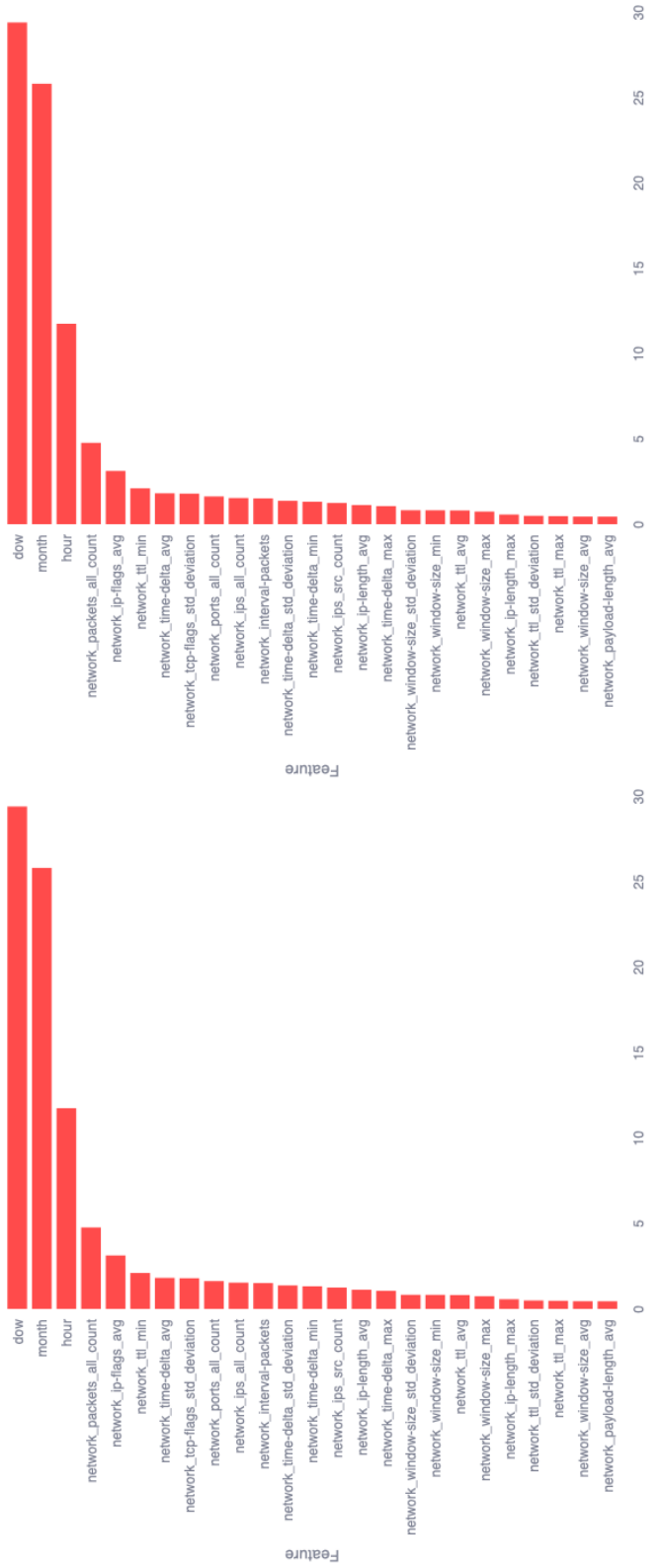


Figure 5.14 – Illustration of the top 25 most important features comparison between RAW (left) and PP (right) for GBDT-CatBoost on CIC-IIoT under stratified splitting (60/20/20)

Key observations

Across both datasets, standardized preprocessing makes CatBoost dramatically more efficient while leaving accuracy essentially unchanged on DNP3 and only marginally lower on CIC-IIoT. Training time drops by 82.6% on DNP3 and 96.8% on CIC-IIoT, inference latency reduces by 77 to 87 % on every split. Model size shrinks slightly on DNP3 (8.8%) and collapses on CIC-IIoT (93.6 %). ROC–AUC remains perfect (100 %) throughout. The CIC-IIoT metrics decline by a few tenths of a percentage point after preprocessing (0.2 to 0.7 pp across accuracy, F1, precision and recall), whereas DNP3 shows no practical change.

The feature importance plots explain these patterns. On CIC-IIoT RAW, high cardinality identifiers such as Src IP and Flow ID are among the most influential features. After preprocessing they disappear, with influence shifting to generic flow statistics (e.g. Tot Fwd Pkts, Flow IAT Max) and calendar cues (hour, dow). On DNP3 RAW, timestamp and device_name dominate, after preprocessing the raw timestamp is decomposed and down weighted and calendar derivatives (dow, month, hour) lead with a long tail of network aggregates. In both datasets an Unnamed: 0.1 index column ranks highly, which is evidence of a spurious artifact that PP should excise.

The PR curves and confusion matrices for CIC-IIoT are consistent with the tables. RAW curves are essentially flat at precision at approx. 100% with AP=1.000 for all classes. PP curves remain extremely strong but show a small contraction for ddos and mitm, which aligns with the 0.5 to 0.7 pp drops in precision, F1 and recall. Confusions remain rare and mostly appear within plausible pairs (e.g. ddos and dos).

Discussion

CatBoost is designed to thrive with categorical and missing data and it can extract signal from high cardinality features by building target statistics and split candidates, that other tree learners cannot. That strength is visible in the RAW runs. On CIC-IIoT, the model leans heavily on Src IP and Flow ID and on DNP3 it exploits timestamp and device_name. These variables are excellent short term discriminators in curated benchmarks but they are also leakage prone. Device names are scenario specific, raw timestamps encode the scripted schedule of synthetic attacks and flow identifiers and IPs can memorize session boundaries. The near-perfect RAW PR curves and the uniformly perfect ROC–AUC, suggest separability that is too clean for messy deployments.

A.R.C.'s standardized preprocessing intentionally reduces that risk. It drops id like columns, derives calendar features from timestamps, bounds categorical explosion and compacts the feature space (approx. 35% on DNP3 and approx. 34% on CIC-IIoT). With those constraints, CatBoost can no longer lean on brittle

identifiers and must base splits on flow level statistics and calendar structure. The small declines on CIC-IIoT are the cost of removing suspect shortcuts. In exchange, the models are an order of magnitude smaller and faster and the learned structure is far more defensible. The PP feature importance ranks are dominated by interpretable aggregates such as packet counts and inter arrival times rather than ids. On DNP3, where the dataset is balanced and clean, the PP and RAW scores are indistinguishable, implying that preprocessing preserves quality while still delivering large efficiency gains.

Finally, the prominence of Unnamed: 0:1 across importance plots highlights an ingestion flaw. An index column slipped into modeling and carries spurious predictive power. Even though preprocessing already trims the feature space, explicitly dropping any auto generated index columns at load time is advisable to prevent accidental leakage.

In sum, with CatBoost the standardized pipeline trades a negligible amount of headline accuracy on CIC-IIoT for substantial gains in efficiency and robustness, while leaving the already clean performance intact. After preprocessing importance shifts away from brittle identifiers to stable network statistics. PR curves remain excellent with minor contraction on the most ambiguous classes. Confusion matrices stay nearly perfectly diagonal. This is exactly the behavior A.R.C. aims to induce, fast, compact and auditable models that rely on portable signals rather than dataset quirks.

