TYPE Original Research
PUBLISHED 21 November 2022
DOI 10.3389/frvir.2022.1021932

Cloud-based cross-platform collaborative augmented reality in flutter

Lars Carius, Christian Eichhorn*, Linda Rudolph, David A. Plecher and Gudrun Klinker

FAR Augmented Reality Research Group, TU Munich, Munich, Germany

Check for updates

OPEN ACCESS

EDITED BY
Kiyoshi Kiyokawa,
Nara Institute of Science and Technology (NAIST), Japan

REVIEWED BY
Francesco De Pace,
Polytechnic University of Turin, Italy
Ralf Doerner,
Hochschule RheinMain, Germany

*CORRESPONDENCE
Christian Eichhorn,
christian.eichhorn@tum.de

SPECIALTY SECTION
This article was submitted to Augmented Reality, a section of the journal Frontiers in Virtual Reality

RECEIVED 17 August 2022
ACCEPTED 07 November 2022
PUBLISHED 21 November 2022

CITATION
Carius L, Eichhorn C, Rudolph L, Plecher DA and Klinker G (2022), Cloud-based cross-platform collaborative augmented reality in flutter.
Front. Virtual Real. 3:1021932.
doi: 10.3389/frvir.2022.1021932

Augmented Reality (AR) as a technology in the business area is utilized in new frontiers such as collaborative real-time experiences and cloud-based solutions. However, there is still a strong tendency towards game engines, which hinders widespread adoption for businesses. We present a collaborative AR framework (Flutter plugin) aimed at lowering the entry barriers and operating expenses of AR applications. A cross-platform and cloud-based solution combined with a web-based content management system (cloud) is a powerful tool for non-technical staff to take over operational tasks such as providing 3D models or moderating community annotations. To achieve cross-platform support, the AR Flutter plugin builds upon ARCore (Android) and ARKit (iOS) and unifies the two frameworks using an abstraction layer written in Dart. In this extensive description we present an in-depth summary of the concepts to realize the framework and prove its performance being on the same level as the native AR frameworks. This includes application-level metrics like CPU and RAM consumption and tracking-level qualities such as keyframes per second used by the underlying SLAM algorithm, detected feature points, and area of tracked planes. Our contribution closes a gap in today's technological landscape by providing an AR framework with the familiar development process of cross-platform apps. Building upon on a content management system (cloud) and AR can be a game changer to achieve business objectives, while being not restrained to stand-alone single-purpose apps. This will trigger a potential paradigm shift for previously complex-to-realize applications relying on AR, e.g., in production and planning. The AR Flutter plugin is fully open-source, the code can be found at: https://github.com/CariusLars/ar_flutter_plugin.

KEYWORDS
augmented reality, collaboration, cross-platform, cloud, flutter

1 Introduction

Recent advances in Augmented Reality (AR) technology such as Google's cloud anchors allow for a fundamentally new means of communication: Anchoring digital information in the real world, retrievable by anyone visiting the respective location. On-demand 3D scans using the smartphone camera overcome the need for hand-built

3D environment models and make this technology widely applicable. These are just some examples of the tremendous progress of AR over the past few years that cumulate in the increasingly widespread adoption of the technology by both private consumers and business users. While increasingly powerful smartphones are an ideal medium for applications aiming at large user groups, heavy reliance on game engine-based development toolchains like Unity has left behind developers of business applications and has created a technological landscape dominated by the entertainment industry. We discovered that the technology can create considerable value once it is used as a tool in operative business and not primarily as a means of entertainment. Marking drop-off locations for deliveries to chaotic construction sites using AR is one of many examples that proved to be a profitable business case. To utilize this potential, AR functionality needs to be embedded—as just one of many features—into applications that serve a larger purpose. With the frameworks currently available, this is not sufficiently possible. Furthermore, this could help to convert demanding research into applications for the end user, e.g., environment specific AR realized through machine learning (Marchesi et al., 2021). There are a plethora of approaches trying to solve this challenge, but they either lack the inclusion of cutting-edge features such as anchoring objects to real-world places or heavily restrict generality and thus applicability by focusing development on game scenarios only. A go-to AR framework for cross-platform app development is still missing.

Hence, we aimed to fill this gap in today's technological landscape by developing an AR framework with the following key characteristics:

- Familiar Environment: Our framework takes the form of a plugin for an established cross-platform app development toolchain and is not based on a game engine SDK to facilitate the use of AR as a tool in non-game apps and allow developers to utilize the technology of AR in their existing apps.
- Cross-Platform: Our framework is agnostic to the operating system the app will be compiled to. Android- and iOS-specific implementations are handled behind the scenes, hence a single codebase can be used for the integration of AR features.
- Open-Source: This guarantees low entry barriers for businesses and developers looking to utilize the potential of AR.
- Collaboration: Our framework supports the latest features such as objects anchored in real-world locations that are viewable from multiple devices. The integration of location-based anchoring for collaborative AR experiences, including a flexible networking backbone, is key for the commercial usage of AR.

This AR framework tries to answers to previously defined framework concepts for Mixed Reality such as Scalability, Interoperability, Extensibility, Convenience and Quality Assurance with the greater picture of Integrability in mind (Weber et al., 2022).

As a result, the plugin will contribute a valuable tool to the AR developer community and enable more businesses and app developers to utilize the potential of location-based AR without having to renounce technical possibilities and performance advantages that originate from using an established cross-platform app development toolchain. With our work, we aim to facilitate the development of complex AR-based workflows such as facility design and management in production and planning scenarios while, at the same time, alleviating the need for excessive computing power and pre-defining the territory of deployment. For example, in the case of Horst et al. (2021) it was necessary to develop two applications for the online continuing medical education. A Unity app provided the AR capability, while the learning management system was web-based and relied on a Flutter implementation. Having to look after two completely different systems makes the development and future maintenance difficult.

2 Related work

To lay a solid foundation for choosing a suitable technology stack, we reviewed AR applications from various fields and analyzed currently available AR frameworks as well as persistent anchor technologies. We focused especially on collaborative AR in which multiple users are located in a shared physical space and interact with the same virtual objects, often from different points of view (Kaufmann, 2003).

In recent years, handheld AR has been used for entertainment and also in a serious context (Plecher et al., 2022). In research we could identify trends in the area of AR multi-user applications such as (Serious) Games like Oppidum (Plecher et al., 2019) or AR-Escape (Plecher et al., 2020). Another trend evolves around the idea of utilizing AR for sports to augment the human body and allowing the user to receive superhuman abilities. The Superhuman Sports genre provided game concepts such as Catching the Drone (Eichhorn et al., 2020) or League of Lasers (Miedema et al., 2019). Early research in the area of collaborative AR has provided a rich foundation for today's diversification of concepts (Schmalstieg et al., 2002). With the advances of machine learning AR will in the future be merged with this technology to form context sensitive AR which has a better understanding of the surroundings like surface types or objects (Marchesi et al., 2021).

The 2020 XR Industry Insight report by VR Intelligence, however, indicates that a majority of AR companies see the largest potential in industrial use cases (Bonasio, 2019). The scenarios in which businesses can use AR are manifold, Figure 1

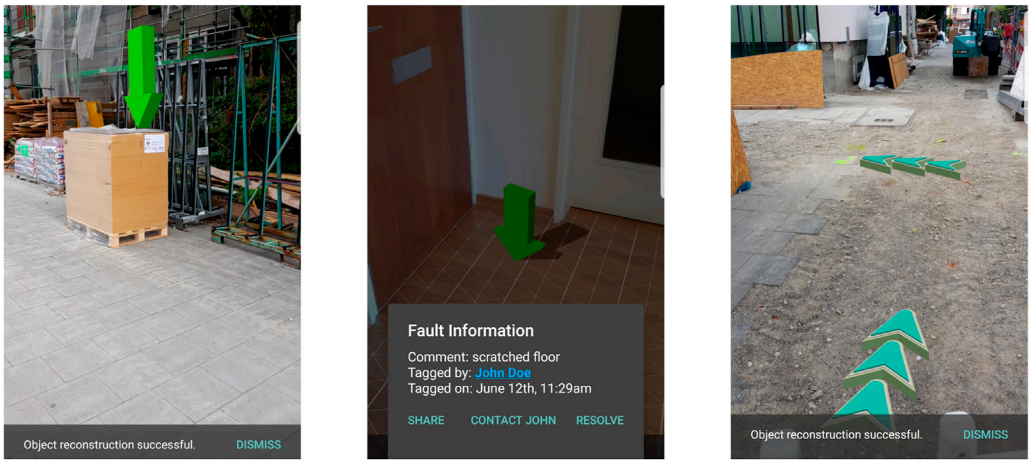


FIGURE 1
Examples for AR-based industrial process optimization: Dropsite marking (left), fault management (middle), delivery instructions (right).

illustrates several examples of the technology's potential to increase productivity and facilitate communication in asynchronous work processes.

