TinyEdge

Authors

Wenzhao Zhang

Yuxuan Zhang

Hongchang Fan

Yi Gao

Wei Dong

Jinfeng Wang

Zhejiang University China. Alibaba-Zhejiang University Joint Institute of Frontier Technologies Technologies

Enabling Rapid Edge System Customization for IoT Applications: Quick Overview

1) Using a top-down approach for designing the software and estimating the performance of a customized edge system under different hardware specifications.

1) Developers select and configure modules to specify the critical logic of their interactions, without dealing with the specific hardware or software.

1) TinyEdge automatically generates the deployment package and estimates the performance after sufficient profiling

1) TinyEdge provides a unified customization framework for modules to specify their dependencies, functionalities, interactions, and configurations

Usage

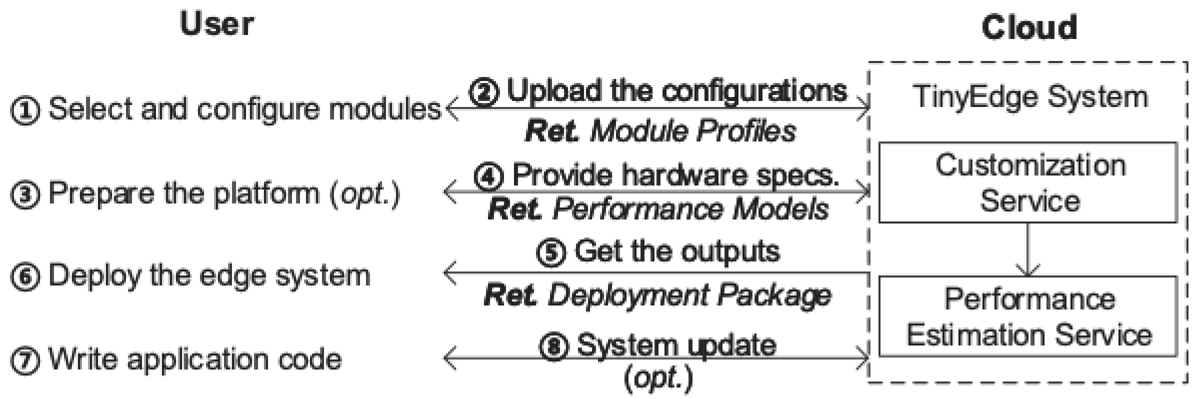
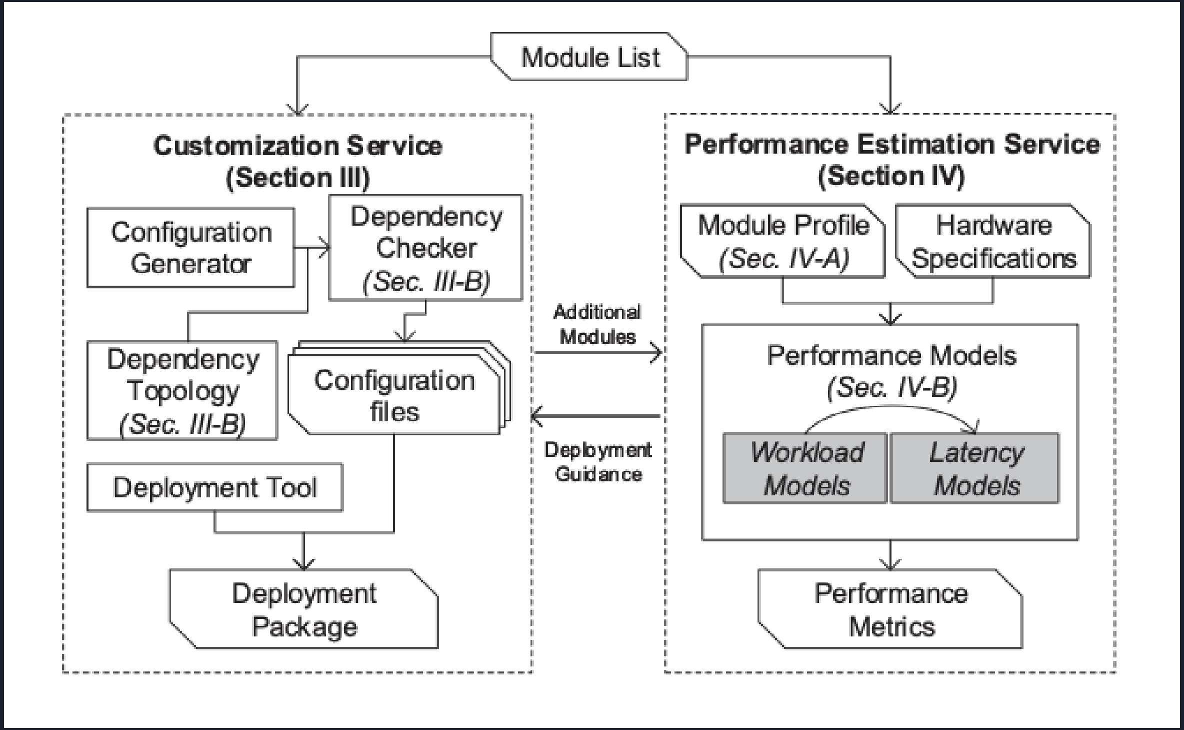