5.2.3 Support vector machines (SVM)

The support vector machines study quantifies the effect of A.R.C.'s standardized preprocessing under stratified splitting. Table 5.8 reports efficiency footprint (training time, serialized model size and effective feature count) for RAW and PP with percentage deltas. Table 5.9 reports accuracy, F1, precision, recall, ROC–AUC, and per split inference time for Train/Validation/Test, with paired differences (Δ). The accompanying figures visualize the test split behavior, precision–recall curves by class (figure 5.16) and normalized confusion matrices (figure 5.15), making the tabulated deltas and remaining error modes directly interpretable.

Table 5.8 – Overview of resource footprint of SVM before (RAW) and after A.R.C.'s standardized preprocessing (PP) under a 60/20/20 stratified split

KPI	DNP3			CIC-IIoT		
	Raw	PP	Δ	Raw	PP	Δ
Training time	1,117 s	0,296 s	-73,47 %	19310.300 s	472.068 s	-97,56 %
Model size	5,22 MB	583,91 KB	-83,69 %	420.16 MB	3.69 MB	-99,12 %
Feature count	85	55	-35,29 %	86	57	-33,72 %

Table 5.9 – Overview of predictive performance of SVM with (PP) and without (RAW) preprocessing on train, validation and test under a 60/20/20 stratified split

KPI	Split	DNP3			CIC-IIoT		
		RAW	PP	Δ	RAW	PP	Δ
Accuracy	Train	98,77 %	98,69 %	-0,08 pp	92,35 %	99,00 %	+6,65 pp
	Validation	98,50 %	98,16 %	-0,34 pp	91,53 %	98,89 %	+7,36 pp
	Test	98,36 %	98,45 %	+0,08 pp	91,66 %	98,87 %	+7,20 pp
F1	Train	98,77 %	98,61 %	-0,16 pp	85,82 %	97,58 %	+11,76 pp
	Validation	98,50 %	98,18 %	-0,32 pp	83,85 %	97,39 %	+13,54 pp
	Test	98,36 %	98,47 %	+0,11 pp	84,11 %	97,31 %	+13,20 pp
Precision	Train	98,77 %	98,55 %	-0,23 pp	94,77 %	96,81 %	+2,05 pp
	Validation	98,53 %	98,29 %	-0,24 pp	94,35 %	96,62 %	+2,28 pp
	Test	98,41 %	98,55 %	+0,14 pp	93,98 %	96,50 %	+2,52 pp
Recall	Train	98,77 %	98,77 %	0,00 pp	80,44 %	98,42 %	+17,98 pp
	Validation	98,50 %	98,23 %	-0,27 pp	77,90 %	98,23 %	+20,33 pp
	Test	98,37 %	98,50 %	+0,13 pp	78,20 %	98,20 %	+19,99 pp
ROC-AUC	Train	99,97 %	99,99 %	+0,02 pp	96,79 %	99,97 %	+3,18 pp
	Validation	99,96 %	99,98 %	+0,02 pp	93,61 %	99,96 %	+6,36 pp
	Test	99,94 %	99,97 %	+0,02 pp	93,85 %	99,96 %	+6,11 pp
Inference time	Train	0,7201 s	0,2008 s	-72,1 %	349,2450 s	42,9810 s	-87,7 %
	Validation	0,2431 s	0,0694 s	-71,5 %	115,6134 s	14,3567 s	-87,6 %
	Test	0,2473 s	0,0690 s	-72,1 %	118,1575 %	14,3384 %	-87,9 %

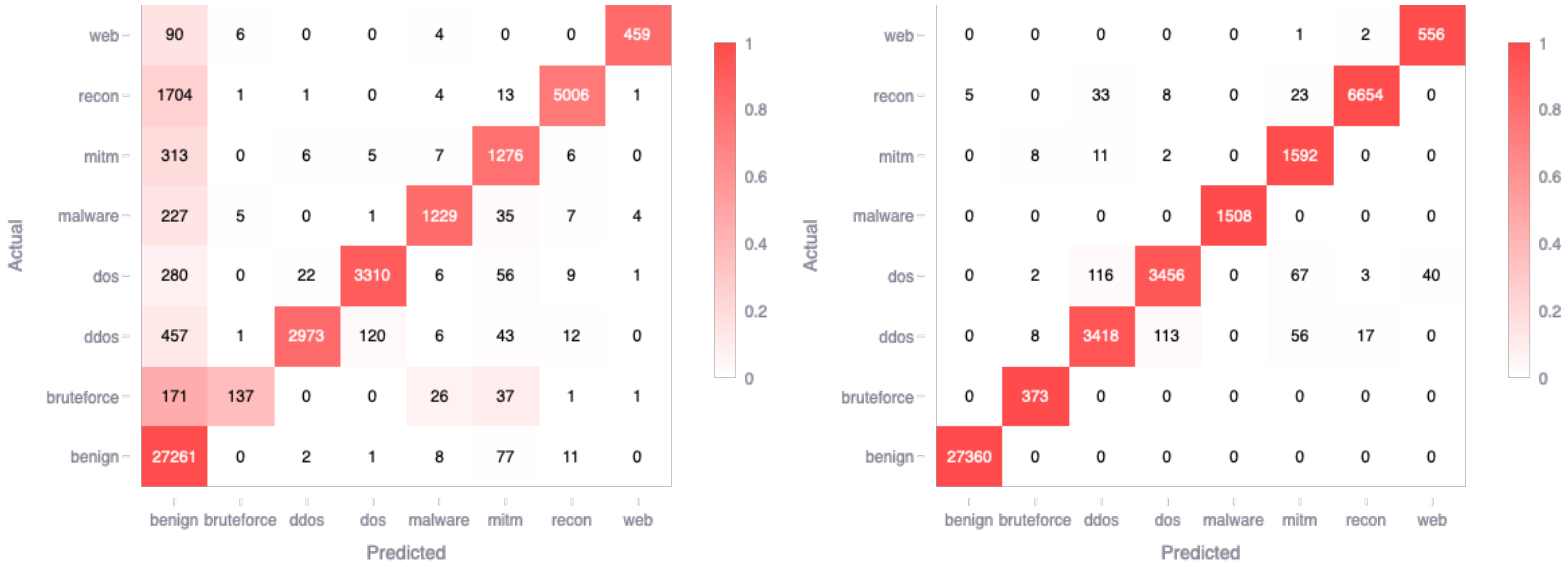


Figure 5.15 – Illustration of the confusion matrix comparison between RAW (left) and PP (right) for SVM on CIC-IIoT under stratified splitting (60/20/20)