2.1 The field of collaborative augmented reality

In recent academic research, technological advances have pushed the boundaries of collaborative AR. In their paper CARS, Zhang et al. (2018) present a method for leveraging the physical vicinity of users in a collaborative AR session for reducing latency through peer-to-peer sharing of annotation data. Egodagamage and Tuceryan (2018) introduce a framework for distributed monocular SLAM that identifies matching features between the camera feeds of different devices and constructs a shared map of the environment. They showcase a collaborative AR application based on their method and introduce a benchmarking dataset. A different approach to creating collaborative AR sessions on-the-fly is manifested in SynchronizAR (Huo et al., 2018), a system based on ultra-wide bandwidth distance measurement units that allows to orchestrate multiple independent SLAM devices without having to share feature maps or employ external trackers by calculating transformations between the coordinate system of the involved devices. In an approach to decrease latency and mitigate the effect of battery drain in AR applications, Ren et al. (2019) employ edge computing to offload processing steps from the cloud backend to an edge layer, for example, for faster feature matching using intelligent caching. A well-known example for a state-of-the-art network architecture for edge-based collaborative augmented and virtual reality applications is ArenaXR (Pereira et al., 2021). It operates on a combination of REST and PubSub network patterns to allow interactive virtual spaces. It has a heavy focus on scalability. However, due to its system architecture it cannot be integrated with small effort in existing applications but demands re-implementations. Additionally, due to its dependency on WebXR, currently it is not supported in iOS Safari Browsers and is therefore not applicable for iOS devices. Aside from novel frameworks, recent publications also showcase the opportunities cloud-based collaborative AR creates. Using real-time spatial reconstruction methods, Piumsomboon et al. (2017) demonstrate collaboration between an AR user and a remote collaborator joining in VR. Zillner et al. (2018) present another comprehensive AR remote collaboration system between AR users and remote maintenance experts based on dense scene reconstructions streamed from the AR user. Their system allows remote staff to freely explore the scene and place real-world-anchored annotations and animations into the AR user's environment to create intuitive instructions. Further examples for the digital collaboration opportunities AR creates are Mourtzis et al. (2020) AR-based product design framework that enables interdisciplinary engineering teams to co-develop and visualize product concepts in a natural way and HoloCity (Lock et al., 2019), an application to collaboratively analyze the cityscapes dataset using AR visualizations for overlaying data from different sources and visually detecting trends. In the production and planning sector, localization-based AR collaboration has largely been approached within pre-defined boundaries: Baek et al. (2019), for example, present an AR system for indoor facility management based on comparing images from the user's camera view with an existing BIM model of the building using a server-side GPU-accelerated deep.

TABLE 1 Comparison of commonly used AR frameworks.

	Supported cross-platform development frameworks	Supports Android and iOS	f: Free o: Open source	Markerless cloud AR
ARCore	Unity, Unreal Engine, React Native[a], Flutter[a]	iOS only partly	f, o	Yes
ARKit	Unity, Unreal Engine, React Native[a], Flutter[a]	iOS only	f	Yes
AR Foundation	Unity	Yes		Yes
Vuforia AR SDK	Unity	Yes		Yes
ViroReact	React Native	Yes	f, o	
EasyAR Sense	Unity	Yes		Yes
Wikitude AR SDK	Unity, Flutter, Cordova, Xamarin, NativeScript, Ionic, React Native	Yes		Yes
MAXST AR SDK	Unity	Yes		
Onirix AR Studio	Unity	Yes		
ARToolkit+	Unity	Yes	f, o	
ARToolkitX		Yes	f, o	
Xzimg	Unity	Yes		
Kudan AR SDK	Unity	Yes		
NativeScript AR Plugin	NativeScript[a]	iOS only	f, o	
Amazon Sumerian	JS-based Frameworks	Yes		
DroidAR		Android only	f, o	
VisionLib	Unity	Yes		
Placenote	Unity	Android only	f	Yes

[a]Community plugins of the AR framework exist for the cross-platform development kit.

2.2 Mobile augmented reality frameworks and the path towards collaboration

Common mobile AR frameworks gradually incorporate novel approaches from academia and make them available to a wide variety of devices. We provide an overview in Table 1, focusing on development for mobile devices as most modern smartphones support AR functionality and mobile apps are a highly flexible solution suitable for most use cases. Recently VR to experience entertainment (immersive VR, fills the complete visual area) is combined with the sensation of physical presence (AR) to enrich virtual immersive environments (Augmented Virtuality). Therefore even if AR and VR are still used separately, the option to change realities provides a useful foundation for multi-user shared environments (SEs). In terms of realization of truly mixed reality applications, where e.g., a person is interacting in VR with a person who using an AR view, game engines such as Unity are still the norm (Oriti et al., 2021; Kumar et al., 2021). In a similar context Keshavarzi et al. (2020) build a solution to find a common accessible virtual ground for remote multi-user interaction scenarios (AR-VR). Early examples of the idea to build AR-VR frameworks date back to the MORGAN project (Ohlenburg et al., 2004), where requirements such as scalability and platform independence for multi-user applications were still major challenges. Considering the compatibility with widely used cross-platform app development frameworks and the extensive functionality, ARCore, ARKit, ViroReact, and Wikitude AR SDK were potential candidates for our framework. While ARCore and ARKit share most of their features, they take different approaches to collaboration. ARCore's cloud anchors allow the synchronization of feature descriptors over the cloud while ARKit's collaborative sessions build on offline processing of anchor data shared between devices. Automatic multi-device AR experiences are possible in ARKit as well, but require the involved devices to be in physical vicinity during the entire duration of the AR experience, persistent augmentations in the form of location anchors rely on pre-recorded location imagery which is currently only available in a selection of US cities[1]. Through later adoption ARCore's cloud anchors were brought on iOS devices so users can share AR experiences[2]. Therefore the cloud anchor API allows to develop for both platforms. ViroReact is based on the commonly used React framework, includes viromedia's own renderer, and supports ARKit and ARCore. Due to its lack of collaborative features and the end of viromedia in 2019, ViroReact was deemed unsuitable in the context of our work. Finally, the Wikitude AR SDK is an all-in-one development platform for AR applications and supports the broadest choice of app development frameworks

1 Apple Inc. ARKit Documentation, 2021.

2 https://developers.google.com/ar/develop/ios/cloud-anchors/quickstart.

TABLE 2 Comparison of commonly used cross-platform app development frameworks.

	React native	Flutter	Ionic	Xamarin	Unity3D	NativeScript
Owner	Facebook	Google	Community	Microsoft	Unity Technologies	Community
Open-Source	Yes	Yes	Yes	Yes	No	Yes
Language	JS	Dart	HTML5, JS	C#	C#	TS, JS, HTML, CSS
GitHub Statistics	92.600 stars	110.000 stars	42.800 stars	5.100 stars	—	19.600 stars
	20.400 forks	15.600 forks	13.300 forks	1.900 forks		1.400 forks
Free (Commercial Use)	Yes	Yes	Yes	No	No	Yes
Developer Share 2020 (Statista Inc., 2021)	42%	39%	18%	14%	11%	5%
Trend Compared to 2019 (Statista Inc., 2021)	±0%	+9%	−10%	−12%	−1%	−6%
Supported AR Toolboxes	ViroReact, Wikitude	Wikitude, ARCore[a], ARKit[a]	Wikitude, AR.js	Wikitude	AR-Foundation, Vuforia, EasyAR, Xzimg, Wikitude, MAXST, Onirix, ARToolkit+, Kudan AR SDK, Visionlib, Placenote	NativeScript AR Plugin[a]

[a]Community plugins of the AR framework exist for the cross-platform development kit.

and operating systems. It allows for cloud-based marker synchronization and augmentation based on geographical location information such as GPS, but is ruled out by many small businesses due to its hefty price tag.

