

asn_ml_risk.py — Machine Learning for ASN Vulnerability & Propagation Risk

1. Purpose & Role in the Platform

`asn_ml_risk.py` builds a machine-learning–based risk score per ASN, estimating how likely an ASN is to be:

- **Vulnerable** (weak origin validation and poor routing hygiene), and
- **unsafe as a source for forged or hijacked BGP announcements**, considering both:
 - **origin-security and acceptance controls** (RPKI coverage, IRR hygiene, strict filtering behavior, ROV enforcement), and
 - **propagation characteristics** (observed reachability and AS-PATH quality).

The output is a **continuous risk score in the range [0, 1]**, where:

- **0.0** indicates *low vulnerability and propagation risk* (strong controls, good hygiene), and
- **1.0** indicates *high vulnerability and propagation risk* (weak controls, poor hygiene).

The model follows a **weakly supervised learning approach**:

- High-confidence **GOOD / BAD labels** are generated automatically using strict, multi-metric rules applied only to ASNs with complete and reliable data.
- A **LightGBM classifier** is then trained on these extreme cases to learn a smooth, probabilistic risk function that generalizes to all ASNs.

In addition to the raw risk score, the pipeline explicitly models **evidence quality** by tracking how complete the underlying security data is for each ASN.

The resulting scores are persisted in the database in a dedicated table:

```
asn_ml_risk(  
    asn,  
    risk_score,  
    data_completeness,  
    score_confidence,  
    label_note,  
    updated_at  
)
```

Where:

- `risk_score` is the ML-derived vulnerability/ propagation risk in [0,1],
- `data_completeness` quantifies the fraction of security signals available for that ASN,
- `score_confidence` provides a qualitative confidence level (LOW / MEDIUM / HIGH) derived from data completeness,
- `label_note` records the supervision regime used (`weak_supervision_same_pot`),
- `updated_at` records the UTC timestamp of score computation.

This score, together with its associated evidence-quality metadata, is then available to higher-level components (dashboards, APIs, reports) as the **final, synthetic ASN-level vulnerability and propagation-risk indicator**.

2. Data Sources & Features

2.1 Base ASN Dataset

The script expects a populated `asn_data` table (main ASN inventory and feature store).
If `asn_data` does not exist or is empty, the script exits with an error.

From `asn_data`, it uses:

- `asn`
- `reachability_propagation`
- `as_path_len`

If `reachability_propagation` or `as_path_len` are missing, the script creates them as `NaN` and later fills them with defaults.

The initial base frame is:

```
base = asn_data[["asn", "reachability_propagation", "as_path_len"]]
```

2.2 Joining Additional Feature Tables

The script then augments this base frame with additional per-ASN features from other tables:

- From `rpki_results`:
 - `rpki_coverage_pct`
(RPKI coverage ratio per ASN – fraction of its originated space covered by valid ROAs)
- From `irr_results`:
 - `irr_coverage_exact_pct` → renamed to `irr_routeobject_coverage`
(exact IRR route-object coverage for that ASN's prefixes)
- From `asn_filter_features`:
 - `strict_filter_score` → renamed to `bgp_filters_strict`
(strict filtering score, i.e., how aggressively the ASN enforces BGP hygiene / anti-leak / anti-bogon policies)
- From `asn_rov_score` (control-plane ROV module):

- `rov_enforcement_score`
(empirical ROV enforcement score in `[0, 1]` derived from observed blocked/leaked invalids vs opportunities)

All merges are left-joins on `asn`.

If a table does not exist, the corresponding column is created as `NaN` and handled later.

At this point, the feature set includes:

- `asn`
- `reachability_propagation`
- `as_path_len`
- `rpki_coverage_pct`
- `irr_routeobject_coverage`
- `bgp_filters_strict`
- `rov_enforcement_score`

2.3 Normalization & Missing Data Handling

This stage ensures that all ASN-level features are numerically consistent, robust to heterogeneous data sources, and safe for both automatic labeling and ML training.

The design explicitly separates **numeric value normalization** from **data availability signaling**, avoiding biased assumptions about missing data.

RPKI scale normalization

If `rpki_coverage_pct` exists and its maximum value is greater than 1.5, the script assumes the metric is expressed on a 0..100 scale and rescales it to 0..1:

```
if base["rpki_coverage_pct"].max() > 1.5:  
    base["rpki_coverage_pct"] = base["rpki_coverage_pct"] / 100.0
```

This guarantees consistent interpretation of RPKI coverage values across heterogeneous upstream data sources.

Numeric coercion

All feature columns are explicitly converted to numeric dtype:

```
base[col] = pd.to_numeric(base[col], errors="coerce")
```

Any malformed, non-numeric, or unexpected values are converted to **NaN**.

Infinity handling

Positive and negative infinity values are replaced with **NaN** to prevent contamination of the feature space:

```
base = base.replace([np.inf, -np.inf], np.nan)
```

Missingness indicators

Before any imputation is applied, the script generates explicit binary indicators capturing whether each original signal was observed:

```
base["has_rpki"] = base["rpki_coverage_pct"].notna().astype(int)
```

```
base["has_irr"] =  
base["irr_routeobject_coverage"].notna().astype(int)
```

```
base["has_rov"] = base["rov_enforcement_score"].notna().astype(int)
```

```
base["has_strict"] = base["bgp_filters_strict"].notna().astype(int)
```

```
base["has_reach"] =  
base["reachability_propagation"].notna().astype(int)
```

```
base["has_aspath"] = base["as_path_len"].notna().astype(int)
```

These indicators allow the model to distinguish between:

- **genuinely low observed values**, and
- **values introduced through imputation due to missing data**.

Missingness is therefore preserved as a **first-class signal**, not erased during preprocessing.

Evidence quality layer (data completeness & confidence)

In addition to per-feature missingness indicators, the pipeline explicitly models **overall evidence quality** for each ASN.

A **data completeness score** is computed as the mean availability of all six core signals:

```
data_completeness =  
mean(has_rpki, has_irr, has_rov, has_strict, has_reach, has_aspath)
```

Based on this value, a qualitative **score confidence** level is assigned using a conservative mapping:

- **HIGH** if $\geq 5 / 6$ signals are present
- **MEDIUM** if $\geq 3 / 6$ signals are present
- **LOW** otherwise

This layer does not alter the numeric feature values used by the model.

Instead, it provides **explicit, interpretable metadata** describing how much empirical evidence supports each ASN's risk score.

