Samsung PRISM Project Report: On-Device Personalization (ODP) Package Implementation

Final Report

Work-let Name: Secure ML model state teansfer using Android ODP

Worklet Details

- Worklet ID:
23OD25

2. College Name:
Vellore Institute of Technology, Chennai

Team

1. College Professor(s):
   Dr Pattabiraman.V / pattabiraman.v@vit.ac.in
   Dr. Rajiv Vincent / rajiv.vincent@vit.ac.in
2. Students:
   1. Adarsh Gaurav / adarsh.gaurav2021@vitstudent.ac.in
   2. Goyam Jain / goyam.jain2021@vitstudent.ac.in
   3. Raghav Matta/raghav.matta2021@vitstudent.ac.in
   4. Rishu Singh / rishuravi.singh2021@vitstudent.ac.in
   5. Pranav Joshi / pranav.joshi2021@vitstudent.ac.in
3. Department: School of Computer Science and Engineering (SCOPE)

Introduction :

The Samsung PRISM (Preparing and Inspiring Student Minds) project is an initiative aimed at fostering collaboration between academia and industry to solve real-world problems. This project provides students with the opportunity to work on cutting-edge technology and gain hands-on experience in software development. Our team was assigned a problem statement that focused on implementing an On-Device Personalization (ODP) package with a layered API structure.

On-device personalization is an emerging technology that allows applications to provide personalized experiences to users by processing data directly on the device. This approach ensures privacy and security since user data does not need to be sent to the cloud. The project involved several key tasks, including building the ODP package, verifying its functionality on an emulator, understanding the architecture, and developing a test application.

Our specific problem statement required us to work within the Android Open Source Project (AOSP) environment to integrate the ODP service, ensuring it runs smoothly on an emulator. This task required a deep understanding of the AOSP build system, Android services, and the new ODP APIs. We faced several challenges, including the lack of documentation and device configuration issues, but we managed to make significant progress in understanding and implementing the ODP package.

This report details the steps we took, the progress we made, and the challenges we encountered. It serves as a comprehensive overview of our efforts and provides insights into the complexities involved in integrating new services into the AOSP environment.

Objectives :

The primary objectives of the project were:

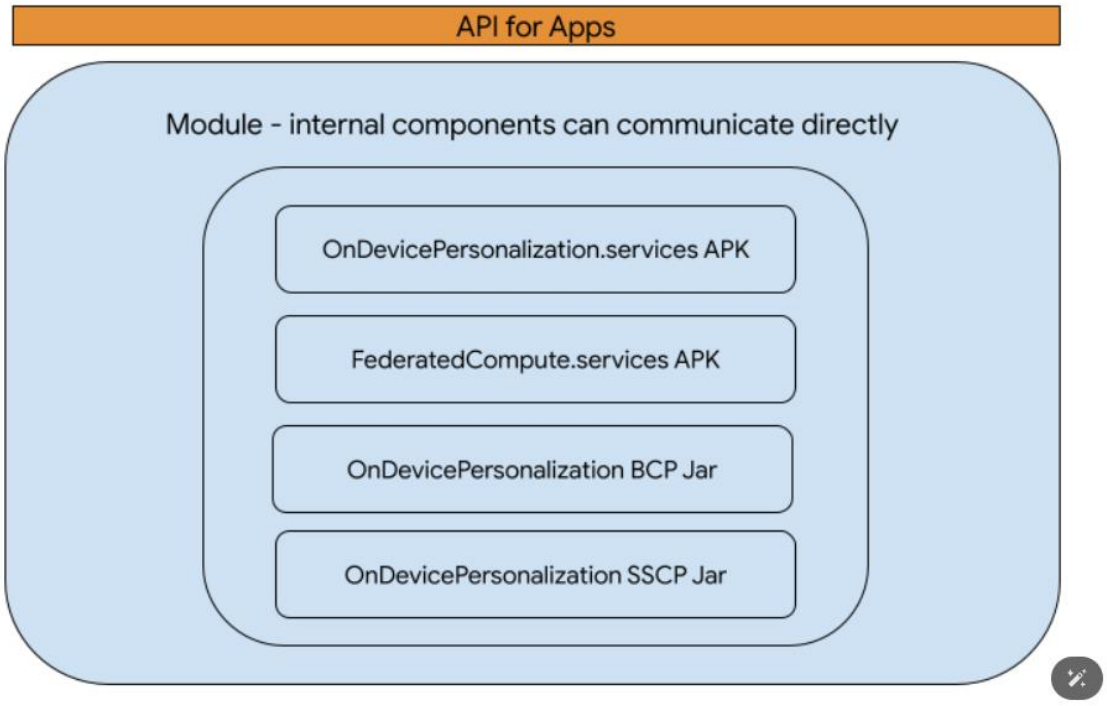
- *Build the ODP package after the AOSP build, if not part of the emulator.*
- *Verify whether the emulator can run the ODP service and, if not, install and make the service runnable.*
- *Check module components, understand the architecture, and prepare a design flow.*

