

# Erebus: Access Control for Augmented Reality Systems

Sanket Goutam\*, Yoonsang Kim\*, Amir Rahmati, Arie Kaufman

Stony Brook University

# Two form factors for building AR Systems

## Standalone



Oculus Quest 2



HoloLens 2

## Companion Device

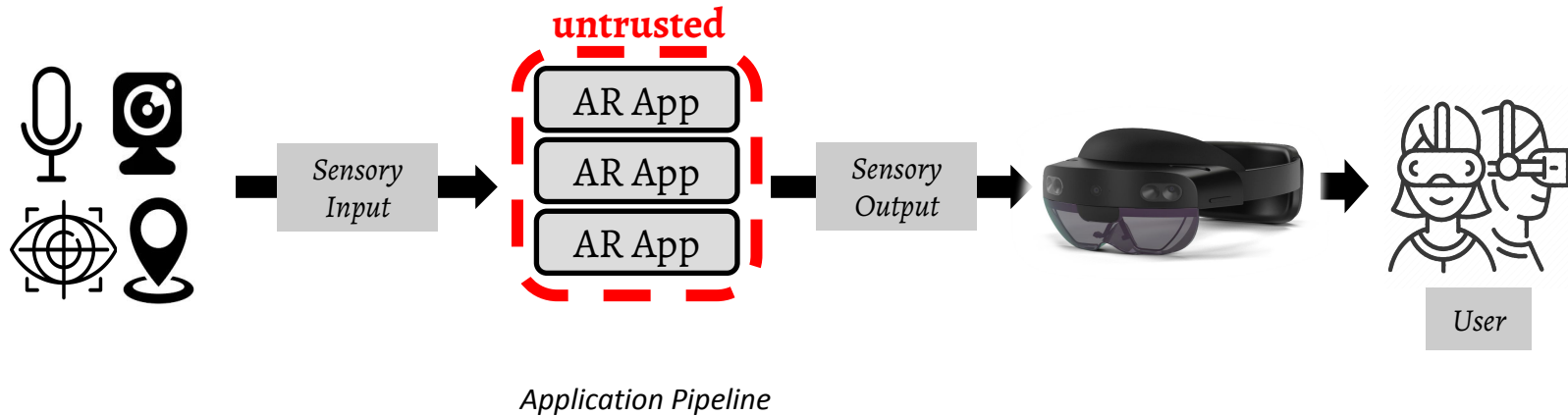


Rokid Air Pro

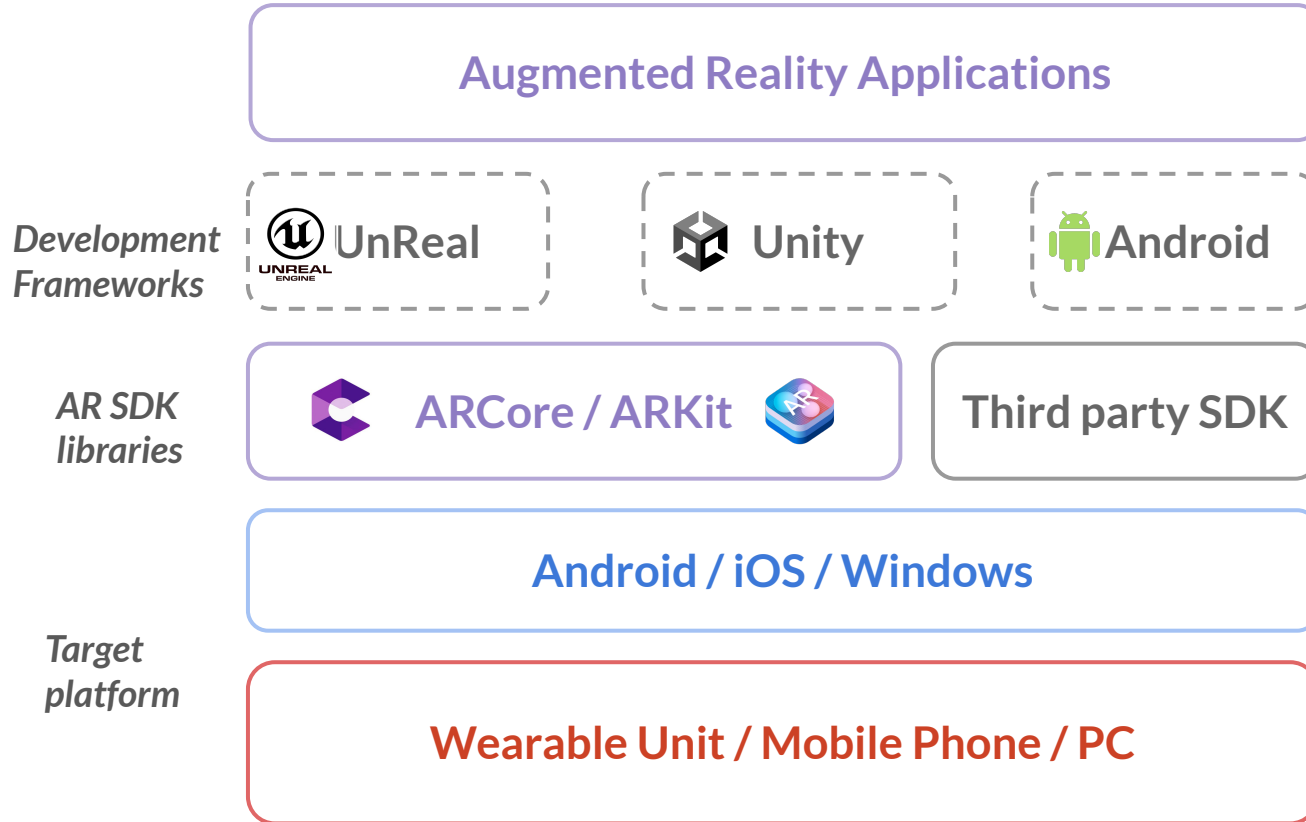


Toshiba  
dynaEdge

# Applications derive information from device sensors.



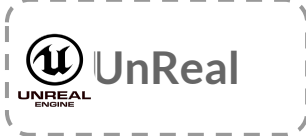
