Session: System for AI

Nazar: Monitoring and Adapting ML Models on Mobile Devices

Wei Hao, Zixi Wang, Lauren Hong, Lingxiao Li, Nader Karayanni, AnMei Dasbach-Prisk, Chengzhi Mao, Junfeng Yang, Asaf Cidon, Columbia University

ASPLOS'25

汇报人：孙铂钛

2025.09.18

Content

- Background
- Related work
- Motivation
- Design
- Evaluation
- Limitations
- Conclusion

BackGround

- 越来越多的ML模型被部署在用户的移动设备上，以实现低延迟推理和离线操作（图像分类、文本建议、语音识别等）。
- **核心挑战：性能不可见 (Lack of Visibility)**
  - 模型部署后，ML Operator无法追踪其在真实应用中的准确率。
  - 开发者没有用户的ground truth，因为用户不会手动标注数据。

How is the model really performing?

辅助驾驶应用

动物识别

BackGround

问题根源：数据漂移（Data Drift）

- 用户环境多变，导致推理数据的分布偏离了训练数据。
- 例如：天气变化、设备摄像头质量差异等。

两种常见类型

1、协变量漂移（Covariate Drift）

- $p_{test}(x) \neq p_{train}(x)$，$but\ \ p_{test}(y|x) = p_{train}(y|x)$
- **场景变了，规则没变。**例如，训练时多为晴天照片，但推理时多为雪天照片。

2、标签漂移（Label Drift）

- $p_{test}(y) \neq p_{train}(y)$，$but\ \ p_{test}(x|y) = p_{train}(x|y)$
- **类别比例变了，但类别样子没变。**例如，训练时汽车和行人比例是1:1，但部署在高速公路上，真实比例变为100:1。

(a) Wildlife animal example.

(b) Cityscapes example.

协变量漂移

Related Works

领域	代表工作	关注点	局限性
部署前验证	ML-EXray(MLSys'22)	在部署前发现并修复模型可能存在的问题（量化过程是否引入错误，数据预处理流程是否存在bug）	无法处理部署后，由环境变化而引发的各种数据漂移问题。
	Data Validation(arXiv)		
部署后适配	Ekya((NSDI'22)	在边缘服务器上实现持续学习和再训练，以便适应新的数据分布	1、依赖于强大的硬件； 2、通常需要有标签数据。
部署后监控	NannyML	为单个、云端部署的机器学习模型诊断数据漂移问题	1、不适用于移动设备这样大规模、分布式的部署场景； 2、缺乏自动化的适配能力，只能发现、不能自动解决。

Motivation

◻Motivation

- 现有的漂移检测/适配算法有一定限制：需要标签数据、假设单一漂移源 …
- 需要一个**无需用户反馈、全自动、端到端**的解决方案。

Nazar: The first end-to-end system for continuously monitoring and adapting models on mobile devices without requiring feedback from users。