- *Install the test package, understand API implementation, and list all system API sequence flows.*
- *Prepare a test app after understanding the test implementation.*

Our Learnings about Device-Personalization

The OnDevicePersonalization module, introduced in Android 13, provides a set of building blocks developed with user privacy as their core tenet, to support development of APKs that offer a personalized experience for their users. Examples of the building blocks provided include a policy engine to guard the ingress, egress, and allow-listed operations of user data. User controls can be expressed as policies that are enforced by this policy engine. Another example of the building blocks provided includes various federated computations, such as federated learning and federated analytics, that enable collaborative training of machine learning models and analysis of local raw data without central data collection.

OnDevicePersonalization attempts to create a developer experience that removes bottlenecks that arose from data collection, consent, control and compliance. This allows OEMs and app developers to focus on the novel and semantically interesting parts of their applications and take advantage of the super-rich and real-time data that's available only on the devices.



Task Breakdown and Progress

Task 1: Build ODP Package after AOSP Build

Objective:

Build the ODP package post-AOSP build if it is not part of the emulator.

Steps Taken:

Install Required Packages:

1. To begin with, we installed the necessary packages on an Ubuntu system to set up the AOSP build environment:

```
/sudo apt-get update
//sudo apt-get install git-core gnupg flex bison build-essential zip
curl zlib1g-dev gcc-multilib g++-multilib libc6-dev-i386 lib32ncurses5-
dev x11proto-core-dev libx11-dev lib32z1-dev libgl1-mesa-dev libxml2-
utils xsltproc unzip
```

2. Initialize the Repo Tool:

We then initialized the Repo tool by downloading it and setting the necessary permissions:

```
mkdir ~/bin
PATH=~/bin:$PATH
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
chmod a+x ~/bin/repo
```

3. Initialize the AOSP Source:

We created a directory for AOSP, moved into it, and initialized the AOSP source:

```
mkdir ~/aosp
cd ~/aosp
repo init -u https://android.googlesource.com/platform/manifest -b android-14
repo sync -j$(nproc --all)
```

4. Configure Environment:

Configured the build environment:

```
source build/envsetup.sh
```

5. Choose a Target:

Selected a target to build, specifically for an emulator build:

```
lunch aosp_x86-eng
```

Outcome:

The ODP package was successfully built and integrated into the AOSP build system.

Task 2: Verify Emulator ODP Service

Objective:

Check whether the emulator can run the ODP service, and if not, install the service and make it runnable.

Steps Taken:

1. Run Emulator: After building the AOSP for the emulator, we ran the emulator to verify if it could run the ODP service:

2. Verification: Checked the availability of the ODP service in the emulator. This involved inspecting the running services and confirming the presence of the ODP service.