2.3 Summary

When conceptualizing our AR framework, we were faced with a choice of cross-platform app development toolkits presented in Table 2. While Unity is most commonly used for AR and VR applications, it contradicts our requirement of using environments familiar to non-game app developers. Besides Unity, Flutter and React Native are the most popular platforms for app development. Both pursue the goal of targeting multiple operating systems using a single codebase and are well-suited for professional cross-platform applications. While Flutter utilizes an embedder layer to run the Flutter engine on the device and ships with its own UI elements, React Native relies on a bridge between the application's Javascript code and the operating system's native UI functionalities. In direct comparison, the different approaches cause Flutter to have the upper hand in terms of performance (Coninck, 2019; InVerita, 2020), while React Native is the more mature framework and thus easier to find developers for (Skuza et al., 2019).

In general, the current AR landscape is fragmented into a plethora of frameworks with limited applicability. Some lack the inclusion of collaborative features, others are incompatible with app development frameworks familiar to business developers. Flutter is an uprising framework that continues to gain popularity, but except for some single-platform plugins without collaborative features (Leuschenko, 2021; Francesco, 2022), there are no competitive solutions for the creation of AR applications. With the creation of our AR Flutter plugin for collaborative AR experiences, we intend to solve the aforementioned problems. Choosing Flutter as the underlying platform brings cross-platform support while preventing companies from having to hire game developers. We build upon on ARCore and ARKit and unify both frameworks using an overarching layer in Dart (cross-platform programming language) to ensure continuous access to cutting-edge functionality. For maximum flexibility, we aim to build the plugin in a cloud-agnostic fashion, with the small exception of cloud anchors for the distribution of visual feature representations, for which we use Google's Cloud Anchor API.

3 AR flutter plugin

In Figure 2 a high-level overview of the AR Flutter plugin's software architecture is shown, which is structured into two main parts: A unified, cross-platform API providing an interface to applications that use the plugin, and platform-specific implementations for Android and iOS that perform logic that cannot be abstracted to the Flutter level. The exposed section of the framework contains widgets that can be included in a client app's widget tree to enhance the UI and AR managers that handle all functionality and logic related to AR and serve as the control instruments of the plugin. Communication with the platform-specific implementations happens only *via* the plugin's API. In this structure, Flutter's platform channel and platform view system can be viewed as an implementation of the adapter pattern. The ARView, for example, makes use of the platform view adapters to expose functionalities of the underlying ARActivity on Android or the ARUIViewController on iOS

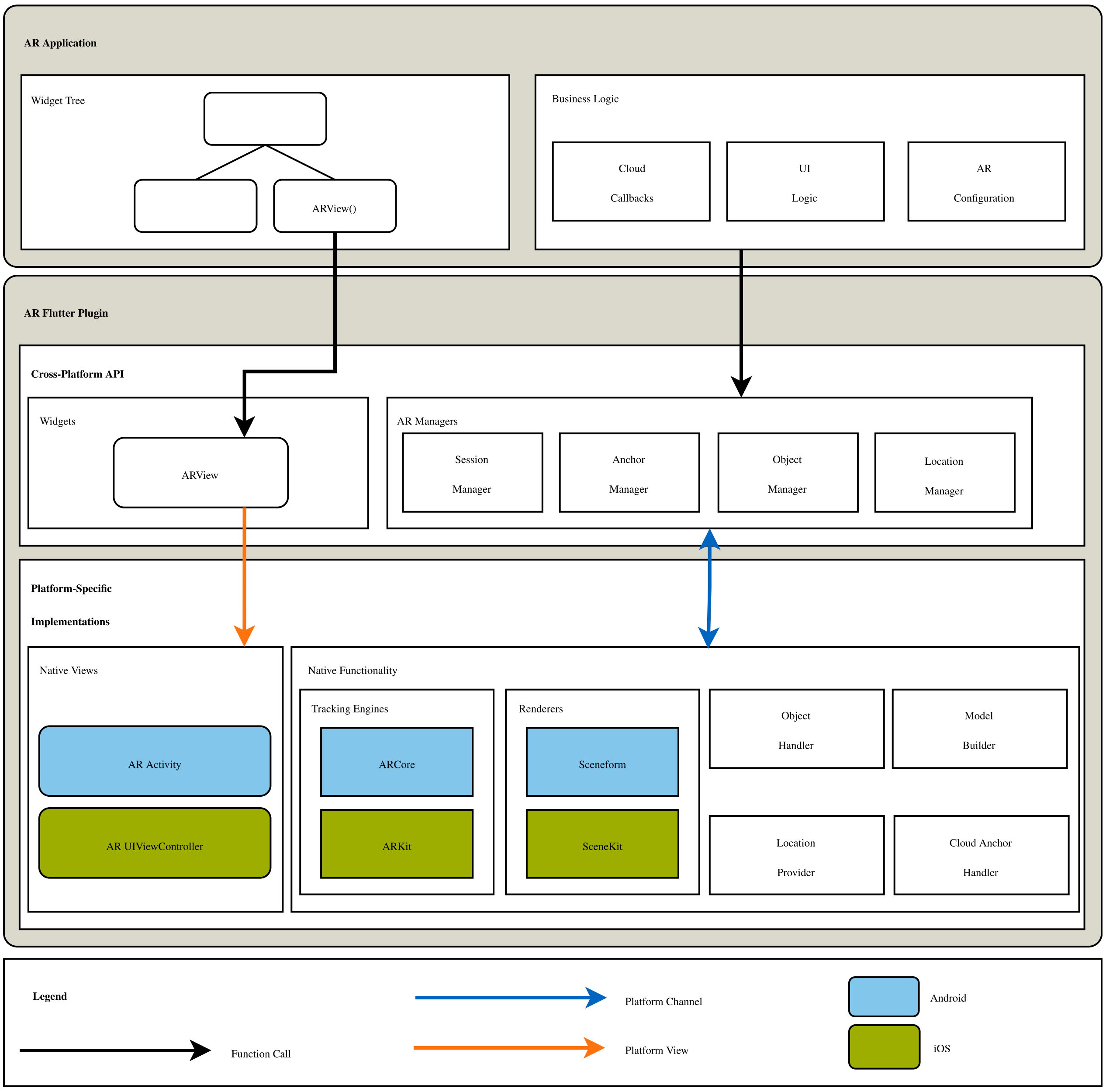


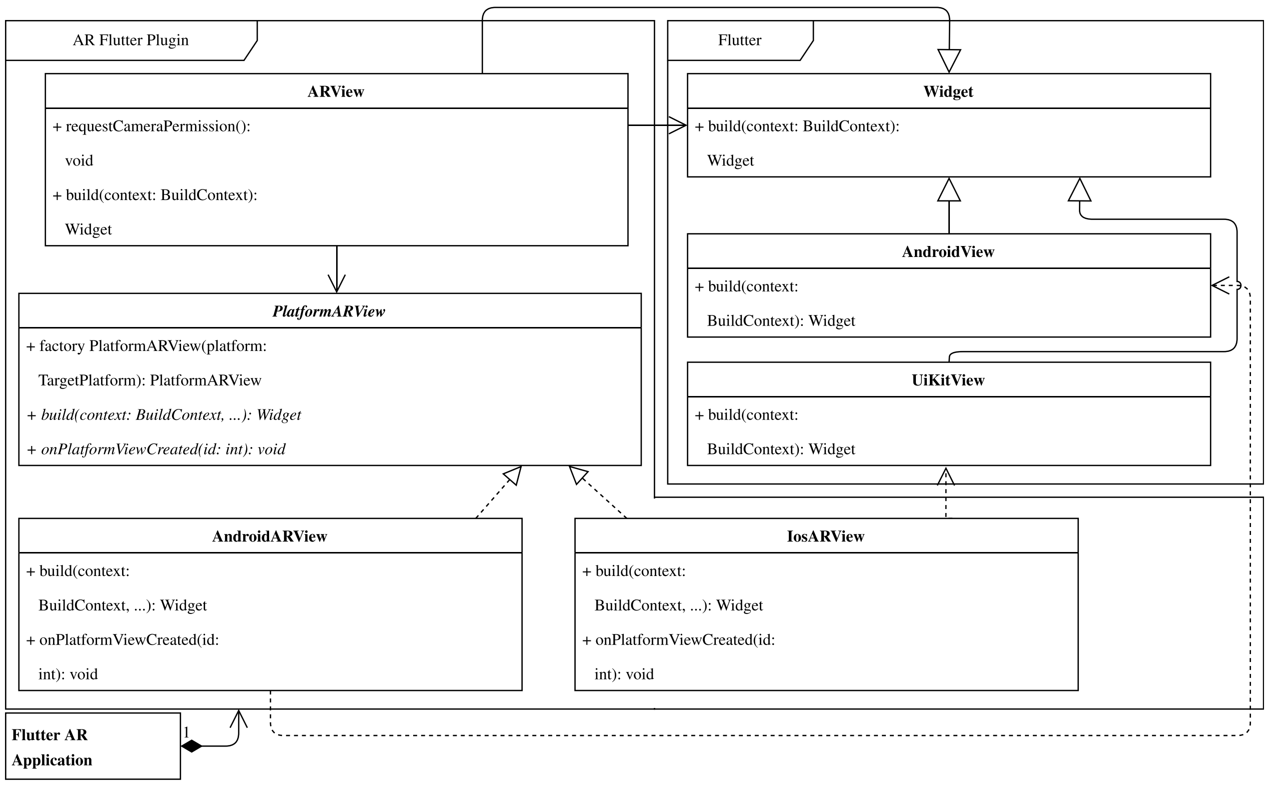
FIGURE 2
High-level software architecture of the AR Flutter plugin (Carius et al., 2022).

