ALARM:
An Adaptive Android Malware Detection Framework with Leiden-based Clustering and Mixture-of-Experts Classification

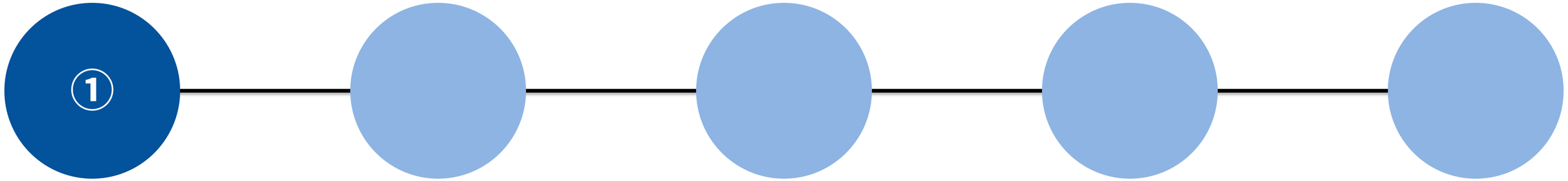
2026.03.24

Kyoungmin Roh
Seungmin Lee
Seong-je Cho
Youngsup Hwang

CSOS Lab.

Contents

1. Introduction



The Android Threat Landscape & Problems

Introduction

The Android Malware Problem

① Scale

3.5B+

Android users worldwide

Largest OS ecosystem.
Millions of new APKs uploaded
to Google Play every year.
Malware authors target this scale aggressively.

② ML Defense

Static Features

ML model as defense

API calls, permissions, and opcodes extracted from APKs.
Classifiers trained on historical data work, until they don’t.

③ The Crisis

Concept Drift

Models decay over time

Malware Evolves.
Feature distributions shift year over year.

Introduction

Why Existing Models Fail: Concept Drift

What is Concept Drift?

- Data distributions shift over time
- Performance of the model trained with historical data degrades

① Malware Tactics Evolve

New evasion strategies, obfuscation patterns, and API usage emerge year over year. The model's decision boundary becomes outdated.

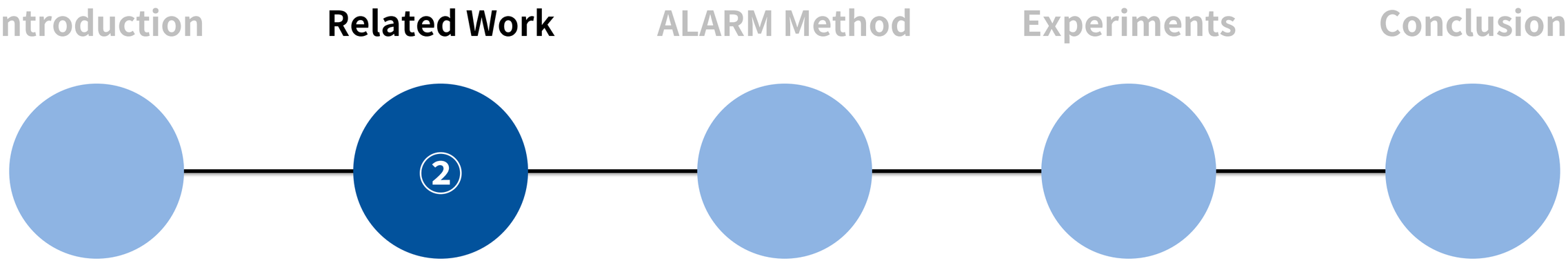
② Distribution Shift

Statistical distribution of API calls in malicious apps changes.

③ Retraining is impractical

Google Play hosts millions of apps. Frequent retraining requires massive labeling effort and compute, not viable in production.

2. Related Work



Prior Works & Gaps

Related Work

Four Streams of Prior Work – and What's Missing

① Concept Drift Handling

Periodic retraining,
Feature engineering for robust model.

Gap: High compute cost. Instability. Retraining-free solutions remain rare
and rely on heuristic threshold tuning or feature reweighting.

② Static Feature Detection

API counts, permissions, opcodes,
Control-flow graphs.

Gap: APIs treated independently — loses co-occurrence context. High dimensionality. Sensitive to distribution shift.

③ Graph-based Analysis

API call graphs, Louvain, Graph Neural Network.

Gap: Some methods cause data leakage: clusters built over full dataset
(train+test mixed) → inflated, non-reproducible performance.

④ Mixture-of-Experts (MoE)

Local Specialists handle different subspaces.
Gating mechanism aggregates predictions.

Gap: Applied widely in NLP — rarely in cybersecurity. Sparse MoE cannot reflect every expert's opinions.
No data-driven structural derivation of expert communities.

Related Work

Key Question

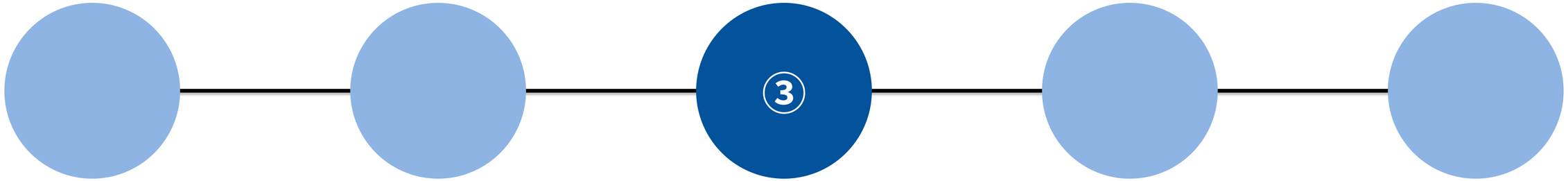
“Can we accurately detect evolving Android Malware – without ever retraining the model?

The limitations in the previous studies:

- Concept drift solutions require costly retraining → not scalable at Google Play scale
- Misses API’s semantic information → operator can’t know why this sample is malicious
- Sparse MoE cannot reflect every expert’s opinions