Design--Overview

“检测-分析-适配”三阶段闭环

1、端侧检测（On-Device Detection）

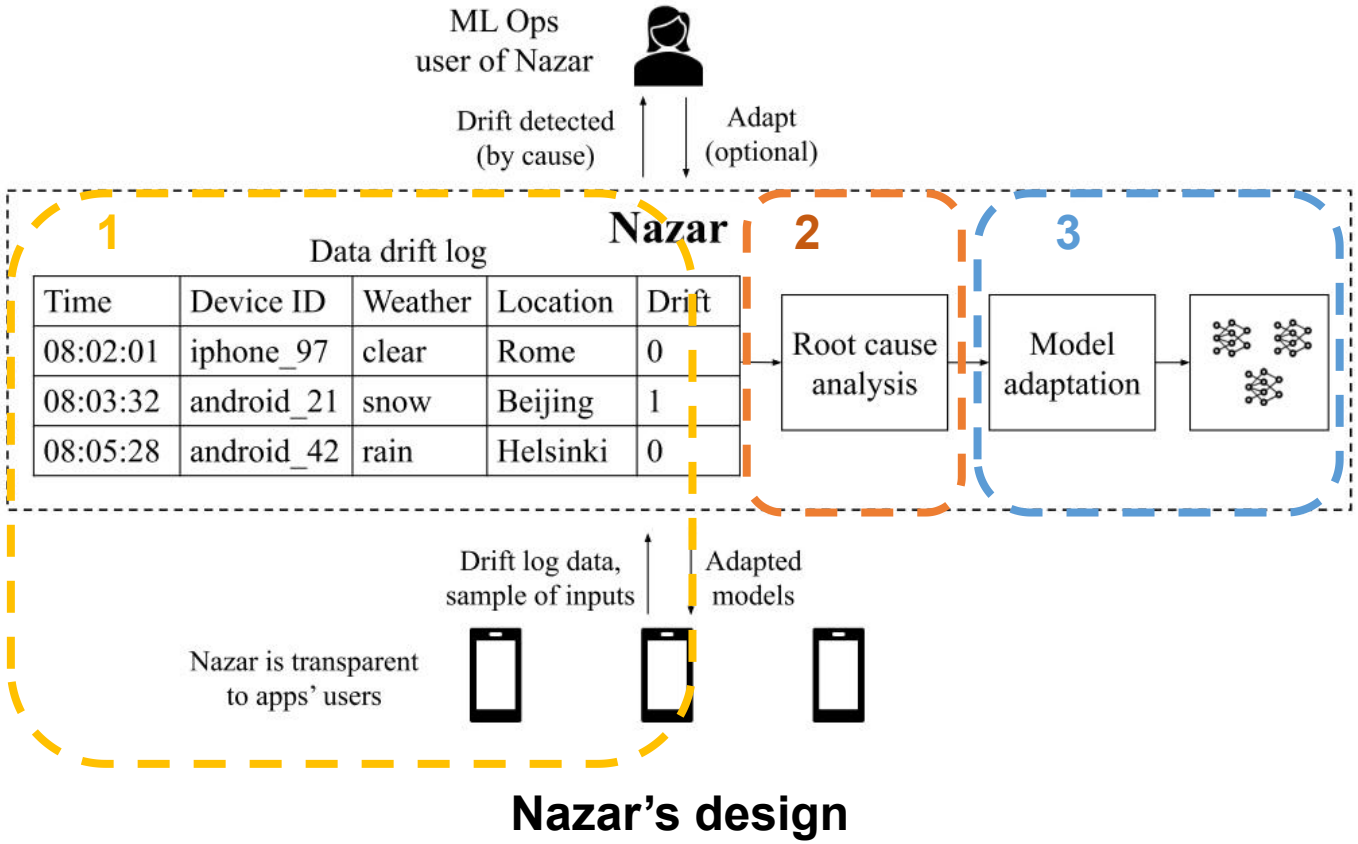
- 在用户手机上轻量级算法**检测数据漂移**，并将日志和少量采样数据上传。

2、云端分析（Cloud Analysis）

- 在云端聚合日志，对日志进行**根因分析**。

3、按因适配（By-Cause Adaptation）

- 针对每个根因，利用无标签的采样数据，训练一个专属的、轻量化的**适配模型**，并推送回设备。



Nazar’s design

Design1--On-Device Data Drift Detection

目标：在端侧以较低的开销检测漂移。

主流的数据漂移检测算法：

- **需要外部数据集**：假设能提前准备一个“漂移数据集”，这在现实中不切实际。
- **需要辅助模型**：需要在手机上同时运行第二个模型来辅助检测，资源开销大。
- **需要反向传播**：需要在推理后进行额外的反向计算，会使单次推理时间增加两倍。

	Threshold [21]	KS-test [48]	OE [22]	Odin [35]	MD [31]	SSL [23]	CSI [58]	GOdin [25]
No secondary dataset	✓	✓	✗	✗	✗	✓	✓	✓
No secondary model	✓	✓	✓	✓	✓	✗	✗	✓
No backpropagation	✓	✓	✓	✗	✓	✓	✓	✗
No batching	✓	✗	✓	✓	✓	✓	✓	✓

不同数据漂移检测算法比较

Design1--On-Device Data Drift Detection

□Threshold vs. KS-test

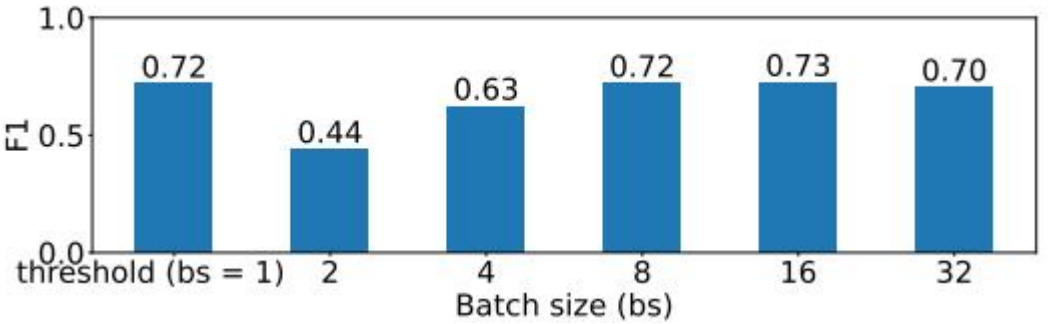
- **基于最大Softmax概率 (MSP) 的阈值检测**：对单次输出的置信度分数（MSP）应用一个简单的阈值。
- **KS-test**：对一批输出的置信度分数进行统计检验。

