A Unity-based Offline Task Management Application with Local Storage and Local Notifications

Yi-Heng Wu [1], Jing-Lun Wang [1], and Lung-Fa Wu [2,*]

[1] Department of Computer Information and Network Engineering & Master Program, Lunghwa University of Science and Technology, Taiwan
[2] Department of Electrical Engineering, National Ilan University, Taiwan
Email: zzz7136@gmail.com (Y.-H.W.); junlun0423@gmail.com (J.-L.W.); lonfawu0405@gmail.com (L.-F.W.)
*Corresponding author

***Abstract*—Task management applications are widely used for personal productivity and reminder scheduling. However, many existing solutions depend on cloud services and account synchronization, which may be unsuitable for offline or privacy-sensitive environments. This study presents a Unity-based offline task management application that integrates JavaScript Object Notation (JSON)-based local storage and Android local notifications. The objective is to evaluate the feasibility of using Unity, a game engine, for a non-gaming mobile productivity application. The system supports task creation, browsing, deletion, local data persistence, and scheduled notifications. Experiments on an Android smartphone evaluated notification reliability, notification delay, storage performance, and resource consumption. Results showed notification success rates of 100.0%, 96.7%, and 93.3% in foreground, background, and screen-locked conditions, respectively, with average delays ranging from 0.42 s to 3.21 s. JSON read/write operations remained below 20 ms for up to 500 tasks. The Android Package (APK) file size was 39.9 MB and 121 MB after installation. The findings indicate that Unity is a feasible platform for lightweight offline task management applications, although trade-offs exist in resource usage and local storage security.**

***Keywords*—task management, mobile application, local notification, offline system, unity**

I. Introduction

Task management applications have become increasingly important in supporting personal productivity, time management, and daily activity organization. As mobile technologies continue to evolve, users expect applications to provide convenient access, intuitive interfaces, and reliable functionality across different usage scenarios. Alshehri *et al*. [1] investigated user concerns and preferences regarding mobile health applications and reported that usability, privacy, and accessibility remain critical factors influencing user acceptance and long-term engagement. These findings highlight the importance of designing mobile applications that effectively balance functionality with user-centered requirements.

Recent advances in mobile application development have expanded beyond traditional software architectures to include immersive and interactive technologies. Vijay *et al.* [2] demonstrated the integration of artificial intelligence and augmented reality in a mobile-based virtual dressing system, illustrating the capability of modern application frameworks to support complex user interactions and real-time processing. Similarly, Rey *et al*. [3] emphasized the importance of system monitoring and performance optimization in maintaining application efficiency and ensuring a satisfactory user experience. These studies indicate that application performance, responsiveness, and reliability are essential considerations in modern software design.

Commercial task management applications already provide mature functions such as task creation, list browsing, reminders, categorization, and cross-device synchronization. However, many of these systems are designed around cloud-based services, account synchronization, and ecosystem integration. Although this design is useful for general productivity purposes, it may not be suitable for all usage scenarios. In low-connectivity environments, training environments, enterprise intranets, or privacy-sensitive situations, users may require a task management system that can operate locally without relying on continuous network access or external servers.

The practical motivation for this study therefore stems from the need for lightweight offline task management. In daily use, users may need to create, view, and delete task records even when network connectivity is unstable or unavailable. Locally stored task data provide immediate access to information while reducing dependence on cloud-based services. For small-scale personal task management, JavaScript Object Notation (JSON)-based storage is a simple and lightweight implementation approach. However, this design also requires careful

Manuscript received April 12, 2026; revised May 20, 2026; accepted July 10, 2026; published November 10, 2026.

discussion because flat JSON files do not provide the same data integrity, transaction safety, or scalability as mobile database systems.

From a software engineering perspective, cross-platform development has become a widely adopted strategy for reducing development costs and improving deployment flexibility. El-Kassas *et al*. [4] presented a comprehensive taxonomy of cross-platform mobile application development approaches and discussed the trade-offs among portability, performance, and maintainability. Among the available frameworks, Unity has gained popularity not only in game development but also in non-gaming interactive applications due to its integrated development environment, C#-based programming support, and cross-platform deployment capability. In addition, Unity allows developers to integrate user interface components, data management logic, and notification-related modules within a single development environment. This makes Unity a suitable experimental platform for educational prototyping, lightweight application development, and future extensions toward interactive visual interfaces, 3D task visualization, or Extended Reality (XR)-based productivity environments.

In distributed and resource-constrained environments, efficient task allocation and synchronization mechanisms also play an important role in system reliability. Wang *et al.* [5] proposed a conflict-reduced task offloading scheme for edge computing-enabled Internet of Things (IoT) networks, highlighting the importance of effective task scheduling and resource management. Although the application domain differs from personal productivity software, the underlying concept of task coordination remains relevant for managing scheduled operations and notifications in mobile applications. Furthermore, Sukarsa *et al.* [6] developed a synchronization model for multiplayer educational games and demonstrated the significance of maintaining data consistency and reliable communication among distributed components. Their findings provide useful insights into the design of systems that require dependable data handling and event management.

A. Research Gap

Despite the growing adoption of mobile productivity applications, most existing task management systems rely heavily on cloud synchronization and online services. These commercial systems mainly emphasize account integration, cross-device synchronization, and mature productivity services. In contrast, limited research has examined the feasibility and technical trade-offs of implementing an offline-first task management application using Unity with local data persistence and scheduled notification mechanisms.

Specifically, the technical trade-offs of using Unity for non-gaming mobile productivity applications remain underexplored. Unity is commonly associated with 2D and 3D game development rather than conventional text-based productivity tools. Therefore, its use for a standard User Interface (UI)-based task management application requires technical examination. In this study, Unity is selected not because it is more resource-efficient than native Android, Flutter, or React Native frameworks, but because it provides a unified development environment, cross-platform deployment capability, and potential extensibility toward interactive visual interfaces.

In particular, it remains necessary to evaluate whether a Unity-based offline implementation can provide acceptable notification reliability, local storage performance, and resource consumption under common Android usage scenarios. Equally important is clarifying the limitations of this implementation in terms of storage scalability, Android notification lifecycle behavior, and local data security. Therefore, this study focuses on a lightweight offline implementation that combines Unity, JSON-based local persistence, and Android local notification scheduling.