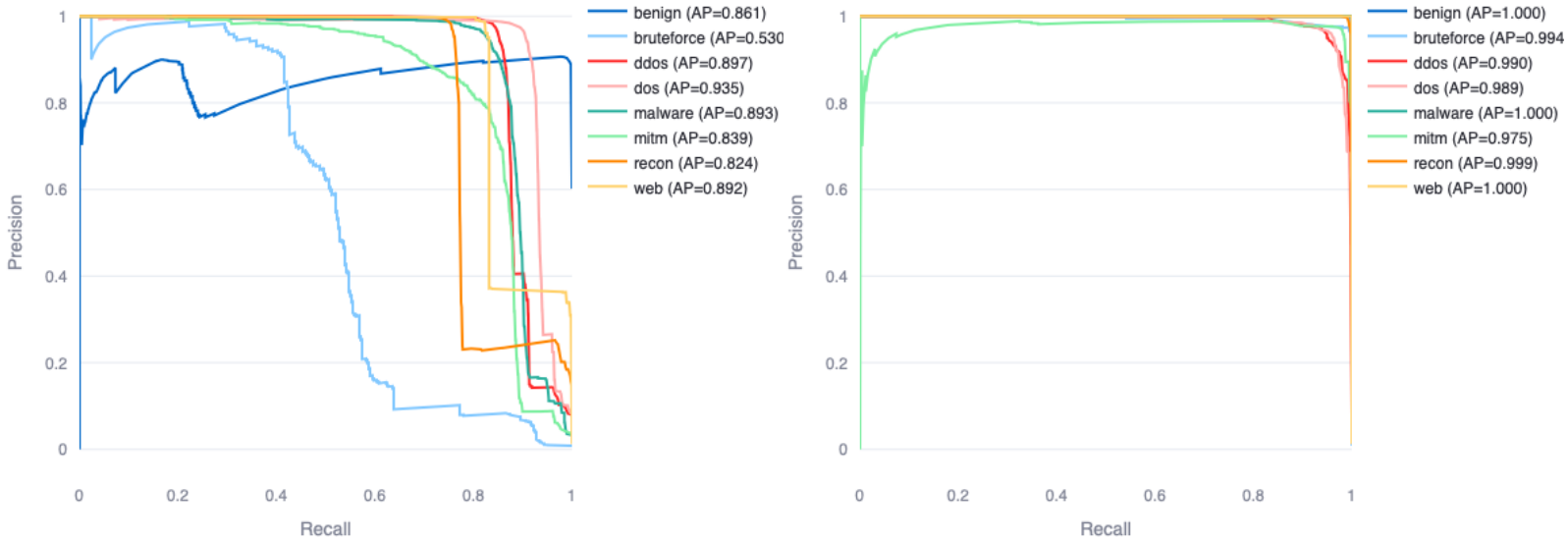


Figure 5.16 – Illustration of the PR curve comparison between RAW (left) and PP (right) for SVM on CIC-IIoT under stratified splitting (60/20/20)

Key observations

Across both datastes, A.R.C.'s preprocessing dramatically reduces SVM footprint. Training time drops by 73% on DNP3 and by 97.6% on CIC-IIoT. Model size shrinks by 83.7% and 99.1% respectively, with a one third reduction in effective features in both cases. On the balanced DNP3 set, predictive metrics are already near maximum, preprocessing nudges validation accuracy and F1 slightly down but improves test accuracy and F1 by 0.1 pp, with ROC–AUC unchanged at 99.95 to 99.97%. On CIC-IIoT, PP transforms SVM from a middling baseline to near state of the art. Test accuracy rises from 91.66% to 98.87%, F1 from 84.11% to 97.31%, precision from 93.98% to 96.50% and recall from 78.20% to 98.20%. Inference latency reduces by approx. 88% on CIC-IIoT and approx. 72% on DNP3.

The figures mirror these deltas. On CIC-IIoT, the RAW PR curves show large areas under the curve only for a few classes, bruteforce (AP=0.53), recon (AP=0.82), mitm (AP=0.84) and malware (AP=0.89) are weak. After PP, PR curves align with the top right across the board with AP= approx.0.99–1.00 for most classes and approx. 0.975 for mitm. The RAW confusion matrix reveals systematic mistakes, heavy leakage of recon into benign, and cross confusions between ddos and dos. The PP matrix is almost perfectly diagonal, with residual, symmetric ddos dos confusions and small tails in recon and web, exactly the classes that still have slightly lower APs than the rest.

Discussion

The SVM results underscore how sensitive margin based learners are to data hygiene and representation. On CIC-IIoT, RAW performance is bottlenecked by scale heterogeneity (high cardinality and irregular features) and imbalance amplified decision boundaries. The model attains good precision on several classes but misses a large fraction of positives, reflected in the low recall (78.20%) and the bowed PR curves for bruteforce, recon, and mitm. A.R.C.'s preprocessing addresses these failure modes in three ways. First, consistent numeric standardization and bounded one hot and categorical handling place features in comparable ranges, allowing a linear margin to form without any single variable dominating the kernel. Second, cleaning and compacting the feature set, by removeing id like and sparse nuisance features that create fragmented margins and many support vectors. Third, class aware stratification ensures minority presence in all splits, stabilizing the hyperplane selection during training. The joint effect is a dramatic reduction in support vector complexity (reflected in the approx. 99% drop in model size on CIC-IIoT) and a step-change in recall (+20 pp on validation, +19.99 pp on test), while precision also oncreases by a couple of points. The PR curves become near rectangular and the confusion matrix turns almost perfectly diagonal.

The residual mistakes after PP are instructive. The ddos dos pair remains the primary confusion channel in the CIC-IIoT matrix, with a small, symmetric spill in both directions. This is consistent with traffic flow similarity between volumetric and service-exhaustion behaviours when reduced to aggregate statistics. It suggests either more discriminative features for rate dynamics or temporal context might be needed to fully separate them. A smaller tail persists for recon misclassified as benign, which is plausible when probing resembles normal scanning or when flow windows are short. Threshold tuning or cost-sensitive calibration could further mitigate this without sacrificing overall accuracy.

DNP3 behaves differently because it is balanced, schema clean, and comparatively small. Both RAW and PP SVMs saturate. Micro-changes around 0.1–0.2 pp on test indicate that once features are already tidy and well-scaled, standardization yields little headroom for linear margin gains. Nevertheless, even on DNP3 the footprint benefits are clear. Faster training and substantially smaller models with the same AUC and virtually identical accuracy and F1. This reinforces the view that, for SVMs, PP's strongest value is efficiency and generality on normal noisy datasets like CIC-IIoT, while it does no harm on already curated datasets.

Operationally, the efficiency improvements are as important as the quality gains. On CIC-IIoT, training time reduces from approx. 5.4 hours to under 8 minutes, and inference time per split drops by approx. 88%, enabling iterative experimentation and real-time use that would be impractical otherwise. Combined with the near perfect PR curves and the diagonal confusion matrix, these results provide convergent evidence that A.R.C.'s standardized preprocessing is a precondition for SVMs to perform competitively on heterogeneous industrial datasets.

5.2.4 Multi layer perceptrons (MLP)

The multi layer perceptrons study quantifies the effect of A.R.C.'s standardized preprocessing under stratified splitting. Table 5.10 reports efficiency footprint (training time, serialized model size and effective feature count) for RAW and PP with percentage deltas. Table 5.11 reports accuracy, F1, precision, recall, ROC–AUC, and per split inference time for Train/Validation/Test, with paired differences (Δ). The accompanying figures visualize the test split behavior, precision–recall curves by class (figure 5.18) and normalized confusion matrices (figure 5.17), making the tabulated deltas and remaining error modes directly interpretable.