□问题：KS-test需要批处理，会在移动端会引发一系列问题

- 如何凑齐一批数据？要等多长时间？数据不够怎么办？

□最终选择：阈值法

- MSP < 阈值 → 低置信度 → 可能发生漂移



KS-检验不同batch size下的F1分数

上传内容：漂移检测结果 (True/False)、元数据 (Metadata：设备ID、地点、时间、天气、模型版本等)、一小部分被采样的输入数据（用于后续适配）

Design2--Root Cause Analysis (Overview)

□目标：从海量日志中精准**找出漂移的根本原因**（root causes）。

□Nazar的三阶段流程

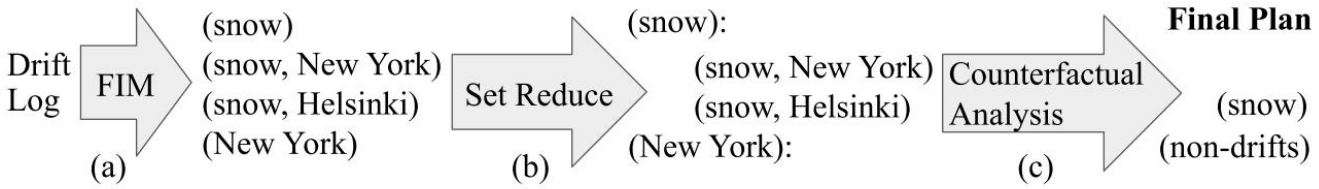
1、频繁项集挖掘（Frequent Itemset Mining, FIM）：广泛撒网，找出所有潜在原因。

2、集合约减（Set Reduce）：消除子集冗余。

3、反事实分析（Counterfactual Analysis）：消除重叠冗余。

Time	Device ID	Weather	Location	Drift
06:02:01	android_42	clear-day	Helsinki	False
06:02:23	android_21	clear-day	New York	False
06:04:55	android_21	clear-day	New York	True
08:03:32	android_21	snow	New York	True
11:05:01	android_42	snow	Helsinki	True

漂移日志



根因分析流程

Design2-1--Frequent Itemset Mining (FIM)

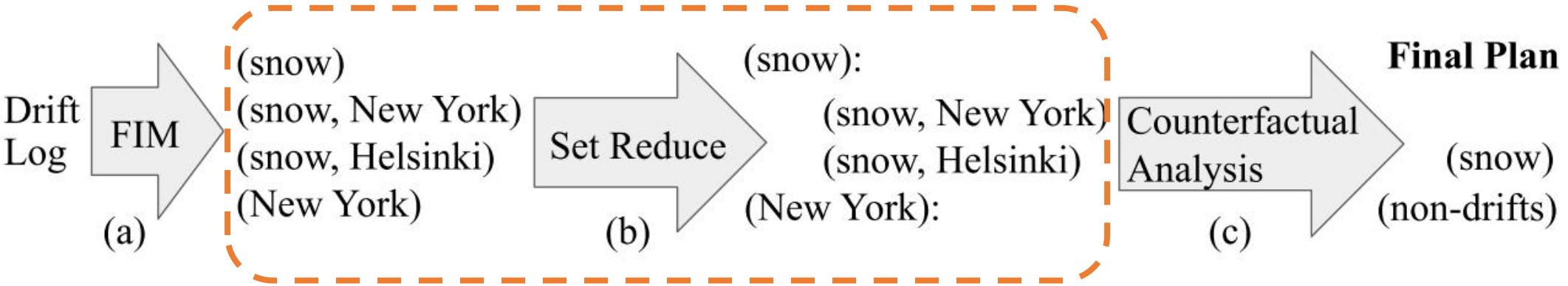
- **方法：**采用Apriori算法寻找与 Drift=True 频繁共同出现的属性组合（即“频繁项集”）。
- **使用4个指标来筛选和排序**
  - ✓ Occurrence (Occ)：该原因在所有日志中出现的频率。
  - ✓ Support (Sup)：在所有漂移日志中，该原因的占比。
  - ✓ Confidence (Conf)：出现该原因时，发生漂移的概率。
  - ✓ Risk Ratio (RR)：出现该原因相比不出现时，漂移风险提升的倍数。**RR最能衡量原因的重要性。**
- 输出：一个包含大量重叠信息的**候选原因列表。**

Time	Device ID	Weather	Location	Drift
06:02:01	android_42	clear-day	Helsinki	False
06:02:23	android_21	clear-day	New York	False
06:04:55	android_21	clear-day	New York	True
08:03:32	android_21	snow	New York	True
11:05:01	android_42	snow	Helsinki	True