to the outside world. Different low-level implementations are unified into an easy-to-use and platform-independent API. The same holds for the AR managers: They abstract the interactions with highly platform-specific features like tracking or rendering into a unified interface by using platform channel functionality as an adapter between the Flutter API and the Swift or Kotlin code sectors of the plugin. The following sections further elaborate on the individual building blocks of the AR Flutter plugin.

3.1 Augmented reality view

The ARView class is the core UI element of the AR Flutter plugin. On instantiation, the class returns a widget that can be included in the client app's widget tree to use the AR functionality provided by the plugin.

The inner structure of the ARView class is visualized in Figure 3. ARView inherits from Flutter's Widget class. Its main objectives are providing a wrapper around PlatformARView to



AR Flutter Plugin
ARView
+ requestCameraPermission(): void
+ build(context: BuildContext): Widget
PlatformARView
+ factory PlatformARView(platform: TargetPlatform): PlatformARView
+ build(context: BuildContext, ...): Widget
+ onPlatformViewCreated(id: int): void
AndroidARView
+ build(context: BuildContext, ...): Widget
+ onPlatformViewCreated(id: int): void
IosARView
+ build(context: BuildContext, ...): Widget
+ onPlatformViewCreated(id: int): void
Flutter
Widget
+ build(context: BuildContext): Widget
AndroidView
+ build(context: BuildContext): Widget
UiKitView
+ build(context: BuildContext): Widget
Flutter AR Application
1

FIGURE 3
UML diagram of the ARView structure (incomplete).

ensure permissions such as camera access and returning a widget by instantiating a PlatformARView and invoking its build method. The class PlatformARView implements the factory pattern (Gamma et al., 1994). Using the factory constructor provided by Dart, the abstract class can be used as a common interface to external code while separate subclasses for each supported OS provide concrete implementations in the form of subclasses. AndroidARView and IosARView are the classes which implement the PlatformARView interface for their respective operating systems Android and iOS. The classes provide wrappers around Flutter's widgets AndroidView and UiKitView which embed native view elements into the Flutter widget tree. For AR applications, this is necessary because there are no suitable rendering and tracking systems available in the Flutter framework itself. In the context of the MVC pattern, the concept can be pictured as outsourcing the view component to the native platforms while the components controller and model remain in Flutter's responsibility. The implementations of PlatformARView also override the function onPlatformViewCreated. The function is used to instantiate all required managers shown in Figure 2 and return references to them to the Flutter application initially calling ARView.

3.2 Session management

All settings and functionalities regarding the AR session running on the underlying OS are orchestrated by the plugin's ARSessionManager. Some, like tracking configuration, platform channel identifiers, and function callbacks are supposed to persist throughout the entire session, so they are passed to the initializer. Other options might be subject to changes during a running session, for example, as a result of user interaction. These include debugging options like plane visualization and toggles to start or stop handling gesture events and can be updated at any time during the session. Tracking visual features of the user's environment is essential for the creation of realistic AR experiences (Bostanci et al., 2013). We focus on tracking planes as collaborative AR requires environment features to persist both long periods of time and highly varying points of view to ensure stable reconstruction of virtual scenes attached to real-world features. The AR Flutter plugin introduces a simple PlaneDetectionConfig data type which can be set to none, horizontal, vertical, or horizontalAndVertical when instantiating the ARSessionManager which sets the search focus for planes in the frame. Aside from configuring the tracking setup, the session manager is also responsible for exposing a set of debugging options such as showWorldOrigin, showFeaturePoints and showPlanes to help developers test their AR applications, check the tracking quality, or aid end-users in understanding how certain AR functionalities work. To interact with a three-dimensional scene through a two-dimensional interface such as a smartphone screen, gestures like screen taps have to be translated into meaningful actions. When a user taps on a 2D screen location at which the plane is rendered in the current frame, the AR Flutter plugin has to project the screen position into 3D space and return the plane that intersects with the virtual ray cast from the user's tap location, a process referred to as hit testing and handled by the plugin's ARSessionManager. The AR Flutter plugin differentiates between node taps, which indicate the intention to interact with a virtual object in the scene, and taps on trackables such as detected planes or feature points from the environment. We adapted the AR hit result class from Oleksandr Leuschenko's ARKit Flutter Plugin (Leuschenko, 2021) for use in a multi-platform setting aiming to facilitate collaborative AR experiences. The ARSessionManager interprets the incoming hit testing data from the platform-specific implementations and instantiates a Flutter object of the custom class ARHitTestResult to avoid having to pre-define the behavior of the AR Flutter plugin when tap gestures are recognized. Instead, callbacks can be used to let the users of the plugin decide what actions are to be triggered.

3.3 Object management

The core feature of any AR application is the placement and observation of virtual objects in the scene. Object management in the AR Flutter plugin consists of two main modules: The ARNode class representing a Flutter abstraction of AR objects along with its corresponding platform-specific handlers and the ARObjectManager that exposes all functionalities regarding AR objects as a cross-platform Flutter API.

In the context of AR, nodes represent hierarchically structured scene objects that are defined by their transformation relative to a local or world coordinate system and their renderable, which in turn can be a 3D model or any other visual representation (MacWilliams et al., 2003). The plugin's ARNode is an adaption of the class ARKitNode from Oleksandr Leuschenko's ARKit Flutter Plugin (Leuschenko, 2021) and has five key properties: Name, type, URI, transformation, and data. On instantiation, an ARNode is named with a unique key used for user reference and association of rendering and gesture events on the platform side. To convey renderable loading information to the ArModelBuilder, each node has a type, which is an instance of the plugin's data type NodeType, and a URI containing a path to a local 3D model file or a URL pointing to an online resource. The ARNode's transformation, which is stored as a four-dimensional affine transformation matrix, defines its pose relative to its parent. Finally, the member variable data of type map provides a convenient location to store any object-related information required by an application such as visibility state or on-tap text. Following the single source of truth principle, the ARNodes in Flutter are the main node objects, the representations on the platform level are dependent on their Flutter counterpart and cannot be explicitly called or modified using the plugin's API. To prepare an ARNode object for being sent through a platform channel or to be uploaded for sharing, it is serialized into the JSON format. Any updates made to an ARNode's transformation trigger a call to the platforms to update the nodes rendered on the screen.

The ARObjectManager handles all functionality related to ARNodes. Its key functions are adding and removing nodes to the AR scenes of the underlying platforms by attaching or detaching them to plane anchors or the parent scene and keeping the platform's node representations in synchrony with their counterparts on the Flutter side. In addition, the object manager handles node taps by invoking callback set by users of the AR Flutter plugin.