ALARM is the Solution

3. ALARM Method



Proposed Architecture

ALARM Method

Solutions for the Limitations

The gap in the field and its solution

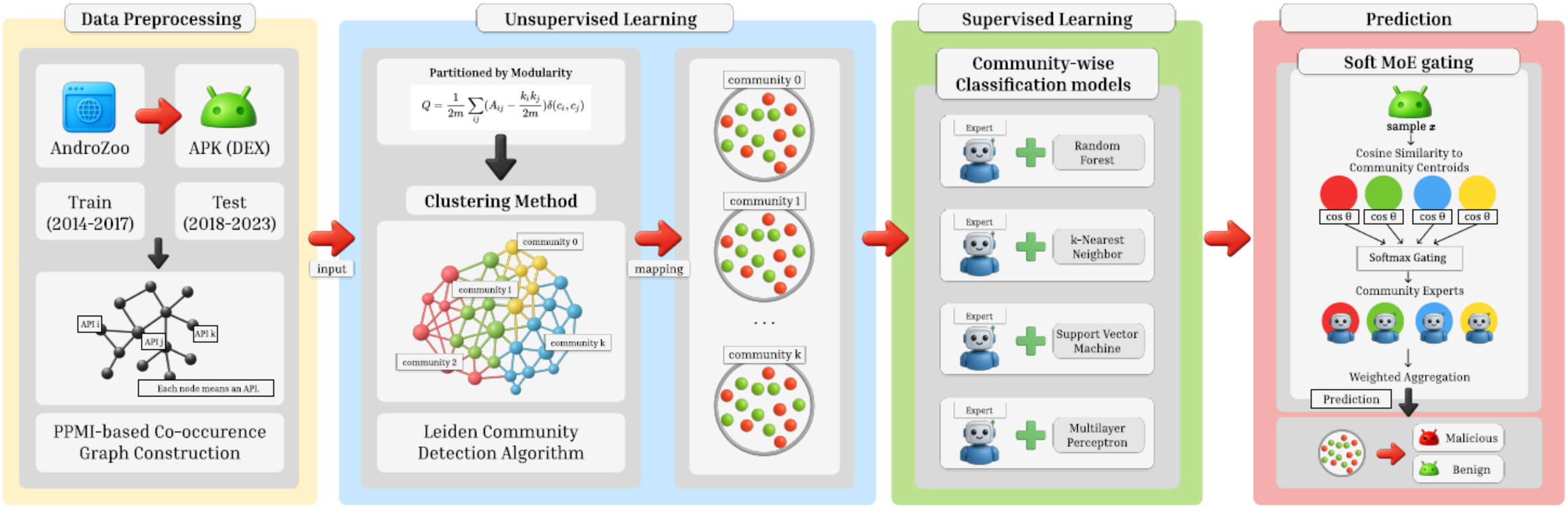
- Concept drift solutions require costly retraining → not scalable at Google Play scale
  - ➢ **Make a retraining-free architecture**
  - ➢ **Uses only historical data to make clusters**

- Misses API's semantic information → operator can't know why this sample is malicious
  - ➢ **Construct PPMI API Co-occurrence graph to reflect API's semantic meaning**
  - ➢ **Use the Social Network Analysis (SNA) algorithm to detect a cluster**

- Sparse MoE cannot reflect every expert's opinions
  - ➢ **Uses Soft MoE to reflect every expert's opinions**

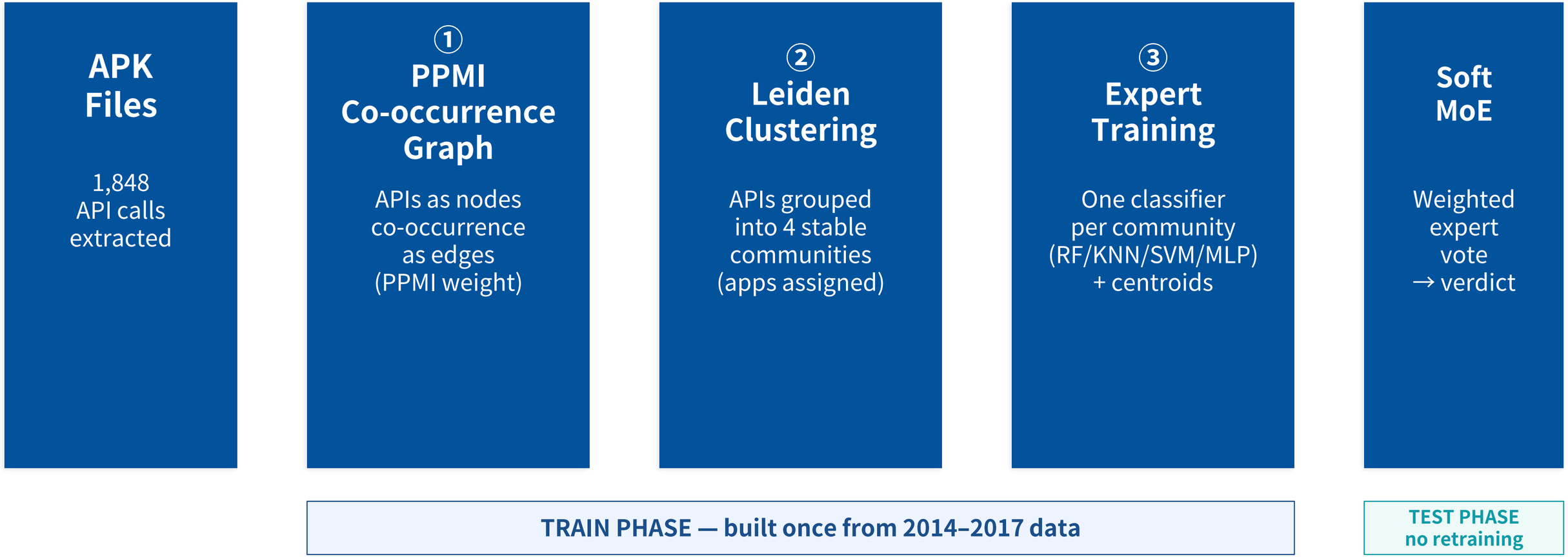
ALARM Method

Overall Architecture



ALARM Method

ALARM Framework Flowchart



The model is built once — then the Soft MoE routing adapts to every new APK without any updates.

ALARM Method

① Why PPMI graph? Because Individual APIs don't reflect semantic information

✗ Simple API Frequency-based approach

App A:

sendSMS | getLocation | startActivity | openCamera

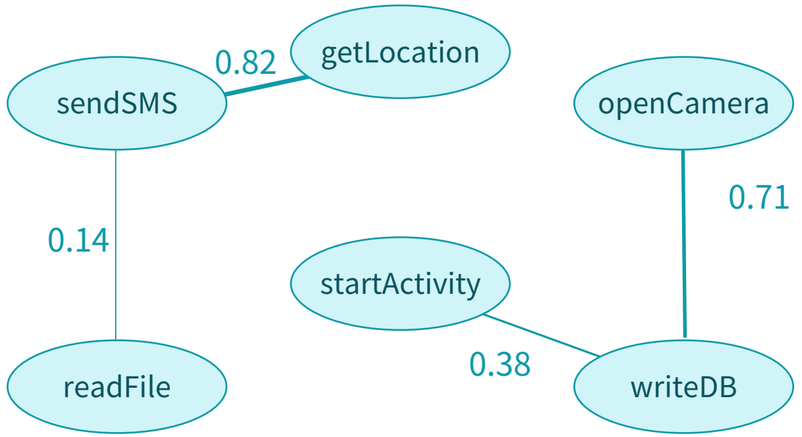
App B:

sendSMS | getLocation | startActivity | openCamera

Count model sees them as IDENTICAL

...but App A calls sendSMS → getLocation together
App B calls openCamera → startActivity together
Completely different behavior!

✓ PPMI Co-occurrence Graph using APIs



Edge weight = how much more often
APIs co-occur than by chance.
Thick edge = strong behavioral link.

PPMI = 0 if no meaningful link → only real relationships kept

ALARM Method

② Why Social Network Analysis (SNA)?

The Social Network Analysis (SNA)

In SNA, individuals (nodes) and their relationships (edges) reveal hidden community structure. Repeated interactions expose social roles.
→ We apply the same principle: each app is a person, each API call is a social activity. APIs that co-occur reveal the app's behavioral fingerprint.