Rank	Metrics				Attributes		
	Occ	Sup	RR	Conf	Weather	Location	Device ID
0	0.4	0.67	3	1	snow	-	-
1	0.2	0.3	2	1	snow	-	android_21
2	0.2	0.3	2	1	snow	-	android_42
3	0.2	0.3	2	1	snow	New York	-
4	0.2	0.3	2	1	snow	Helsinki	-
5	0.4	0.7	1.3	0.67	-	-	android_21
6	0.4	0.7	1.3	0.67	-	New York	-
7	0.4	0.7	1.3	0.67	-	New York	android_21
8	0.2	0.33	0.75	0.5	-	-	android_21
9	0.2	0.33	0.75	0.5	-	-	android_42
10	0.2	0.33	0.75	0.5	-	Helsinki	-
11	0.2	0.33	0.75	0.5	clear-day	-	android_21
12	0.2	0.33	0.75	0.5	-	Helsinki	android_42
13	0.2	0.33	0.75	0.5	clear-day	New York	-
14	0.2	0.33	0.75	0.5	-	Helsinki	-
15	0.2	0.33	0.33	0.33	clear-day	-	-

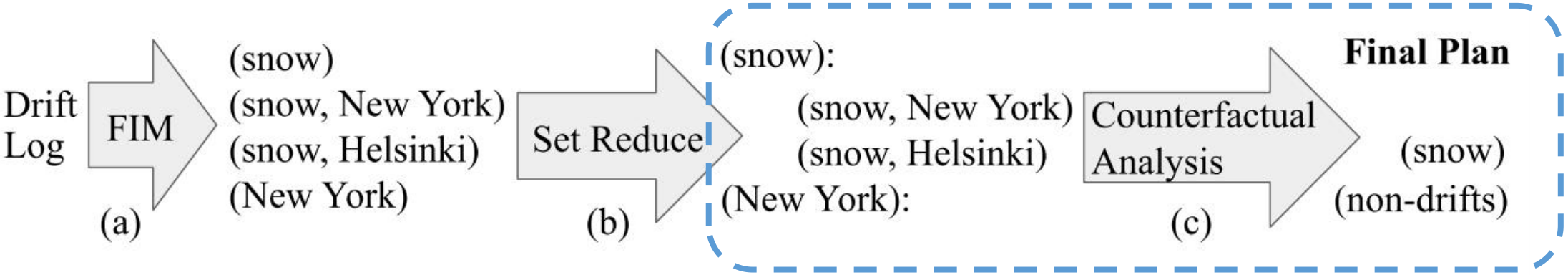
Example of FIM results

Design2-2--Set Reduce



- **目标：解决子集冗余。**例如，如果 {snow} 是一个强原因，那么 {snow, New York} 也会被识别，但后者只是前者的一个特例。
- **做法：**将更具体的子集原因 {snow, New York} 合并到**其排名更高的超集原因** {snow} 中，避免重复 。

Design2-3--Counterfactual Analysis



- **目标：解决重叠冗余。**例如，{New York} 看起来是原因，可能只是因为纽约冬天经常下雪，其影响已被 {snow} 这个原因覆盖了。
- **做法：反事实分析验证** {New York} 这个原因的独立性。假设我们已经修复了 {snow} 导致的所有漂移，再看 {New York} 本身是否还显著？如果不显著了，就将其剔除 。

Design3--By-Cause Adaptation

- **目标：针对已识别出的根本原因生成专门的模型更新版本。**
- **前提：**必须使用自监督方法，因为没有用户标注的标签。
- **自监督：采用TENT算法（ICLR'21），通过最小化模型在这批漂移数据上预测结果的熵，**来调整模型参数（轻量的批归一化BN层）进行适配。目标是让模型对新数据做出更“自信”的预测。
- **问题：**如果只对单个样本优化，模型会作弊，直接将最可能的类别概率强行改为100%，这是一种无效的过拟合。
  - **Nazar的方案：**通过批处理来解决。模型必须找到一组能同时降低一批不同样本预测熵的通用参数更新，从而避免对单个样本作弊。
- **如何提高效率：**只适配和更新轻量的批归一化层 (BN layers)，而非整个模型，极大降低了更新开销。对于ResNet50模型，一个BN层版本（约0.4MB）比一个完整模型（约92MB）要小217倍 。

Design3--By-Cause Adaptation

- **发现：**针对每个原因分别适配一个专属模型（By-cause），远优于将所有漂移数据混合在一起盲目适配一个通用模型（Adapt-all）。
- **原因：**强迫模型适应多个不同分布会导致欠拟合。

Methods	Average Accuracy (%)
No-adapt	38.7
By-cause (TENT)	61.5
By-cause (MEMO)	42.3
Adapt-all (TENT)	42.4
Adapt-all (MEMO)	30.3

- By-cause：准确率显著提升 (TENT：+22.8%)
- Adapt-all：效果很差，甚至降低准确率 (MEMO：-8.4%)