While the ARObjectManager provides a consistent interface for node handling to the user, the actual functionality is implemented on each platform to account for the differences on Android and iOS. On both systems, the data contained in the platform channel messages is deserialized, yielding a four-dimensional affine transformation matrix on iOS and a triple of position vector, scale vector, and rotation quaternion on Android. The ARObjectManager's addNode function is handled asynchronously by the AR Flutter plugin by unpacking the data sent through the platform channel, requesting a renderable from the ArModelBuilder according to the node's type and attaching it to the root scene of the AR session or an anchor. Loading of GLTF2 and GLB models from the app bundle, the device's file system, or the internet is supported for maximum flexibility. Assets required for all users can be included by developers, models needed for flexible AR experiences can be managed separately in a cloud system, sparing businesses from having to update the codebase. We describe a content management system utilizing the flexible model loading capabilities to provide a desktop GUI in Section 4. Both Sceneform (ARCore) and SceneKit (ARKit) allow recursively searching the scene tree for objects matching a specified name, allowing the plugin to handle transformationChanged and removeNode requests.

To summarize, the heavily different requirements and APIs of Sceneform (ARCore) and SceneKit (ARKit) result in the AR Flutter plugin having to perform rather complex operations behind the scenes. By moving all these platform-specific implementations behind common data structures and consistent, platform-independent Flutter methods, however, our framework provides an easy-to-use interface for developers using AR without requiring knowledge about which platform their app is eventually used on.

3.4 Anchor management and cloud synchronization

Anchors allow interweaving virtual content with real-world environments. Instead of describing poses in the digital scene, anchors encapsulate a position and orientation in the physical space, thus being constantly updated to keep their relative position to real-world objects after initial placement using world coordinate transformations. Due to their ability to persist over time and viewpoint changes, we focus on plane anchors for creating collaborative AR experiences in the AR Flutter plugin.

Similar to the handling of nodes, the AR Flutter plugin introduces an abstraction of AR anchors to create a unified Flutter wrapper around the implementation of anchors in ARCore and ARKit. Derived from the factory pattern architecture of Oleksandr Leuschenko's ARKit Flutter Plugin (Leuschenko, 2021), we use an abstract ARAnchor class and separate subclasses for each anchor type. When users request a specific type of anchor, e.g., a plane anchor at the location of a hit test result, subclasses can be directly instantiated. For automatic anchor creation by the plugin, the factory method ARAnchor.fromJson acts as a constructor to instantiate a suitable subtype from a serialized platform channel object.

To offer a central handler for AR anchors, the ARAnchorManager exposes a set of cross-platform functions such as addAnchor and removeAnchor to the API of the AR Flutter plugin. To add or remove anchors, serialized representation in the JSON format are sent through Flutter's platform channels. While on Android, the platform-level implementation of adding and removing anchors programmatically is straightforward, on iOS, we had to implement custom extensions of ARKit and SceneKit's rendering loop to externally manage anchors and therewith achieve the level of abstraction required by the cross-platform AR Flutter.

To enable collaborative AR experiences (Kaufmann, 2003), the AR Flutter plugin contains functionality to upload and download anchor objects from a cloud service using the Google Cloud Anchor API. Local 3D scans used to define anchors can be stored in a cloud system and the current scene can be compared to already uploaded anchors in order to download content previously placed in the scene. To enable the storage of anchors for up to 365 days, the AR Flutter plugin implements keyless authentication. On Google's Android OS, the process is well integrated and straightforward to implement, a novel library for JWT-based authentication was developed to add the functionality on the iOS side of the plugin.

To ensure a consistent cross-platform experience, functionality regarding the uploading and downloading of anchors to or from the Google Cloud Anchor API is abstracted into the ARAnchorManager on the Flutter side of the plugin. The ARAnchorManager offers two functions: uploadAnchor and downloadAnchor. All cloud interactions are handled asynchronously to avoid blocking the app during the process, so both functions accept callbacks that are triggered once the underlying platform notifies the ARAnchorManager of the completion of the process. As visualized in Figure 4, to upload an anchor, developers can pass an ARAnchor object to the upload function and specify the number of days the anchor should be stored in the cloud. The manager serializes the anchor object and sends it through a platform channel for further handling. On completion of the upload process, the ARAnchorManager's onCloudAnchorUploaded platform channel method is called. Downloading a cloud anchor is implemented in a similar, asynchronous fashion: Developers can call the downloadAnchor function with a cloud anchor ID, thus triggering the download process on the underlying platform. If the features of the requested cloud anchor match the current scene and the download succeeds, the ARAnchorManager's platform channel method onAnchorDownloadSuccess is executed. The function receives a serialized anchor from the underlying platform and uses it to instantiate the corresponding Flutter counterpart. Under the hood, both the Android and the iOS section of the AR Flutter plugin contain a CloudAnchorHandler class to deal with the platform-specific requirements of Google's cloud anchor libraries. While the specific implementation varies on the platforms, they share a common structure consisting of four key elements: Wrappers around the upload and download functions of Google's cloud anchor library, a bookkeeping logic to track the progress on cloud processes, an update function used to regularly check for completed cloud anchor tasks, and listeners to notify on task completion.

The processes described in the previous paragraphs illustrate a core functionality of the AR Flutter plugin regarding collaborative AR as defined by Kaufmann (2003): Uploading, storing, and downloading visual feature data to transfer it between the local physical scene and the Google Cloud Anchor Service. To implement individual anchor distribution systems such as location-based anchor querying using anchors' GPS coordinates provided by the AR Flutter plugin's ARLocationManager, users can reference the cloud anchor ID to store additional information. All in all, the anchor management approach of the AR Flutter plugin complements the abstraction of AR nodes introduced in Section 3.3. Lifting the functionality to the Flutter interface of the plugin increases the complexity of platform-level implementations, but provides developers with a unified cross-platform API and greatly reduces the know-how required to include AR in one's application without foregoing state-of-the-art features.

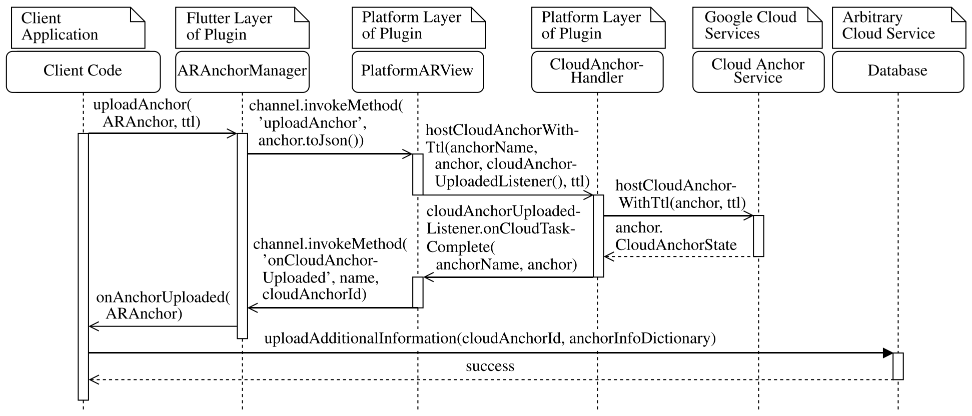

Client Application
Flutter Layer of Plugin
Platform Layer of Plugin
Platform Layer of Plugin
Google Cloud Services
Arbitrary Cloud Service
Client Code
ARAnchorManager
PlatformARView
CloudAnchor-Handler
Cloud Anchor Service
Database
uploadAnchor(ARAnchor, ttl)
channel.invokeMethod('uploadAnchor', anchor.toJson())
hostCloudAnchorWith-Ttl(anchorName, anchor, cloudAnchor-UploadedListener(), ttl)
hostCloudAnchor-WithTtl(anchor, ttl)
anchor.CloudAnchorState
cloudAnchorUploaded-Listener.onCloudTask-Complete(anchorName, anchor)
channel.invokeMethod('onCloudAnchor-Uploaded', name, cloudAnchorId)
onAnchorUploaded(ARAnchor)
uploadAdditionalInformation(cloudAnchorId, anchorInfoDictionary)
success

FIGURE 4
UML sequence diagram of the anchor upload process.

4 Cloud-based content management system

Medium-sized businesses often struggle to adopt innovative technology and sustain the required level of service in everyday business due to a lack of capital or trained personnel (Lee and Baek, 2011). We identified content creation and management as the main reason for recurring workload and present a content management solution that greatly reduces running costs by managing all content externally in a cloud backend that ties into the flexible interface of the AR Flutter plugin, thus covering the entire value chain of integrating AR into business processes.