Fig. 1. Workflow overview of TinyEdge.

Design


Module List
Customization Service
(Section III)
Configuration Generator
Dependency Checker
(Sec. III-B)
Dependency Topology
(Sec. III-B)
Configuration files
Deployment Tool
Deployment Package
Additional Modules
Deployment Guidance
Performance Estimation Service
(Section IV)
Module Profile
(Sec. IV-A)
Hardware Specifications
Performance Models
(Sec. IV-B)
Workload Models
Latency Models
Performance Metrics

Customization Service

System Level Customization

User first selects modules and provides necessary configurations for them

TinyEdge dependency checker makes sure the dependencies are satisfied

System Level Customization: Key Techniques

In order to reduce module configuration time, TinyEdge proposes two configuration reduction methods

1) Configuration classification
2) Global Configuration Sharing

In order to resolve fine-grained module dependency dynamically, TinyEdge proposes a dependency checking mechanism which contains checking policies and a checking algorithm.

Application Level Customization

User writes application code using TinyEdge Domain Specific Language (DSL)

TinyEdge runtime parses the code into different parts and generates message queue topics, then distribute each part to designated execution modules or engines

Dynamic Topic Generation

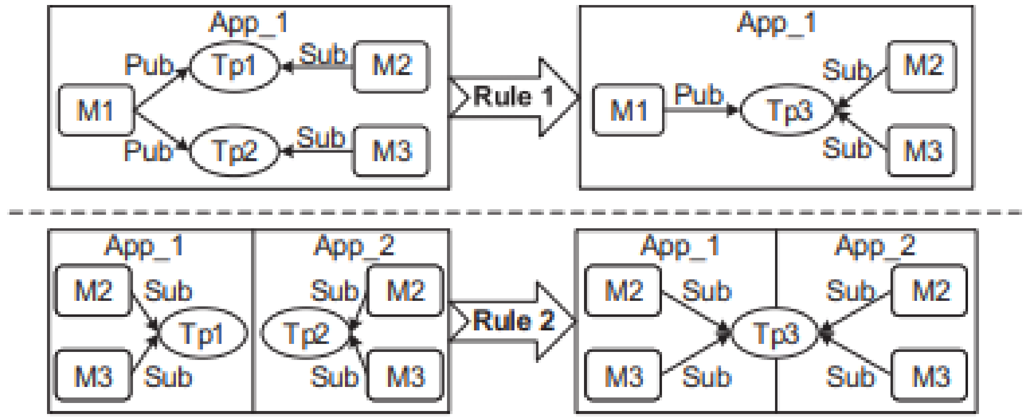


Fig. 8. TinyEdge topic generation mechanism, where M represents Module, Tp represents Topic, App represents Application.

Performance Estimation Service

Module Profile

Contains key information source for the performance models

- Module category
- Customization information (functionality and configuration)
- Performance models for specific module functions

Then TinyEdge splits modules

- System modules - make up essential part of edge system like device mgmt and logging
- Processing modules - take charge of the computation part of an edge system like data filter and video analyzing
- Connecting modules - responsible for accessing IoT devices or transmitting data to the cloud

Performance models

TinyEdge chooses to model latency and workload because

1) Both general performance metrics
2) Can describe two essential edge computing features

Does not measure accuracy

Implementation

Module Selection

Conducted small investigation of 20+ edge computing related papers and 6 mainstream industrial edge computing platforms

Findings:

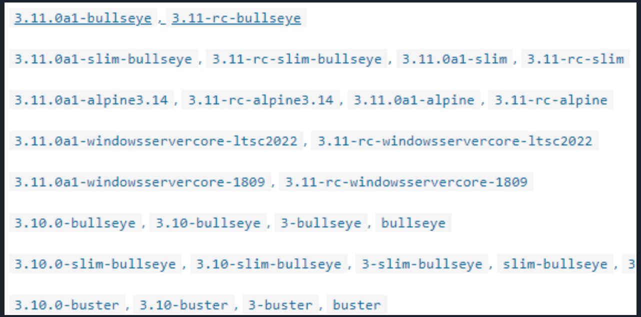
The main functionalities of edge systems for IoT applications include:

1) Connector for both IoT devices and cloud services -> HTTP, Modbus, Bluetooth, integrate EMQ
2) Data processing (e.g. stream analytics and ML) -> simple data filter and object recognition
3) Traditional database -> MySQL
4) Security -> device authentication module

Module Compaction, OS tools, and Runtime

1) Smaller base images

1) Optimized Dockerfile



```
3.11.0a1-bullseye , 3.11-rc-bullseye
3.11.0a1-slim-bullseye , 3.11-rc-slim-bullseye , 3.11.0a1-slim , 3.11-rc-slim
3.11.0a1-alpine3.14 , 3.11-rc-alpine3.14 , 3.11.0a1-alpine , 3.11-rc-alpine
3.11.0a1-windowsservercore-ltsc2022 , 3.11-rc-windowsservercore-ltsc2022
3.11.0a1-windowsservercore-1809 , 3.11-rc-windowsservercore-1809
3.10.0-bullseye , 3.10-bullseye , 3-bullseye , bullseye
3.10.0-slim-bullseye , 3.10-slim-bullseye , 3-slim-bullseye , slim-bullseye , 3
3.10.0-buster , 3.10-buster , 3-buster , buster
```

TinyEdge runtime -> enables customized system to update

- Message queue engine and serverless engine (Apache Kafka)
- Connector (OpenFaas)

Evaluation

Evaluation Cases

1) Data connection and visualization (IoT)

1) Intelligent data processing (EI)

1) Hybrid-analysis system (GIoTTO)