专模型专用！为“雪天”适配的模型，不应用于“雨天”。

Design3--By-Cause Adaptation

- **按因适配（By-cause）会产生多个模型版本，如何管理?**
- **策略：**云端维护一个模型池，使用LRU策略淘汰旧版本，并将最新的、最重要的几个版本组合推送到设备。
- **选择规则：**选择一个与当前输入**元数据属性匹配数量最多**的模型。
  - 例：当输入元数据为 {snow, New York} 时，系统会优先选择为 {snow, New York} 适配的模型，而不是只为 {snow} 适配的模型。

Evaluation

实验设置

- 数据集
  - Cityscapes：模拟自动驾驶场景
  - Animals：基于ImageNet，模拟野生动物识别App，并引入天气漂移和类别偏移
- 模型
  - ResNet18, ResNet34, ResNet50
- Baseline
  - no-adapt：从不更新的原始模型
  - adapt-all：将所有漂移数据混合，只适配一个通用模型
- 硬件配置
  - 漂移日志：运行在Amazon Aurora上。
  - 漂移分析：作为AWS Lambda函数运行（256MB）
  - 模型适配：运行在一个 p3.2xlarge EC2 实例上，an NVIDIA Tesla V100 GPU 。

评估目标

- Q1：检测器效果?
- Q2：根因分析准确性?
- Q3：适配方法性能?
- Q4：系统能否演进?
- Q5：端到端准确率?
- Q6：性能和扩展性?

(a) Wildlife animal example.

(b) Cityscapes example.

Evaluation1--Detection Algorithm Performance

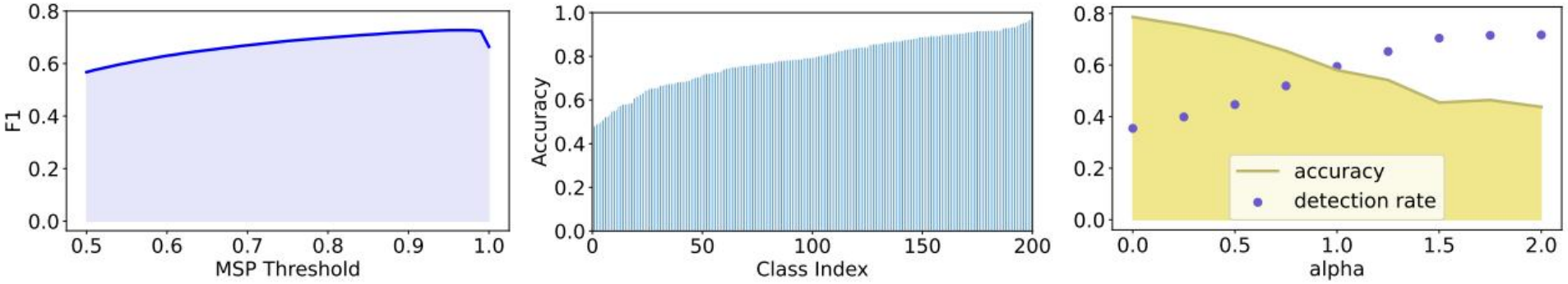
Q1: How well does Nazar's detection algorithm detect different types of data drift?

□评估方法

- 在包含一半干净数据、一半漂移数据的测试集上，评估MSP阈值检测器的F1分数。

□实验发现:

- 对于合成漂移：检测器性能稳定，当MSP阈值设为0.9时，F1分数达到峰值0.73。
- 对于类别偏移：当数据中某个类别的比例急剧增加（高skew）时，模型准确率会从78.7%降至43.8%，而漂移检出率则从0.35显著提升至0.72，证明检测器对此类漂移敏感。
- 对于真实漂移：在真实的雨天驾驶数据集上，检测器依然有效，F1分数达到0.67。



(a) F1 score by MSP threshold. (b) Accuracy across classes. (c) Rate of detection and accuracy vs skew.

Evaluation2--Root Cause Analysis Effectiveness

Q2: How effective is Nazar's root cause analysis in identifying the root cause of drift?

口评估方法

- 使用**Fowlkes-Mallows Score (FMS)** 来衡量Nazar找到的原因与ground truth的相似度。
  （FMS用于衡量两组聚类结果之间的相似性，为1代表完美匹配）
- 在8个包含不同天气组合的漂移场景下进行测试。

口实验发现

- 简单的FIM方法（第一行）在某些复杂场景下表现不佳。
- Nazar的全套三阶段流程（FIM+集合约减+反事实分析）始终得分最高（最后一行）。
- 在除Snow外的所有7个场景中，Nazar的FMS分数都达到了1.0，在Snow场景下也从0.773提升到了0.874。

Analysis Methods/ Ground Truth Drifts	None	Rain	Snow	Fog	Fog & Snow	Fog & Rain	Snow & Rain	Snow, Rain & Fog
FIM	1	0.919	0.773	1	0.891	0.921	0.926	0.917
FIM with Set Reduction	1	0.919	0.773	1	0.891	0.921	0.934	0.919
FIM with Set Reduction and CF Analysis	1	1	0.874	1	1	1	1	1

Table 5. Evaluation of root cause analysis algorithms with Fowlkes-Mallows Score (1 is optimal).