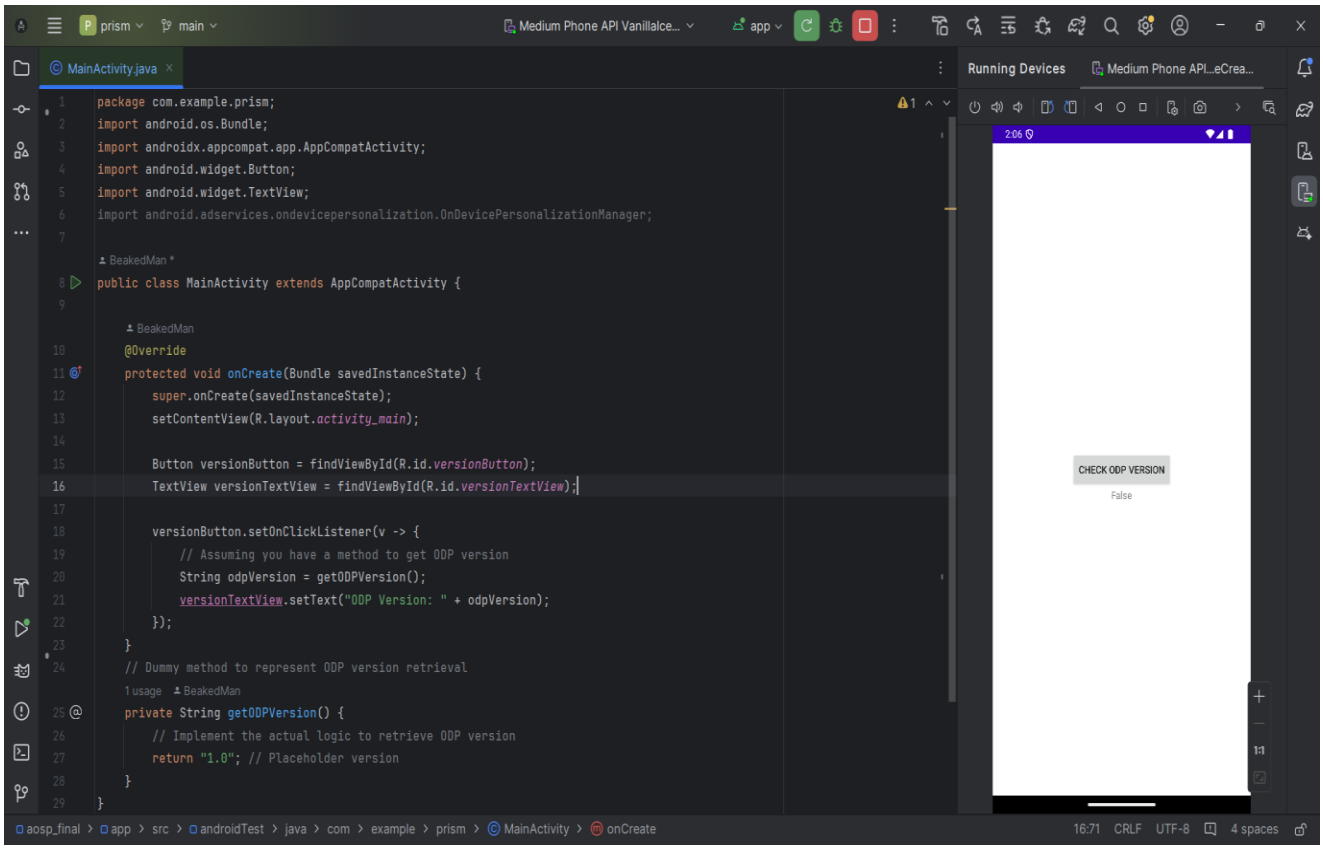
3. Service Installation: When the service was not found, we manually installed the ODP service. This required configuring the emulator environment to accept the new service installation.