B. Research Questions

Based on the above research gap, this study addresses the following Research Questions (RQ):

RQ1: Can Unity be used as a feasible non-gaming framework for developing an offline-capable mobile task management application?

RQ2: What is the resource-consumption characteristics of a Unity-based task management application in terms of Android Package (APK) file size, Random-Access Memory (RAM) usage, Central Processing Unit (CPU) usage, and battery consumption?

RQ3: How reliable are JSON-based local persistence and Unity-based local notification scheduling under common Android usage scenarios, including foreground, background, and screen-locked operation?

RQ4: What are the major limitations of using JSON-based local storage and Unity-based notification scheduling for lightweight offline task management?

C. Contributions and Goals of This Study

Accordingly, the contributions and goals of this study are summarized as follows:

(1) This study implements a Unity-based offline task management application as a non-gaming mobile productivity application case study.
(2) The proposed system integrates JSON-based local persistence and Android local notification scheduling for lightweight offline task management.
(3) The system is evaluated through notification reliability, notification delay, JSON storage performance, resource consumption, and preliminary observational usability assessment.
(4) The study discusses the technical trade-offs and limitations of using Unity for standard UI-based mobile task management, including resource footprint, Android notification lifecycle constraints, and local data security considerations.

II. Literature Review

A. Research on Task Management Tools

Digital task management tools have been widely studied as practical systems for supporting task organization, workflow coordination, and reminder-based planning.

Bellotti *et al*. [7] examined how users record, organize, and complete to-do tasks and derived design implications for the development of personal task list managers. Esposito *et al*. [8] further investigated the use of a smartphone-based to-do list application in an intensive care unit and found that mobile task management tools could support workflow coordination and reduce task omission in high-pressure work environments. Khaokaew *et al*. [9] also reported that modern knowledge workers rely heavily on digital tools to manage tasks in multitasking work contexts, and that systems supporting priority handling and flexible scheduling can better respond to users' productivity needs. These findings indicate that task management applications function not only as recording tools but also as workflow-support mechanisms.

User requirements for task management tools are also highly individualized. Nygaard *et al.* [10] showed that users' daily prospective memory needs vary according to personal habits, task types, and contextual demands, suggesting that reminder and to-do list systems should provide flexibility rather than fixed interaction flows. Similarly, Sanders and Boot [11] found that users are sensitive to reminder format, visual presentation, and technology support preferences, which may influence whether a task management system is accepted and continuously used. These studies suggest that system usability, reminder clarity, and interface design are important factors in task management application development.

In addition to general users, task management and reminder systems have also been examined in elderly and cognitively demanding contexts. Baric *et al*. [12] reported that older adults using an interactive digital calendar with mobile phone reminders perceived such tools as helpful for structuring daily routines and remembering scheduled activities. Andreassen *et al.* [13] further demonstrated that an interactive digital calendar with mobile reminders could support people with cognitive impairment by helping them manage everyday tasks and reduce memory-related difficulties. These findings underscore the relevance of reminder-based mobile tools in both general productivity and assistive technology contexts.

B. Design Principles of Digital Reminder Systems

Reminder mechanisms are a core component of mobile task management systems because they directly influence whether users can complete tasks at the intended time. Haj *et al*. [14] showed that smartphone-based calendars can be used to support prospective memory rehabilitation in patients with Alzheimer's disease, suggesting that mobile reminder systems can play a practical role in daily memory support. Scullin *et al*. [15] also demonstrated through a randomized controlled trial that smartphone reminder technology can improve prospective memory functioning, especially in situations where users need to remember future intentions at specific times. These studies indicate that mobile reminders are not merely notification features but can serve as cognitive support tools.

From an interaction design perspective, Brewer *et al.* [16] emphasized that effective reminder systems should consider how users formulate, interpret, and act upon reminders. Their study showed that reminder tools should provide appropriate timing, context, and feedback to reduce the likelihood of missed tasks. A systematic scoping study by Sriranganathan *et al.* [17] further indicated that digital planning-based technologies can support memory-related functioning among older adults with mild cognitive impairment, particularly when tools integrate planning, reminder, and task-tracking functions. These findings suggest that reminder design should consider both technical scheduling and user comprehension.

Recent application-oriented studies also highlight the need for personalization and offline-capable design. Torino *et al*. [18] proposed the MañanaList App as a mobile to-do list application for efficient and personalized task management, emphasizing the importance of task creation, reminder scheduling, and user-centered task organization. Shama *et al*. [19] reviewed existing note-taking and task-management applications and pointed out that many current systems focus on basic task recording and interface design, while still leaving room for improvement in offline access, data control, and flexible reminder mechanisms. Therefore, a task management application should be evaluated not only by whether it can create and display tasks, but also by whether its reminder mechanism, data storage, and usage context are technically reliable.

C. Offline-First and Local Data Persistence in Mobile Applications

Offline-first design is an important consideration for mobile applications used in unstable or unavailable network environments. Escobar-Velásquez *et al.* [20] investigated eventual connectivity issues in Android applications and noted that mobile applications are frequently used in situations where internet connectivity is unavailable or unreliable; inappropriate handling of such connectivity issues may lead to bugs, crashes, and reduced user confidence. These finding positions offline operation as a key design requirement for mobile task management applications.

From an Android application architecture perspective, offline-first applications should remain usable without a reliable network connection and should present local data immediately instead of waiting for network calls [21]. Android's official architecture documentation further emphasizes that, in offline-first design, a local data source such as a database is recommended as the source of truth to maintain consistent and correct data across application layers [22]. In the context of task management, this means that users should still be able to create, view, modify, and delete task records even when cloud services are unavailable.

Local persistence mechanisms can be implemented using different approaches, including simple JSON files, SQLite-based databases, Room, Realm-like embedded databases, or other local storage frameworks. Song *et al*. [23] compared Android persistence frameworks and noted that persistence is a fundamental function of mobile applications, while execution time,

energy consumption, and programming effort should be considered when selecting a persistence mechanism. This suggests that local storage should be evaluated not only by implementation simplicity but also by performance and long-term maintainability.