Table 5.10 – Overview of resource footprint of MLP before (RAW) and after A.R.C.'s standardized preprocessing (PP) under a 60/20/20 stratified split

KPI	DNP3			CIC-IIoT		
	Raw	PP	Δ	Raw	PP	Δ
Training time	0,982 s	0,767 s	-21,83 %	133,903	433,403	+233,67 %
Model size	1,26 MB	808,75 KB	-37,30 %	394,68 MB	23,69 MB	-94,00 %
Feature count	85	76	-10,59 %	86	90	+4,65 %

Table 5.11 – Overview of predictive performance of MLP with (PP) and without (RAW) preprocessing on train, validation and test under a 60/20/20 stratified split

KPI	Split	DNP3			CIC-IIoT		
		RAW	PP	Δ	RAW	PP	Δ
Accuracy	Train	100,00 %	100,00 %	0,00 pp	94,44 %	99,75 %	+5,31 pp
	Validation	99,52 %	99,72 %	+0,20 pp	90,38 %	99,58 %	+9,20 pp
	Test	99,80 %	99,65 %	-0,15 pp	90,40 %	99,49 %	+9,09 pp
F1	Train	100,00 %	100,00 %	0,00 pp	91,90 %	99,40 %	+7,50 pp
	Validation	99,52 %	99,67 %	+0,14 pp	82,32 %	98,90 %	+16,57 pp
	Test	99,80 %	99,66 %	-0,14 pp	82,92 %	98,75 %	+15,84 pp
Precision	Train	100,00 %	100,00 %	0,00 pp	97,07 %	99,22 %	+2,15 pp
	Validation	99,53 %	99,65 %	+0,12 pp	94,22 %	98,63 %	+4,41 pp
	Test	99,80 %	99,66 %	-0,14 pp	94,12 %	98,37 %	+4,25 pp
Recall	Train	100,00 %	100,00 %	0,00 pp	87,64 %	99,58 %	+11,94 pp
	Validation	99,52 %	99,69 %	+0,16 pp	75,11 %	99,17 %	+24,06 pp
	Test	99,80 %	99,66 %	-0,14 pp	75,56 %	99,16 %	+23,61 pp
ROC-AUC	Train	100,00 %	100,00 %	0,00 pp	98,94 %	100,00 %	+1,05 pp
	Validation	100,00 %	100,00 %	0,00 pp	93,63 %	99,99 %	+6,37 pp
	Test	99,98 %	99,99 %	+0,01 pp	93,84 %	99,99 %	+6,15 pp
Inference time	Train	0.0160 s	0,0064 s	-59,8 %	0.8602 s	7,4604 s	+767,3 %
	Validation	0,0111 s	0,0028 s	-74,2 %	0.4828 s	0,9827 s	+103,5 %
	Test	0,0113 s	0,0025 s	-77,9 %	0.4462 s	1,3130 s	+194,3 %

Figure 5.17 – Illustration of the confusion matrix comparison between RAW (left) and PP (right) for MLP on CIC-IIoT under stratified splitting (60/20/20)

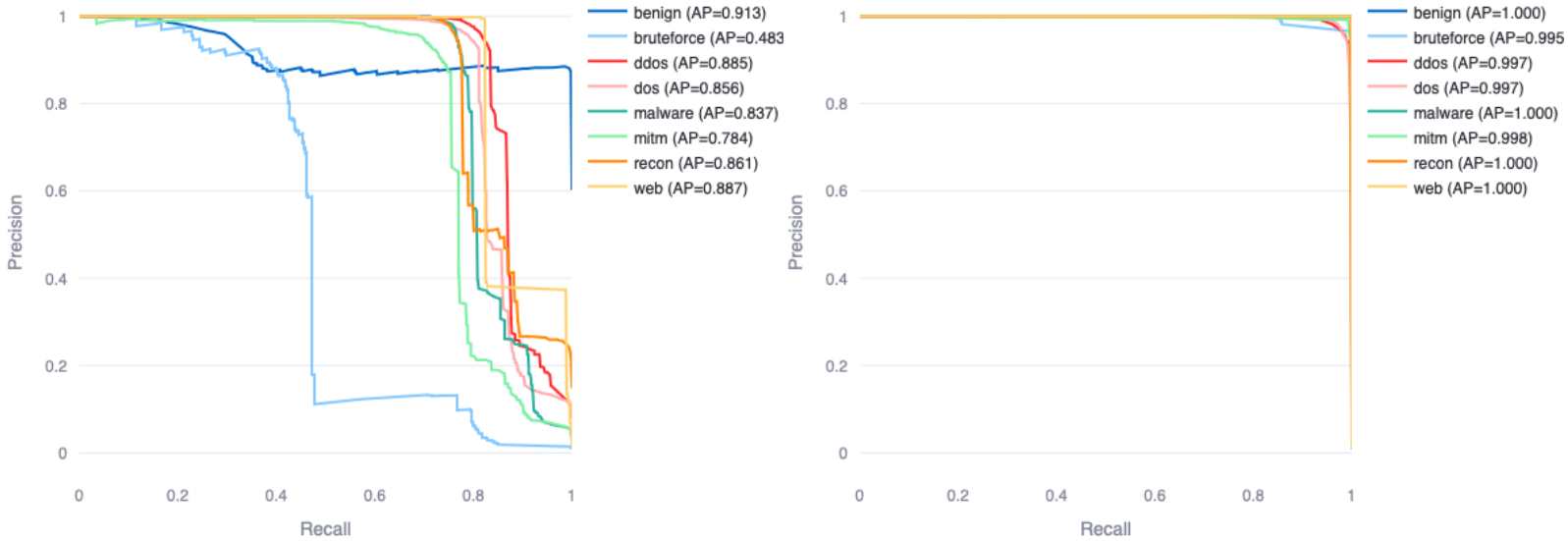


Figure 5.18 – Illustration of the PR curve comparison between RAW (left) and PP (right) for MLP on CIC-IIoT under stratified splitting (60/20/20)

Key observations

Standardized preprocessing strengthens the MLP on the noisier CIC-IIoT dataset while leaving the already clean DNP3 essentially unchanged. On DNP3, accuracy, F1, precision, recall and ROC–AUC on validation and test move by only a few hundredths of a percentage point and sometimes dip marginally, consistent with reduced overfitting rather than a true loss of generalization. Efficiency improves modestly, training time drops from 0,982,s to 0,767,s, model size reduces by approx. 37%, and the effective feature set is slightly smaller (85 to 76).

On CIC-IIoT, the picture is different. Test accuracy climbs from 90,40% to 99,49% (+9,09 pp), F1 from 82,92% to 98,75% (+15,84 pp) and recall from 75,56% to 99,16% (+23,61 pp). ROC–AUC increases by approx. 6 pp to essentially 100,00. The precision–recall curves on the test split confirm the shift, with RAW preprocessing, minority attacks such as bruteforce, mitm, recon and the DoS variants bow downward with AP in the 0,48 to 0,89 range, whereas the PP curves cluster at the upper right with AP = approx. 0,995 to 1,000. The confusion matrices tell the same story, substantial off diagonal spillover into benign and between attack families under RAW nearly vanishes under PP, yielding an almost perfectly diagonal matrix.