While the flexible, cloud-ready API of the AR Flutter plugin allows for any kind of web-based content management to be attached, our exemplary architecture satisfies two main requirements: Providing users with a graphical desktop interface and allowing both content used in the app's UI, such as 3D model files, and data uploaded by users, such as AR annotations and corresponding text, to be managed remotely. Instead of being an add-on directly related to the mobile application, the content management system is a standalone instance that only interacts with the cloud services the AR Flutter plugin uses. Content-related data flows include 3D models stored on a user-defined server, visual feature data streamed to and from the Google Cloud Anchor Service, and descriptive annotation data such as GPS coordinates or user comments.

To provide an intuitive and adaptable UI for managing content stored in our Firestore Realtime Database, we utilized the open-source software Firetable[3]. The web application offers a spreadsheet-like UI to view, modify, add, or delete data stored in Firestore collections and can easily be deployed on one's own server.

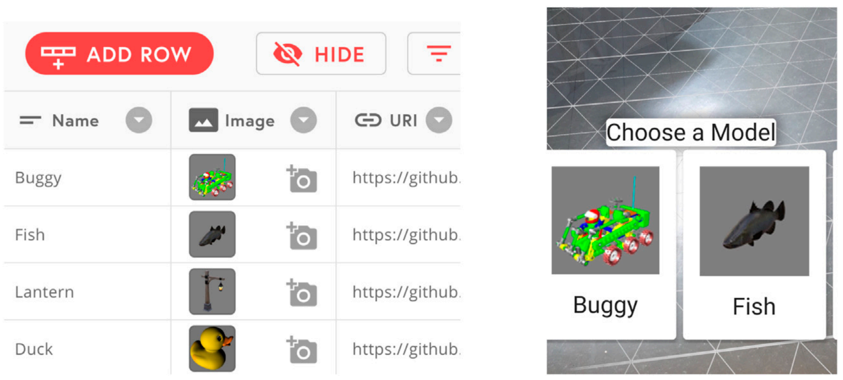
In Figure 5 one of the example applications included in the AR Flutter plugin is visualised that uses the content management system for external model management. New 3D models can easily be added to existing applications in the spreadsheet-like interface. This content management architecture enables the operations team to modify the content available to users of the application without having to task the technical development team with updates to the source code. Further examples of the opportunities external content management provides are making one's Firetable instance publicly available to allow users to add 3D models or moderating AR annotations and on-click texts in an AR-based social network, either manually or through the utilization of cloud functions for automatic updates.

The provided examples demonstrate the versatility of content management systems used in cooperation with the AR Flutter plugin, especially their contribution to fostering interactive and distributed multi-user AR experiences based on cloud anchor technology.

5 Evaluation

The usage of cross-platform app development frameworks has many benefits, however, it can require additional resources to achieve performance comparable to native applications due to additional software layers being introduced. To assess whether or not the performance of AR applications is significantly impacted by the user of Flutter and the AR Flutter plugin, apps using our framework are benchmarked against comparable native applications.

3 FiretableProject. Firetable, 2021.



Name	Image	URI
Buggy		https://github.
Fish		https://github.
Lantern		https://github.
Duck		https://github.

FIGURE 5
Managing models with our content management system[4]: Web interface using Firetable (left); In-app model menu (right).

Ideally, pre-recorded data should be used to achieve perfectly comparable results in benchmark testing. However, ARCore and ARKit take into account multiple real-time sensor measurements such as camera, accelerometer, gyroscope, and light sensor data. While in ARCore, a session can be recorded and replayed, the same is not possible in ARKit. Moreover, the two frameworks process sensor data differently, rendering the creation of a cross-platform solution for recording and playing back AR session data a research question that future work could cover. To still provide comparable data for the performance benchmark tests, we used a standardized real-world environment. All AR session tests are performed in the same room using fixed, purely artificial lighting, a coarsely pre-defined device trajectory, and a constant spatial setup. We used a MacBook Pro 2020 for compilation and profiling, a Samsung Galaxy S8 for testing on Android, and an iPad Pro 2018 for iOS tests.

5.1 Application performance benchmark

The first analysis of our plugin's performance assesses application-level metrics in typical AR scenarios. In our experiments, the user opens the application which initializes the AR view, scans the room by moving the device in all three spatial directions, and taps on a predefined surface to place a renderable virtual object loaded from a glTF file contained in the application bundle. Finally, the user moves around the AR object to view it from all sides. During the entire process, tracked planes are visualized using a custom texture. To generate a performance baseline measurement to which the Flutter applications are compared, the test is carried out with three separate applications which were developed to all provide the same user experience: A dedicated cross-platform Flutter application developed using the AR Flutter plugin, a native Android AR application based on the AR sample app by Google, and a native iOS AR application based on the ARKit demo app by AppCoda[5]. We adapted the native apps to use the same render engines and model formats as the AR Flutter plugin and visualize tracked planes using a custom texture. A notable observation made during the development of the benchmarking applications is the fact that the Flutter app based on the AR Flutter plugin was, by far, the most straightforward to set up and required next to no technical understanding of AR. Aside from import statements and autogenerated supplementary files and using Visual Studio Code's standard code formatting rules, the final application contains only 53 lines of project code.

We present application-level performance measurements in two categories: Build metrics and AR session performance.

Cross-platform applications add a layer of abstraction to the source code and thus tend to introduce additional build steps and cause application bundles to take up more disk space. To study the effect of using Flutter in conjunction with the AR Flutter plugin instead of native approaches for creating AR applications, build times and application sizes of the three benchmarking apps are presented in Table 3 and Table 4. Each build was performed three times to obtain average values. Before each run, the project's build folder and the application on the device were deleted to produce clean and complete builds without any steps skipped due to existing pre-compiled fragments from earlier runs.

The results indicate that the usage of Flutter and the AR Flutter plugin incurs overhead in both build times and application sizes. On Android, both measurements roughly double when using the cross-platform approach. On iOS, the build time increases even more drastically, while absolute

5 AppCoda. ARKit demo for horizontal planes detection.

TABLE 3 Average build times and resulting application sizes of the benchmarking apps on an android device.

	Average build time in release mode (s)	Application size on device (MB)
Flutter AR application	23.14	42.80
Native android AR application	10.07	20.76
Flutter virtual campus rally application	32.54	46.23
Native android virtual campus rally application	20.68	24.12

TABLE 4 Average build times and resulting application sizes of the benchmarking applications on an iOS device.

	Average build time in release mode (s)	Application size on device (MB)
Flutter AR application	62.64	33.4
Native iOS AR application	17.61	16.8

TABLE 5 Performance measurements during the run of an AR session benchmark of flutter and native apps on android and iOS devices.

	Average CPU utilization (%)	Maximum CPU utilization (%)	Average RAM utilization (MB)	Maximum RAM utilization (MB)	Average frame time
Flutter AR application on android	29	49	440	492	—
Native android AR application	28	46	428	399	—
Flutter AR application on iOS	16	23	410	450	16.7 ms
Native iOS AR application	16	19	305	323	16.7 ms

application sizes are smaller but also roughly double when switching to Flutter. While, in absolute terms, even the Flutter-based application compiles relatively fast, analyzing the share of the different build steps in the total time yields interesting insights: Considerable portions of the AR Flutter plugin's build times can be traced back to its own dependencies, for example, the packages used for permission handling, hinting towards further optimization potential. From benchmarking tests using the rally application, we deduct that the more complex an application becomes, the more the difference in build time and application size diminishes between the native and the cross-platform approach. At this point, it is worth mentioning that exact numbers cannot be scientifically compared between native and Flutter applications because the exact implementation influences the resulting data. Demonstrated trends and conclusions drawn in this section, however, are still representative of the influence of cross-platform frameworks on build time and application size.

While build time and application size are good indicators for an app's code efficiency, it is ultimately judged by users in terms of responsiveness, running smoothly, and not draining the battery. To investigate the performance achievable with Flutter apps using the AR Flutter plugin, a series of measurements were taken during the benchmark tests described above to evaluate the AR Session Performance: Average and maximum CPU utilization, average and maximum RAM utilization, and, on iOS, average frame time.