ALARM Method

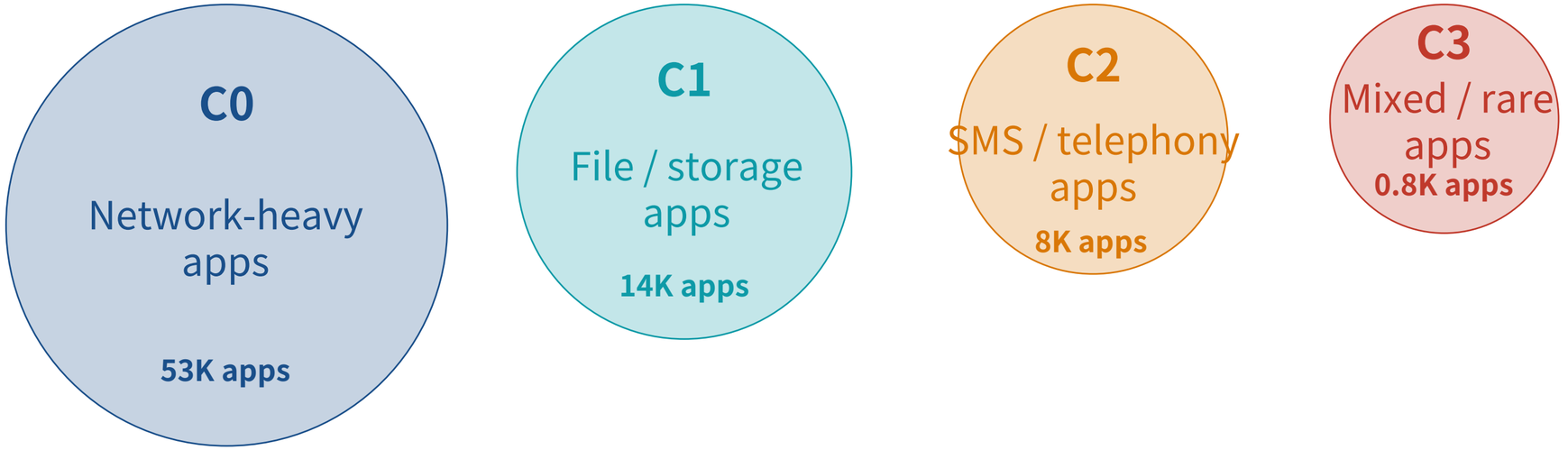
② Clustering apps by similar behavior using SNA

✗ Louvain

May produce disconnected clusters · Resolution limit problem · Less stable

✓ Leiden (our choice)

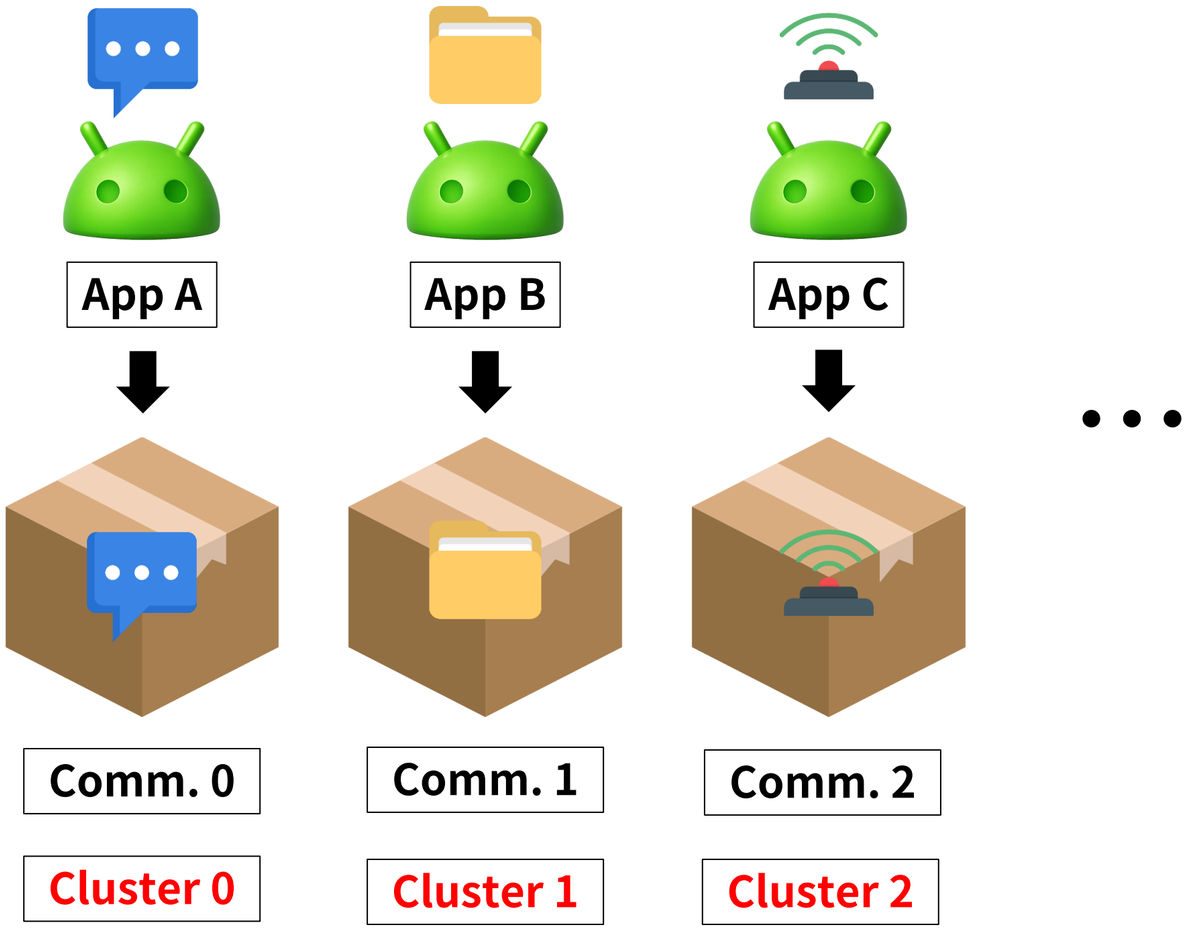
Guaranteed well-connected · Better modularity · Temporally stable communities



Temporal stability: these 4 community structures hold their shape year over year — even as individual malware variants change.

ALARM Method

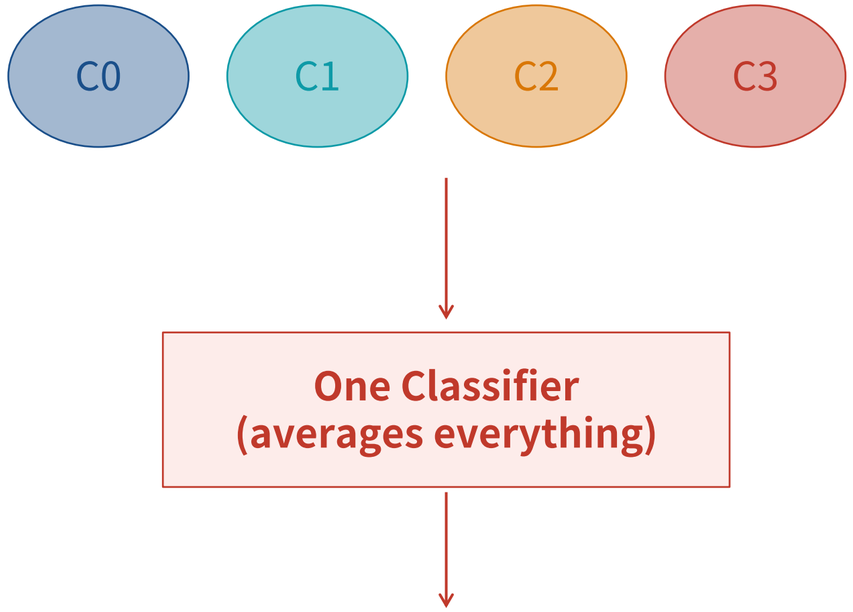
② Clustering apps by similar behavior using SNA