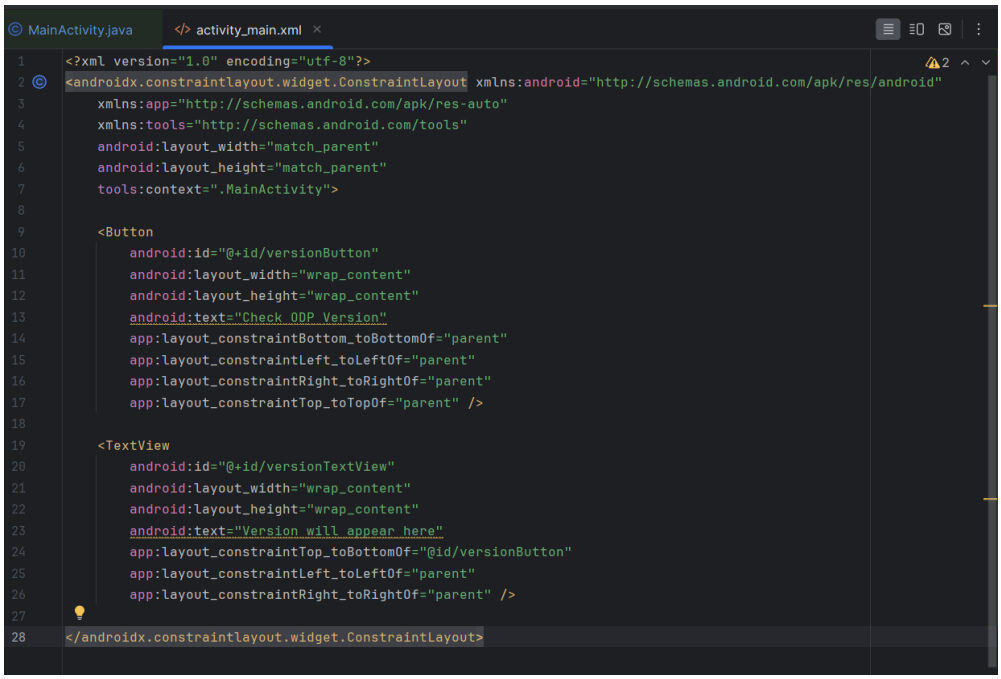
4. Service Execution :
   Made necessary configurations to ensure the service was runnable. This included setting appropriate permissions and ensuring the service started without issues.
5. ODP service verification and Version Retrieval