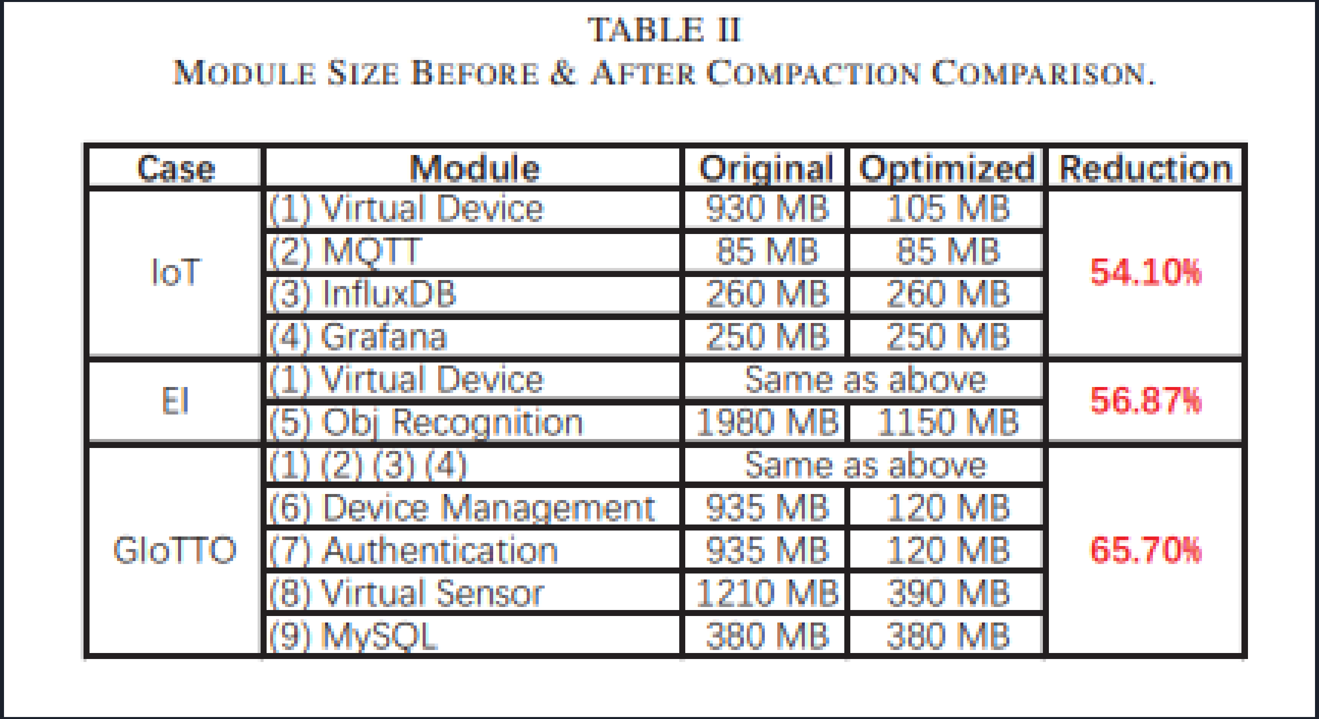


TABLE II
MODULE SIZE BEFORE & AFTER COMPACTION COMPARISON.

Case	Module	Original	Optimized	Reduction
IoT	(1) Virtual Device	930 MB	105 MB	54.10%
	(2) MQTT	85 MB	85 MB	
	(3) InfluxDB	260 MB	260 MB	
	(4) Grafana	250 MB	250 MB	
EI	(1) Virtual Device	Same as above		56.87%
	(5) Obj Recognition	1980 MB	1150 MB	
GIoTTO	(1) (2) (3) (4)	Same as above		65.70%
	(6) Device Management	935 MB	120 MB	
	(7) Authentication	935 MB	120 MB	
	(8) Virtual Sensor	1210 MB	390 MB	
	(9) MySQL	380 MB	380 MB	

Module Size Before and After Compaction Comparison

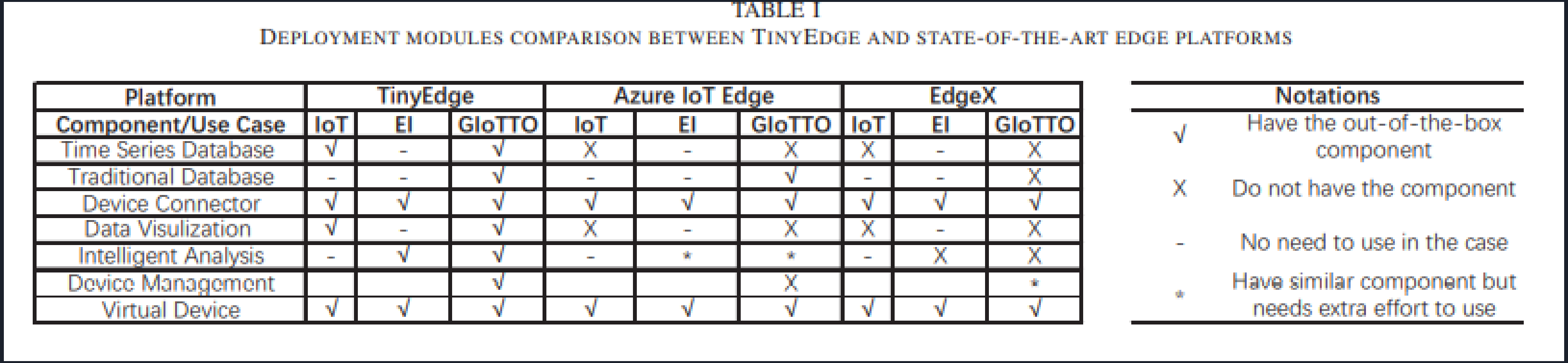


TABLE I
DEPLOYMENT MODULES COMPARISON BETWEEN TINYEDGE AND STATE-OF-THE-ART EDGE PLATFORMS

Platform	**TinyEdge**			**Azure IoT Edge**			**EdgeX**		
Component/Use Case	**IoT**	**EI**	**GIoTTO**	**IoT**	**EI**	**GIoTTO**	**IoT**	**EI**	**GIoTTO**
Time Series Database	√	-	√	X	-	X	X	-	X
Traditional Database	-	-	√	-	-	√	-	-	X
Device Connector	√	√	√	√	√	√	√	√	√
Data Visulization	√	-	√	X	-	X	X	-	X
Intelligent Analysis	-	√	√	-	*	*	-	X	X
Device Management			√			X			*
Virtual Device	√	√	√	√	√	√	√	√	√