# How are these applications developed?



# Dichotomy between data required and access requested.

## Augmented Reality Applications

*Development Frameworks*



Developers use high-level APIs to access sensor data.

*AR SDK libraries*



## Android / iOS / Windows

*Target platform*



Permission enforcement applied only on the target platform.

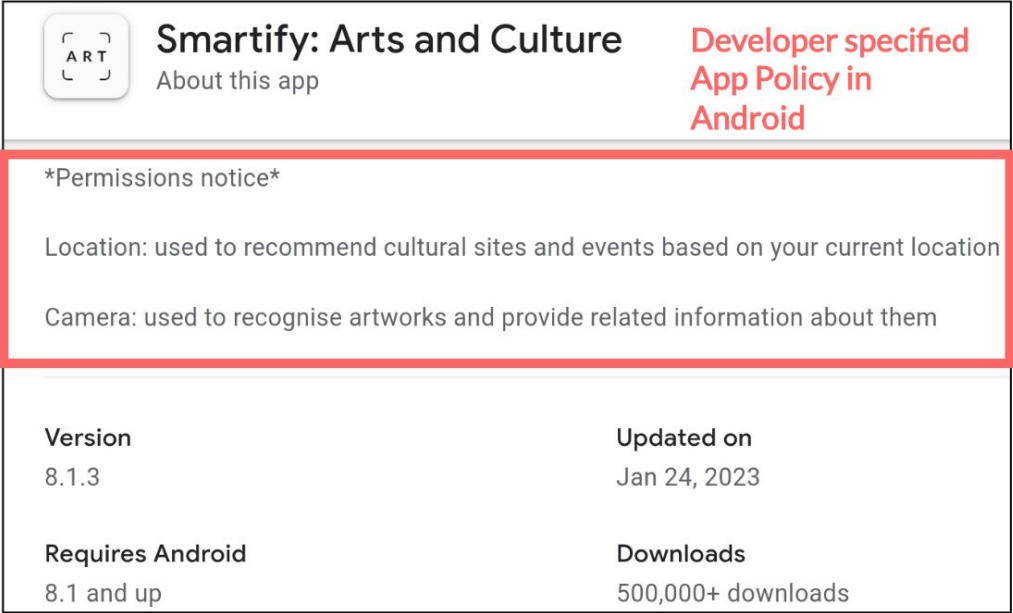
# Permission Control similar to Smartphone OS.