The results of the performance benchmark on the Android device are presented in Table 5. In essence, the cross-platform application based on the AR Flutter plugin performs on the same level as the native Android application. On iOS, solely the average and maximum RAM utilization exhibit an increase for the Flutter application since the AR Flutter plugin provides extensive functionality which, in this specific benchmarking application, is not required but still takes up memory. In absolute terms, however, this increase of 100 MB is negligible, especially taking

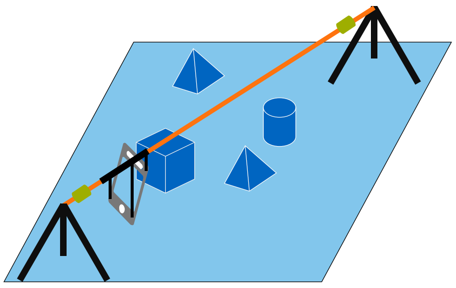


FIGURE 6
Test setup for the tracking quality benchmark.

into account that most modern phones have at least 4 GB of memory.

5.2 Tracking quality benchmark

The performance experiments described in Section 5.1 indicate that cross-platform applications using the AR Flutter plugin run just as smoothly as modern platform-specific apps using the native versions of ARCore and ARKit. However, AR frameworks are highly complex structures, so appearances can be deceptive. To reduce the processing load on the device and stop the application from jerking, incoming sensor readings such as camera frames can be analyzed less extensively or even skipped entirely, resulting in worse augmentation quality while superficial measurements like the device's frame rate remain unchanged. Thus, to test the quality of the augmentation and, therefore, the scene understanding performance offered by the AR Flutter plugin, we conducted a series of experiments directly measuring the number of features processed by the benchmarking applications.

The experiments were performed on the Samsung phone only as the iPad used in the previous benchmark test would not produce meaningful results because its processing power exceeds the demands of the benchmark applications by far, resulting in the AR frameworks never having to adjust tracking performance for the sake of retaining UI speed (see constant frame time in Table 5).

TABLE 7 Plane tracking quality benchmark.

	Total tracked AR plane area
Android native AR application	8.19 ± 0.67 m^2
Flutter AR application	8.15 ± 0.89 m^2

As this experiment highly depends on constant conditions, we introduced additional boundaries: A static scene is traversed by the devices at a fixed angle and speed using a rope anchored on two tripods, each tracked run takes exactly 10 s (see Figure 6). The benchmarking applications were slightly modified to log the total number of AR frames (quantity of SLAM updates) and detected feature points, the number of detected feature points per AR frame (quality of SLAM updates), and the total detected plane area.

While we minimized the number of background processes running on the device, effects of external factors such as CPU base-load and device temperature cannot be ruled out. Hence, Table 6 and Table 7 show mean and standard deviation of five consecutive runs.

The evaluation of low-level tracking features to measure the AR Flutter plugin's impact on a SLAM performance level yields a similar number of AR frames for both applications. While the total number of feature points, and, consequently, the average number of feature points per AR frame, is slightly lower in the cross-platform application, the values' mean ± standard deviation intervals still overlap. Based on this data, it can be concluded that the SLAM algorithm's performance suffered a negligible impact through the additional layer of abstraction our plugin introduces.

The plane area detected by the tracking algorithms directly translates to the area available for the user to place AR objects onto, rendering it a valuable criterion for AR performance. The results in Table 7 reinforce the findings of the previous section: The differences in the detected plane area of the native and the cross-platform application are insignificantly small, leading to the conclusion that cross-platform applications utilizing our plugin allow for the same level of augmentation quality as state-of-the-art native apps. While the results obtained using the applications developed for testing purposes might not reflect the performance of every possible AR use case, they certainly provide insights on the overall trends and the general

TABLE 6 Feature point tracking quality benchmark.

	Total number of AR frames	**Total number of feature points**	**Feature points per AR frame**
Android native AR application	298 ± 12	41,000 ± 3,107	137 ± 8
Flutter AR application	300 ± 9	38,057 ± 2,324	126 ± 7

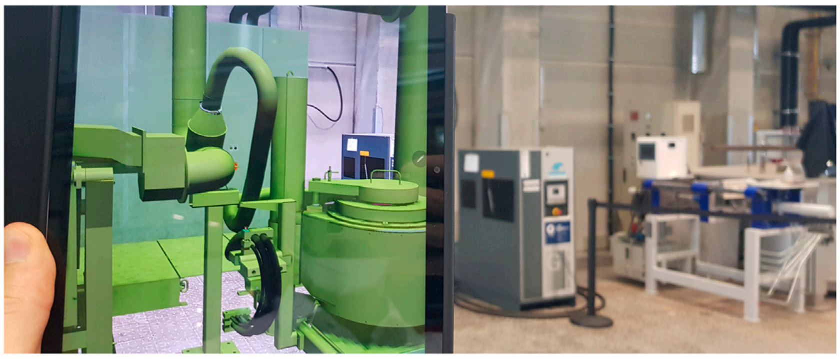


FIGURE 7
Augmentation of an industrial site with the flutter plugin on an android tablet using a life-sized CAD model of the machinery (Carius et al., 2022).

performance comparison between realizing an AR use case in a cross-platform versus a native approach.

5.3 Community reception

Developing the AR Flutter plugin open source led to the opportunity of generating feedback even before the first version was published. Community interest in the AR Flutter plugin continuously grew with the extent of features increasing and, at the time of writing, is reflected by the repository having 205 GitHub stars, 109 forks, and ongoing discussions about future features. On Flutter's official package management system, the AR Flutter plugin is currently among the top 8% of the most used packages over the past 60 days[6], indicating that it is already used in many cross-platform cloud-based AR applications. The feedback received in this early phase shows that the framework satisfies a real need in the AR market and forms the foundation of a community-supported cross-platform AR framework with a large userbase.

6 Future work

During our research, we discovered areas worth exploring further and possibilities for building upon the developed framework. The most apparent area for future development is the addition of features to the AR Flutter plugin itself. Building on the community feedback, preferred features are image anchors that allow the placement of content relative to a predefined 2D image, and face anchors, which enable the dynamic augmentation of people's faces with digital content. Additionally, with the help of an early contributor, the foundation was already built to include anchors into the plugin that depend solely on GPS data without requiring visual feature points, allowing for content to be placed in the far distance.

Another branch of possible improvement is the expansion to further cloud anchor providers. As some businesses might not be comfortable with storing data on Google's servers, offering developers additional choices of backends for storing collaborative anchors, e.g., Azure's spatial anchors, could further increase the impact of the AR Flutter plugin. If more providers of cloud anchor functionality emerge, the interaction with the backends could also be outsourced to the client code using callbacks and interfaces.

A final suggestion for future research is the development of a cross-platform AR benchmarking framework. As mentioned in Section 5, currently, AR sessions cannot be recorded and replayed uniformly on Android and iOS devices to obtain standardized testing conditions. A study on the topic should first examine the exact sensors used by ARCore and ARKit to establish an environment estimation and deduct a hardware setup used to record a standardized benchmarking dataset of AR scenarios. Finally, software to replay the session data into ARCore- and ARKit-based applications and record performance metrics should be developed. A test framework of this kind would benefit the future development of both tracking algorithms and higher-level AR frameworks and would help developers choose the AR software solution best suited to the task at hand.

Aside from further scientific work, our AR Flutter plugin allows for the creation of novel applications and games based on the concept of cloud-based collaborative AR, ranging from industrial optimization tools to superhuman sports.

6 https://pub.dev/packages/ar_flutter_plugin.

7 Conclusion

We introduced a novel cross-platform AR framework based on the UI development kit Flutter. The framework's focus lies on providing state-of-the-art cloud-based collaborative AR functionality that works seamlessly on both Android and iOS while exposing a common, intuitive API that allows non-expert developers to utilize the potential of AR as a feature of their application. The implementation consists of the AR Flutter plugin and a web-based content management system and is designed to achieve flexibility through a modular approach. Our analyses find that there is no significant difference in the application-level performance and the augmentation quality between a native approach and the cross-platform framework contributed by this work. The AR Flutter plugin's versatility could already be proven through the realization of an industrial machinery augmentation project (Figure 7).

Our contribution facilitates a paradigm shift in the way complex location-based AR workflows can be established: Instead of having to resort to preparation-heavy approaches involving 3D models of the deployment environment and a computation backbone (Baek et al., 2019), our plugin allows to utilize AR annotations for tasks like facility management in unknown terrain on standard smartphone hardware. Our approach builds on a modular architecture, which, combined with a reliance on efficient patterns and the support of a large community of developers, has the potential to bundle the technological capabilities of various frameworks into a powerful single source of AR functionality.

Data availability statement

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found in the article/supplementary material.