The main trade off is computational. For CIC-IIoT, MLP training time increases (133,903 s to 433,403 s) and per split inference latency rises as well, even though the serialized model becomes approx. 94% smaller. In short, PP yields very large accuracy recall gains on CIC-IIoT, slight efficiency gains on DNP3, but notably higher compute cost for MLP on CIC-IIoT.

Discussion

The DNP3 results mirror expectations for a balanced, schema clean dataset. PP removes brittle signals and standardizes inputs, which trims model size (1,26 MB to 808,75 KB) and speeds inference by approx. 60 to 78% without materially changing performance. The tiny dips on train and test reflect less memorization after eliminating identifier like or unstable columns and applying consistent scaling. The near ceiling ROC–AUC (approx. 100,00%) before and after PP indicates the task is already easy and rank ordering was never the limiting factor.

On CIC-IIoT, PP transforms the MLP from a moderately performing classifier into an almost perfectly calibrated detector. The RAW precision recall plot exhibits clear minority class collapse, bruteforce (AP=0,53) and mitm and recon (AP=0,78 to 0,86). After PP, every class curve alligns with the top right corner. This aligns with the confusion matrices, under RAW, thousands of recon and DoS samples leek into benign, under PP, those off diagonal counts collapse and each attack class is retrieved with near perfect precision and recall. These effects drive the +23,61 pp jump in recall and +15,84 pp in F1 on the test split.

Why does PP help the MLP so much here? Two mechanisms are plausible and consistent with A.R.C.'s preprocessing catalogue. First, scale normalization and outlier handling make the loss surface better conditioned for gradient descent, allowing the same architecture to separate classes that were previously entangled by heterogeneous feature scales and heavy-tailed distributions. Second, disciplined encoding of categorical/identifier fields reduces leakage and puts more weight on meaningful traffic statistics; the PR/CM improvements indicate the network is no longer keying on spurious sequence or index artefacts that previously blurred minority attacks into *benign*.

The notable regression in compute on CIC-IIoT deserves comment. Although the model artifact is far smaller (RAW: 394,68,MB, PP: 23,69,MB), training and inference are slower. This is consistent with a denser input representation after preprocessing (one hot and bounded categorical encodings and fully standardized float tensors) and with CPU bound dense matrix multiplies dominating both fit and forward pass. Even a modest rise in effective dimensionality (RAW: 86, PP: 90) can increase MLP cost substantially.

The MLP study reinforces a pattern seen throughout this case study. On clean, balanced data (DNP3), PP improves efficiency and model hygiene while keeping metrics flat. On noisier, moderately imbalanced data (CIC-IIoT), the same standardized pipeline delivers large, operationally meaningful gains in recall and F1, clearly visible in PR curves and confusion matrices, at the expense of higher compute for MLP specifically. Given the magnitude of the accuracy gains and the dramatic reduction in model footprint, this compute trade off is often acceptable.

5.2.5 Logistic regression (LR)

The multi layer perceptrons study quantifies the effect of A.R.C.'s standardized preprocessing under stratified splitting. Table 5.12 reports efficiency footprint (training time, serialized model size and effective feature count) for RAW and PP with percentage deltas. Table 5.13 reports accuracy, F1, precision, recall, ROC–AUC, and per split inference time for Train/Validation/Test, with paired differences (Δ). The accompanying figures visualize the test split behavior, precision–recall curves by class (figure 5.20) and normalized confusion matrices (figure 5.19), making the tabulated deltas and remaining error modes directly interpretable.

Table 5.12 – Overview of resource footprint of LR before (RAW) and after A.R.C.'s standardized preprocessing (PP) under a 60/20/20 stratified split

KPI	DNP3			CIC-IIoT		
	Raw	PP	Δ	Raw	PP	Δ
Training time	1,111 s	1,379 s	+24,15 %	30,853 s	8,442 s	-72,64 %
Model size	201,65 KB	6,49 KB	-96,78 %	391,45 MB	5,91 KB	-100,00 %
Feature count	85	55	-35,29 %	86	57	-33,72 %

Table 5.13 – Overview of predictive performance of LR with (PP) and without (RAW) preprocessing on train, validation and test under a 60/20/20 stratified split

KPI	Split	DNP3			CIC-IIoT		
		RAW	PP	Δ	RAW	PP	Δ
Accuracy	Train	99,66 %	99,30 %	-0,36 pp	92,36 %	97,18 %	+4,82 pp
	Validation	98,84 %	99,01 %	+0,17 pp	86,89 %	97,22 %	+10,33 pp
	Test	99,11 %	99,22 %	+0,11 pp	86,87 %	97,15 %	+10,27 pp
F1	Train	99,66 %	99,17 %	-0,48 pp	85,98 %	93,39 %	+7,40 pp
	Validation	98,84 %	99,04 %	+0,20 pp	75,17 %	93,54 %	+18,37 pp
	Test	99,11 %	99,25 %	+0,13 pp	75,41 %	93,35 %	+17,94 pp
Precision	Train	99,66 %	99,04 %	-0,62 pp	92,69 %	91,37 %	-1,32 pp
	Validation	98,88 %	99,06 %	+0,18 pp	87,35 %	91,67 %	+4,33 pp
	Test	99,14 %	99,26 %	+0,12 pp	87,27 %	91,39 %	+4,12 pp
Recall	Train	99,66 %	99,34 %	-0,32 pp	81,08 %	95,90 %	+14,82 pp
	Validation	98,84 %	99,05 %	+0,16 pp	67,91 %	95,85 %	+27,93 pp
	Test	99,11 %	99,25 %	+0,14 pp	68,09 %	95,80 %	+27,71 pp
ROC-AUC	Train	100,00 %	99,99 %	-0,01 pp	98,30 %	99,80 %	+1,50 pp
	Validation	99,84 %	99,99 %	+0,15 pp	92,12 %	99,80 %	+7,68 pp
	Test	99,98 %	99,97 %	-0,01 pp	92,20 %	99,79 %	+7,60 pp
Inference time	Train	0.0207 s	0,0019 s	-90,9 %	0.6846 s	0,0135 s	-98,0 %
	Validation	0,0105 s	0,0005 s	-95,3 %	0.3728 s	0,0024 s	-909,4 %
	Test	0,0105 s	0,0004 s	-96,4 %	0.3737 s	0,0019 s	-99,5 %

Figure 5.19 – Illustration of the confusion matrix comparison between RAW (left) and PP (right) for LR on CIC-IIoT under stratified splitting (60/20/20)