ALARM Method

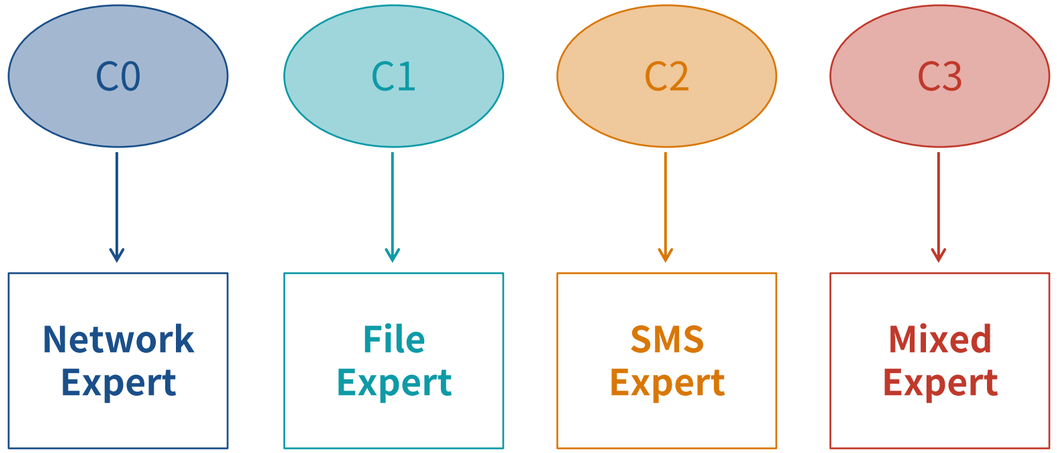
③ One Expert per Cluster

✗ One Global Model (Prior Study)



Fails when one sub-group's distribution shifts

✓ Cluster-Specific Experts (Our work)



Each expert trained on its own cluster's data only

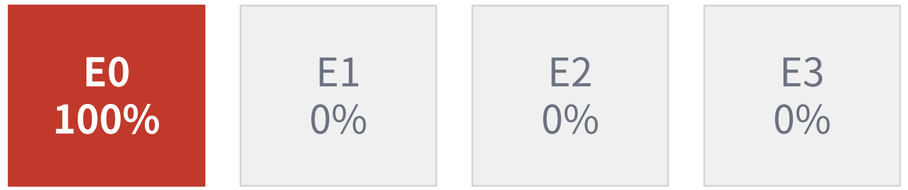
Local patterns captured stay stable under drift

ALARM Method

④ Soft MoE: Every Experts gets a voice

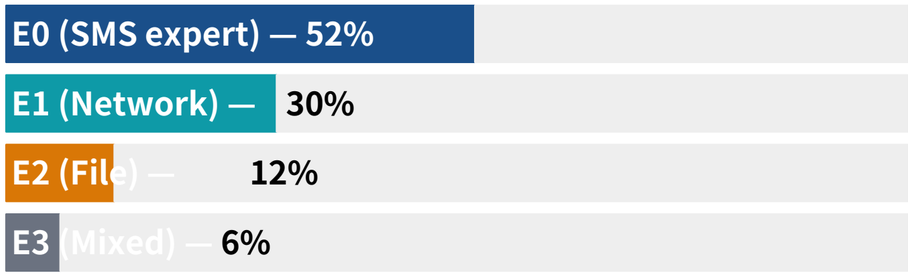
Why Soft MoE?

✗ Sparse MoE (Prior Work)



One expert wins all. Small perturbation → total flip.

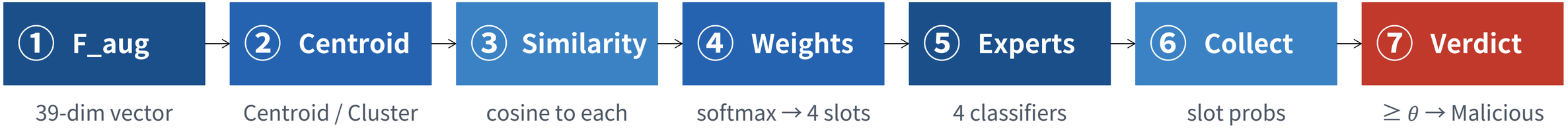
✓ Soft MoE (our choice)



Cosine Similarity → weight · smooth boundary transitions

ALARM Method

How ALARM Decides: Benign or Malicious?



Centroid

The median position of all training APKs in one cluster.
One 39-dim point per cluster — computed once, never changed at test time.

Slot

One position in the 4 results.
Each slot holds: cluster ID · similarity score · weight · P(malicious).

Nothing changes at test time — centroids, clusters, and experts are all frozen from training.

ALARM Method

① Each APK represents a fingerprint with 39 dimensions

7 dims based on the heuristic features

32 dims based on structured features

F_aug (1 × 39)

Graph features — what they measure:

GTV ratio	Smooth on benign graph vs. malware graph?
Centrality	How close to high-influence APIs?
Role score	Hub, bridge, or peripheral in API graph?
GTV benign	Raw graph smoothness — benign network
GTV malware	Raw graph smoothness — malware network
Mass benign	Overlap with benign API usage patterns
Mass malware	Overlap with malware API usage patterns