In this study, JSON-based local storage is adopted as a lightweight implementation choice for a small-scale offline task management prototype. JSON files are easy to inspect and simple to implement, making them suitable for storing basic task records such as task content, scheduled time, and task status. However, JSON files do not provide built-in transaction safety, indexing, query optimization, or strong data integrity mechanisms. In contrast, SQLite supports atomic commit, meaning that all changes within a transaction either occur completely or do not occur at all, even if interruption or failure occurs during the transaction process [24]. Therefore, JSON-based storage should be understood as a lightweight prototype-level solution rather than a complete replacement for local database systems. For larger-scale task management systems, SQLite, Room, or other embedded local databases would be more appropriate choices.

D. Unity for Non-Gaming Mobile Applications

Unity is widely known as a development platform for 2D/3D games, simulations, and interactive applications. However, its role can extend beyond conventional game development. Biørn-Hansen *et al*. [25] argued that cross-platform mobile development frameworks can reduce development cost and support multi-platform deployment, but they may also introduce performance overhead compared with native Android implementation. This finding is relevant to the present study because Unity is also a cross-platform development environment, and its use for a standard UI-based productivity application should therefore be evaluated in terms of APK size, memory usage, CPU usage, and battery consumption.

Unity's strength lies in its component-based development model, visual editor, cross-platform deployment capability, and support for interactive visual content. Juliani *et al*. [26] further argued that modern game engines can serve as general configurable platforms for developing rich visual, physical, task-based, and interactive environments. Although their study focused on intelligent agents and simulation environments, it supports the broader idea that Unity can function as more than a game engine when the target application requires interaction, visualization, or future extension toward 2D/3D and XR-based interfaces.

For the notification module, Unity also provides an official Mobile Notifications Package. According to Unity's documentation, this package supports scheduling local one-time or repeatable notifications on Android and iOS [27]. This makes it technically feasible to implement local reminder functions inside a Unity-based mobile application. Nevertheless, using Unity for a standard UI-based task management application still involves trade-offs. Its rendering-oriented runtime may introduce a larger package size and higher resource consumption than native Android, Flutter, or React Native implementations. Accordingly, Unity is adopted in this study as an experimental development framework for examining the feasibility and limitations of a game engine in non-gaming offline mobile productivity applications.

E. Comparative Analysis of Existing Task Management Systems

Existing commercial task management systems already provide mature functions for task creation, reminder scheduling, and cross-device synchronization. Google Tasks is closely integrated with the Google ecosystem and allows users to track to-dos that sync across devices, while also supporting integration with Google Workspace services such as Gmail and Calendar [28]. This design suits users already embedded in that ecosystem. However, its design is mainly centered on account-based synchronization and cloud service integration rather than user-controlled local data persistence.

Microsoft to Do similarly emphasizes cross-device and productivity ecosystem integration. According to Microsoft's official description, Microsoft to Do allows users to sync lists across the web, iOS, Android, and Windows devices [29]. This design is useful for users who need consistent task access across multiple devices and Microsoft services. However, its architecture remains closely tied to Microsoft account-based synchronization and cloud-connected productivity services.

Todoist also supports offline use, but its offline mechanism is still designed around eventual synchronization. According to Todoist's official documentation, offline mode is automatically activated when the user has no internet access, and any changes made offline are automatically synchronized with the server after the device reconnects to the internet [30]. Therefore, although Todoist provides offline functionality, this offline mode is primarily a temporary local operation followed by cloud synchronization.

Compared with these commercial task management systems, the proposed Unity-based application is designed as an experimental implementation case rather than a commercial replacement for Google Tasks, Microsoft to Do, or Todoist. The proposed system focuses on local JSON-based data persistence, Android local notification scheduling, offline operation, and empirical evaluation of technical trade-offs.

F. Summary

Based on the reviewed literature and system comparison, digital task management and reminder systems have been shown to support task organization, prospective memory, and daily routine management. Existing studies also indicate that reminder timing, interface clarity, personalization, and user context are important factors affecting system usability. However, many existing commercial task management systems are designed around cloud synchronization, account integration, and ecosystem-based productivity services.

In contrast, the proposed system focuses on a lightweight offline implementation using Unity, local JSON persistence, and Android local notification scheduling. The literature also suggests that offline-first design, local persistence, resource consumption, and

notification reliability should be evaluated as technical factors in mobile application development. Therefore, this study contributes an experimental implementation case and empirical evaluation of a Unity-based offline task management application, with emphasis on notification reliability, local storage performance, resource consumption, and system limitations.

III. Methodology

This study adopts a system design and implementation approach to develop a mobile task management application. The research process consists of three phases: system architecture design, implementation, and evaluation.

The proposed system is developed using the Unity engine and C# programming language. It integrates local JSON-based data persistence with mobile notification mechanisms to support offline task management and real-time reminders. System performance is evaluated through real-device testing and observational user evaluation to assess usability and reliability.

A. Development Environment and Tools

The system is developed using Unity 2021 as the primary development platform, with Android smartphones as the target deployment environment. The Unity engine provides cross-platform capabilities for user interface development and resource management, and supports built-in features such as JSON serialization and mobile notification packages, which facilitate local data storage and local notification implementation.

The application is programmed in C#, utilizing Unity's Application Programming Interface (API) to handle user interface control, file operations, and local notification scheduling. The development and testing environment includes a Windows 10 personal computer for system development and simulation, as well as Android mobile devices for real-device testing. This setup allows the application to be evaluated under practical mobile operating conditions.

B. System Architecture and Data Flow

The proposed system adopts a modular layered architecture consisting of three main components: the data persistence layer, the application logic and User Interface (UI) control layer, and the notification management layer. As illustrated in Fig. 1, these modules collaborate to support core functionalities, including task creation, storage, visualization, and reminder notification.



Fig. 1. Architecture and data flow of the proposed system. JSON: JavaScript Object Notation.

The *JsonDataSystem* represents the data persistence layer and is responsible for storing users' task data locally in JSON format, as well as providing interfaces for data creation, retrieval, and deletion. The corresponding workflow is illustrated in Fig. 2. To support different execution environments, this module dynamically selects appropriate file storage paths. Specifically, *Application.dataPath* is used within the Unity Editor, while *Application.persistentDataPath* is applied on Android devices. Task data are organized using a custom-defined *ReminderList* structure, which contains multiple *ReminderData* entries. Each entry records essential attributes, such as task content and reminder time, enabling efficient data storage, retrieval, and management.