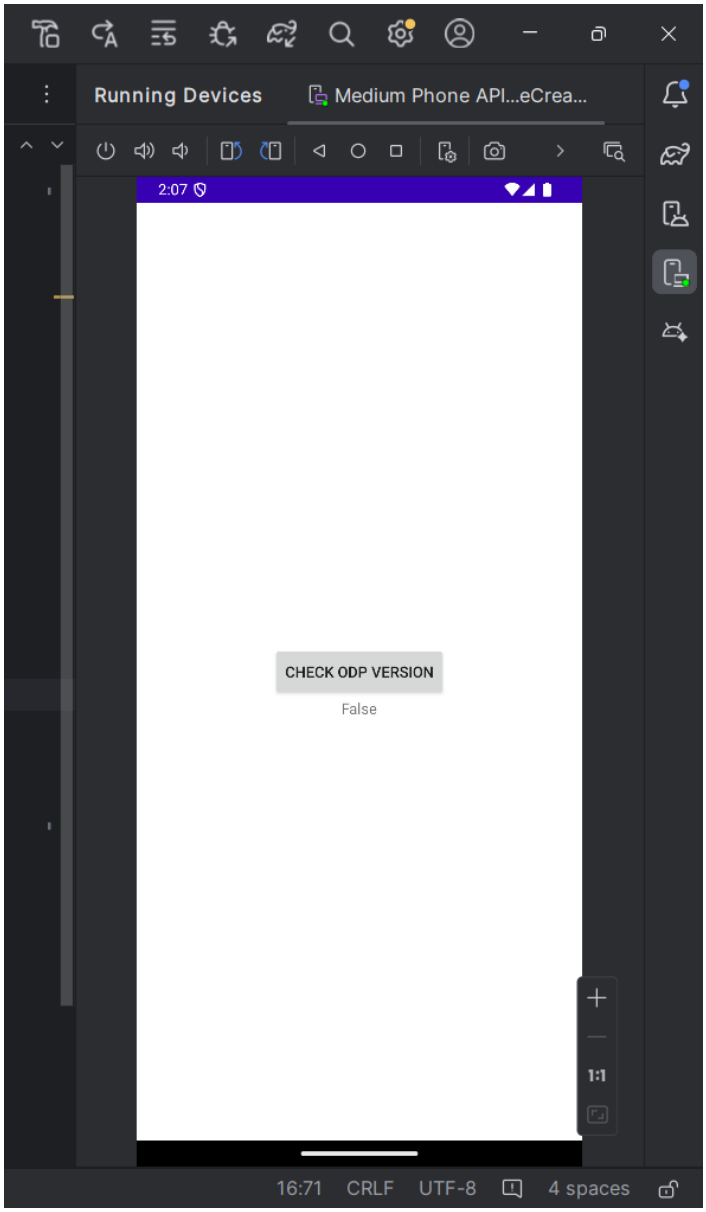
To verify the successful installation and operation of the ODP service, we developed a simple Android application to retrieve and display the ODP manager version. The following code was used:

```
package com.example.prism;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import android.widget.Button;
import android.widget.TextView;
import
android.adservices.ondevicepersonalization.OnDevicePersonalizationManager;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button versionButton = findViewById(R.id.versionButton);
        TextView versionTextView = findViewById(R.id.versionTextView);

        versionButton.setOnClickListener(v -> {
            // Assuming you have a method to get ODP version
            String odpVersion = getODPVersion();
            versionTextView.setText("ODP Version: " + odpVersion);
        });
    }
    // Dummy method to represent ODP version retrieval
    private String getODPVersion() {
        // Implement the actual logic to retrieve ODP version
        return "1.0"; // Placeholder version
    }
}

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/versionButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Check ODP Version"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/versionTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Version will appear here"
        app:layout_constraintTop_toBottomOf="@id/versionButton"
```

```
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```







Outcome:

The emulator was configured to run the ODP service successfully. The application verified the operation by retrieving and displaying the ODP manager version.

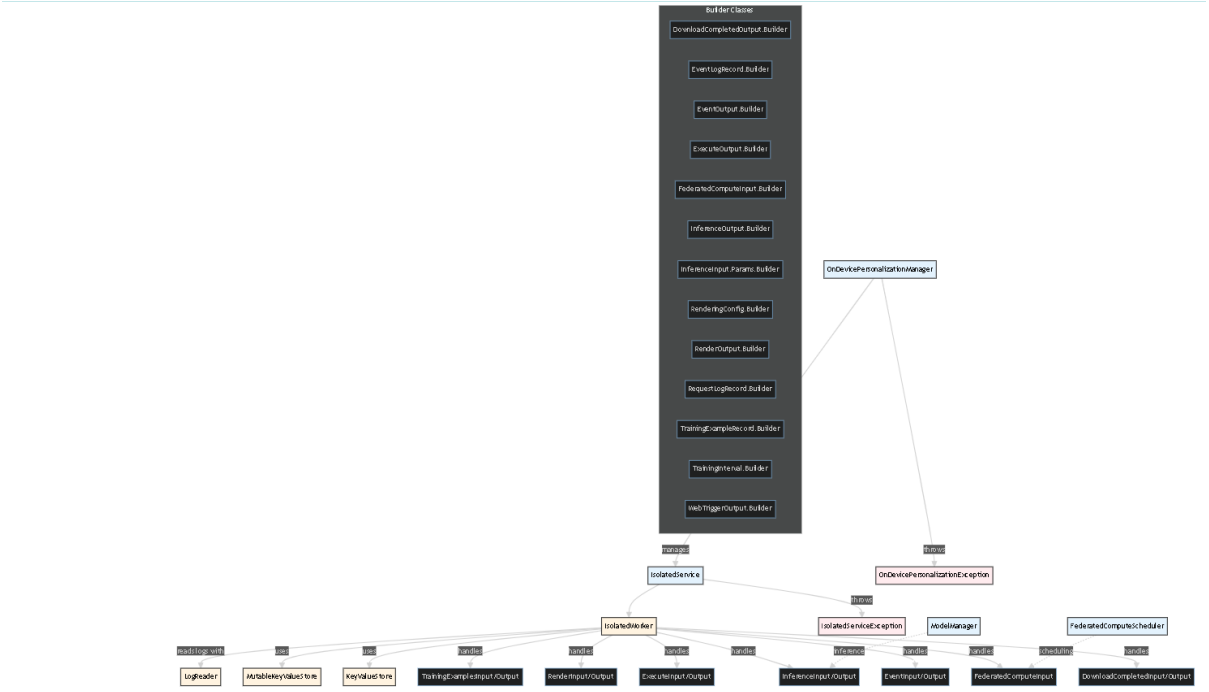
Task 3: Understand Architecture and Prepare Design Flow

Objective:

Check module components, understand the architecture, and prepare a design flow.

Steps Taken:

1. Component Analysis: Analyzed the various components of the ODP modules.
2. Architecture Study: Studied the architecture to understand how the components interact.
3. Design Flow Preparation: Prepared a detailed design flow diagram to represent the system architecture.



Flowchart Explanation :

1. ***OnDevicePersonalizationManager (ODP):*** This is the entry point for apps to interact with isolated services. It manages instances of IsolatedService.IsolatedService: Represents services that run in an isolated process managed by the ODP. This service acts as a container for the IsolatedWorker, which directly handles request execution.

2. ***IsolatedWorker:*** A crucial component that implements various interfaces to perform specific tasks such as downloading, event handling, execution of commands, and rendering.
3. ***Handles downloads:*** Processes download completion data and possibly alters service behavior based on download status.
4. ***Handles events:*** Processes event data, which could involve logging or operational changes based on event specifics.
5. ***Executes commands:*** Takes execution input and performs the requested action, returning the outcome.
6. ***Handles rendering:*** Processes rendering requests to prepare visual output for the app.

Data Stores:

1. ***KeyValueStore and MutableKeyValueStore:*** Interfaces used by the IsolatedWorker for accessing persistent key-value data, essential for state management and configuration settings.