AR Device Type	Device Name	Platform	Access Control Mechanism
Standalone Wearable	Meta Quest 2 [43]	Android	App Manifest
	Microsoft HoloLens 2 [44]	Windows	App Manifest, Policy CSP
	Magic Leap 2 [16]	Android	
	Google Glass Enterprise [23]	Android	
	ThirdEye X2 MR Smart Glasses [22]	Android	
	Vuzix Blade AR [70]	Android	
	Snap Spectacles [67]	Android	
	Raptor AR Headset [19]	Android	
	Kopin Solos [36]	Android	
Xiaomi Smart Glasses [68]	Android		
With a Companion Device	Lenovo ThinkReality A3 [40]	Android	
	Epson Moverio [18]	Android	
	Toshiba dynaEdge [63]	Windows	No AC mechanism
	Rokid Air Pro [52]	Android, iOS	App Manifest
	NReal Light [46]	Android	No AC mechanism
	Viture One [69]	Android	No information available
Dream Glass Flow [66]	Android, iOS	No information available	


```

<uses-feature android:name="android.hardware.camera"
  android:required="true" />
<uses-permission android:name="android.permission.record_audio"
  android:required="true" />
<uses-feature android:name="android.hardware.location.GPS"
  android:required="true" />
<uses-feature android:name="android.hardware.sensor.heartrate"
  android:required="true" />
    
```

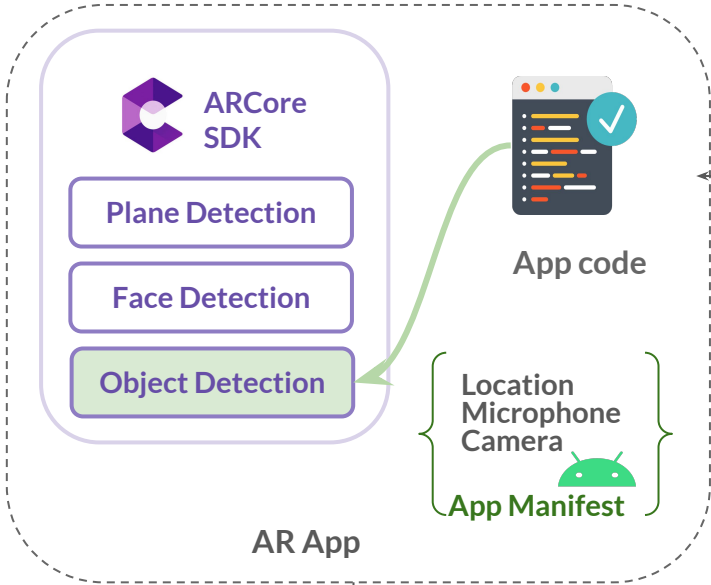
# Developer specifies an access policy to user on Play Store.



The screenshot shows the app page for 'Smartify: Arts and Culture'. The app icon is a square with 'ART' inside. The title is 'Smartify: Arts and Culture' and there is a link 'About this app'. A red text label 'Developer specified App Policy in Android' is positioned to the right. A red-bordered box highlights the permissions notice section, which includes the text '\*Permissions notice\*', 'Location: used to recommend cultural sites and events based on your current location', and 'Camera: used to recognise artworks and provide related information about them'. Below this, a table provides technical details: Version 8.1.3, Updated on Jan 24, 2023, Requires Android 8.1 and up, and Downloads 500,000+.