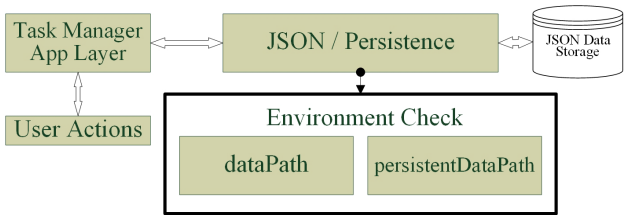


Fig. 2. Data management system flowchart.

The *ToListManager*, which serves as the application logic and UI control layer, is responsible for handling user interaction logic and managing the task list. This module monitors user interface events, such as button clicks, collects user inputs including task content and scheduling information, and controls the real-time updating of the task list. When users create or edit tasks through the UI, the *ToListManager* first performs data validation to ensure the correctness and completeness of the input before proceeding with further processing.

After validation, the *ToListManager* invokes the *JsonDataSystem* to write or update task data in the local JSON file and synchronizes the user interface to immediately display the newly created or modified task items. At the same time, the *ToListManager* dynamically generates UI components for the task list (e.g., creating new task entries) and binds event listeners to the delete button of each item, allowing users to remove tasks directly through the interface. During the UI update process, the system continuously compares the current time with each task's deadline. If a task has exceeded its scheduled time, it is marked with a red "!" icon; otherwise, a green "O" icon is displayed to indicate that the task remains within its valid timeframe. This color-coded visual feedback mechanism provides real-time information on task urgency, enabling users to quickly distinguish between overdue and active tasks.

The *NotificationSystem* represents the notification management layer and is responsible for scheduling and managing local notifications on mobile devices. When a user creates a task with a specified reminder time, the *ToListManager* invokes the *ScheduleNotification()* method in the *NotificationSystem* to register the task content and scheduled time as a local notification. The *NotificationSystem* uses Unity's Mobile Notifications Package to establish notification channels and schedule

notifications on the Android platform. This allows reminder messages to be scheduled for delivery at the designated time, including the background execution conditions examined in this study. When a task is deleted, the *ToListManager* notifies the *NotificationSystem* to execute the *CancelNotification()* method, which cancels any previously scheduled but not yet triggered notifications associated with the task. This prevents outdated reminders from being displayed after the task has been removed. In addition, the *NotificationSystem* checks and requests user permission for notifications during application startup to support correct reminder operation. The notification scheduling workflow is illustrated in Fig. 3. The notification permission handling process is illustrated in Fig. 4.

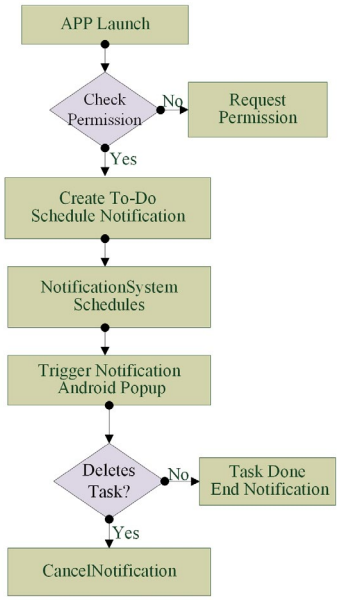


Fig. 3. Notification scheduling workflow diagram.

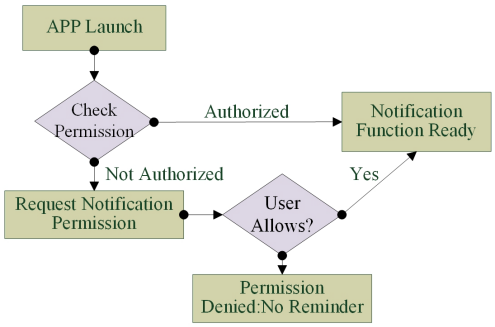


Fig. 4. Notification permission flowchart.

C. User Interaction Flow

To enhance clarity and usability, this study designs an intuitive user interaction workflow. The system supports three primary user operations: task creation, task list loading, and task deletion, as illustrated in Fig. 5. The processing procedure for each operation is as follows.

When a user clicks the "Add" button on the main interface, the system displays an input form that allows the user to enter task content and specify a reminder time. Once the user confirms the input by pressing the "Save" button, the corresponding functions in the ToListManager are triggered. The system first performs input validation to check data correctness. Subsequently, the *JsonDataSystem* is invoked to append and store the new task information in the local JSON file. The user interface is then updated to reflect the newly created task item in the task list. If the task includes a scheduled reminder time, the *ToListManager* simultaneously calls the *ScheduleNotification()* method in the *NotificationSystem* to register the task as a local notification. This allows a reminder message to be delivered at the specified time on the mobile device.

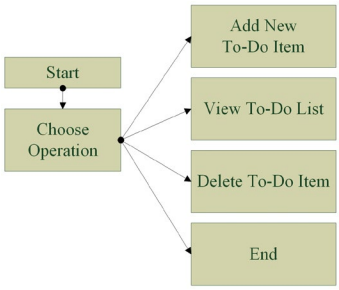


Fig. 5. User interaction flowchart.

Whenever the application is launched or resumed from the background, the system automatically loads the existing task list. The *ToListManager* invokes the *GetJson()* method of the *JsonDataSystem* to retrieve all locally stored task data. The system then dynamically generates the corresponding number of UI components for each task item, including task descriptions, scheduled time, status indicators, and delete buttons. Event listeners are again bound to each delete button so that user interactions correctly trigger the removal procedure. Through this dynamic loading mechanism, the displayed task list remains current each time the application is launched or resumed.

When a user clicks the delete button associated with a task item in the list, the user interface triggers the *deleteItem()* method in the *ToListManager* to process the deletion request. The method first removes the corresponding task from the currently loaded data structure. Subsequently, the *JsonDataSystem* invokes the *DeleteJson()* method to remove the associated record from the local JSON file. At the same time, the *ToListManager* notifies the *NotificationSystem* to execute the *CancelNotification()* method, which cancels any pending notifications associated with the deleted task, again preventing outdated reminders from appearing. Finally, the system updates the user interface to reflect the removal of the task item, providing immediate visual feedback to the user.