Imputation rules

Missing numeric values are filled using robust median-based imputation, computed independently for each feature:

```
rpki_med    = median(rpki_coverage_pct)

irr_med     = median(irr_routeobject_coverage)

rov_med     = median(rov_enforcement_score)

strict_med  = median(bgp_filters_strict)

reach_med   = median(reachability_propagation)

aspath_med  = median(as_path_len)


fill_map = {

    "rpki_coverage_pct":      rpki_med,

    "irr_routeobject_coverage": irr_med,

    "rov_enforcement_score":  rov_med,

    "bgp_filters_strict":    strict_med,

    "reachability_propagation": reach_med,

    "as_path_len":          aspath_med,

}
```

Each median is computed only from valid observed values and is guarded against NaN or infinite results.

Rationale

- Missing values are **not interpreted as worst-case insecurity**.

- Forced zero-imputation is deliberately avoided to prevent systematic bias against ASNs with incomplete observability.
 - Uncertainty is modeled explicitly via `has_*` indicators and evidence-quality metadata rather than implicitly via extreme numeric defaults.
 - Median imputation preserves the global feature distribution and is robust to outliers.
 - This design allows the model to learn whether **absence of data itself carries predictive signal**, without conflating missingness with poor security posture.
-

Resulting feature matrix

After normalization and imputation, the resulting feature frame (`base`) contains one row per ASN with all features fully numeric and no remaining `NaN` values.

Core features:

- `rpki_coverage_pct`
- `rov_enforcement_score`
- `irr_routeobject_coverage`
- `bgp_filters_strict`
- `reachability_propagation`
- `as_path_len`

Missingness indicators:

- `has_rpki`
- `has_irr`
- `has_rov`

- `has_strict`
- `has_reach`
- `has_aspath`

Evidence quality metadata:

- `data_completeness`
- `score_confidence`

This feature matrix is used consistently and unchanged for:

- **automatic label generation (weak supervision)**, and
 - **final ML inference and risk scoring**.
-

3. Automatic Label Generation (Weak Supervision)

The function `auto_label(df)` constructs a **high-precision weak-supervision** training set by labeling only **extreme ASNs** that clearly belong to one of two classes:

- **GOOD (label = 0)**: low hijack feasibility / low final impact
- **BAD (label = 1)**: high hijack feasibility / high final impact

Labels are intentionally conservative: **only ASNs with complete observability across all labeling dimensions are considered**, and **only those meeting strict extreme criteria are labeled**. All other ASNs remain unlabeled and are scored later via probabilistic inference by the trained model.

This version uses an “**all-metrics / same-pot**” design: the labeling criteria treat **acceptance friction**, **hygiene posture**, and **propagation capability** symmetrically, reflecting an end-to-end threat model focused on final impact.

3.1 Labeling Eligibility (High-Precision Filter)

Automatic labels are generated only for ASNs that have **complete data** for all signals used in labeling. An ASN is eligible for labeling only if the following signals are present:

- `rpki_coverage_pct`
- `irr_routeobject_coverage`
- `rov_enforcement_score`
- `bgp_filters_strict`
- `reachability_propagation`
- `as_path_len`

Formally, the eligible subset is constructed as:

```
eligible = df[
    (df["has_rpki"] == 1)
    & (df["has_irr"] == 1)
    & (df["has_rov"] == 1)
    & (df["has_strict"] == 1)
    & (df["has_reach"] == 1)
    & (df["has_aspath"] == 1)
].copy()
```

ASNs missing any of these signals are excluded from labeling and are never assigned GOOD or BAD labels. They are evaluated later only through probabilistic inference by the trained model.

If no eligible ASNs exist, the function returns an empty labeled set and (optionally) prints:

```
[LABEL] eligible (complete data): 0
```

3.2 Propagation Thresholds (Distribution-Adaptive Quantiles)

Unlike fixed cutoff designs, the propagation-related thresholds in this labeling scheme are derived from the **empirical distribution** of eligible ASNs, avoiding brittle hardcoding across datasets.

The script computes:

- `reach_hi`: 75th percentile of `reachability_propagation`
- `reach_lo`: 25th percentile of `reachability_propagation`
- `aspath_lo`: 25th percentile of `as_path_len`
- `aspath_hi`: 75th percentile of `as_path_len`

Implementation:

```
reach_hi = _safe_quantile(eligible["reachability_propagation"], 0.75,  
fallback=0.55)
```

```
reach_lo = _safe_quantile(eligible["reachability_propagation"], 0.25,  
fallback=0.25)
```

```
aspath_lo = _safe_quantile(  
    eligible["as_path_len"], 0.25,  
    fallback=_safe_median(eligible["as_path_len"], 4.0)  
)
```

```
aspath_hi = _safe_quantile(  
    eligible["as_path_len"], 0.75,  
    fallback=_safe_median(eligible["as_path_len"], 4.0)  
)
```

This ensures that “high propagation” and “low propagation” are defined **relative to the observed population**, while retaining safe fallback behavior if quantile computation fails.

3.3 “BAD” ASNs (label = 1)

An eligible ASN is labeled **BAD (label = 1)** when it simultaneously satisfies:

Weak acceptance friction (permissive environment):

- `bgp_filters_strict <= 0.30`
- `rov_enforcement_score <= 0.30`

Weak hygiene posture:

- `rpki_coverage_pct <= 0.10`
- `irr_routeobject_coverage <= 0.30`

Strong propagation capability (distribution-aware):

- `reachability_propagation >= reach_hi`
- `as_path_len <= aspath_lo`

Formally:

```
bad = (  
  (eligible["bgp_filters_strict"] <= 0.30)  
  & (eligible["rov_enforcement_score"] <= 0.30)  
  & (eligible["rpki_coverage_pct"] <= 0.10)  
  & (eligible["irr_routeobject_coverage"] <= 0.30)  
  & (eligible["reachability_propagation"] >= reach_hi)
```

```
    & (eligible["as_path_len"] <= aspath_lo)
)
```

Interpretation: these ASNs represent environments where a forged announcement is likely to experience **low resistance at acceptance points** while also possessing **strong structural conditions for rapid and wide propagation**.

3.4 “GOOD” ASNs (label = 0)

An eligible ASN is labeled **GOOD (label = 0)** when it simultaneously satisfies:

Strong acceptance friction (restrictive environment):

- `bgp_filters_strict >= 0.80`
- `rov_enforcement_score >= 0.60`

Strong hygiene posture:

- `rpki_coverage_pct >= 0.80`
- `irr_routeobject_coverage >= 0.70`

Weak propagation capability (distribution-aware):

- `reachability_propagation <= reach_lo`
- `as_path_len >= aspath_hi`

Formally:

```
good = (
    (eligible["bgp_filters_strict"] >= 0.80)
    & (eligible["rov_enforcement_score"] >= 0.60)
    & (eligible["rpki_coverage_pct"] >= 0.80)
```

```
& (eligible["irr_routeobject_coverage"] >= 0.70)

& (eligible["reachability_propagation"] <= reach_lo)

& (eligible["as_path_len"] >= aspath_hi)

)
```

Interpretation: these ASNs represent environments where forged announcements face **high acceptance friction** and also lack strong structural conditions for wide propagation, yielding **low end-to-end hijack impact**.