	<b>Smartify: Arts and Culture</b> About this app	Developer specified App Policy in Android
<b>*Permissions notice*</b>		
Location: used to recommend cultural sites and events based on your current location		
Camera: used to recognise artworks and provide related information about them		
<b>Version</b>	8.1.3	<b>Updated on</b>
		Jan 24, 2023
<b>Requires Android</b>	8.1 and up	<b>Downloads</b>
		500,000+ downloads

# User installs the app on their device.

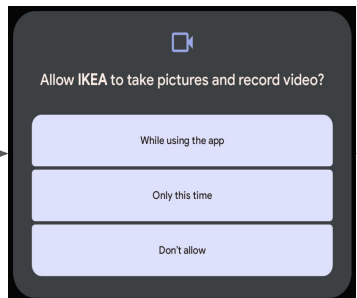


Access sensor data.

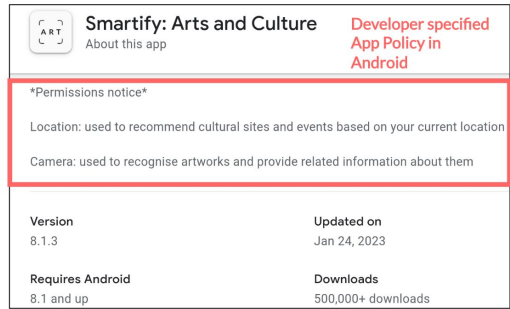


Device Sensors

Prompt user to grant access to sensors.




Grants permissions on first use.