D. Experimental Setup

To strengthen the empirical basis of the proposed system, this study conducted a set of real-device tests to evaluate notification reliability, local JSON storage performance, and system resource consumption. The evaluation was designed to verify whether the Unity-based implementation could provide acceptable performance for

offline task management under common Android usage scenarios.

The experimental tests were conducted on a Samsung Galaxy A51 smartphone running Android 13 with One UI 5.1. The device was selected as a representative mid-range Android smartphone to evaluate whether the proposed Unity-based task management application could operate under practical mobile hardware conditions.

The experimental environment is summarized in Table I. The application was developed using Unity 2021 and C# and deployed on the Android platform for real-device testing. The tests were conducted under three application states: foreground execution, background execution, and screen-locked mode. These scenarios were selected because task reminder applications are frequently used under different mobile execution conditions, especially when the application is not actively displayed on the screen.

TABLE I. EXPERIMENTAL ENVIRONMENT

Item	Description
Development platform	Unity 2021
Programming language	C#
Target platform	Android
Testing device	Samsung Galaxy A51
Android version	Android 13 / One UI 5.1, verified from device software information
Test repetitions	30 per notification scenario
Task data sizes	10, 50, 100, and 500 task records
Main evaluation metrics	Notification success rate, notification delay, JSON read/write time, Android Package (APK) size, RAM usage, CPU usage, and battery consumption

E. Evaluation Metrics and Formulas

To quantitatively evaluate the proposed Unity-based offline task management application, this study adopts four categories of performance metrics, namely notification reliability, notification delay, JSON storage performance, and resource consumption. These metrics are derived from established software quality assessment standards and computer system performance evaluation methodologies [31–33].

First, notification success rate is used to evaluate the reliability of scheduled local notifications under different application states. Following the software reliability assessment principles defined in ISO/IEC 25010 [31], the notification success rate is calculated as:

$$Notification\ Success\ Rate = \frac{N_s}{N_t} \times 100\% \quad (1)$$

where N_s represents the number of successfully triggered notifications, and N_t represents the total number of scheduled notification tests.

Second, notification delay is used to measure the latency between the scheduled notification time and the actual notification display time. According to standard computer system performance analysis methodologies [32], notification delay is defined as:

$$Notification\ Delay = T_a - T_s \quad (2)$$

where T_a is the actual notification display time and T_s is the scheduled notification time. The average notification delay for each testing scenario is calculated as:

$$Average\ Delay = \frac{1}{n}\sum_{i=1}^{n}(T_{a,i} - T_{s,i}) \quad (3)$$

where n denotes the number of notification tests, $T_{a,i}$ is the actual notification display time of the i-th test, and $T_{s,i}$ is the scheduled notification time of the i-th test.

Third, JSON storage performance is evaluated by measuring the execution time required for read, write, and delete operations under different task data sizes. Following common software performance evaluation practices [32], the average processing time is calculated as:

$$Average\ Processing\ Time = \frac{1}{n}\sum_{i=1}^{n} T_i \quad (4)$$

where T_i denotes the processing time of the i-th test and n is the total number of repeated tests.

Finally, system resource consumption is evaluated using APK file size, installed storage usage, idle RAM usage, idle CPU usage, and battery consumption. Following smartphone energy consumption evaluation approaches [33], battery consumption rate is calculated as:

$$Battery\ Consumption\ Rate = \frac{B_S - B_e}{T} \quad (5)$$

where B_s is the battery percentage at the beginning of the test, B_e is the battery percentage at the end of the test, and T is the testing duration in hours.

These metrics provide objective and reproducible evidence for evaluating system reliability and performance, thereby reducing dependence on subjective assessments such as perceived stability or user feedback [31–33].

IV. RESULTS AND EVALUATION

A. System Implementation and Results

The proposed application was successfully implemented and deployed on Android smartphones. The system provides the main functions required for personal task management, including task creation, task browsing, task deletion, local JSON-based data persistence, and scheduled local notifications. When a user creates a task, the application stores the task content and reminder time in a local JSON file. When the scheduled time is reached, the system triggers an Android local notification through Unity's Mobile Notifications Package.

This section focuses on measurable system behavior and empirical evaluation results. The evaluation includes notification reliability, local storage performance, system resource consumption, and preliminary user observation. The implemented task management application interface is shown in Fig. 6.

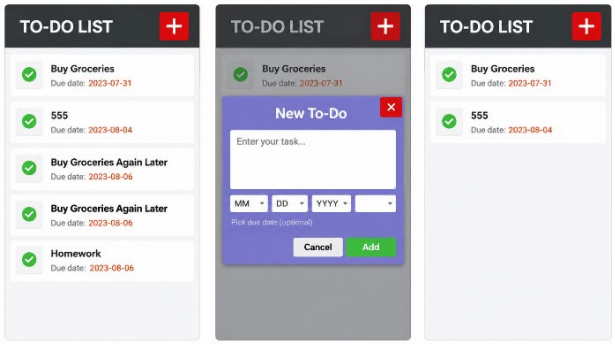


Fig. 6. Screenshot of the task management application interface.

B. Notification Reliability Test

To evaluate the reliability of the notification mechanism, 30 notification tests were conducted under each of the following scenarios: foreground execution, background execution, and screen-locked mode. Each test scheduled a local notification at a predefined time, and the actual notification trigger time was recorded manually. The notification delay was calculated as the difference between the scheduled time and the actual display time.

As shown in Table II, the notification mechanism achieved stable performance in the foreground scenario, with all 30 notifications successfully triggered. In the background execution scenario, 29 out of 30 notifications were successfully triggered, resulting in a success rate of 96.7%. Under the screen-locked condition, 28 out of 30 notifications were successfully triggered, with a success rate of 93.3%. The average delay increased from 0.42 s in the foreground state to 1.68 s in the background state and 3.21 s in the screen-locked state.

TABLE II. NOTIFICATION RELIABILITY TEST RESULTS

Scenario	Number of Tests	Successful Notifications	Success Rate	Average Delay
App in foreground	30	30	100.0%	0.42 s
App in background	30	29	96.7%	1.68 s
Screen locked	30	28	93.3%	3.21 s

The notification mechanism successfully delivered reminders across different application states. In particular, the foreground and background scenarios demonstrated high notification reliability. However, notification behavior may still be affected by Android power management mechanisms such as Doze Mode and App Standby. Therefore, the notification results should be interpreted within the tested device and operating system environment.