3.5 Label Assignment and Training Set Construction

After computing `good` and `bad`, the labeled training set is built as:

```
df_labeled = eligible[good | bad].copy()

df_labeled["label"] = np.where(bad.loc[df_labeled.index], 1, 0)
```

Only ASNs that are clearly GOOD or clearly BAD are used for supervised training. All intermediate or ambiguous ASNs are intentionally excluded from labeling to avoid introducing noise into the training process.

3.6 Labeling Transparency and Debug Output

For transparency, when `verbose=True`, the labeling function prints:

- total number of ASNs
- number of eligible ASNs (complete data)
- propagation thresholds used (`reach_lo`, `reach_hi`, `aspath_lo`, `aspath_hi`)
- number of GOOD ASNs (label = 0)
- number of BAD ASNs (label = 1)
- total labeled ASNs used for training

Output format:

```
print(f"[LABEL] total ASNs: {len(df)}")

print(f"[LABEL] eligible (complete data): {len(eligible)}")

print(
    f"[LABEL] thresholds: reach_lo={reach_lo:.3f},
    reach_hi={reach_hi:.3f}, "
    f"aspath_lo={aspath_lo:.3f}, aspath_hi={aspath_hi:.3f}"
)

print(f"[LABEL] labeled GOOD (0): {(df_labeled['label'] == 0).sum()}")
print(f"[LABEL] labeled BAD (1): {(df_labeled['label'] == 1).sum()}")
print(f"[LABEL] used for training: {len(df_labeled)}")
```

This provides immediate visibility into label coverage, threshold adaptation behavior, and class balance.

4. Model Training (LightGBM)

The ASN vulnerability classifier is trained using **LightGBM gradient-boosted decision trees**, optimized for tabular risk modeling under weak supervision.

The training process operates exclusively on high-confidence labeled ASNs generated by the automatic labeling logic described in Section 3.

4.1 Feature Vector

The model operates on a unified, per-ASN feature vector in which **all metrics are treated symmetrically**.

No manual prioritization or weighting is imposed; instead, the model learns the relative importance of each signal directly from data.

The full feature set used for both training and inference is:

```
feature_cols = [  
    "bgp_filters_strict",  
    "rov_enforcement_score",  
    "reachability_propagation",  
    "as_path_len",  
    "rpki_coverage_pct",  
    "irr_routeobject_coverage",  
    "has_rpki",  
    "has_irr",  
    "has_rov",  
    "has_strict",  
    "has_reach",  
    "has_aspath",  
]
```

Feature categories

Although all features are jointly learned, they can be conceptually grouped as follows:

Acceptance and containment signals

- `bgp_filters_strict` — observed strictness of inbound prefix filtering
- `rov_enforcement_score` — observed ROV enforcement behavior at ASN boundaries

Propagation capability signals

- `reachability_propagation` — observed global propagation strength of announcements
- `as_path_len` — observed AS-PATH length (shorter paths imply stronger reach)

Hygiene and policy posture signals

- `rpki_coverage_pct` — fraction of originated prefixes covered by valid ROAs
- `irr_routeobject_coverage` — fraction of originated prefixes covered by IRR route objects

Missingness indicators

- `has_rpki`
- `has_irr`
- `has_rov`
- `has_strict`
- `has_reach`
- `has_aspath`

These binary indicators explicitly encode whether each signal was originally observed or imputed, allowing the model to distinguish between:

- genuinely low observed values, and
- values introduced through median-based imputation due to incomplete observability.

No feature is excluded from training based on category; all signals participate jointly in learning the final risk function.

4.2 Train / Validation Split

The automatically labeled dataset is split into training and validation subsets using a **stratified random split**:

```
X_train, X_val, y_train, y_val = train_test_split(
    X,
    y,
    test_size=0.25,
    random_state=random_state,
    stratify=y,
)
```

Split properties:

- **25%** of labeled ASNs are reserved for validation
- `stratify=y` preserves the GOOD / BAD class ratio across splits
- `random_state` ensures reproducibility across runs

Only **high-confidence labeled ASNs** (as defined in Section 3) are included in this split. Unlabeled ASNs never participate in training or validation.

4.3 LightGBM Configuration

The classifier is trained using LightGBM with gradient-boosted decision trees.

Core model parameters:

```
params = {
    "objective": "binary",
    "metric": ["auc", "binary_logloss"],
```

```
"boosting_type": "gbdt",  
"learning_rate": 0.05,  
"num_leaves": 31,  
"feature_fraction": 0.9,  
"bagging_fraction": 0.9,  
"bagging_freq": 5,  
"seed": random_state,  
"verbose": -1,  
}
```

Training configuration:

- `num_boost_round = 800`
- `early_stopping_rounds = 80`, evaluated on the validation set
- Evaluation metrics: **ROC AUC** and **binary logloss**
- Optional `verbose_eval = 50` when verbose output is enabled

This configuration balances expressiveness and regularization, making it well-suited for learning non-linear interactions between acceptance, hygiene, and propagation signals under weak supervision.

4.4 Validation Metrics and Outputs

After training, the model evaluates performance on the validation set:

Probability prediction:

```
y_val_pred = model.predict(X_val, num_iteration=model.best_iteration)
```

ROC AUC computation:

```
auc = roc_auc_score(y_val, y_val_pred)
```

Optional classification report (threshold = 0.5):

```
y_val_bin = (y_val_pred >= 0.5).astype(int)

print(classification_report(y_val, y_val_bin, digits=3))
```

The training function returns:

- `model` — the trained LightGBM classifier
- `auc` — validation ROC AUC score

These metrics quantify how effectively the model separates **high hijack-feasibility ASNs** from **low hijack-feasibility ASNs** under the assumed end-to-end threat model.

5. Predicting Risk for All ASNs

Once trained, the model is applied to **all ASNs**, not only the ones used for training.

This ensures that every ASN in the database receives a vulnerability and propagation-risk assessment, including those that were not eligible for automatic labeling.

The function `predict_all_asns(model, df_all)` performs the following steps:

Feature extraction

The function extracts the full per-ASN feature matrix from `df_all` using **the exact same `feature_cols` vector** employed during model training.