Spectral embedding

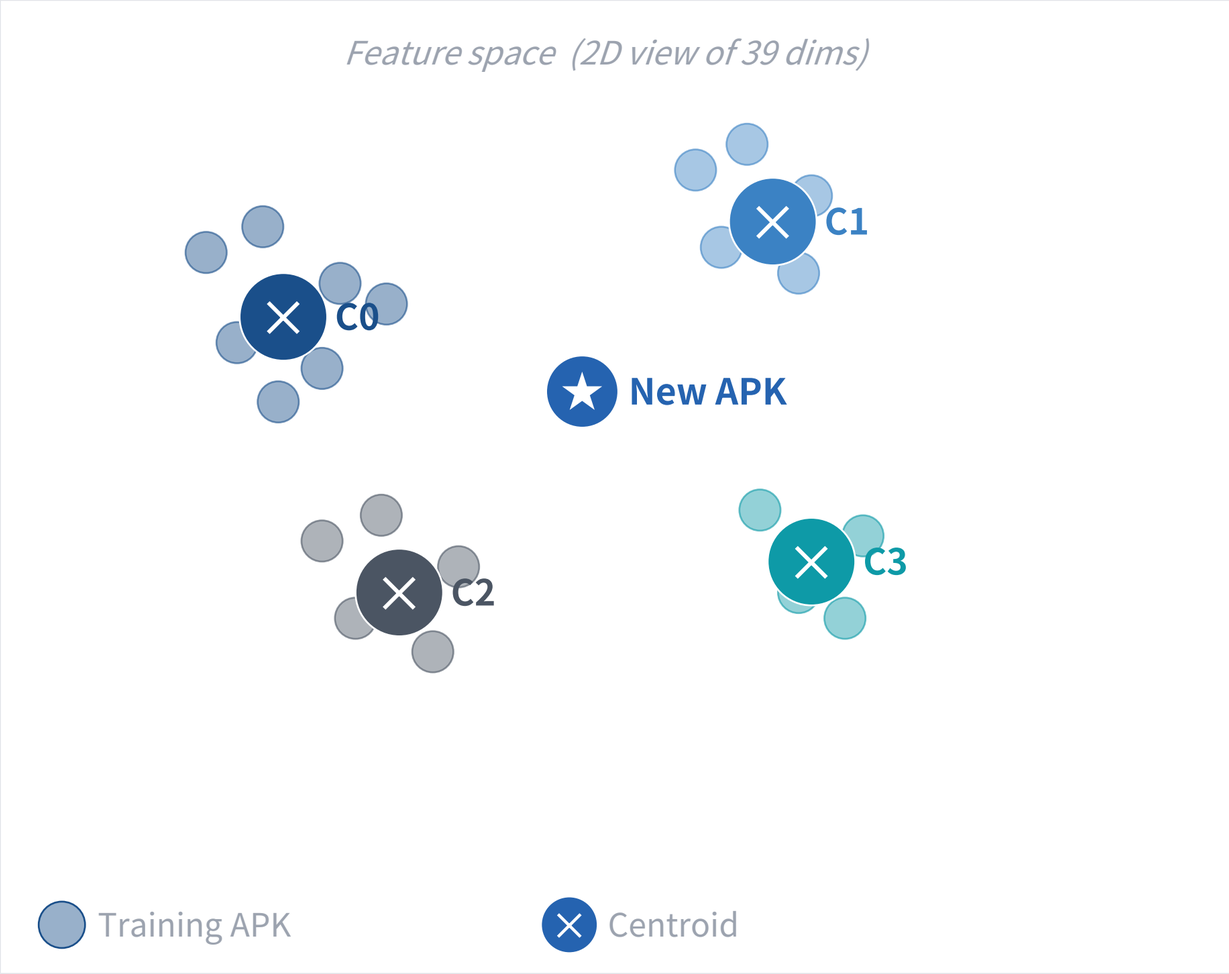
Projects APK into PPMI graph's eigen-space (32 dimensions).

Captures structural role in API co-occurrence graph.

Computed from training data only — fixed at test time.

ALARM Method

② Each cluster includes one centroid



C0 — **Network-heavy**
53,000 apps → 1 centroid (1×39)

C1 — **File / storage**
14,000 apps → 1 centroid (1×39)

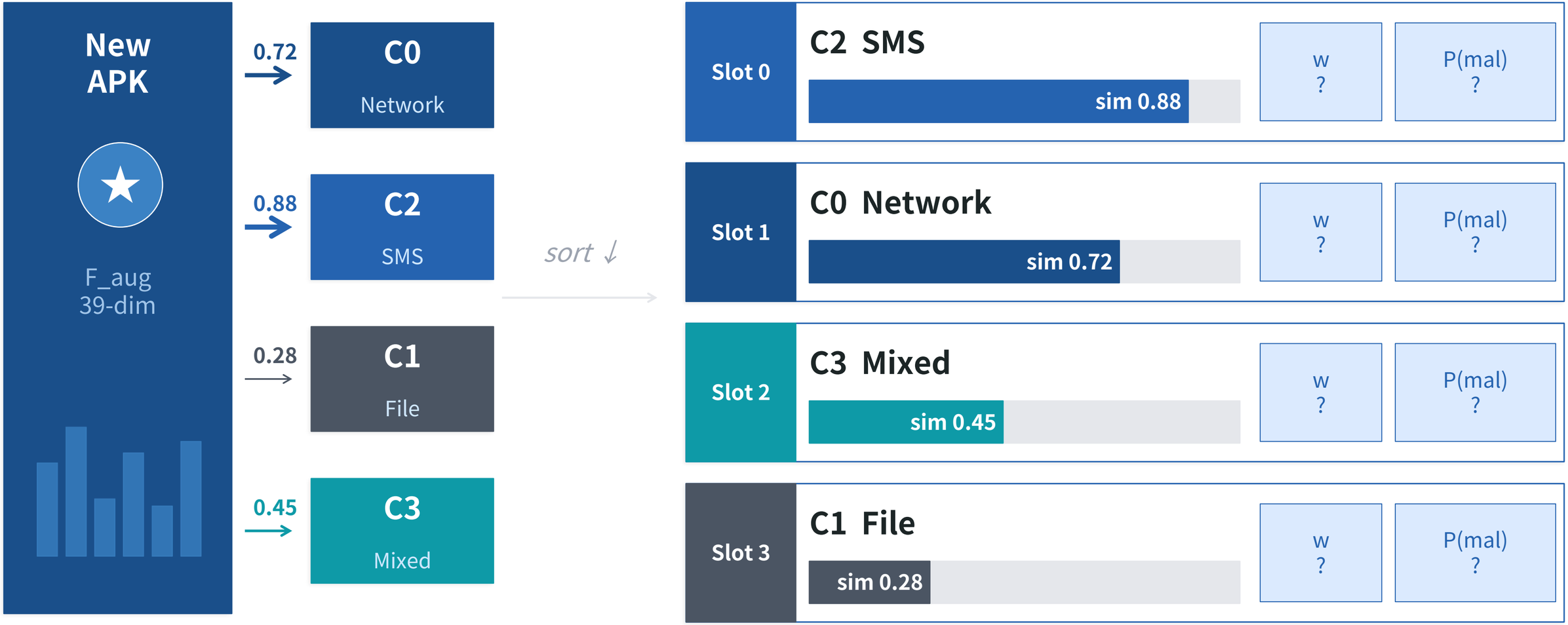
C2 — **SMS / telephony**
8,000 apps → 1 centroid (1×39)

C3 — **Mixed / rare**
800 apps → 1 centroid (1×39)

Cmat = 4 rows × 39 cols (fixed at test time)