Notations	
√	Have the out-of-the-box component
X	Do not have the component
-	No need to use in the case
*	Have similar component but needs extra effort to use

Customization related results

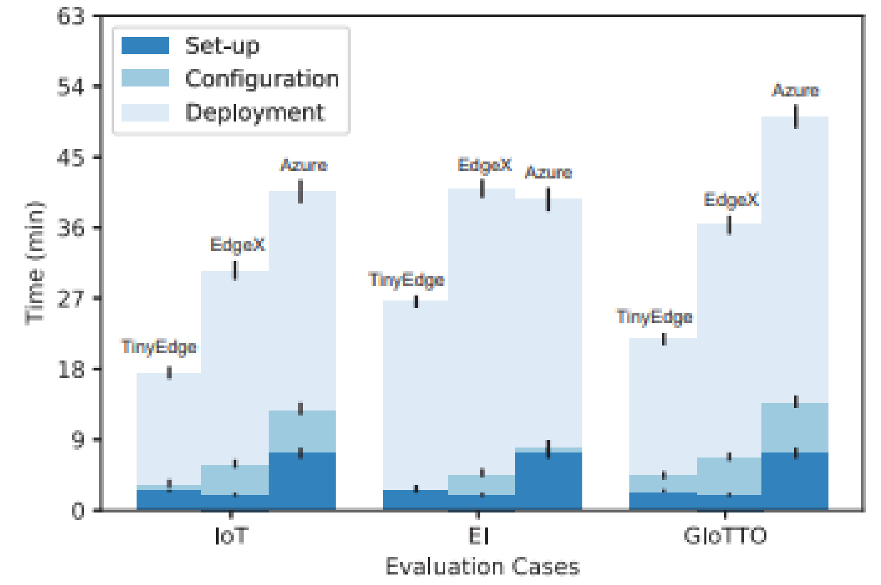


Fig. 10. System-level customization time comparison among TinyEdge, EdgeX, and Azure IoT Edge.

System level customization results

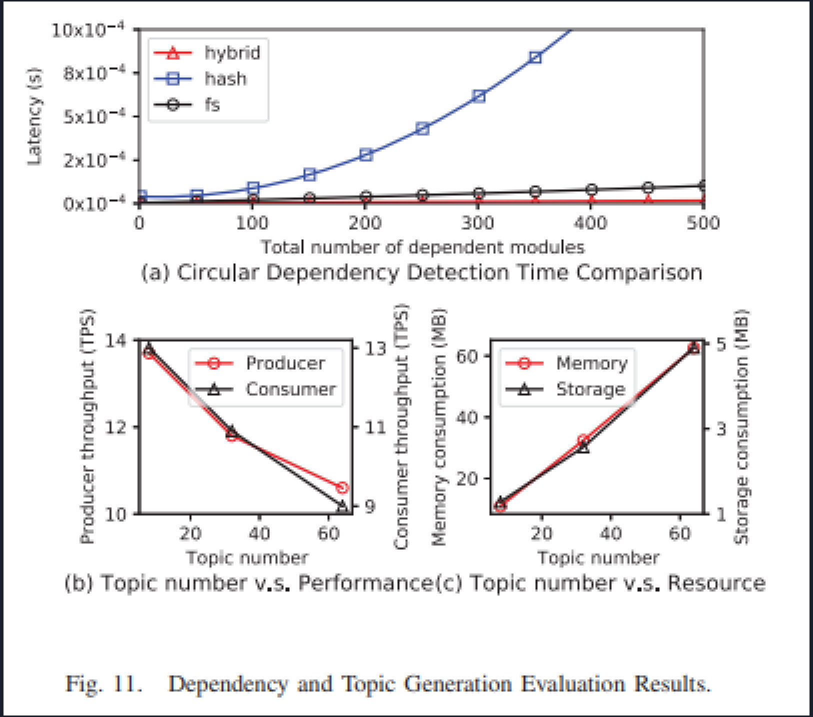


Fig. 11. Dependency and Topic Generation Evaluation Results.