Evaluation3--Adaptation Performance

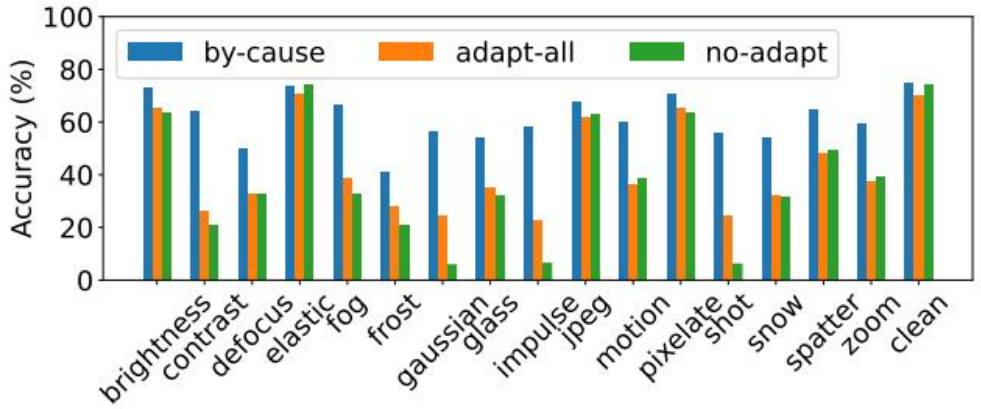
Q3: How does by-cause adaptation perform compared to prior approaches?

口评估方法

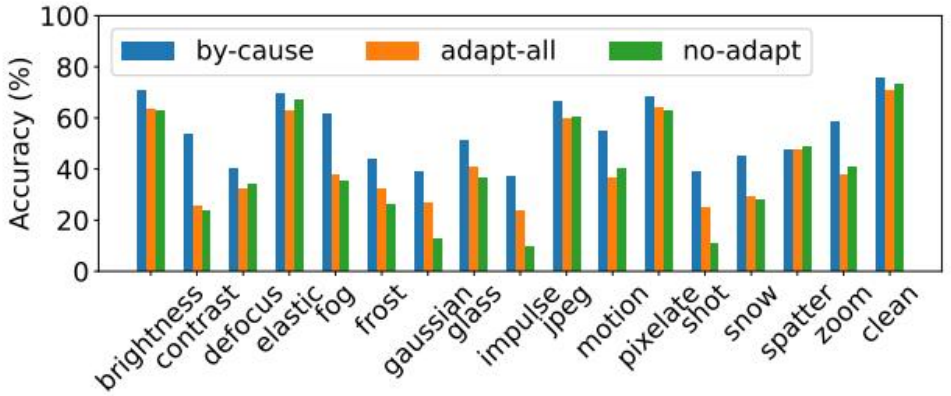
- 在理想条件下（root cause已被识别），对比三种适配策略在16种不同漂移类型上的准确率 。
- By-cause (Nazar) vs. Adapt-all (基线) vs. No-adapt (基线)。

口实验发现

- By-cause绝大部分场景下表现得最好。



(a) Accuracy of adaptation methods ($S_{adapt} = S_{test}$).



(b) Accuracy of adaptation methods ($S_{adapt} \neq S_{test}$).

Evaluation4--Evolving Drift Detection

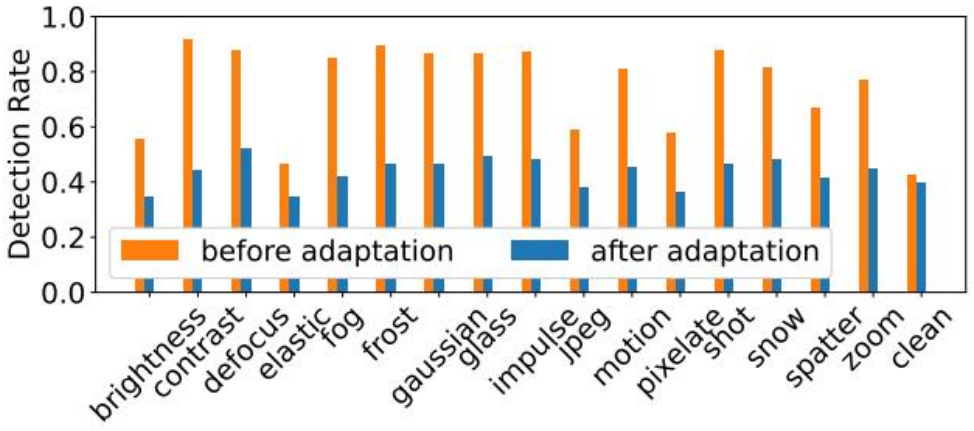
Q4: Does the detector evolve with model adaptation?