By reporting both notification success rate and average delay, the evaluation provides measurable evidence for assessing the reliability of the notification module.

C. JSON Storage Performance Test

To evaluate the performance of the JSON-based local storage mechanism, this study tested four task data sizes: 10, 50, 100, and 500 task records. For each data size, the system measured the average time required to write, read, and delete task data. The purpose of this test was to determine whether flat JSON storage is sufficient for small-scale personal task management.

TABLE III. JSON STORAGE PERFORMANCE TEST RESULTS

Number of Task Records	Average Write Time	Average Read Time	Average Delete Time	JSON File Size
10	1.8 ms	1.2 ms	1.6 ms	2.1 KB
50	3.7 ms	2.8 ms	3.4 ms	10.6 KB
100	6.4 ms	5.1 ms	5.9 ms	21.3 KB
500	18.6 ms	15.3 ms	17.2 ms	104.8 KB

As shown in Table III, the JSON file size increased from 2.1 KB for 10 task records to 104.8 KB for 500 task records. The average write time increased from 1.8 ms to 18.6 ms, while the average read time increased from 1.2 ms to 15.3 ms. The average delete time also increased as the number of task records increased, ranging from 1.6 ms for 10 records to 17.2 ms for 500 records.

The results indicate that JSON-based persistence is well suited to lightweight task management applications with relatively small datasets. For small and medium task sizes, the read and write times remained within an acceptable range for interactive mobile use. The growth in access time and file size at larger record counts, however, suggests that while JSON storage is easy to implement and appropriate for lightweight offline use, it may not be the most suitable solution for larger-scale task management systems.

Therefore, JSON persistence is interpreted as a lightweight storage approach for small-scale offline task management rather than a fully scalable data architecture. For future versions, SQLite or Realm would be more appropriate for improving data integrity, transaction safety, and scalability.

D. System Resource Consumption

Because Unity is primarily designed for 2D/3D interactive applications, its use in a standard task management application requires technical justification. Therefore, this study measured the basic resource consumption of the proposed Unity-based implementation, including APK file size, installed storage usage, idle RAM usage, idle CPU usage, and battery consumption.

As shown in Table IV, the APK file size of the proposed Unity application was 39.9 MB, and the installed storage usage on the Android device was 121 MB. During idle operation, the application used approximately 184 MB of RAM and maintained an average idle CPU usage of 1.8%. Battery consumption was approximately 3.2% per hour under the tested idle observation condition.

TABLE IV. RESOURCE CONSUMPTION RESULTS

Resource Metric	Measured Value
APK file size	39.9 MB
Installed storage usage	121 MB
Idle RAM usage	184 MB
Idle CPU usage	1.8%

These measurements show that the Unity-based implementation incurs a measurable runtime overhead relative to lightweight mobile application frameworks, yet Unity remains a feasible development environment for offline task management prototypes, while requiring careful consideration of runtime footprint and platform-specific notification behavior.

Although Unity may introduce higher resource consumption, it also provides cross-platform development capabilities, integrated UI management, and potential extensibility toward interactive 2D/3D or XR-based productivity interfaces. The choice of Unity therefore represents a deliberate technical trade-off, not a claim that it can universally replace native mobile development frameworks.

E. Preliminary Observational User Evaluation

Because the initial user evaluation was conducted through observation rather than a formal questionnaire, this study reports the results as a preliminary observational evaluation. The purpose of this evaluation was to examine whether users could complete the major task management operations and understand the visual task status indicators.

The observed operations included task creation, task browsing, task deletion, recognition of overdue tasks, and confirmation of notification scheduling. During the observation, users were able to complete the basic task flow without additional training. The red "!" indicator for overdue tasks and the green "O" indicator for active tasks were generally understood by users. Immediately after creating a task, several users also attempted to verify whether the reminder had been successfully scheduled, suggesting that clearer confirmation feedback may improve user confidence. The preliminary observational user evaluation results are summarized in Table V.

TABLE V. OBSERVATIONAL USER EVALUATION SUMMARY

Observed Item	Observation Result
Task creation	Users could complete the task creation process with minimal guidance.
Task browsing	Users could identify task content and scheduled time from the task list.
Task deletion	Users could remove existing tasks through the delete button.
Status recognition	Users generally understood the red and green visual indicators.
Notification confirmation	Some users attempted to confirm whether reminders were successfully scheduled.
Improvement suggestions	Search, task categorization, tagging, recurring tasks, and clearer confirmation feedback were suggested.

The observational results suggest that the proposed system provides an understandable interaction flow for basic task management. However, because this evaluation did not use a standardized questionnaire, the results should be interpreted as preliminary usability observations rather than statistically validated user evaluation results. To address this limitation, future work will administer a formal System Usability Scale (SUS) or Likert-scale questionnaire to a larger participant sample.

F. Summary of Evaluation Results

Overall, the evaluation results indicate that the proposed Unity-based offline task management application can support basic task creation, browsing, deletion, local storage, and scheduled notification functions. The notification test provides measurable evidence regarding reminder reliability, while the JSON performance test shows that the local storage mechanism is acceptable for lightweight task data. The resource consumption results also clarify the technical trade-off of using Unity for a non-gaming mobile application.

Compared with purely qualitative observation, the combined tests provide a more objective evidentiary basis for the feasibility claims of this study.

V. DISCUSSION AND LIMITATIONS

A. Limitations of JSON-Based Local Storage

The proposed system uses JSON-based local storage because the target application is a lightweight personal task management tool with a relatively small amount of data. As discussed in Section II, this approach is simple to implement, easy to inspect, and sufficient for basic task records.

However, this design also has several limitations. First, flat JSON files do not provide transaction safety. If the application is unexpectedly terminated during file writing, data corruption may occur. Second, JSON storage may not scale well when the number of task records increases significantly. Third, JSON does not provide built-in query optimization, indexing, or relational data management. Therefore, while JSON is acceptable for lightweight offline task management, it should not be considered a complete replacement for local databases such as SQLite or Realm.

In future work, the system can be extended by adopting SQLite or Realm to improve data integrity, transaction safety, and scalability. This would be especially important if the application were to support advanced functions such as task categorization, tagging, search, recurring tasks, and cross-device synchronization.