Dependency checking efficiency comparison results

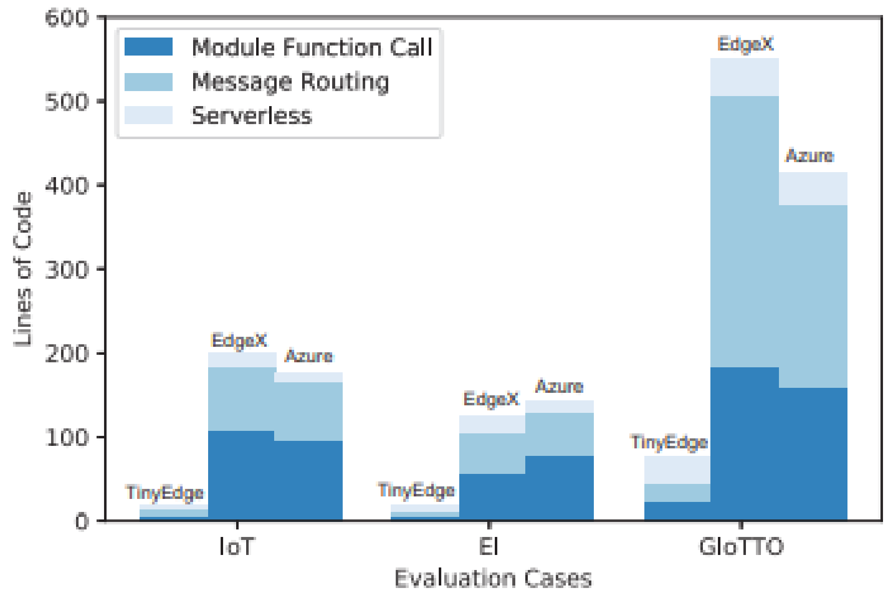


Fig. 12. Application-level customization code comparison among TinyEdge, EdgeX, and Azure IoT Edge.

Application-level customization results

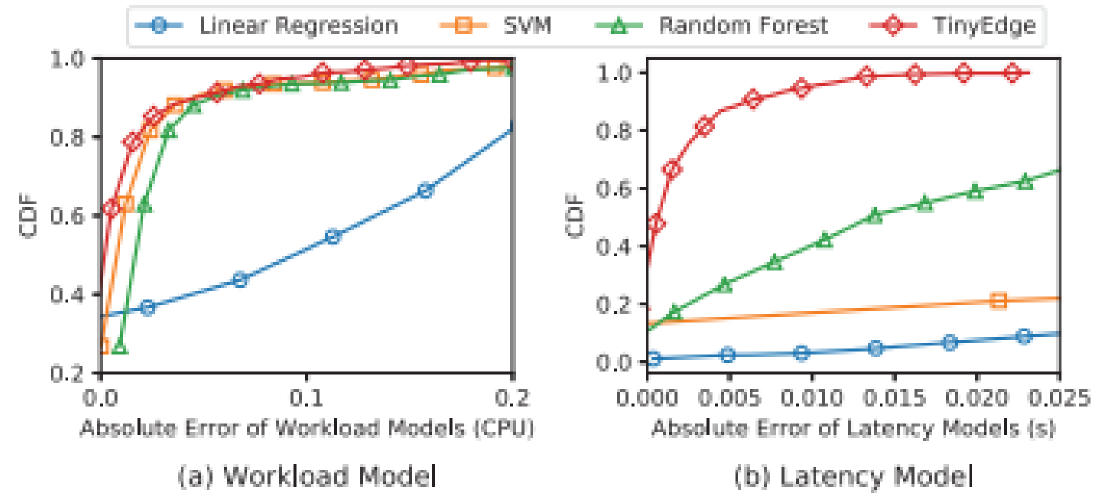


Fig. 13. Performance Modeling Comparison Between TinyEdge and Conventional ML Methods.

Performance estimation related results

Final Results

Final results show that TinyEdge reduces customization time of edge systems, reducing 44.15% of customization time and 67.79% lines of on average while giving accurate performance estimation in various settings

Discussion

1) Effectiveness of evaluation cases and components. Are they representative?
2) Dependency checking efficiency?
3) Irrelevant data? (# of topics vs performance and resources)
4) Effectiveness of configuration reduction methods

Most importantly:

1) What did we learn?
    - Customization time of set up, configuration, and deployment can be reduced
    - The number of lines of code to customize can be reduced
        - Follow up: Is this a valuable metric?
    - Can more accurately estimate performance for the user
1) Does what we learned provide value for the IoT Industry?