# Malicious app can **violate app policy**.

 **Smartify: Arts and Culture** Developer specified App Policy in Android  
About this app

---

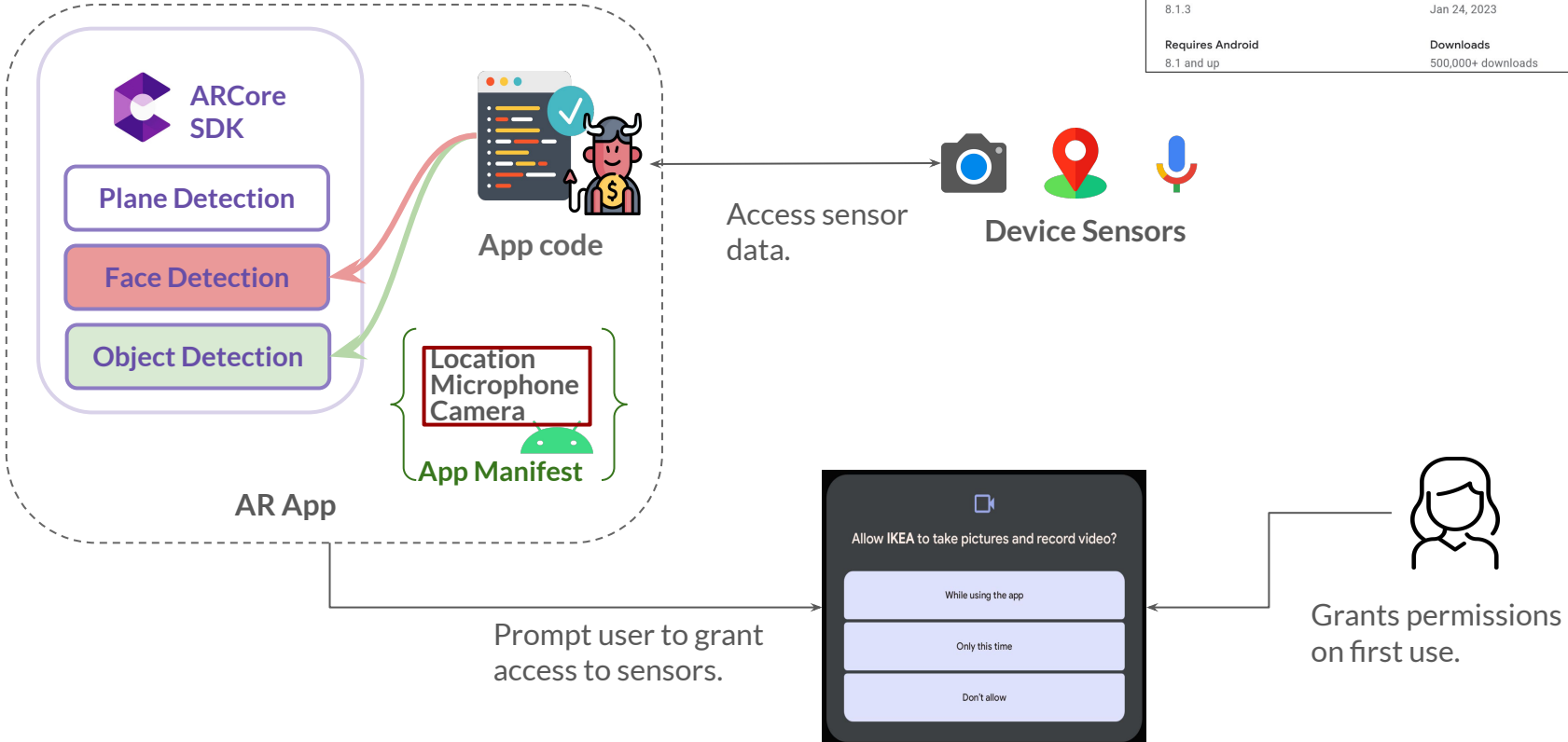
**\*Permissions notice\***

Location: used to recommend cultural sites and events based on your current location

Camera: used to recognise artworks and provide related information about them

---

<b>Version</b> 8.1.3	<b>Updated on</b> Jan 24, 2023
<b>Requires Android</b> 8.1 and up	<b>Downloads</b> 500,000+ downloads



# Can we reimagine Access Control for VisionOS?

SPATIAL COMPUTING —

## Unity's visionOS support has started to roll out—here's how it works

A closed beta will admit developers gradually over the coming weeks.