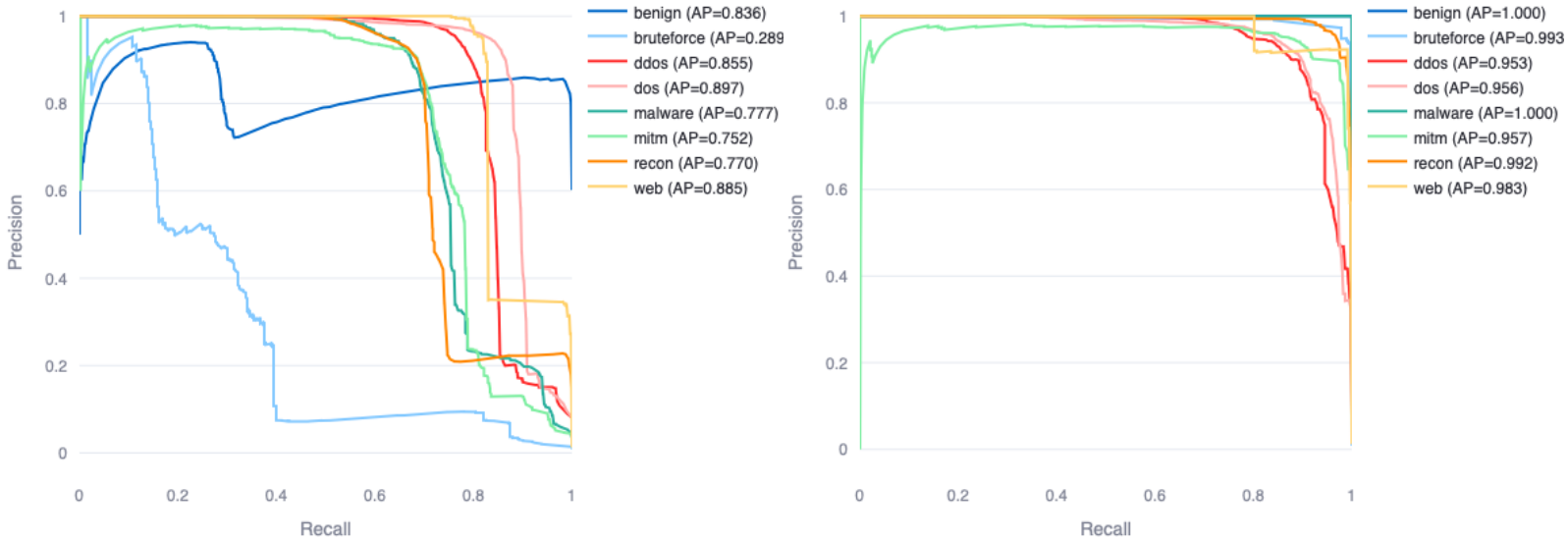


Figure 5.20 – Illustration of the PR curve comparison between RAW (left) and PP (right) for LR on CIC-IIoT under stratified splitting (60/20/20)

Key observations

Across both datasets, standardized preprocessing turns LR into a much lighter and faster model and, on the noisier CIC–IIoT dataset, a far more accurate one. On DNP3, PP mainly trims footprint, model size reduces from 201.7 KB to 6.5 KB (approx. 97%) with a one-third drop in effective features (RAW: 85, PP:55), while predictive metrics move only a few tenths of a point, accuracy +0.11 pp, F1 +0.13 pp, recall +0.14 pp. The tiny training-split dips indicate reduced overfitting rather than a loss of generalization. On CIC-IIoT, the impact is higher, test accuracy increases by +10.27 pp, F1 by +17.94 pp, and recall by +27.71 pp, with a modest precision gain (+4.12 pp) and a substantial ROC–AUC lift (+7.60 pp). These gains are mirrored in the figures, the RAW precision–recall curves bow and peel away at high recall, most notably bruteforce (AP=0.65) and recon (AP=0.89), while the PP curves concentrate near the upper right (AP=0.99 to 1.00 for most classes). The RAW confusion matrix shows heavy spillover of recon and DoS variants into benign. The PP matrix cleans the diagonal with only small residual errors. Efficiency follows suit, inference time drops by approx. 91 to 96% on DNP3 and approx. 99 to 99.5% on IIoT across all splits. Training time is essentially unchanged on DNP3 (+24% from a sub second baseline) but falls by 73% on CIC-IIoT and model size shrinks from 391.5 MB to 5.9 KB (approx. 100%) there.

Discussion

The tables and figures tell a coherent LR story, PP standardizes scales, removes or regularizes high cardinality identifiers and curbs one hot explosion, producing a well conditioned linear problem. On the clean, balanced DNP3 set, LR already ranks examples nearly perfectly (ROC–AUC approx. 100.00% in both regimes). PP therefore manifests mainly as efficiency, fewer and better behaved inputs reduce the parameter vector and accelerate dot product inference; the slight reductions on the training split suggest that PP strips away cues the RAW model could memorize without affecting validation test.

CIC–IIoT behaves differently because RAW features include heavy-tailed magnitudes and session identifier fields that a linear model can't safely exploit. They distort optimization and calibration, leading to poor recall. PP fixes these pathologies: after type safe casting, robust scaling and controlled encodings, class boundaries are closer to linearly separable, which is exactly where LR excels. That is why recall surges by approx. 28 pp and F1 by approx. 18 pp and why the PR curves shift toward the top right. The confusion matrices make the operational impact concrete, the RAW model mislabels large swaths of recon and DoS traffic as benign, while the PP model removes those off diagonals, retaining only minor bleed. At the same time, the LR footprint becomes negligible (hundreds of megabytes brcome a few kilobytes) and

per batch inference drops from approx. 0.37 s to approx. 0.002 s on CIC-IIoT test, an improvement that enables real-time deployment.

In short, for LR the standardized pipeline does not merely polish an already good classifier, on CIC-IIoT style data it unlocks the model by delivering well scaled, leakage free inputs, yielding large gains in accuracy while shrinking artifacts and latency to near-trivial levels and on DNP3 it preserves near ceiling performance with a smaller, faster and more defensible model.

5.2.6 Summary of the impact analysis

The table 5.14 summarizes the predictive performance across all models on the test subset. The table 5.15 summarizes the resource footprint across all models before and after A.R.C.'s standardized preprocessing under a 60/20/20 stratified split.
Across models and datasets, standardized preprocessing (PP) consistently improves robustness and comparability, with effect sizes driven by dataset cleanliness. On the more stable, balanced DNP3 dataset, predictive metrics shift only marginally under PP, often by a few hundredths, while efficiency typically improves (smaller models and faster inference) thanks to duplicate handling, feature pruning and consistent scaling. On CIC-IIoT, which exhibits stronger diversity and drift, PP delivers large, practically relevant gains, Minority classes are recovered, off diagonal confusion shrinks, and scores rise (e.g. for MLP the test F1 increases from 82.92% to 98.75% and recall from 75.56% to 99.16%, with PR curves moving to the upper right for previously confused minority attacks). These effects were observed under paired Raw vs. PP runs with identical splits and tuning budgets, following the evaluation plan in Chapter 4.

Efficiency impacts depend on model family. Tree ensembles and linear models typically see reduced model size and steady or lower inference latency after PP; deep MLPs can incur higher training and inference cost on CIC-IIoT (despite smaller serialized artifacts) in exchange for much higher recall and precision on minority attacks.

Overall, the case study confirms the thesis premise, a standardized preparation catalog yields cleaner features, stronger minority detection and more reproducible comparisons, especially on noisier, more variable data.