This guarantees strict feature consistency between training and inference.

Risk probability estimation

The trained LightGBM model is then applied to all ASNs to compute `proba_bad`, the estimated probability that an ASN belongs to the **BAD (high-risk)** cluster learned during weak supervision:

```
proba_bad = model.predict(X_all, num_iteration=model.best_iteration)
```

This value lies in the range **[0, 1]** and represents a smooth, probabilistic risk estimate rather than a hard classification.

Result frame construction

The function builds a result frame that combines the ML-derived risk score with explicit evidence-quality metadata:

```
result = df_all[[
    "asn",
    "data_completeness",
    "score_confidence"
]].copy()

result["asn_risk_ml"] = proba_bad
```

The resulting output contains, for each ASN:

- `asn` — the autonomous system number,
- `asn_risk_ml` — the continuous ML risk score in [0,1],

- `data_completeness` — the fraction of underlying security signals available for that ASN,
 - `score_confidence` — a qualitative confidence level (LOW / MEDIUM / HIGH) derived from data completeness.
-

Interpretation of the risk score

The field `asn_risk_ml` is interpreted as an **approximate probability** that an ASN behaves similarly to the high-risk (BAD) extreme cluster defined during weak supervision, from a combined **vulnerability and propagation-risk** perspective.

The accompanying evidence-quality fields (`data_completeness`, `score_confidence`) provide essential context for interpreting this probability, allowing downstream consumers to distinguish between:

- high-risk scores supported by strong empirical evidence, and
- high or low scores produced under limited data availability.

Together, these outputs form a complete, interpretable ASN-level risk assessment suitable for operational use, analytics, and reporting.

6. Writing Scores to the Database

The function `write_scores_to_db(conn, df_scores)` persists the final ML-derived ASN vulnerability and propagation-risk scores into the SQLite database using idempotent UPSERT semantics.

This guarantees safe re-execution, atomic updates, and consistent storage of model outputs across runs.

6.1 Table Definition

Before inserting any data, the function ensures that the destination table exists with the full, current schema:

```
CREATE TABLE IF NOT EXISTS asn_ml_risk (  
    asn                INTEGER PRIMARY KEY,  
    risk_score         REAL NOT NULL,  
    data_completeness REAL NOT NULL,  
    score_confidence   TEXT NOT NULL,  
    label_note         TEXT,  
    updated_at         TEXT NOT NULL  
);
```

This schema enforces the following guarantees:

- exactly one unique row per ASN (`asn` as primary key),
- atomic updates on repeated executions,
- explicit persistence of both risk score and evidence-quality metadata,
- provenance and annotation support via the `label_note` field.

6.1.1 Lightweight Migration Guard

To support backward compatibility with existing deployments, the function performs a lightweight schema check at runtime:

- `PRAGMA table_info(asn_ml_risk)` is used to inspect existing columns,
- missing columns (`data_completeness`, `score_confidence`) are added automatically via `ALTER TABLE` with safe default values.

This allows the script to be executed seamlessly on databases created with earlier schema versions, without manual migrations.

6.2 Timestamp Generation

At write time, a current UTC timestamp is generated using ISO 8601 format with a **Z** suffix:

```
now = datetime.utcnow().isoformat(timespec="seconds") + "Z"
```

This timestamp represents the exact moment when the vulnerability risk score was computed and persisted, enabling precise temporal tracking of score updates.

6.3 Row Construction

For each ASN, a row is constructed with the following structure:

```
(  
    asn,  
    asn_risk_ml,  
    data_completeness,  
    score_confidence,  
    "weak_supervision_same_pot",  
    now  
)
```

Where:

- **asn** is the autonomous system number,
- **asn_risk_ml** is the continuous vulnerability / propagation-risk score output by the trained LightGBM model,

- `data_completeness` quantifies the fraction of available security signals used for that ASN,
- `score_confidence` is a qualitative confidence level (`LOW`, `MEDIUM`, `HIGH`) derived from data completeness,
- `label_note` is explicitly set to the constant string `"weak_supervision_same_pot"`, documenting the supervision and labeling regime used,
- `updated_at` records the UTC timestamp of score generation.

The `label_note` field is intentionally populated for all rows to ensure traceability and auditability.

6.4 UPSERT Semantics

Insertion uses SQLite's UPSERT logic to guarantee safe repeatability:

```
INSERT INTO asn_ml_risk
    (asn, risk_score, data_completeness, score_confidence, label_note,
    updated_at)
VALUES (?, ?, ?, ?, ?, ?)
ON CONFLICT(asn) DO UPDATE SET
    risk_score          = excluded.risk_score,
    data_completeness  = excluded.data_completeness,
    score_confidence   = excluded.score_confidence,
    label_note         = excluded.label_note,
    updated_at         = excluded.updated_at;
```

This ensures that:

- the first execution inserts exactly one row per ASN,
 - subsequent executions update all relevant fields atomically,
 - duplicate rows cannot be created, regardless of how many times the script is run.
-

6.5 Operational Implications

- The pipeline is fully idempotent and safe to re-run as input data or model parameters evolve.
- Risk scores and evidence-quality metadata are always updated together, ensuring internal consistency.
- The database remains in a valid, query-ready state after every execution.

This enables future extensions such as:

- distinguishing between different supervision or labeling regimes,
 - tracking model evolution over time,
 - performing controlled comparisons across experimental runs.
-

6.6 Design Rationale

Unlike earlier designs where annotation fields were optional or unused, the current implementation explicitly persists:

- the risk score,
- the quality of underlying evidence,
- and the supervision regime used to produce it.

As a result, the database functions not only as a score store, but also as a lightweight provenance and evidence layer for ASN-level vulnerability and propagation-risk assessments, supporting long-term maintainability, governance, and operational trust.

Important note : Handling of Missing or Incomplete ASN Data

ASNs with missing, incomplete, or partially unavailable security data are handled explicitly and transparently by design.

In BGP security analysis, missing information about an ASN does not imply malicious intent or poor operational practices.

At the same time, it does not provide sufficient evidence to assert that the ASN operates in a secure or well-filtered environment.

From a defensive routing-risk assessment perspective, incomplete observability represents uncertainty, not inherent risk.

The model therefore avoids both optimistic (“safe by default”) and pessimistic (“unsafe by default”) assumptions.

Missing values are not automatically interpreted as weak security and are not forced to worst-case values.

Instead, missingness is modeled explicitly through dedicated binary indicators (`has_*` features), allowing the classifier to learn whether the *absence of evidence itself* carries predictive value.