Author contributions

Conceptualization: CE and LC; Software: LC; Writing, review and editing: CE, LC, LR, and DP; Supervision; CE, DP, and GK; All authors have read and agreed to the published version of the manuscript.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Baek, F., Ha, I., and Kim, H. (2019). Augmented reality system for facility management using image-based indoor localization. *Automation Constr.* 99, 18–26. doi:10.1016/j.autcon.2018.11.034

Bonasio, A. (2019). Report: XR Industry insight 2019-2020. Available at: https://medium.com/edtech-trends/report-xr-industry-insight-2019-2020-5-a7a7ed9c63c (Accessed June 22, 2022).

Bostanci, E., Kanwal, N., Ehsan, S., and Clark, A. F. (2013). User tracking methods for augmented reality. *Int. J. Comput. Theory Eng.* 5, 93–98. doi:10.7763/IJCTE.2013.V5.654

Carius, L., Eichhorn, C., Plecher, D. A., and Klinker, G. (2022). "Cloud-based cross-platform collaborative ar in flutter," in 2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW) (Christchurch, NZ: IEEE), 682–683.

Coninck, B. D. (2019). Flutter versus other mobile development frameworks: A UI and performance experiment. Part 2. Available at: https://blog.codemagic.io/flutter-vs-android-ios-xamarin-reactnative/ (Accessed June 22, 2022).

Egodagamage, R., and Tuceryan, M. (2018). Distributed monocular visual slam as a basis for a collaborative augmented reality framework. *Comput. Graph.* 71, 113–123. doi:10.1016/j.cag.2018.01.002

Eichhorn, C., Jadid, A., Plecher, D. A., Weber, S., Klinker, G., and Itoh, Y. (2020). "Catching the Drone - a tangible augmented reality game in superhuman sports," in 2020 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct) (Recife, Brazil: IEEE), 24–29. doi:10.1109/ismar-adjunct51615.2020.00022

Francesco, G. M. D. (2022). ARCore flutter plugin. Available at: https://github.com/giandifra/arcore_flutter_plugin (Accessed June 22, 2022).

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software.* Boston, MA, USA: Addison-Wesley.

Horst, R., Fenchel, D., Retz, R., Rau, L., Retz, W., and Dörner, R. (2021). "Integration of game engine based mobile augmented reality into a learning management system for online continuing medical education," in INFORMATIK 2020.

Huo, K., Wang, T., Paredes, L., Villanueva, A. M., Cao, Y., and Ramani, K. (2018). "Synchronizar: Instant synchronization for spontaneous and spatial collaborations in augmented reality," in UIST '18: Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology (New York, NY, USA: Association for Computing Machinery), 19–30. doi:10.1145/3242587.3242595

InVerita (2020). Flutter vs React native vs native: Deep performance comparison. Available at: https://bit.ly/3nAgpMF (Accessed June 22, 2022).

Kaufmann, H. (2003). Collaborative augmented reality in education. *Tech. Rep.*

Keshavarzi, M., Yang, A. Y., Ko, W., and Caldas, L. (2020). "Optimization and manipulation of contextual mutual spaces for multi-user virtual and augmented reality interaction," in 2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR) (Piscataway, NJ, USA: IEEE), 353–362.

Kumar, T., Sharma, S., Sharma, A., Malhotra, J., and Gupta, V. (2021). "Using flutter to develop a hybrid application of augmented reality," in *Computational intelligence for information retrieval* (Boca Raton, FL, USA: CRC Press), 141–156.

Lee, J., and Baek, S. I. (2011). Adoption of internet technologies in small business. *Int. J. Digital Manag.*

Leuschenko, O. (2021). ARKit flutter plugin. Available at: https://github.com/olexale/arkit_flutter_plugin (Accessed June 22, 2022).

Lock, O., Bednarz, T., and Pettit, C. (2019). "Holocity – exploring the use of augmented reality cityscapes for collaborative understanding of high-volume urban sensor data," in VRCAI '19: The 17th International Conference on Virtual-Reality Continuum and Its Applications in Industry (New York, NY, USA: Association for Computing Machinery). doi:10.1145/3359997.3365734

MacWilliams, A., Reicher, T., Klinker, G., and Bruegge, B. (2003). *Design patterns for augmented reality systems*.

Marchesi, G., Eichhorn, C., Plecher, D. A., Itoh, Y., and Klinker, G. (2021). Envslam: Combining slam systems and neural networks to improve the environment fusion in ar applications. *ISPRS Int. J. Geoinf.* 10, 772. doi:10.3390/ijgi10110772

Miedema, N. A., Vermeer, J., Lukosch, S., and Bidarra, R. (2019). "Superhuman sports in mixed reality: The multi-player game league of lasers," in 2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR) (Osaka, Japan: IEEE), 1819–1825.

Mourtzis, D., Siatras, V., Angelopoulos, J., and Panopoulos, N. (2020). An augmented reality collaborative product design cloud-based platform in the context of learning factory. *Procedia Manuf.* 45, 546–551. doi:10.1016/j.promfg.2020.04.076

Ohlenburg, J., Herbst, I., Lindt, I., Fröhlich, T., and Broll, W. (2004). "The morgan framework: Enabling dynamic multi-user ar and vr projects," in Proceedings of the ACM symposium on Virtual reality software and technology, 166–169.

Oriti, D., Manuri, F., Pace, F. D., and Sanna, A. (2021). Harmonize: A shared environment for extended immersive entertainment. *Virtual Real.* 1–14, 1–14. doi:10.1007/s10055-021-00585-4

Pereira, N., Rowe, A., Farb, M. W., Liang, I., Lu, E., and Riebling, E. (2021). "Arena: The augmented reality edge networking architecture," in 2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR) (Bari, Italy: IEEE), 479–488.

Piumsomboon, T., Dey, A., Ens, B., Lee, G., and Billinghurst, M. (2017). "[poster] covar: Mixed-platform remote collaborative augmented and virtual realities system with shared collaboration cues," in 2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct), 218–219. doi:10.1109/ISMAR-Adjunct.2017.72

Plecher, D. A., Eichhorn, C., Köhler, A., and Klinker, G. (2019). "Oppidum-a serious-ar-game about celtic life and history," in International Conference on Games and Learning Alliance (Berlin, Germany: Springer), 550–559.

Plecher, D. A., Ludl, M., and Klinker, G. (2020). "Designing an ar-escape-room with competitive and cooperative mode," in *GI VR/AR workshop* (Bonn, Germany: Gesellschaft für Informatik eV).

Plecher, D., Eichhorn, C., and Klinker, G. (2022). *Roar-role of augmented reality in serious games and superhuman sports*.

Ren, J., He, Y., Huang, G., Yu, G., Cai, Y., and Zhang, Z. (2019). An edge-computing based architecture for mobile augmented reality. *IEEE Netw.* 33, 162–169. doi:10.1109/MNET.2018.1800132

Schmalstieg, D., Fuhrmann, A., Hesina, G., Szalavári, Z., Encarnaçao, L. M., Gervautz, M., et al. (2002). The studierstube augmented reality project. *Presence. (Camb).* 11, 33–54. doi:10.1162/105474602317343640

Skuza, B., Mroczkowska, A., and Wlodarczyk, D. (2019). Flutter vs. React native – what to choose in 2021? Available at: https://www.thedroidsonroids.com/blog/flutter-vs-react-native-what-to-choose-in-2021 (Accessed June 22, 2022).

Statista Inc (2021). Cross-platform mobile frameworks used by software developers worldwide in 2019 and 2020. *Tech. Rep.*

Weber, S., Rudolph, L., Eichhorn, C., Dyrda, D., Plecher, D. A., Klinker, G., et al. (2022). Frameworks enabling ubiquitous mixed reality applications across dynamically adaptable device configurations. *Front. Virtual Real.* 36. doi:10.3389/frvir.2022.765959

Zhang, W., Han, B., Hui, P., Gopalakrishnan, V., Zavesky, E., and Qian, F. (2018). "Cars: Collaborative augmented reality for socialization," in HotMobile '18: Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications (New York, NY, USA: Association for Computing Machinery), 25–30. doi:10.1145/3177102.3177107

Zillner, J., Mendez, E., and Wagner, D. (2018). "Augmented reality remote collaboration with dense reconstruction," in 2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct) (Munich, Germany: IEEE), 38–39. doi:10.1109/ISMAR-Adjunct.2018.00028