Table 5.14 – Overview of resource footprint across all models before (RAW) and after A.R.C.'s standardized preprocessing (PP) under a 60/20/20 stratified split

KPI	Model	DNP3 RAW	DNP3 PP	DNP3 Δ	CIC-IIoT RAW	CIC-IIoT PP	CIC-IIoT Δ
Accuracy	RF	99,59 %	99,79 %	+0,20 pp	94,70 %	99,59 %	+4,90 pp
	GBDT	99,93 %	99,93 %	0,00 pp	99,97 %	99,74 %	-0,23 pp
	SVM	98,36 %	98,45 %	+0,08 pp	91,66 %	98,87 %	+7,20 pp
	MLP	99,80 %	99,65 %	-0,15 pp	90,40 %	99,49 %	+9,09 pp
	LR	99,11 %	99,22 %	+0,11 pp	86,87 %	97,15 %	+10,27 pp
F1	RF	99,59 %	99,80 %	+0,20 pp	91,08 %	99,10 %	+8,02 pp
	GBDT	99,93 %	99,93 %	0,00 pp	99,95 %	99,40 %	-0,55 pp
	SVM	98,36 %	98,47 %	+0,11 pp	84,11 %	97,31 %	+13,20 pp
	MLP	99,80 %	99,66 %	-0,14 pp	82,92 %	98,75 %	+15,84 pp
	LR	99,11 %	99,25 %	+0,13 pp	75,41 %	93,35 %	+17,94 pp
Precision	RF	99,60 %	99,80 %	+0,21 pp	97,11 %	98,87 %	+1,76 pp
	GBDT	99,93 %	99,93 %	0,00 pp	99,96 %	99,24 %	-0,72 pp
	SVM	98,41 %	98,55 %	+0,14 pp	93,98 %	96,50 %	+2,52 pp
	MLP	99,80 %	99,66 %	-0,14 pp	94,12 %	98,37 %	+4,25 pp
	LR	99,14 %	99,26 %	+0,12 pp	87,27 %	91,39 %	+4,12 pp
Recall	RF	99,59 %	99,79 %	+0,20 pp	86,56 %	99,34 %	+12,77 pp
	GBDT	99,93 %	99,93 %	0,00 pp	99,94 %	99,57 %	-0,37 pp
	SVM	98,37 %	98,50 %	+0,13 pp	78,20 %	98,20 %	+19,99 pp
	MLP	99,80 %	99,66 %	-0,14 pp	75,56 %	99,16 %	+23,61 pp
	LR	99,11 %	99,25 %	+0,14 pp	68,09 %	95,80 %	+27,71 pp
ROC-AUC	RF	99,99 %	99,99 %	0,00 pp	99,91 %	99,99 %	+0,08 pp
	GBDT	100,00 %	100,00 %	0,00 pp	100,00 %	100,00 %	0,00 pp
	SVM	99,94 %	99,97 %	+0,02 pp	93,85 %	99,96 %	+6,11 pp
	MLP	99,98 %	99,99 %	+0,01 pp	93,84 %	99,99 %	+6,15 pp
	LR	99,98 %	99,97 %	-0,01 pp	92,20 %	99,79 %	+7,60 pp
Inference time	RF	0,0171 s	0,0063 s	-62,9 %	0,5278 s	0,1551 s	-70,6 %
	GBDT	0,0088 s	0,0015 s	-82,5 %	0,1756 s	0,0238 s	-86,5 %
	SVM	0,2473 s	0,0690 s	-72,1 %	118,1575 %	14,3384 %	-87,9 %
	MLP	0,0113 s	0,0025 s	-77,9 %	0.4462 s	1,3130 s	+194,3 %
	LR	0,0105 s	0,0004 s	-96,4 %	0.3737 s	0,0019 s	-99,5 %

Table 5.15 – Overview of predictive performance across all models with (PP) and without (RAW) preprocessing on train under a 60/20/20 stratified split

Model	KPI	DNP3			CIC-IIoT		
		Raw	PP	Δ	Raw	PP	Δ
RF	**Training time**	0,335 s	0.284 s	-15,50 %	14.606 s	5,137 s	-64,83 %
	Model size	4,22 MB	2,44 MB	-42,08 %	411,90 MB s	18,66 MB	-94,47 %
	Feature count	85	55	-35,29 %	86	57	-33,72 %
GBDT	**Training time**	34,672 s	6,039 s	-82,58 %	1146,239 s	36,680 s	-96,80 %
	Model size	5,22 MB	4,76 MB	-8,76 %	55,80 MB	3,58 MB	-93,58 %
	Feature count	85	55	-35,29 %	86	57	-33,72 %
SVM	**Training time**	1,117 s	0,296 s	-73,47 %	19310.300 s	472.068 s	-97,56 %
	Model size	5,22 MB	583,91 KB	-83,69 %	420.16 MB	3.69 MB	-99,12 %
	Feature count	85	55	-35,29 %	86	57	-33,72 %
MLP	**Training time**	0,982 s	0,767 s	-21,83 %	133,903	433,403	+233,67 %
	Model size	1,26 MB	808,75 KB	-37,30 %	394,68 MB	23,69 MB	-94,00 %
	Feature count	85	76	-10,59 %	86	90	+4,65 %
LR	**Training time**	1,111 s	1,379 s	+24,15 %	30,853 s	8,442 s	-72,64 %
	Model size	201,65 KB	6,49 KB	-96,78 %	391,45 MB	5,91 KB	-100,00 %
	Feature count	85	55	-35,29 %	86	57	-33,72 %

5.3 Challenges, limitations and mitigations

Dataset availability and realism

Suitable and real datasets are scarce, most public sets contain synthetic campaigns or lab traffic. This limits external validity and risks overfitting to artefacts rather than operational behaviour. This is mitigated by, using two structurally distinct datasets, applying schema and label validators to surface artefacts (duplication, mixed types, bursty timing) and reporting split aware label coverage so readers can judge realism. Nevertheless, conclusions remain conditioned on these sources, releasing dataset cards and configuration manifests eases future replication on new data.

Application performance (Streamlit & session state)

All data and intermediates reside in Streamlit session_state and most computations run inside the app process. With large tables this causes memory pressure and UI lag. It is mitigated with lazy evaluation (only recomputing when inputs change), down-sampled previews and optional heavy visuals. Intermediate artefacts (features, splits, metrics) are keept to avoid rework across interactions. The current implementation remains constrained by single process Streamlit execution.

Comparisons require training on PP and RAW

Side by side comparisons double training cost, on large data (e.g. SVM and MLP for CIC-IIoT) RAW runs are especially slow (table 5.15). We mitigated by computing RAW on demand. Still, full RAW and PP recomputation is time consuming. In practice, PP can serve as the default, with RAW reserved for ablations.

Time based split and unseen labels

Strict chronological splits produced partitions with unseen classes, making per class metrics and PR and ROC curves undefined for those labels. Therefore time based results are treated as a diagnostic (how class mix evolves) and stratified splits for primary model comparisons were used. Mitigations include reporting label coverage per partition, reading metrics alongside time bin plots.

Metric stability and rare classes

Minority classes below 1% yield unstable per class estimates, small count changes can affec F1 and Recall. Macro averages are emphasized, confusion matrices and PR curves shown and top-25 features as families rather than causal drivers are

interpreted. Still, fine-grained class rankings should be read cautiously, where support is low.