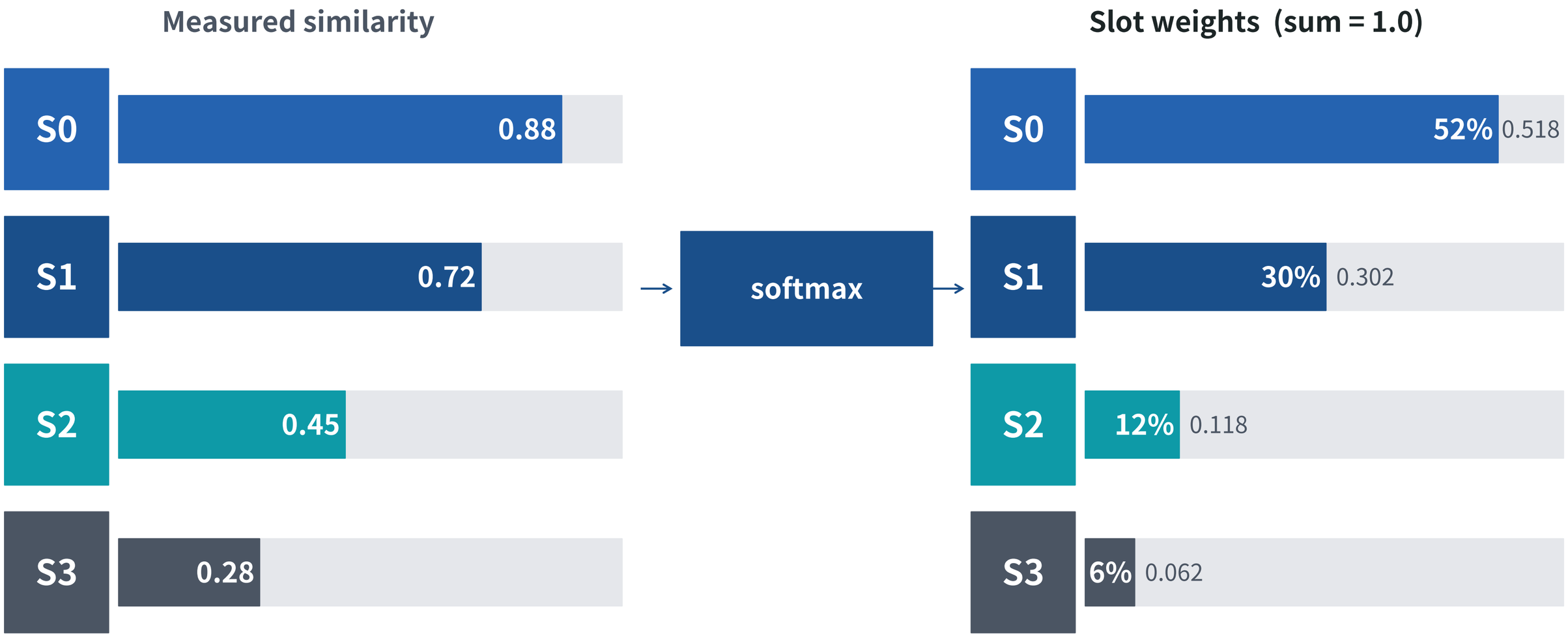
ALARM Method

③ Measure Cosine Similarity → ④ Rank into 4 slots according to similarity



ALARM Method

④ Similarity → proportional weight for each slot



Higher similarity → higher weight. The most similar community influences the verdict most.

ALARM Method

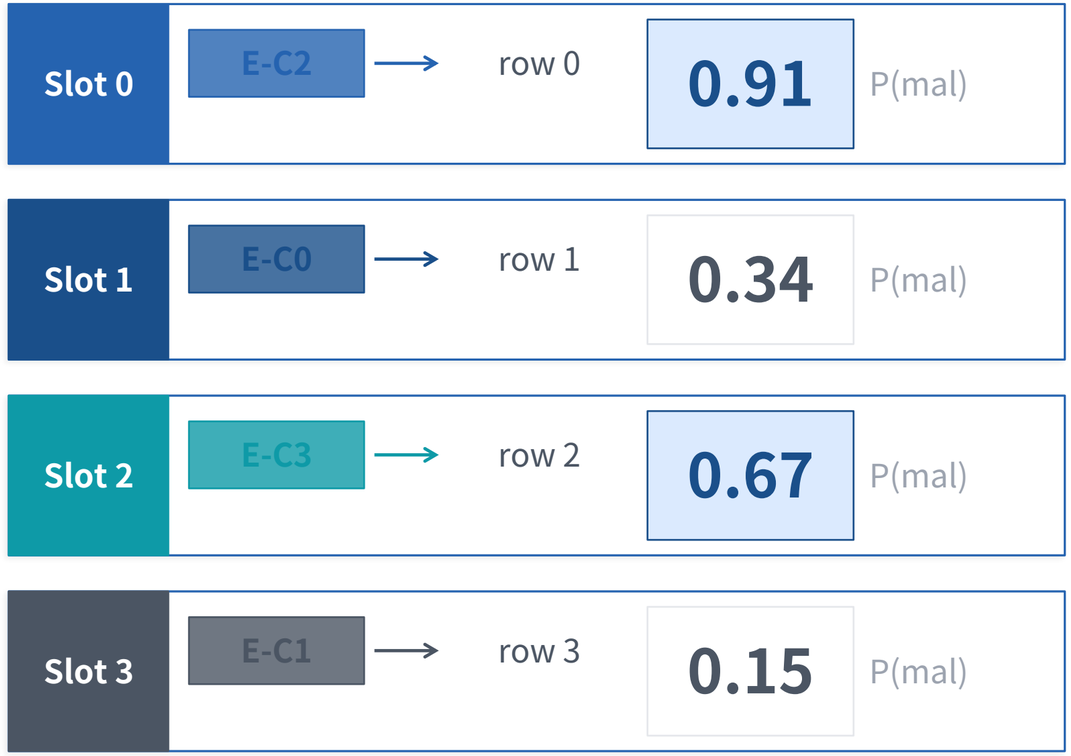
⑤ Each expert scores a test app — ⑥ each slot gets its probability

⑤ P_expert matrix — every expert scores all APKs

	APK 0	APK 1	APK 2	APK 3
E-C2	**0.91**	0.12	**0.88**	**0.75**
E-C0	0.34	**0.82**	0.41	0.29
E-C3	**0.67**	0.55	**0.71**	**0.88**
E-C1	0.15	**0.93**	0.38	0.55

Bold = high P(malicious)

⑥ APK *i* — each slot picks its own expert's score



ALARM Method

⑦ Weighted sum → final decision

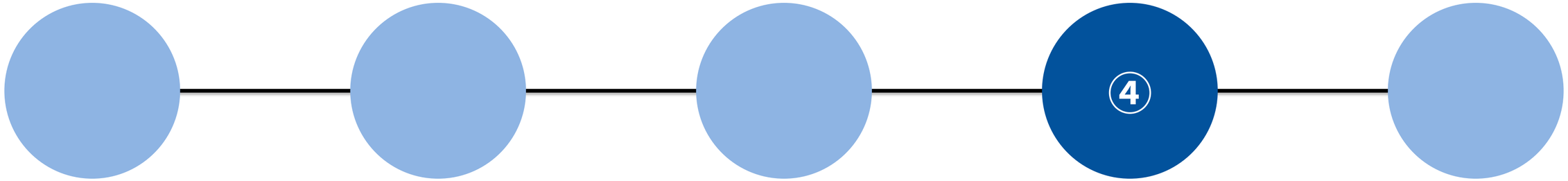
Slot	Weight w	×	P(malicious)	=	Contribution
Slot 0	0.518	×	0.91	=	0.471
Slot 1	0.302	×	0.34	=	0.103
Slot 2	0.118	×	0.67	=	0.079
Slot 3	0.062	×	0.15	=	0.009

0.471 + 0.103 + 0.079 + 0.009 = 0.662

0.662 ≥ θ (0.5) → **MALICIOUS**

4. Experiments

Introduction | Related Work | ALARM Method | **Experiments** | Conclusion



Experimental Setup & Evaluation

Experiments

Evaluation

Experimental Setup

- Dataset
  - AndroZoo Dex File, API Call frequency data
    - Training (2014-2017), Test (2018-2023)