This design prevents systematic bias against ASNs with limited data coverage while still ensuring that lack of reliable security evidence does not automatically result in a low-risk classification.

As a result, missing data influences the predicted risk probabilistically, based on learned patterns in the data, rather than through hard-coded assumptions.

The model does not assume that missing data causes attacks.

It assumes only that incomplete evidence introduces uncertainty, which is reflected in the resulting risk score without forcing a definitive judgment.

7. Command-Line Interface & Execution Flow

The script exposes a **minimal, deterministic command-line interface** designed for batch execution in automated pipelines.

All execution logic is encapsulated in a single entry point (`main()`) and requires **no manual interaction** once invoked.

The pipeline is **single-shot**, fully automated, and safe to execute repeatedly.

7.1 Command-Line Arguments

The following command-line arguments are supported:

`--db` (required)

Path to the SQLite database file containing all ASN-level feature tables (e.g., `asn_data.db`).

The database must already contain the upstream-derived tables required by the model, including (but not limited to):

- `asn_data`
- `rpki_results`
- `irr_results`
- `asn_filter_features`
- `asn_rov_score`

If the database or required tables are missing, the script terminates immediately with an explicit error.

`--min-labeled` (default: 200)

Minimum number of automatically labeled ASNs (**GOOD + BAD**) required to proceed with model training.

If the number of labeled ASNs produced by the automatic labeling stage is below this threshold, the script exits with an error to prevent training on insufficient or statistically unstable supervision.

This guard ensures robustness and prevents accidental training on degenerate datasets.

--print-summary

If set, prints a short summary of the **top 20 ASNs with the highest predicted vulnerability risk score** after inference.

This option is intended for:

- quick sanity checks,
- exploratory inspection,
- operational visibility.

It **does not affect** model training, inference, or persistence.

7.2 Execution Flow

The complete execution flow is implemented in the `main()` function:

```
def main():
    args = parse_args()
    conn = sqlite3.connect(args.db)

    print("[ML] Loading ASN features...")
    df_all = load_asn_features(conn)

    print("[ML] Generating automatic labels (high precision; all
metrics)...")
    df_labeled = auto_label(df_all, verbose=True)

    if df_labeled.empty or "label" not in df_labeled.columns:
        print("[ERR] No labeled samples produced by auto_label().",
file=sys.stderr)
        print("[HINT] Ensure your source tables are populated and
thresholds are not too strict.", file=sys.stderr)
        sys.exit(1)
```

```

    if pd.Series(df_labeled["label"]).nunique() < 2:
        counts =
df_labeled["label"].value_counts(dropna=False).to_dict()
        print(f"[ERR] auto_label() produced only one class: {counts}",
file=sys.stderr)
        print("[HINT] Relax thresholds or switch to a relaxed labeling
mode to get both GOOD and BAD samples.", file=sys.stderr)
        sys.exit(1)

    if len(df_labeled) < args.min_labeled:
        print(
            f"[ERR] Too few automatically labeled ASNs
({len(df_labeled)} < {args.min_labeled}).",
            file=sys.stderr,
        )
        print("[HINT] Lower --min-labeled, relax thresholds, or ensure
all source tables are populated (including AS_PATH).")
        sys.exit(1)

    print("[ML] Training LightGBM model...")
    model, auc = train_model(df_labeled, verbose=True)
    print(f"[ML] Model trained. Validation AUC = {auc:.4f}")

    print("[ML] Predicting ML risk score for all ASNs...")
    df_scores = predict_all_asns(model, df_all)

    print("[ML] Writing scores into DB...")
    write_scores_to_db(conn, df_scores)

    conn.close()

    if args.print_summary:
        print("[ML] Example scores (top 20):")
        print(
            df_scores.sort_values("asn_risk_ml", ascending=False)
            .head(20)
            .to_string(index=False)

```

```
)  
  
print("[ML] Done. Scores stored in asn_ml_risk")
```

7.3 Pipeline Semantics

The script executes a **single-shot, end-to-end pipeline** with the following semantics:

Feature Loading

- Loads all ASN-level features from the SQLite database.
 - Applies normalization, missing-data handling, and feature preparation exactly as described in Sections 2 and 3.
-

Automatic Label Generation

- Applies **high-precision weak supervision** using the *all-metrics / same-pot* labeling strategy.
 - Labels only extreme ASNs as **GOOD** or **BAD**.
 - Leaves all ambiguous ASNs unlabeled.
-

Safety Validation

Before training, the script enforces **three mandatory safety checks**:

1. Labeled data exists (`df_labeled` is non-empty).
2. Both classes are present (GOOD and BAD).
3. The number of labeled ASNs meets `--min-labeled`.

Any violation causes **immediate termination with explicit diagnostics**.

Model Training

- Trains a LightGBM classifier using **only high-confidence labeled ASNs**.
- Uses stratified train/validation split, early stopping, and regularization.
- Reports validation ROC AUC for transparency.

Inference

- Applies the trained model to **all ASNs**, including unlabeled ones.
- Produces a continuous vulnerability risk score in `[0, 1]`.

Persistence

- Stores results in the `asn_ml_risk` table.
- Uses **UPSERT semantics**, making repeated executions safe and idempotent.
- Each run updates `risk_score`, `label_note`, and `updated_at`.

Optional Reporting

- When `--print-summary` is enabled, prints the top 20 highest-risk ASNs.
- This step is observational only and does not affect stored results.

7.4 Operational Characteristics

- No interactive input is required.

- The pipeline is deterministic given:
 - the same database state,
 - the same random seed.
 - Safe to run repeatedly (idempotent database updates).
 - All error conditions fail fast with explicit, actionable messages.
 - Suitable for:
 - cron jobs,
 - CI/CD pipelines,
 - scheduled analytics,
 - large-scale security monitoring workflows.
-

8. Interpretation & Limitations

8.1 Interpretation

Low **risk_score** (~0.0)

ASNs that receive a low vulnerability risk score typically exhibit a combination of:

- strong or well-established acceptance controls (strict filtering and/or ROV enforcement),
- limited or moderate propagation capability,
- and generally good routing hygiene signals (RPKI and IRR coverage), when such data is available.

These ASNs represent environments where forged announcements are either:

- unlikely to be accepted, **or**

- unlikely to propagate widely even if accepted.

As a result, they are operationally difficult to abuse as attack origins and present a low expected impact in hijack scenarios.

High **risk_score** (~1.0)

ASNs that receive a high risk vulnerability score tend to resemble the **BAD extreme cluster** used during weak supervision and typically show:

- permissive acceptance behavior (weak strict filtering and weak ROV enforcement),
- strong reachability and propagation capability,
- and/or incomplete or weak routing hygiene signals (low RPKI / IRR coverage).