Scope of learning methods

The current framework supports supervised models only, scenarios with sparse or no labels are out of scope. Planned mitigations (future work) include integrating unsupervised detectors (Local Outlier Factor, Isolation Forest) and adding a native CICFlowMeter path from PCAP to standardized flows to broaden applicability and reduce preprocessing friction.

Chapter 6

Conclusion and future work

This thesis addressed a persistent source of friction in cybersecurity ML, data processes that are ad hoc, leaky and hard to reproduce. In response, A.R.C. was build, a modular framework that standardizes the path from raw telemetry to auditable evaluation. It unifies ingestion, schema and label validation, split strategies that treat time as first class, a disciplined preprocessing catalogue, and an export layer that records every decision (splits, configs, models, metrics) for exact replay.

The case study demonstrates where standardization matters most. On DNP3, a already clean and balanced dataset, preprocessing leaves predictive metrics essentially unchanged while reducing model size and speeding inference. On CIC-IIoT, a noisier and imbalanced dataset, the same steps yield large, practical gains, recall and F1 rise, PR curves improve, and confusion off diagonals shrink, effects that persist under stratified and time-based splits, with transparent efficiency trade-offs.

Equally important are the guardrails, early schema checks prevent leakage; split audits surface unseen labels and drift, paired RAW vs. PP runs make fairness explicit and artifact exports turn results into shareable, verifiable study units. Beyond this case study, A.R.C.'s validators, split audits and artifact schema can serve as a community quality bar for dataset releases and paper submissions, yielding standardized dataset cards, split manifests and replayable configs. Applied consistently across labs, these conventions would enable cross dataset benchmarks and make replication packages first class research outputs rather than ad hoc supplements.These practices raise the floor for future work by making comparisons traceable and disagreements testable.

Limits remain. The datasets are synthetic, the study focuses on supervised learning and near default models, absolute scores should not be read as family ceilings. Still, the constraints are explicit and logged, clarifying attribution to the data process rather than the learner.

Next steps are clear, test A.R.C. on real data by ingesting anonymized field traces from operational ICS and smart grid networks, extend beyond supervised learning to unsupervised and continual settings under the same metric suite and integrate CICFlowMeter into the pipeline to generate standardized flow features from PCAPs with full provenance.

In short, careful preparation is not overhead but infrastructure and A.R.C. makes that infrastructure practical and reusable.

List of Figures

List of Tables

Bibliography

[1] R. H. Moulton, G. A. McCully, and J. D. Hastings, "Confronting the reproducibility crisis: A case study of challenges in cybersecurity ai," in *2024 Cyber Awareness and Research Symposium (CARS)*, 2024, pp. 1–6. DOI: 10.1109/CARS61786.2024.10778911.

[2] A. Sundararajan, T. Khan, A. Moghadasi, and A. I. Sarwat, "Survey on synchrophasor data quality and cybersecurity challenges, and evaluation of their interdependencies," *Journal of Modern Power Systems and Clean Energy*, vol. 7, no. 3, pp. 449–467, 2019. DOI: 10.1007/s40565-018-0473-6.

[3] M. M. Alani and T. Baker, "A survey of smart grid intrusion detection datasets," in *Workshop Proceedings of the 19th International Conference on Intelligent Environments (IE2023)*. 2023, pp. 5–13.

[4] R. Haluška, J. Brabec, and T. Komárek, "Benchmark of data preprocessing methods for imbalanced classification," in *2022 IEEE International Conference on Big Data (Big Data)*, 2022, pp. 2970–2979. DOI: 10.1109/BigData55660.2022.10021118.

[5] Canadian Institute for Cybersecurity, *Cic iiot dataset 2025*, University of New Brunswick, 2025. [Online]. Available: https://www.unb.ca/cic/datasets/iiot-dataset-2025.html.

[6] P. Radoglou-Grammatikis, V. Kelli, T. Lagkas, V. Argyriou, and P. Sarigiannidis, *Dnp3 intrusion detection dataset*, 2022. DOI: 10.21227/s7h0-b081. [Online]. Available: https://dx.doi.org/10.21227/s7h0-b081.

[7] *Tensorflow*, https://www.tensorflow.org/, Accessed: 2025-10-01.

[8] *Kubeflow*, https://www.kubeflow.org/, Accessed: 2025-10-01.

[9] *Mlflow*, https://mlflow.org/, Accessed: 2025-10-01.

[10] *Apache airflow*, https://airflow.apache.org/, Accessed: 2025-10-01.

[11] J. C. Mondragon, P. Branco, G. V. Jourdan, et al., "Advanced ids: A comparative study of datasets and machine learning algorithms for network flow-based intrusion detection systems," *Applied Intelligence*, vol. 55, no. 608, 2025. DOI: 10.1007/s10489-025-06422-4.

[12] R. Croft, M. A. Babar, and M. M. Kholoosi, "Data quality for software vulnerability datasets," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 121–133. DOI: 10.1109/ICSE48619.2023.00022.

[13] B. Wu, D. M. Divakaran, and M. Gurusamy, "Uninet: A unified multi-granular traffic modeling framework for network security," *IEEE Transactions on Cognitive Communications and Networking*, pp. 1–1, 2025. DOI: 10.1109/TCCN.2025.3585170.

[14] M. Di Mauro, G. Galatro, G. Fortino, and A. Liotta, "Supervised feature selection techniques in network intrusion detection: A critical review," *Engineering Applications of Artificial Intelligence*, vol. 101, p. 104 216, 2021, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2021.104216.

[15] D. Pinto, I. Amorim, E. Maia, and I. Praça, "A review on intrusion detection datasets: Tools, processes, and features," *Computer Networks*, vol. 262, p. 111 177, 2025, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2025.111177.

[16] N. Moustafa and J. Slay, "Unsw-nb15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942.

[17] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani, et al., "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," *ICISSp*, vol. 1, no. 2018, pp. 108–116, 2018.

[18] F. Cremer et al., "Cyber risk and cybersecurity: A systematic review of data availability," *The Geneva papers on risk and insurance. Issues and practice*, vol. 47, no. 3, p. 698, 2022.

[19] Z. Yang et al., "A systematic literature review of methods and datasets for anomaly-based network intrusion detection," *Computers & Security*, vol. 116, p. 102 675, 2022.

[20] M. A. Bouke and A. Abdullah, "An empirical study of pattern leakage impact during data preprocessing on machine learning-based intrusion detection models reliability," *Expert Systems with Applications*, vol. 230, p. 120 715, 2023.

[21] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE symposium on computational intelligence for security and defense applications*, Ieee, 2009, pp. 1–6.

[22] Z. Wu, X. Zhou, X. Lu, L. Yang, S. Shen, and D. Yan, "Research on encrypted malicious traffic detection in power information interaction: Application of the electricity multi-granularity flow representation learning approach," *Scientific Reports*, vol. 15, no. 1, p. 17 766, 2025.

[23] C. C. Aggarwal, "An introduction to outlier analysis," in *Outlier analysis*, Springer, 2016, pp. 1–34.

[24] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[25] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[26] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[27] G. Ke et al., "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[28] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.