口评估方法

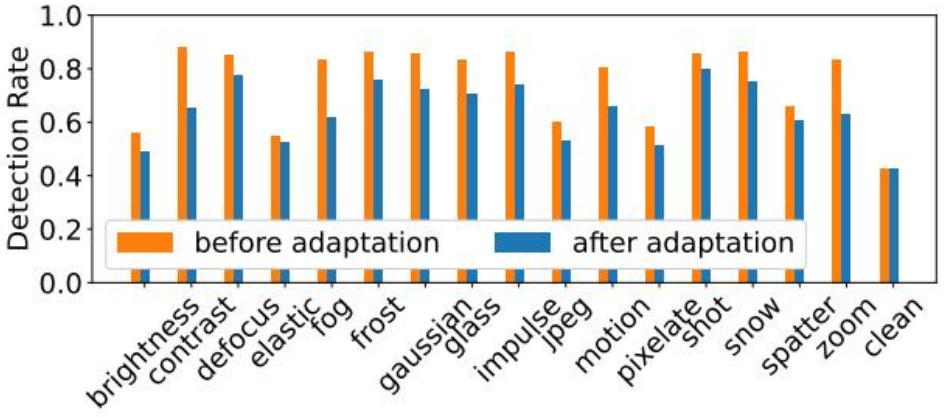
- 测量在模型适配前和适配后，漂移检测器对同一种漂移的检出率变化。

口实验发现

- 当使用了为该漂移专属适配的模型后，检出率显著下降（蓝色），几乎降到了和处理干净数据时一样的水平。



(a) Detection rate of the detector ($S_{adapt} = S_{test}$)



(b) Detection rate of the detector ($S_{adapt} \neq S_{test}$)

Evaluation5--End-to-End Performance

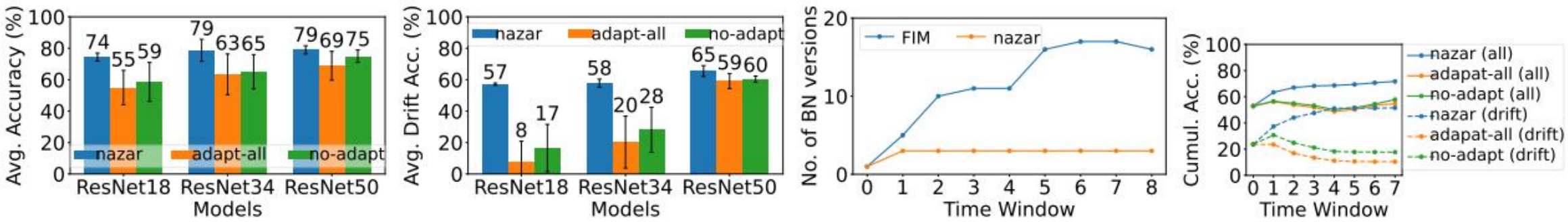
Q5: What is the accuracy in end-to-end workloads?

口评估方法

- 在Cityscapes数据集的流式工作负载上，运行完整的Nazar系统，并与基线对比。

口实验发现

- 总体准确率：Nazar的平均准确率始终最高，相比adapt-all提升了10.1% - 19.4%。
- 漂移数据准确率：在漂移数据上，Nazar的优势更明显，准确率提升最高可达49.5% 。
- 随时间变化：Nazar的累积准确率随时间稳定提升，而adapt-all出现波动和下降。



(a) Average accuracy. **(b)** Average accuracy of drifted data. **(c)** No. of model versions over time. **(d)** ResNet18 accuracy over time.

Figure 8. Experiments on cityscapes dataset. 8a and 8b show the average accuracy with three strategies. 8d shows the trace of accumulated accuracy per time window. 8c shows the number of model versions on user devices without set reduction and counterfactual analysis. Accuracy numbers in the figures are rounded to integers due to space constraint.

Evaluation6--Runtime & Scalability

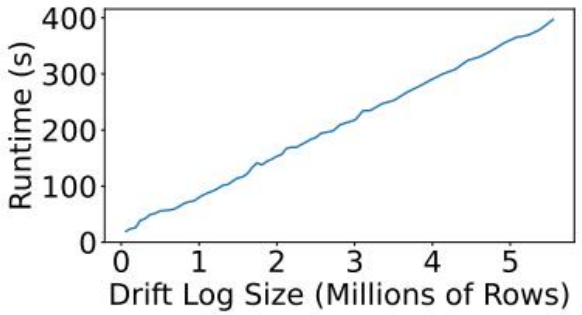
Q6: How quickly does Nazar adapt and how well does the root cause analysis scale?