These ASNs represent environments where forged announcements are both:

- more likely to be accepted, **and**
- more likely to propagate broadly once accepted.

Such conditions increase the feasibility and potential impact of hijacked route announcements.

8.2 Limitations

- **Weak supervision, not incident ground truth**
Labels are automatically derived from high-confidence heuristic extremes based on acceptance and propagation signals.
They are not derived from manual labeling or historical incident datasets, which are scarce and incomplete at Internet scale.
- **High-precision labeling implies partial coverage**
Only ASNs with sufficiently complete and unambiguous data are used for supervision.
Many ASNs remain unlabeled during training and are evaluated only at prediction time.
- **Single train/validation split**
The current implementation uses a single stratified train/validation split rather than k-fold cross-validation.

This choice favors simplicity and operational clarity over exhaustive hyperparameter optimization.

- **Explicit modeling of missing ROV and hygiene data**
`rov_enforcement_score`, RPKI coverage, IRR coverage, strict filtering, and reachability features are **imputed using robust statistics (medians)** when missing. Missingness is **explicitly encoded via presence indicators (`has_*` flags)**, allowing the model to distinguish between weak signals and absent information.

Missing data is therefore treated as **uncertainty**, not as a hard indicator of insecurity.

- **Heuristic thresholds are tunable**
Thresholds used for labeling extremes (e.g., what constitutes “strong” or “weak” filtering or propagation) are based on engineering judgment. They are intentionally conservative and can be refined as additional operational feedback or deployment experience becomes available.

9. Why This ML Design Is Robust and Operationally Defensible

9.1 Conservative, High-Confidence Labeling (Weak Supervision)

The GOOD/BAD labels are applied **only to extreme, unambiguous cases**, and only when sufficient, reliable data is available.

Labeling is intentionally **high-precision and low-recall** by design.

GOOD ASNs represent environments where either:

- acceptance controls are clearly strict (strong filtering and ROV enforcement), **or**
- propagation capability is structurally limited, reducing the potential impact of any forged announcement.

BAD ASNs represent environments where:

- acceptance controls are permissive, **and**

- propagation capability is strong enough to allow wide dissemination of forged routes.

ASNs that do not clearly satisfy one of these extreme conditions are **not forced into a class**. Instead, they remain **unlabeled** and are evaluated only at prediction time.

This has two important consequences:

- **Label noise is minimized**
Only high-confidence extremes are used for supervision, reducing the risk of incorrect labels in ambiguous cases.
- **No artificial ground truth is invented**
The system does not pretend to know the true security posture of every ASN.
Supervision is applied only where the signal is strong enough to justify it.

For security ML problems where no complete ground truth exists, this is a standard, defensible approach.

9.2 Labeling Uses Acceptance and Propagation Signals Explicitly

Label generation intentionally combines **acceptance behavior** and **propagation capability**.

In the current design, the labeling logic uses:

- strict filtering behavior,
- ROV enforcement strength,
- and reachability / propagation capability.

This reflects the operational reality of BGP hijacks:
a forged announcement is only dangerous if it is **both accepted and propagated**.

Importantly:

- ROV enforcement is **explicitly included** in the labeling rules,
- not treated as an isolated or post-hoc signal.

This avoids unrealistic label definitions where ASNs with permissive ROV behavior would still be labeled as “safe” despite clear acceptance weaknesses.

The model therefore learns from **operationally meaningful extremes**, not abstract hygiene-only notions of security.

9.3 Transparent, Explainable Model Class

The model is implemented using **LightGBM (gradient-boosted decision trees)**, a well-established choice for tabular risk scoring.

LightGBM:

- is widely deployed in production security systems,
- supports feature importance and advanced explainability techniques (e.g., SHAP),
- avoids opaque behavior typical of deep neural networks.

This enables:

- per-feature contribution analysis,
- global importance ranking,
- and clear explanations for why a given ASN receives a high or low risk score.

For an engineering and operator audience, this level of transparency is critical for trust and adoption.

9.4 Consistency with Operator Intuition

The learned relationships are aligned with established BGP security intuition:

- Strong strict-filtering behavior → lower acceptance of forged routes → lower risk
- Strong ROV enforcement → reduced acceptance of invalid origins → lower risk

- Strong reachability and propagation → higher potential blast radius → higher risk when acceptance is weak
- Poor hygiene signals (low RPKI / IRR coverage) → increased uncertainty about routing discipline

The model does not rely on a single metric.

Instead, it learns how **multiple weak signals combine** into meaningful risk.

This mirrors how experienced operators reason about routing risk in practice.

9.5 Designed for Incremental Improvement, Not a One-Shot Black Box

The pipeline is intentionally modular and extensible:

- The model can be retrained as new data becomes available.
- Each feature table is computed independently.
- Additional metrics (e.g., data-plane ROV, historical incidents, MOAS frequency) can be added without redesigning the architecture.

This ensures the system evolves alongside the BGP ecosystem rather than being tied to a static snapshot in time.

9.6 Explicit Modeling of Missing and Incomplete Data

Missing or incomplete data is handled **explicitly**, not implicitly.

The design follows two principles:

1. **Missing values are not interpreted as secure behavior**
Coverage-type features (RPKI, IRR, filtering, ROV, reachability) are imputed using robust statistics (medians),
while **explicit presence flags (`has_*`) inform the model about data availability**.
2. **Uncertainty increases risk, but does not force classification**
ASNs with incomplete data are not automatically labeled as BAD.
Instead, the model learns that missing evidence of security reduces confidence in a

low-risk classification.

This approach avoids:

- false assumptions of safety due to missing data,
- and unjustified hard penalties based purely on data gaps.

From a defensive risk-assessment perspective, this treatment of uncertainty is conservative, realistic, and operationally defensible.

10. Automatic Feature Importance Learning in LightGBM

A core strength of the ML vulnerability model is that **it learns the relative importance of all features directly from data**, without any manually assigned weights or predefined priority rules.

This is achieved through LightGBM's gradient-boosted decision tree architecture, which determines feature importance in a fully automated, data-driven, and auditable way.

Below is a clear explanation suitable for network engineers and ML-aware professionals.

1. LightGBM builds hundreds of decision trees (boosted ensemble)

During training, LightGBM constructs a sequence of decision trees.

Each tree attempts to correct the mistakes of the previous trees, gradually improving predictive performance.

At each tree-building step, the model asks:

“Which feature best splits GOOD vs BAD ASNs at this stage?”

and selects the feature that:

- gives the highest information gain,
- reduces classification error the most, or
- increases separation between the classes.