SAMUEL AXON - 7/19/2023, 3:51 PM

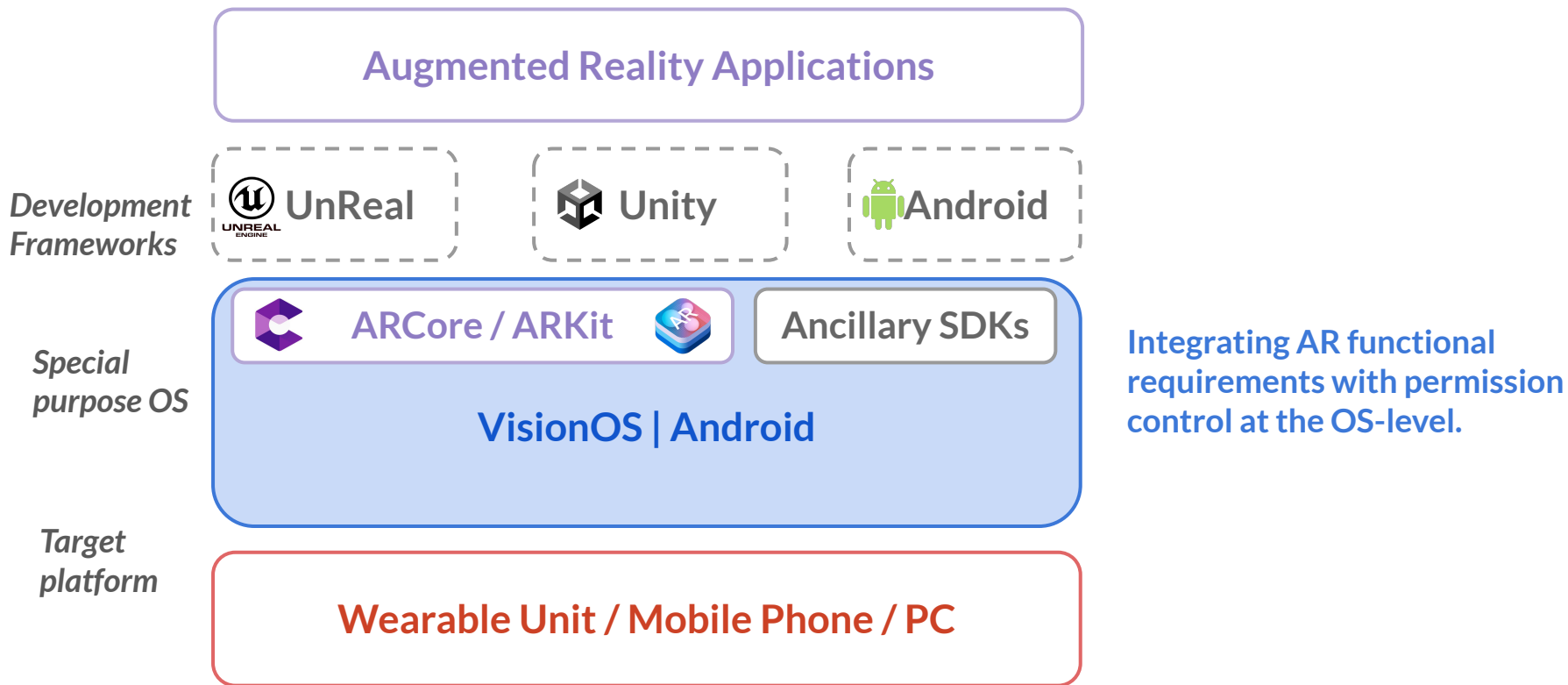


**G1.** How can we regulate direct access to sensors?

**G2.** How to ensure a least privilege access based on developer-specified policy, allowing access to what's required and nothing more?

**G3.** Can we allow users to adjust access based on their requirements?

# Erebus: regulating sensor access at the OS-level



# Erebus: policy specification language that expresses functionality

The screenshot shows the Google Play Store page for the app 'Smartify: Arts and Culture'. The app icon is a square with 'ART' inside. The text 'Smartify: Arts and Culture' is followed by 'About this app' and 'Developer specified App Policy in Android'. A red box highlights the '\*Permissions notice\*' section, which contains two entries: 'Location: used to recommend cultural sites and events based on your current location' and 'Camera: used to recognise artworks and provide related information about them'. Below this, there is a table with app details.

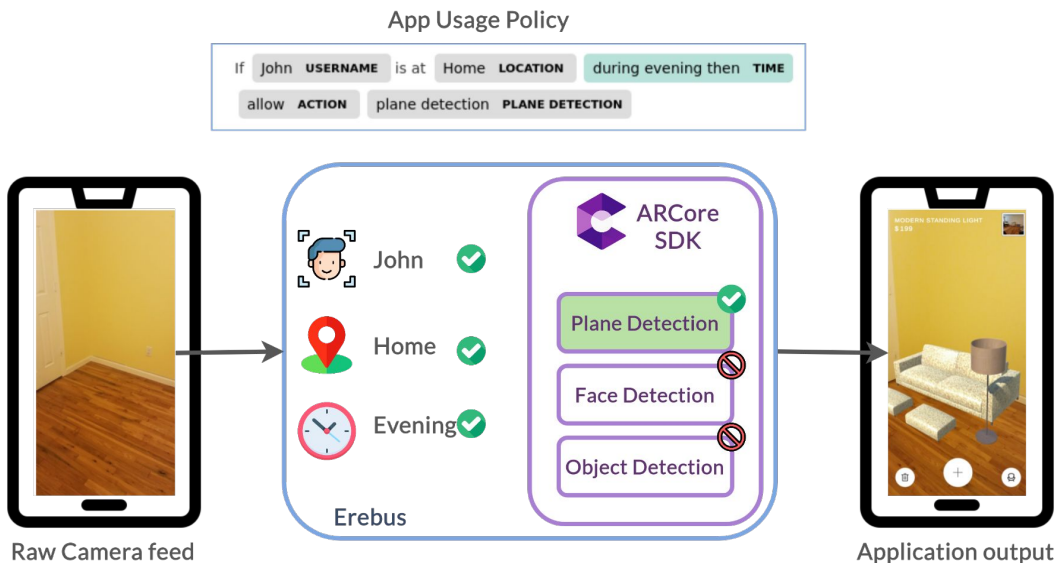
Version	Updated on
8.1.3	Jan 24, 2023
Requires Android	Downloads
8.1 and up	500,000+ downloads

- Coarse-grained access requirement. **(Location, Camera)**
- Functional requirement cannot be enforced by the system **(recognize artworks).**

- Functional description in a semi-structured natural language format.
- Fine-grained permission enforcement.