口评估方法

- 测量从分析到适配完成的端到端延迟。
- 测量根因分析（Root Cause Analysis）模块的运行时间随日志数据量增加的变化情况。

口实验发现

- 运行时间 (Runtime)：端到端延迟约为50分钟，其中大部分时间（超过98%）用于可并行的模型适配，根因分析本身仅需约46秒。
- 扩展性 (Scalability)：根因分析的运行时间与漂移日志的行数几乎是线性关系。



(d) Root cause analysis scalability.

Limitations

1、根本原因的捕获不完全

- 根本原因分析**完全依赖于设备上报的元数据（属性）是否全面**。如果一个真正的漂移原因（如某型号摄像头镜片存在缺陷）没有被作为元数据记录下来，系统就无法直接定位。此时系统可能会将漂移归因于一个相关但不准确的属性（如设备型号）。

2、对特定漂移类型的处理存在短板

- 在面对严重的“类别偏移”时，Nazar的表现可能不如简单的“adapt-all”基线方法。
- 这是因为系统当前的设计没有将“类别偏移”本身作为一个可识别的根本原因，导致在适应过程中可能对不具代表性的数据样本产生过拟合。

3、隐私问题

- 系统需要设备将部分用户输入数据样本上传至云端用于模型适应。

Conclusion

Nazar：首个为移动设备设计的、全自动的端到端 ML 模型监控与适应系统，通过“按因适配”框架，在无需用户标签的情况下，解决了数据漂移导致的模型性能下降问题。

1、提出了一个完整的“检测-诊断-适应”自动化框架

✓ Nazar将轻量级的设备端漂移检测、云端根因分析与自监督模型适应整合，形成了一个可持续演进的自动化系统。

2、核心创新：“按因适配”策略

✓ 论文证明，针对特定漂移原因进行专门优化，比盲目地在所有漂移数据上进行适应更有效。

✓ 通过结合频繁项集挖掘、集合归约与反事实分析，Nazar 能够精准定位漂移的根本原因。

Idea

1、基于此：自适应漂移检测器

- Nazar的漂移检测器采用固定的MSP阈值，对于所有设备和所有漂移类型一视同仁，这可能不是最优的。可以利用Nazar**云端的分析结果，反过来为设备端训练和部署一个更智能、个性化的微型漂移检测器。**
- **数据处理：**Nazar的云端在完成根因分析后，实际上已经拥有了漂移数据和正常数据的样本。利用这些数据，我们可以在云端训练一个轻量级的分类模型。这个模型的输入不只是MSP值，还可以是模型输出Logits向量的更多统计特征，目标是更准确地分辨漂移与否。
- **部署：**针对由同一根因影响的设备组，云端可以将这个专门优化的微型检测器下发给它们，取代原有的固定阈值策略。随着系统不断运行，这个检测器也可以持续地在新数据上进行迭代更新。

Idea

2、与此类似：模型部署性能退化的根因分析

- **目标：**在模型被部署到用户设备之前，分析出**哪种压缩操作导致了模型在哪些特定类型的数据上性能下降。**
- **检测：**让原始模型和压缩后的模型在同一个数据集上运行。检测两个模型预测结果不一致的数据点，可以将这些数据点标记为“性能下降点”。
- **构建日志：**为“性能下降点”构建一个日志。其中的元数据是数据本身的属性（图片的亮度、对比度等和模型变换的操作（第5层被量化为8位、第3个模块被剪枝50%等）。
- **根因分析：**使用Nazar的频繁项集挖掘、集合归约等方法，分析到底是哪些数据属性和哪些模型变换操作的组合，与“性能下降”这个事件高度相关。
- **适应：**为ML Ops生成一份可执行的优化报告。例如：“当前的8位整型量化策略，严重影响了模型在低光照图片上的表现。根因可能在于第3和第5卷积层的权重分布对量化过于敏感。”

Idea

3、由此需要：针对流水线/级联模型的监控与适应

- Nazar目前只针对单个模型。但现实中可能是多个模型以流水线或级联的方式协同工作（例如模型A做物体检测，模型B对检测出的物体做精细分类）。能否将Nazar的问题泛化，**从单模型扩展到模型流水线的监控和适应**?
- **问题泛化**：漂移可能只发生在流水线的某一个环节（如模型B），但其影响会扩散到整个系统。根因分析不仅要回答“为什么漂移”，还要回答“是哪一个模型环节在漂移”。
- **检测**：在流水线的**每个模型后都部署**轻量级的漂移检测。
- **根因分析**：需要**分析来自不同模型环节的漂移日志**，进行因果推断，来定位问题的最初来源。是输入到模型A的数据就漂移了，还是模型A的输出导致了模型B的输入漂移?
- **适应**：**适应策略需要协同。**是只更新有问题的模型B，还是更新上游的模型A，还是同时更新多个模型以达到全局最优?

Q & A

汇报人：孙铂钛