B. Android Notification Lifecycle Limitations

The proposed system uses Unity's Mobile Notifications Package to schedule local notifications on Android devices. This approach is suitable for basic reminder functions. However, Android imposes several background execution and battery optimization restrictions, including Doze Mode, App Standby, and notification channel management. These mechanisms may affect notification timing, especially when the device remains idle for an extended period.

In the current implementation, the notification function was tested under foreground, background, and screen-locked scenarios. However, it should be noted that complete notification persistence cannot be guaranteed under all Android system conditions. In particular, device reboot, aggressive RAM clearing, manufacturer-specific battery optimization, and exact alarm restrictions may affect scheduled notification behavior.

Future implementations may benefit from integrating Android native components such as *BroadcastReceiver*, *WorkManager*, *AlarmManager*, and boot-completed receivers to improve reminder persistence after device reboot or system-level process termination.

C. Security and Privacy Considerations

Local JSON storage may reduce dependence on external cloud transmission, but it should not be interpreted as a complete data security mechanism.

Plaintext JSON files do not provide encryption-level protection. If a device is rooted, compromised, or if local backups are extracted, the stored data may still be exposed. The current implementation is therefore a lightweight local storage design, not a fully secure storage solution.

Future work should incorporate encryption mechanisms, such as Android Keystore-based key management or encrypted local databases, to improve protection for sensitive task data.

D. Future Work

Based on the limitations identified in this study, several directions can be considered for future system improvement. First, the current JSON-based storage mechanism can be replaced with SQLite or Realm to improve data integrity, transaction safety, scalability, and query efficiency. This would be particularly useful when the number of task records increases or when advanced functions such as task categorization, tagging, and search are introduced. Second, the notification mechanism can be further improved by integrating Android native components such as *BroadcastReceiver*, *WorkManager*, *AlarmManager*, and boot-completed receivers.

Third, the security of local task data should be strengthened. As noted in the security discussion above, plaintext JSON files do not provide encryption-level protection. Future versions should consider Android Keystore-based key management or encrypted local databases to protect sensitive task information. Fourth, cloud synchronization can be added as an optional function. While the current system focuses on offline operation, optional cloud synchronization would allow users to access and back up their task data across multiple devices. This function should be designed carefully to balance usability, privacy, and data security.

Finally, future versions can support recurring tasks and intelligent reminder mechanisms. Recurring tasks would allow users to schedule daily, weekly, or monthly reminders, while intelligent reminder mechanisms could adjust notification timing based on user behavior, task urgency, or historical completion patterns. These extensions would make the system more practical for long-term task management.

VI. Conclusion

This paper presented and evaluated a Unity-based offline task management application that integrates JSON-based local persistence and Android local notification scheduling. Rather than proposing a new task management architecture, this work examined the feasibility and limitations of using Unity for a non-gaming mobile productivity application.

Under the tested Android environment, the system reliably supported the full task lifecycle: creation, browsing, deletion, local persistence, and scheduled notification. The notification reliability, JSON storage performance, and resource consumption measurements quantify its behavior under realistic usage conditions. In addition, the preliminary observational user evaluation suggests that users can understand the basic interaction flow and visual task status indicators, although a formal questionnaire-based usability study is still required for stronger statistical validation.

The study also identifies several limitations. JSON-based local storage is simple and suitable for lightweight task data, but it lacks transaction safety and scalability. Android notification behavior may be affected by Doze Mode, App Standby, device reboot, and manufacturer-specific battery optimization. In addition, plaintext JSON storage provides no encryption-level protection. These limitations suggest that future versions should consider SQLite or Realm for local persistence, Android native components for notification reliability, and encryption mechanisms for improved data protection. Overall, the findings provide practical insights into the feasibility of using Unity as a framework for non-gaming mobile productivity applications.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Yi-Heng Wu conducted system development and manuscript writing. Jing-Lun Wang assisted in implementation and testing. Lung-Fa Wu supervised the research and revised the manuscript. All authors approved the final version.

Funding

This work was supported by the Ministry of Education, Taiwan, through the Initiative for Advancing Faculty and Student Practical Competencies in Industry-Academia Colleges (Project Year 114).

References

[1] A. Alshehri, J. O. Alamari, and K. A. Almarhabi, "Assessing user concerns and preferences in the evolution of mobile health applications in Saudi Arabia," *Journal of Advanced Information Technology*, vol. 16, no. 4, pp. 499–509, 2025. doi: 10.12720/jait.16.4.499-509

[2] V. C. Vijay, K. R. S. Keerthivasan, G. Zhong, P. Kumar, and M. Zhao, "Real-time Artificial Intelligence (AI) and Augmented Reality (AR)-driven virtual saree dressing: Mirroring reality in fashion technology," *Journal of Advanced Information Technology*, vol. 16, no. 3, pp. 426–433, 2025. doi: 10.12720/jait.16.3.426-433

[3] W. P. Rey, E. A. G. Cudilla, and R. A. Verdida, "Performance peaks: Monitoring and optimizing systems for efficiency," *Journal of Advanced Information Technology*, vol. 16, no. 11, pp. 1595–1603, 2025. doi: 10.12720/jait.16.11.1595-1603

[4] W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba, "Taxonomy of cross-platform mobile applications development approaches," *Ain Shams Engineering Journal*, vol. 8, no. 2, pp. 163–190, 2017. doi: 10.1016/j.asej.2015.08.004

[5] D. Wang, K. B. A. Bakar, B. Isyaku, and L. Lei, "A conflict-reduced task-to-host matching scheme for task offloading in edge computing-enabled IoT network," *Journal of Advanced Information Technology*, vol. 16, no. 10, pp. 1449–1458, 2025. doi: 10.12720/jait.16.10.1449-1458

[6] I. M. Sukarsa, I. P. A. Bayupati, G. O. D. Ananda, P. W. Buana, and N. W. Wisswani, "An efficient synchronization model in random word educational games with multiplayer platforms," *Journal of Advanced Information Technology*, vol. 14, no. 5, pp. 897–906, 2023. doi: 10.12720/jait.14.5.897-906