2. ***LogReader:*** Utilized by the IsolatedWorker to read logs, which are critical for diagnostics and understanding system behavior.
3. ***Specialized Managers:ModelManager:*** Manages the lifecycle and execution of machine learning models.
4. ***FederatedComputeScheduler:*** Schedules and manages federated computing tasks, which are essential for distributed machine learning operations.
5. ***Exception Handling***: Both IsolatedService and ODP are designed to handle exceptions, which are critical for robust error management in a production environment.
6. This ensures that errors are logged and aggregated properly.

Module Interactions

- ***Between Modules:*** The Policy Engine dictates the operations of other modules by setting rules for data collection, sanitization, and storage. User data flows from the collection module to sanitization, then to storage. The federated computation accesses stored data to perform analytics and update personalization models securely within the TEE.
- ***Data Handling and Privacy in ODP***

- ***Data Dumping:***
  - What Data Could Be Dumped: ODP might temporarily store data such as user preferences, interaction logs with applications, and device settings. This data is primarily used to understand the user's behavior and preferences to tailor the device functionality.
  - How Data Is Dumped: Data collected by ODP is typically processed locally and stored in a secure, encrypted format in the device's local database. Access to this data is tightly controlled by the operating system and the ODP framework.

Privacy Preservation:

- ***Intent of the Service:*** The primary intent of ODP is to personalize the user experience without compromising privacy. It is not typically used for advertising or external commercial purposes but may enhance app suggestions, user interface customization, and device settings optimization based on the user's behavior.
- ***Privacy Mechanisms:*** ODP ensures privacy through local data processing, use of sanitization techniques to remove PII, and employing TEE for secure operations. These mechanisms ensure that the data does not leave the device in a form that could be used to identify the user, aligning with privacy regulations and best practices.

This architecture allows ODP to offer personalized experiences directly on Android devices while upholding the user's privacy, making it a suitable solution for modern privacy-conscious digital environments.

ODP Architecture in Android and Module Interactions

Overview On-Device Personalization (ODP) in Android is designed to enhance user experience by personalizing content and functionality directly on the device. This approach is privacy-centric as it minimizes data transmission to external servers.Key Modules:

1. ***Policy Engine:*** Manages decision-making rules for personalization based on user interactions and device usage patterns. It decides what data is relevant for collection and how it should be processed.
2. ***User Data Collection:*** Gathers data from various sources on the device, such as app usage statistics, device settings, and user preferences without sending this data off the device.

3. ***Data Sanitization:*** Ensures that the collected data is cleaned and anonymized before processing. This module removes personally identifiable information (PII) to maintain user privacy.