Allow **ACTION** this app to detect objects **OBJECT TRACKING**  
that are **QR codes** **OBJECTNAME** only during evening when **TIME**  
I am **USER** Home **LOCATION**

# Erebus: users define *what, when, and where* data can be accessed



System validates app's sensor request based on context-dependent policy specification, ensuring *least-privilege*.

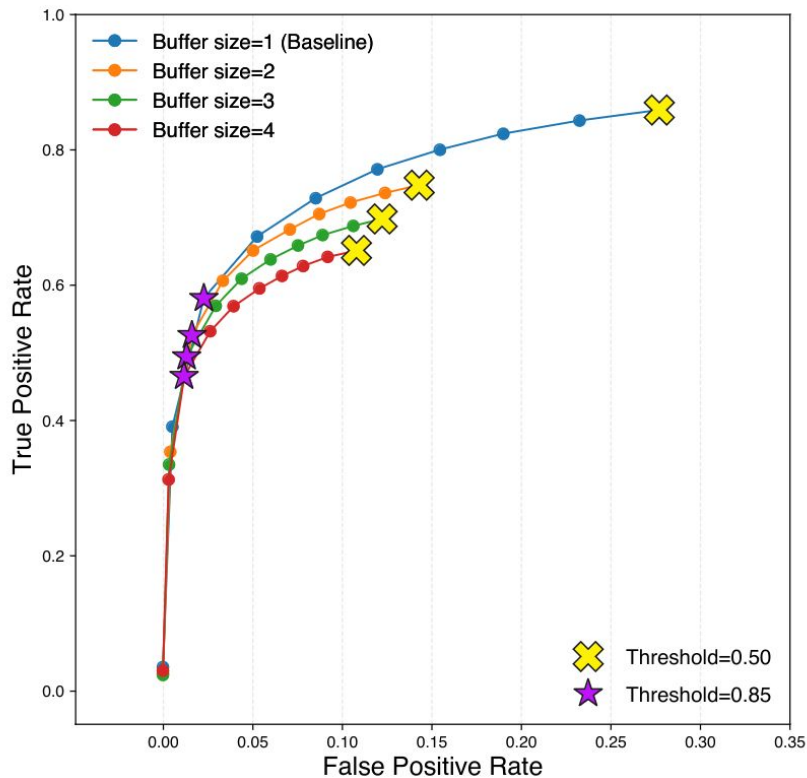
# Erebus: preventing sensitive data leakage

Object detection is an imperfect process. False positives could leak sensitive information to the app.



# How does **Erebus** prevent leaking sensitive data due to false positives?

We leverage *conflation* technique to optimize object detection accuracy and reduce false positives in Erebus.



## Does **Erebus** incur additional latency over API calls?

<b>API Type</b>	<b>Erebus (ms)</b>	<b>Unprotected (ms)</b>
Camera sensor-based API	$0.35 \pm 0.12$	$0.18 \pm 0.04$
Location sensor-based API	$0.22 \pm 0.04$	$<0.01$

By enforcing runtime checks on API calls, there is a small overhead incurred by Erebus but this has no impact on performance.