This selection is completely automatic.

No weights, priorities, or assumptions are hard-coded by the platform.

2. For each tree, LightGBM chooses the best metric and best threshold

Each decision tree evaluates **all features** such as:

- RPKI coverage
- ROV enforcement
- IRR route-object coverage
- strict filtering score
- reachability propagation
- AS_PATH length

For every feature, LightGBM tests many possible thresholds (splits):

Example splits the model might test:

- `rpki_coverage_pct < 0.15?`
- `bgp_filters_strict < 0.40?`
- `irr_routeobject_coverage < 0.50?`
- `rov_enforcement_score < 0.70?`
- `reachability_propagation < 0.20?`

For each candidate split, LightGBM measures:

How much does this split reduce uncertainty in the labels?

(i.e., how well it separates GOOD from BAD ASNs)

The feature + threshold combination that produces the largest reduction in error is automatically chosen.

3. Over hundreds of splits, LightGBM accumulates “importance scores”

Every time a feature is chosen for a split (at any tree and any depth):

- its internal importance counter increases
- proportional to how much that split improved the model

After training hundreds of trees, LightGBM aggregates these counters to produce:

Global Feature Importance Ranking

This ranking tells:

- which metrics matter most
- which ones matter moderately
- which ones contribute less
- which interactions the model has discovered

All of this is learned **directly from the real BGP security data**, not from rules imposed by developers.

4. Why this process is extremely valuable for routing security

✓ No manual weights or arbitrary priorities

The model is not biased by assumptions such as “RPKI is more important than IRR” or “strict filtering matters more than ROV”.

Instead:

The Internet’s behavior determines the weights, not a human designer.

This prevents engineering bias and increases credibility.

✓ Captures non-linear interactions between metrics

Routing-security signals interact in complex ways:

- an ASN with poor strict filtering but high RPKI coverage
- an ASN with good reachability but no IRR objects
- an ASN with moderate filtering but extremely weak ROV adoption

These relationships are often **non-linear**.

LightGBM discovers and models these patterns automatically.

No hand-written formula could capture these interactions reliably.

✓ Auditable and explainable

Because the model uses trees (not deep neural networks), engineers can:

- inspect feature importance
- visualize decision thresholds
- interpret why a given ASN receives a high or low risk score

This level of transparency is extremely important for operational trustworthiness.

✓ Adapts to real Internet conditions

As ROV adoption, IRR hygiene, or RPKI usage change in the real world:

- retraining automatically updates feature importance
- no code changes needed
- no re-tuning manual weights

- no policy rewrites

The model evolves with the Internet.

11. Why Automatic Feature-Importance Learning is Highly Accurate

While Section 10 describes *how* LightGBM determines feature importance, this section explains *why* this data-driven approach achieves high accuracy in the context of ASN vulnerability and propagation-risk modeling. These arguments focus strictly on the predictive reliability of the method and do not repeat the mechanics already covered.

11.1 Optimized Directly for Prediction Accuracy

LightGBM optimizes a well-defined mathematical objective (reducing classification error / increasing AUC).

Because feature importance emerges directly from these optimization steps, the model automatically assigns higher influence to metrics that genuinely improve predictive accuracy in real conditions.

This ensures that importance values are not subjective, not tuned manually, and not affected by operator bias.

Instead, the scoring reflects the statistical behavior of the global routing ecosystem.

11.2 Robustness Against Noise and Incomplete Data

Routing security datasets naturally contain noise, missing values, inconsistent IRR objects, and partial RPKI adoption.

Boosted tree ensembles are specifically known to be highly resilient to:

- noisy features,
- outliers,
- non-uniform measurement quality.

Because LightGBM combines hundreds of weak learners, its feature-importance estimation remains stable even when individual data sources are imperfect — a common scenario in BGP security analytics.

This robustness directly translates into more accurate vulnerability assessments.

11.3 Captures Real-World Interactions That Human-Defined Weights Miss

Routing-security metrics do not act independently.

For example:

- low IRR coverage may matter more for ASNs with already weak RPKI,
- strict filtering may partially compensate for weak reachability,
- ROV enforcement strength interacts with propagation quality in non-linear ways.

A human or static scoring formula cannot anticipate or model these dependencies reliably.

LightGBM naturally discovers these interactions during training, assigning feature importance based on the *combined predictive contribution* across all interaction patterns—not individual metrics in isolation.

This is a major source of increased precision over heuristic scoring.

11.4 Proven Performance on Security-Relevant Tabular Data

Gradient-boosted decision trees are widely considered the highest-performing models for structured datasets in security-sensitive domains, including:

- IP and ASN reputation scoring,
- anti-abuse and anti-fraud systems,
- anomaly detection with noisy telemetry,
- risk classification pipelines.

These domains require models that deliver high AUC, low false positives, and reliable feature attribution.

The same characteristics apply directly to vulnerability and propagation-risk scoring.

Thus, LightGBM's automatic importance estimation is not theoretical — it is validated operationally at massive scale by cloud, CDN, and security providers worldwide.

11.5 Stable and Reproducible Importance Signals

A critical property of LightGBM is that importance rankings are:

- **stable across runs,**
- **reproducible,**
- **insensitive to small perturbations** of the input data,
- **consistent as long as global routing patterns remain stable.**

This stability is essential in an operational context:

engineers can trust that a feature's influence in the model represents a persistent effect in the Internet's routing behavior, not a random fluctuation.

11.6 Naturally Adapts to Global Internet Changes

As RPKI adoption increases, IRR quality evolves, filtering policies shift, or hijack patterns change, a static weighting scheme would become outdated.

LightGBM solves this intrinsically:

- retraining recalibrates importance values automatically,
- no manual re-weighting or redesign is required,
- the model remains accurate as the Internet changes.

This adaptability is crucial for maintaining long-term precision in BGP security analytics.

11.7 Why LightGBM Was Chosen Over Other Models

LightGBM was selected after evaluating the characteristics of ASN-level routing-security data and the operational requirements of this platform.

Several alternative models were considered (logistic regression, random forests, XGBoost, CatBoost, neural networks), but LightGBM provides the best balance of **accuracy, robustness, explainability, and operational practicality**.

1. Superior Accuracy for Tabular, Mixed-Quality Security Data

ASN vulnerability involves heterogeneous features:

- continuous (as_path_len, rpki_coverage_pct),
- semi-discrete (strict filtering scores),
- noisy real-world measurements (reachability propagation),
- potentially sparse features (IRR coverage, ROV enforcement).