4. ***Database for Data Storage:*** Stores sanitized, non-PII data locally on the device. This database is typically lightweight and optimized for fast retrieval to support real-time personalization.
5. ***Federated Computation and Analytics***: Executes data analysis and machine learning models on the device, using the collected data to personalize the user experience. Federated learning might also be used to update and improve models without needing to send personal data back to a server.
6. ***Trusted Execution Environment (TEE) Usage***: Utilizes TEE for secure data management and sharing, ensuring that sensitive operations like data access and processing are isolated from the main operating system to prevent tampering and unauthorized access.

Outcome:

A comprehensive design flow was prepared, illustrating the architecture and component interactions.

Task 4: Install Test Package and Understand API Implementation

Objective:

Install the test package, understand the API implementation, and list all system API sequence flows.

Challenges:

1. Lack of Documentation: There was no comprehensive documentation available for the APIs.
2. Device Configuration Issues: Encountered challenges related to device configuration that hindered progress.

Outcome:

Due to the lack of documentation and device configuration challenges, this task could not be completed. However, initial steps were taken to install the test package and identify key APIs.

Task 5: Prepare Test App

Objective:

Prepare a test app after understanding the test implementation.

Challenges:

1. Incomplete API Understanding: Due to incomplete API documentation, the full implementation of the test app was not feasible.
2. Configuration Issues: Ongoing device configuration challenges further delayed progress.

Outcome:

This task remains incomplete due to the aforementioned challenges. Further work is required to understand the API implementation fully and address the configuration issues.

Conclusion

Our team made significant progress in the initial stages of the project, successfully building the ODP package, configuring the emulator to run the service, and preparing a detailed design flow. However, the lack of comprehensive documentation and device configuration challenges impeded our ability to complete the remaining tasks. Future work will involve resolving these issues to fully understand the API implementation and complete the development of the test app.

NOTE FROM TEAM

We would like to extend our heartfelt gratitude to our Samsung R&D mentors, Himanshu Sir and Sahil Sir, as well as our project guides, Dr. Pattabiraman and Dr. Rajiv Vincent, for their invaluable guidance and support throughout the Samsung PRISM project on implementing the On-Device Personalization (ODP) package. Himanshu Sir and Sahil Sir, your mentorship was instrumental in bridging the gap between academic theory and industry practice, and your technical expertise and patience in addressing our queries were crucial in helping us overcome various challenges. Dr. Pattabiraman and Dr. Rajiv Vincent,

your academic insights and unwavering encouragement provided us with the foundational knowledge and confidence necessary to tackle the complexities of this project. Thank you all for dedicating your time and sharing your knowledge with us, contributing to the success of our project and enriching our learning experience.

**********THANK__YOU**********