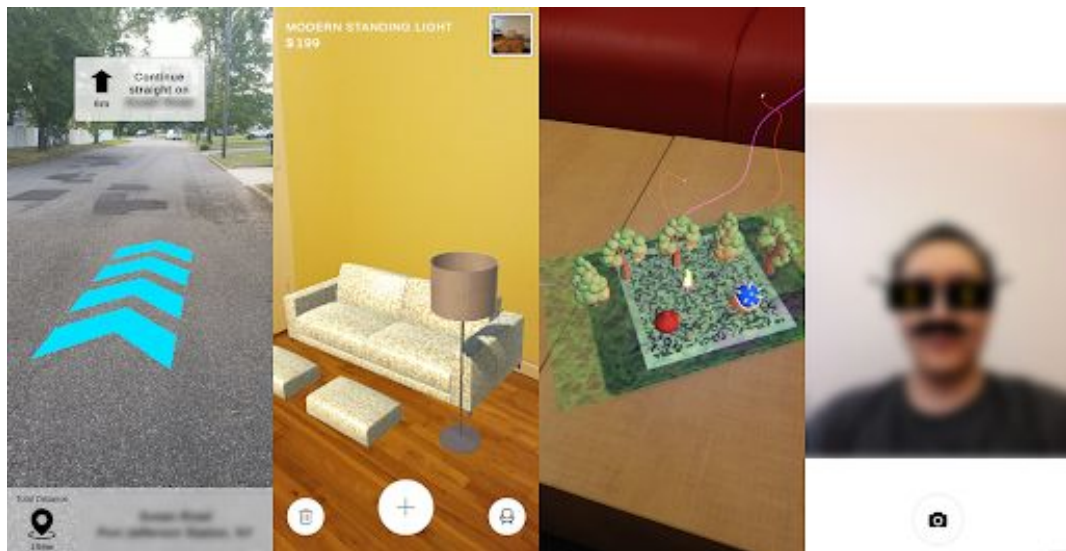


# Does **Erebus** affect app's overall performance?

Component Type	Component	Latency (ms)
Erebus	Object Detection	28.91
	Non-Max Suppression	0.02
	Object Tracking	0.25
	Conflation	0.01
	Whitelisting	0.08
Application	Async GPU Readback (Constant)	181.72
	Application Logic	33.47
Overall Latency		<b>244.46</b>

Our prototype apps were able to run at **~34.16 FPS** with Erebus framework enforcing runtime checks.

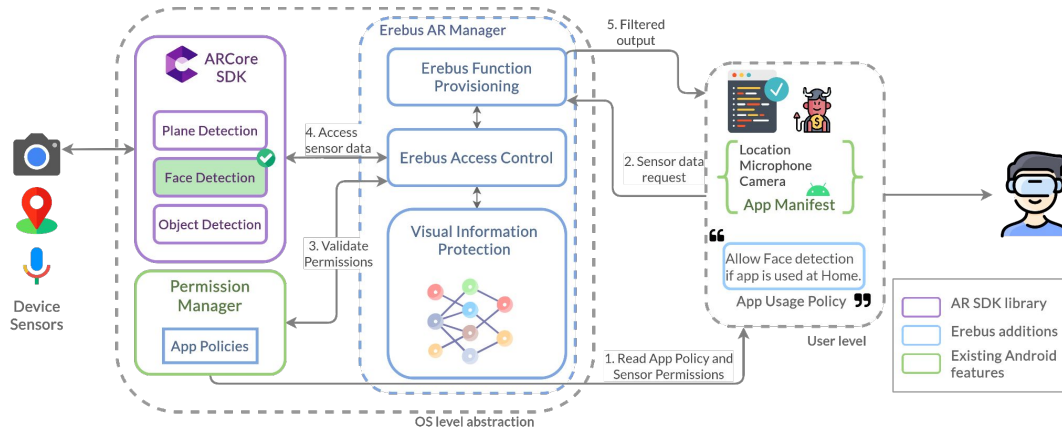
# Erebus: adapting the framework



- Implemented on Google ARCore SDK using Unity Framework.
- Adapted 5 prototype AR applications to our framework.
- We open-source our framework implementation, policy-language design, and prototype applications for developer's reference.



# Erebus: Access Control for Augmented Reality Systems



Sanket Goutam\*, Yoonsang Kim\*, Amir Rahmati, Arie Kaufman

Stony Brook University



[https://github.com/Ethos-lab/erebus-AR\\_access\\_control](https://github.com/Ethos-lab/erebus-AR_access_control)



<https://sgoutam.github.io>



@sanketgoutam