[7] V. Bellotti, B. Dalal, N. Good, P. Flynn, D. G. Bobrow, and N. Ducheneaut, "What a to-do: Studies of task management towards the design of a personal task list manager," in *Proc. the SIGCHI Conference on Human Factors in Computing Systems*, 2004, pp. 735–742. doi: 10.1145/985692.985785

[8] M. Esposito, P.-L. Rocq, E. Novy, and R. Remen, "Smartphone to-do list application to improve workflow in an intensive care unit: A superiority quasi-experimental study," *International Journal of Medical Informatics*, vol. 136, 104085, 2020. doi: 10.1016/j.ijmedinf.2020.104085

[9] Y. Khaokaew, I. Holcombe-James, M. S. Rahaman *et al.*, "Imagining future digital assistants at work: A study of task management needs," *International Journal of Human-Computer Studies*, vol. 168, 102905, 2022. doi: 10.1016/j.ijhcs.2022.102905

[10] M. A. Nygaard, E. Rubínová, J. L. Allan, and L. H. Phillips, "Understanding older adults' experience of prospective memory errors and strategy use," *Applied Cognitive Psychology*, vol. 39, no. 3, e70066, 2025. doi: 10.1002/acp.70066

[11] E. C. Sanders and W. R. Boot, "Technology support preferences for prospective memory among older adults with subjective cognitive complaints," *Innovation in Aging*, vol. 9, no. 11, igaf107, 2025. doi: 10.1093/geroni/igaf107

[12] V. Baric, M. Andreassen, A. Öhman, and H. Hemmingsson, "Using an interactive digital calendar with mobile phone reminders by senior people—A focus group study," *BMC Geriatrics*, vol. 19, no. 1, 116, 2019. doi: 10.1186/s12877-019-1128-9

[13] M. Andreassen, H. Danielsson, H. Hemmingsson, and T. Jaarsma, "An interactive digital calendar with mobile phone reminders (RemindMe) for people with cognitive impairment: A pilot randomized controlled trial," *Scandinavian Journal of Occupational Therapy*, vol. 29, no. 4, pp. 270–281, 2022. doi: 10.1080/11038128.2021.1917654

[14] M. El Haj, K. Gallouj, P. Antoine, and G. Chapelet, "Prospective memory in mobile: Using smartphone-based calendars to rehabilitate prospective memory in patients with Alzheimer's disease," *Current Alzheimer Research*, vol. 21, no. 10, pp. 755–763, 2024. doi: 10.2174/0115672050369534250127104933

[15] M. K. Scullin, W. E. Jones, R. Phenis *et al.*, "Using smartphone technology to improve prospective memory functioning: A randomized controlled trial," *Journal of the American Geriatrics Society*, vol. 70, no. 2, pp. 459–469, 2022. doi: 10.1111/jgs.17551

[16] R. N. Brewer, M. R. Morris, and S. Lindley, "How to remember what to remember: Exploring possibilities for digital reminder systems," in *Proc. the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 3, pp. 1–20, 2017. doi: 10.1145/3130903.

[17] A. Sriranganathan, S. Kathiravelu, T. Li *et al*., "Digital planning-based technologies to support memory-related functioning in older adults with mild cognitive impairment: A systematic scoping study," *Journal of Ageing and Longevity*, vol. 5, no. 4, 42, 2025. doi: 10.3390/jal5040042

[18] M. F. Torino, M. N. Fabro, C. K. M. Pastrana, and M. M. V. Gatdula, "MañanaList app: A mobile to-do list application for efficient and personalized task management," *International Journal of Research and Innovation in Social Science*, vol. 9, no. 9, pp. 2507–2523, 2025.

[19] S. Shama, S. H. M. Shivakumar, S. S. Poojary, S. Kumar, and A. L. Kotian, "Review on existing note-taking and task-management apps," *International Journal of Advanced Computer Science and Technology*, vol. 13, no. 2, pp. 59–64, 2024. doi: 10.30534/ijacst/2024/021322024

[20] C. Escobar-Velásquez, A. Mazuera-Rozo, C. Bedoya, M. Osorio-Riaño, M. Linares-Vásquez, and G. Bavota, "Studying eventual connectivity issues in Android apps," *Empir. Software Eng.*, vol. 27, 22, 2022.

[21] Android Developers. (2026). Build an offline-first app. *Android Developers Documentation*. [Online]. Available: https://developer.android.com/topic/architecture/data-layer/offline-first

[22] Android Developers. (2026). Data layer. *Android Developers Documentation*. [Online]. Available: https://developer.android.com/topic/architecture/data-layer

[23] Z. Song, J. Pu, J. Cheng, and E. Tilevich, "Performance and programming effort trade-offs of Android persistence frameworks," *Journal of Systems and Software*, vol. 146, pp. 99–111 2018.

[24] SQLite. (2026). Atomic commit in SQLite. *SQLite Documentation*. [Online]. Available: https://sqlite.org/atomiccommit.html

[25] A. Biørn-Hansen, C. Rieger, T.-M. Grønli, T. A. Majchrzak, and G. Ghinea, "An empirical investigation of performance overhead in cross-platform mobile development frameworks," *Empirical Software Engineering*, vol. 25, pp. 2997–3040, 2020. doi: 10.1007/s10664-020-09827-6

[26] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," arXiv preprint, arXiv:1809.02627, 2018.

[27] Unity Technologies. (2026). Mobile notifications package. *Unity Documentation*. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.mobile.notifications@latest/

[28] Google. (2026). Learn about Google Tasks. *Google Tasks Help*. [Online]. Available: https://support.google.com/tasks/answer/7675772

[29] Microsoft. (2026). To Do list and task management app. *Microsoft To Do*. [Online]. Available: https://www.microsoft.com/en-us/microsoft-365/microsoft-to-do-list-app

[30] Todoist. (2026). Use Todoist while offline. *Todoist Help Center*. [Online]. Available: https://www.todoist.com/help/articles/use-todoist-while-offline-4rbaZw

[31] International Organization for Standardization, *Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models*, ISO, Geneva, Switzerland, 2011.

[32] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, New York, NY, USA: John Wiley & Sons, 1991.

[33] A. Pathak, Y. C. Hu, and M. Zhang, "Fine-grained power modeling for smartphones using system call tracing," in *Proc. 10th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012, pp. 153–168.

Copyright © 2026 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited (CC BY 4.0).