LightGBM consistently outperforms:

- logistic regression (too linear),
- random forests (weaker handling of feature interactions),
- neural networks (require far more data and are harder to audit)

on this type of dataset.

2. Built-In Handling of Missing and Noisy Inputs

Routing data is imperfect by nature.

LightGBM natively tolerates:

- missing values,
- inconsistent update frequency,
- partial data quality across ASNs.

This robustness is essential for security analytics, where ground truth is rarely clean.

3. Fast Training and Re-Training (Operational Advantage)

For a platform meant to update risk scores regularly, LightGBM provides:

- extremely fast training,
- low computational overhead,
- the ability to retrain daily/hourly if desired.

Neural networks or more complex models would be far slower and harder to maintain.

4. Full Explainability (a Critical Requirement)

Unlike deep learning, LightGBM produces:

- global feature importance,
- per-ASN reasoning (via SHAP),
- tree-based decision paths.

This transparency is essential when the output must be trusted by network operators and validated during security reviews.

5. Captures Complex Non-Linear Interactions

ASN vulnerability is not a linear problem.

Filtering strength, RPKI maturity, IRR hygiene, and ROV enforcement influence each other in complicated ways.

LightGBM captures this naturally, whereas:

- linear models cannot,
- random forests handle it less efficiently,
- neural networks would require far more data.

6. Field-Proven in Large-Scale Security Applications

LightGBM is widely used in:

- anti-abuse scoring pipelines,

- anomaly detection at hyperscalers,
- fraud detection,
- IP/ASN reputation systems.

These are the same types of problems—highly imbalanced, high-stakes, tabular security data—making LightGBM an industry-proven choice.

12. Anti-Overfitting Guarantees (Model Stability & Generalization)

The ML classifier used in this platform is explicitly engineered to minimize overfitting, ensure stable generalization, and produce reproducible ASN vulnerability risk scores across retraining cycles and evolving Internet routing conditions.

Overfitting occurs when a model memorizes training-specific artifacts instead of learning generalizable structural relationships.

To prevent this, the system integrates **multiple complementary safeguards** at the data-selection, training-procedure, and model-configuration levels.

12.1 Independent Validation Split (25% of Labeled ASNs)

During training, the automatically labeled dataset of high-confidence GOOD and BAD ASNs is split into:

- **75% training set**
- **25% validation set**

using stratified sampling:

```
train_test_split(  
    X,  
    y,
```



```
test_size=0.25,  
  
random_state=random_state,  
  
stratify=y  
)
```

Key properties of this split:

- Validation ASNs are **never seen during training**
- The GOOD / BAD class ratio is preserved across splits
- Validation metrics provide an **unbiased estimate of generalization performance**

If the model begins to overfit (i.e., memorize training samples), validation AUC stagnates or degrades while training performance continues to improve, acting as an immediate overfitting signal.

12.2 Early Stopping (Validation-Driven Training Halt)

Training uses LightGBM's built-in early stopping mechanism:

```
early_stopping_rounds = 80
```

This enforces that:

- training halts automatically once validation performance stops improving,
- additional trees are not added if they only reduce training loss but fail to improve validation metrics.

Effect:

- the model converges to the most generalizable structure,

- unnecessary complexity is never introduced,
 - training remains robust even with a high upper bound on boosting rounds (`num_boost_round = 800`).
-

12.3 Controlled Model Complexity (`num_leaves = 31`)

Tree complexity is explicitly bounded:

```
num_leaves = 31
```

Implications:

- each decision tree remains relatively shallow,
- memorization of individual ASNs or rare corner cases is strongly constrained,
- expressive power is achieved through **boosting aggregation**, not tree depth.

This conservative configuration is well-suited for structured routing-security data and prioritizes stability and robustness over excessive expressiveness.

12.4 Regularized Gradient Boosting (Feature & Data Subsampling)

Additional regularization is applied through stochastic subsampling:

```
feature_fraction = 0.9
```

```
bagging_fraction = 0.9
```

```
bagging_freq = 5
```

This ensures that:

- each tree sees only a random subset of features,

- each boosting iteration uses a random subset of labeled ASNs,
- no single feature or ASN can dominate the learning process.

Resulting effects:

- reduced variance,
- improved robustness,
- resistance to spurious correlations in the labeled data.

12.5 Extreme-Cases Training via Weak Supervision

The model is trained **exclusively** on ASNs that are clearly GOOD or clearly BAD, as determined by strict, rule-based weak supervision (Section 3).

Properties of this approach:

- ambiguous or borderline ASNs are never used for supervision,
- labels are high-precision and low-noise,
- training signals are clean and internally consistent.

This “extremes-only” supervision strategy is commonly used in:

- security risk modeling,
- abuse and fraud detection systems,
- anomaly detection pipelines.

It significantly reduces overfitting by eliminating contradictory or weakly defined training examples.

12.6 Fully Explainable Model Architecture

The classifier is based on **gradient-boosted decision trees**, not deep neural networks.

As a result:

- every split is inspectable,
- feature importance can be audited,
- SHAP values can be computed to explain individual ASN risk scores.

Explainability directly contributes to overfitting control:

- memorization artifacts are easily detectable,
- model behavior can be validated against routing-security domain knowledge,
- no opaque black-box failure modes are introduced.

12.7 Stable Predictions Across Time

All model features are derived from empirical, continuously updated routing-security measurements, including:

- RPKI coverage,
- IRR route-object visibility,
- strict filtering behavior,
- ROV enforcement,
- observed propagation characteristics.

When the model is retrained periodically on updated data:

- global score distributions remain stable,
- changes occur only where Internet routing reality has genuinely evolved,
- accidental model drift is avoided.

This ensures that vulnerability risk scores remain:

- comparable across time,
 - operationally reliable,
 - suitable for longitudinal and trend-based analysis.
-

13. Summary

`asn_ml_risk.py` adds a **machine-learning layer** on top of a rich set of BGP security and routing metrics:

- It condenses multiple features (RPKI coverage, IRR coverage, strict filters, ROV enforcement, propagation, AS-PATH length) into a **single continuous risk score per ASN**.
- It is **fully automated and re-trainable**, storing results in a dedicated table for straightforward integration with dashboards, APIs, and downstream analytics.
- The model is designed to be **transparent, conservative and explainable**, aligned with how network operators reason about:
 - origin security,
 - filtering strength,
 - and real-world propagation behavior of hijacked announcements.

From an engineering perspective, the ML design is:

- grounded in operational reality,
- conservative in its assumptions,
- explicit in its heuristics,

- and open to inspection and iterative improvement — making it a **robust and defensible component** of an ASN vulnerability and propagation-risk assessment platform.