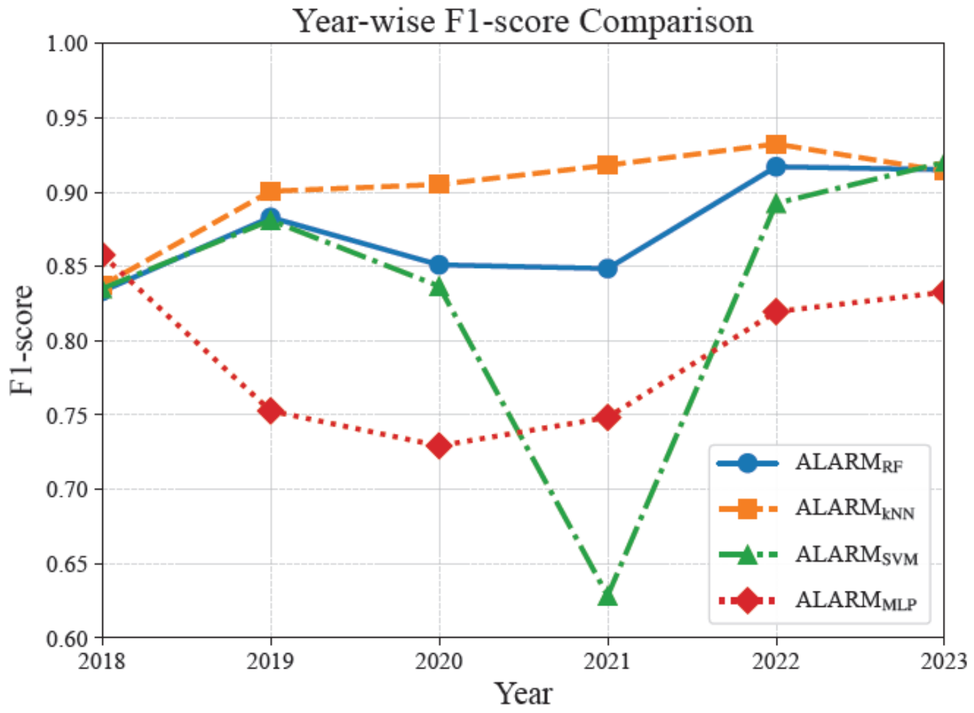
Year	Benign	Malicious	Year	Benign	Malicious
2014	5,000	5,000	2019	3,000	3,000
2015	5,000	5,000	2020	3,000	3,000
2016	5,000	5,000	2021	3,000	3,000
2017	5,000	5,000	2022	3,000	3,000
2018	3,000	3,000	2023	3,000	3,000
Total	23,000	23,000	Total	15,000	15,000

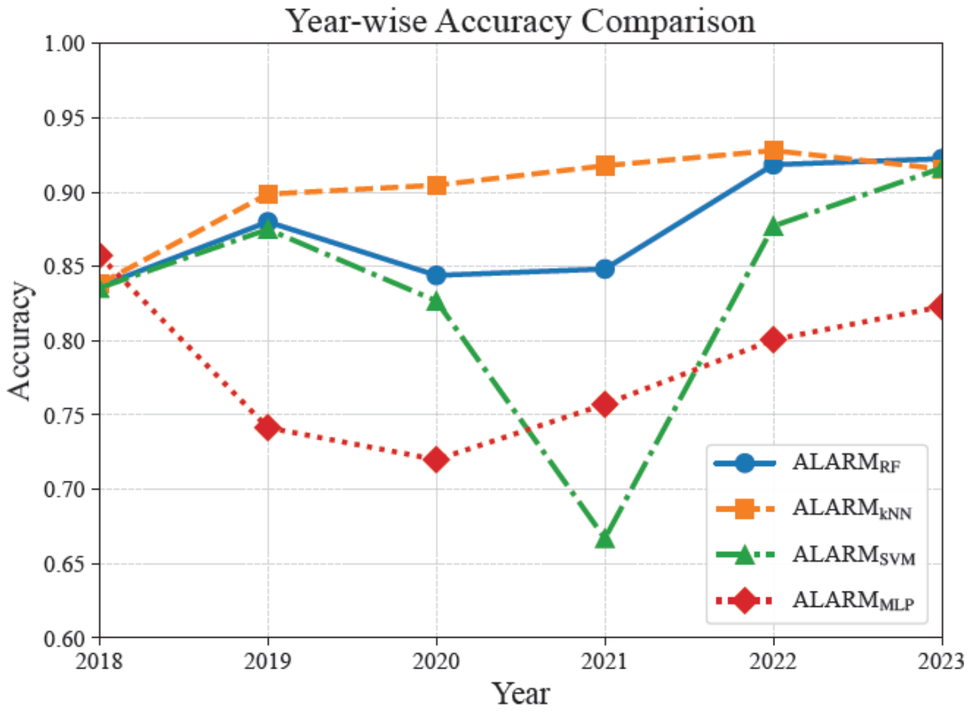
Experiments

Evaluation

Example Classifiers:

- Random Forest (RF), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Multilayer Perceptron (MLP)





Experiments

Evaluation

Example Classifiers:

- Random Forest (RF), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Multilayer Perceptron (MLP)

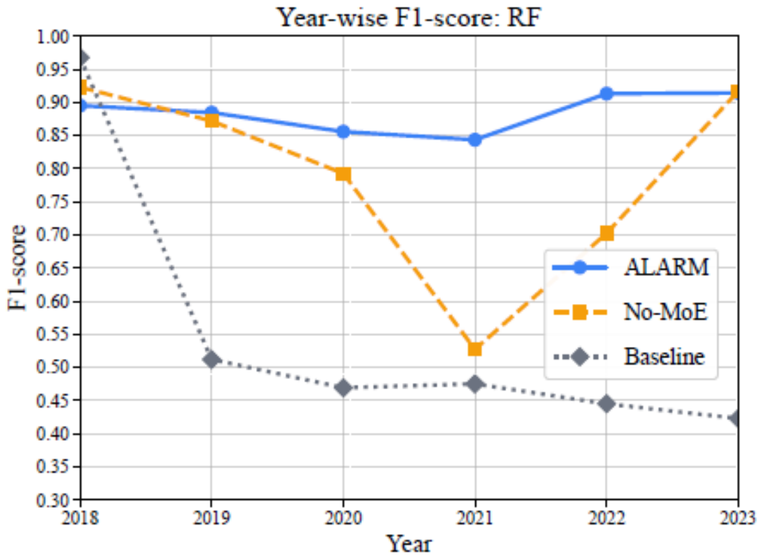
Model	ACC	F1
ALARM_{RF}	0.8750	0.8745
$\text{ALARM}_{\text{kNN}}$	**0.9005**	**0.9008**
$\text{ALARM}_{\text{SVM}}$	0.8330	0.8316
$\text{ALARM}_{\text{MLP}}$	0.7833	0.7860

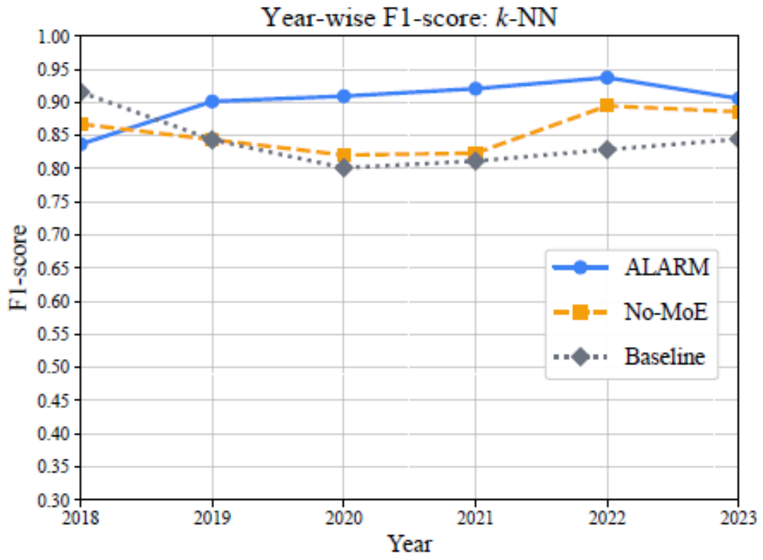
Experiments

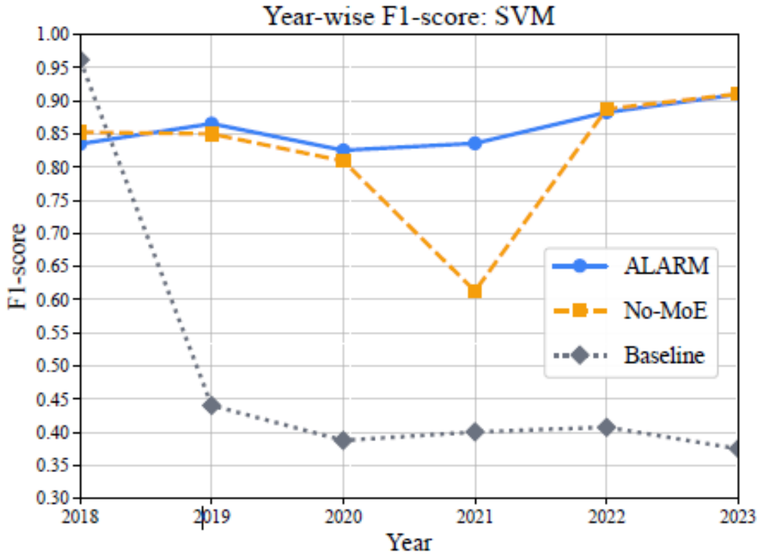
Ablation Study

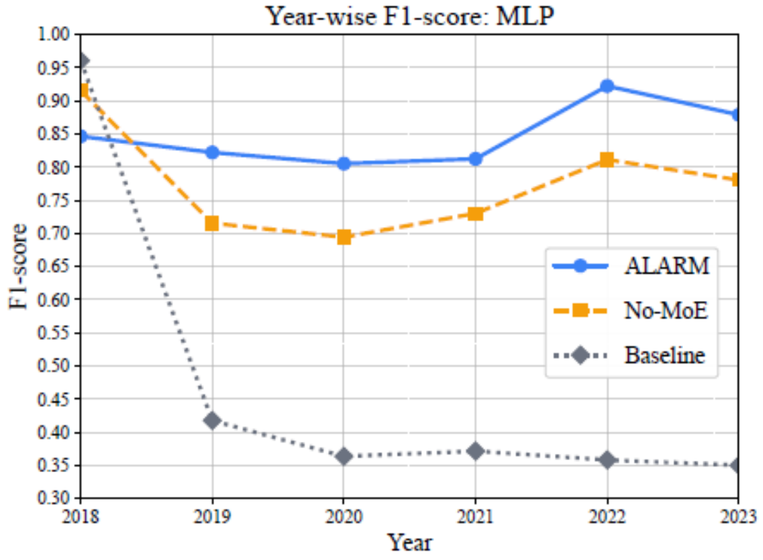
Three Models:

- ALARM: Leiden + Soft MoE
- No-MoE: Leiden
- Baseline: Global Classifier

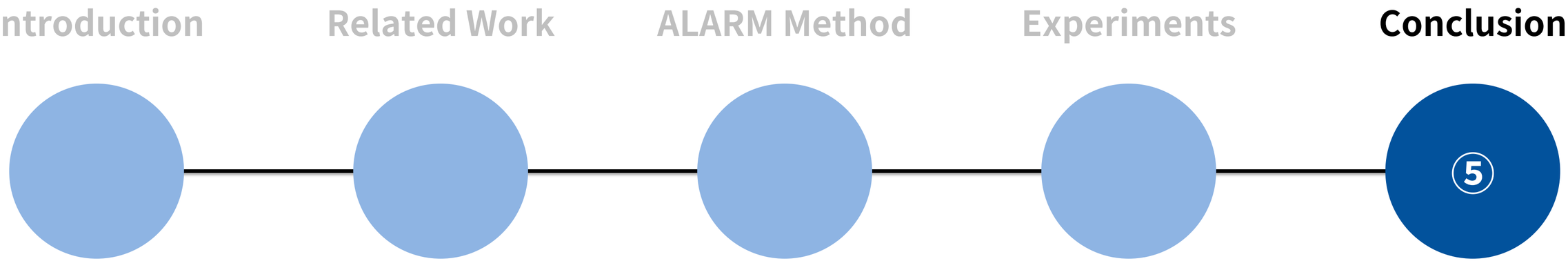

Year-wise F1-score: RF
F1-score
Year
ALARM
No-MoE
Baseline


Year-wise F1-score: k-NN
F1-score
Year
ALARM
No-MoE
Baseline


Year-wise F1-score: SVM
F1-score
Year
ALARM
No-MoE
Baseline


Year-wise F1-score: MLP
F1-score
Year
ALARM
No-MoE
Baseline

5. Conclusion



Discussion & Future Work

Conclusion

Limitations

✗ **Static features only**

Cannot capture runtime obfuscation, dynamic code loading, or behavioral changes that are invisible in the APK's static API calls.

✗ **Cosine-based gating — linear assumption**

Routing assumes communities are linearly separable in feature space. Near-boundary APKs may be misrouted if the manifold is complex.

✗ **Fixed global hyperparameters**

τ and θ are global constants. Under severe drift, per-community adaptive thresholds would likely perform better.

✗ **No theoretical guarantee under extreme drift**

ALARM is empirically validated — but formal generalization bounds under temporal distribution shift are not yet established.

Conclusion

Future Work

→ **Dynamic + static hybrid modeling**

Incorporate runtime behavior to catch obfuscated malware.

→ **Learnable gating mechanism**

Neural MoE / attention-based routing — replace cosine similarity.

→ **Online / continual adaptation**

Incremental model updates while preserving retraining-free core.

→ **Large-scale deployment validation**

Test on production-scale data; validate against adversarial samples.

Takeaway

"Concept drift does not necessarily require retraining — if structure is modeled correctly."

Conclusion

Conclusion

01

Efficient Retraining-free detection under concept drift

ALARM maintains stable detection performance across 6 years of real-world data without ever updating the model after training.

02

PPMI API Structure + Leiden SNA Adaptation + Soft MoE

Leiden clustering captures stable API behavioral groups. Soft MoE dynamically adjusts expert weights per APK — zero retraining required.

03

6-year performance stability (F1 ≈ 0.90)

Year-wise F1 remains robust against temporal drift. In the hardest year (2021), ALARM achieves F1 = 0.844 vs. 0.528.

04

Outperforms baseline model without degradation

Community-specific experts prevent the catastrophic averaging that collapses global classifiers under evolving malware distributions.

Q&A