A Mobile Application for AI Visual Creation

"LocalCanvas" APP

Software Design Document

Project Members:

Kaile Huang 20233801099
Yupeng Wang 20233801067
Siyuan Yang 20233801009
Yijie Shao 20233801041

School of Artificial Intelligence and Data Science
South China Normal University – University of Aberdeen
Intelligent Application Development Course
December 2025

Contents

1 Introduction

1.1 Project Background

With the rapid development of generative artificial intelligence, AI-based image creation and editing tools have become increasingly popular. However, most existing solutions are designed for desktop environments and rely heavily on text-based prompts, which often leads to limited controllability and poor usability on mobile devices. In particular, users face difficulties in precisely specifying editing regions, managing long-running generation tasks, and maintaining a smooth user experience under mobile constraints such as limited memory, unstable network conditions, and fragmented usage scenarios.

1.2 Project Objectives

To address these challenges, this project presents **LocalCanvas**, a mobile AI-based visual creation application developed for the Android platform. LocalCanvas focuses on enabling controllable and intuitive image generation through touch-based interaction. By allowing users to directly define editing regions via masking and configure AI generation workflows in a structured manner, the application bridges the gap between powerful AI generation capabilities and practical mobile usability.

The primary goal of this project is to design and implement a complete end-to-end mobile application that supports image input, region selection, AI-driven generation, result preview, and local persistence. The project emphasizes usability, controllability, and robustness in real mobile usage contexts.

1.3 Project Scope and Limitations

The scope of the project includes user interface design, system architecture, asynchronous task management, data persistence, and cross-version compatibility on Android devices. The application integrates mobile-side interaction (e.g., masking), structured workflow configuration, remote AI generation service invocation, and local result management.

Rather than training AI models, the project concentrates on system integration, interaction design, and engineering robustness. Model training and backend service implementation are considered outside the scope of this work.

Figure 1 illustrates the overall concept and usage scenario of the LocalCanvas application, highlighting the main interaction flow from image selection to result generation and management.

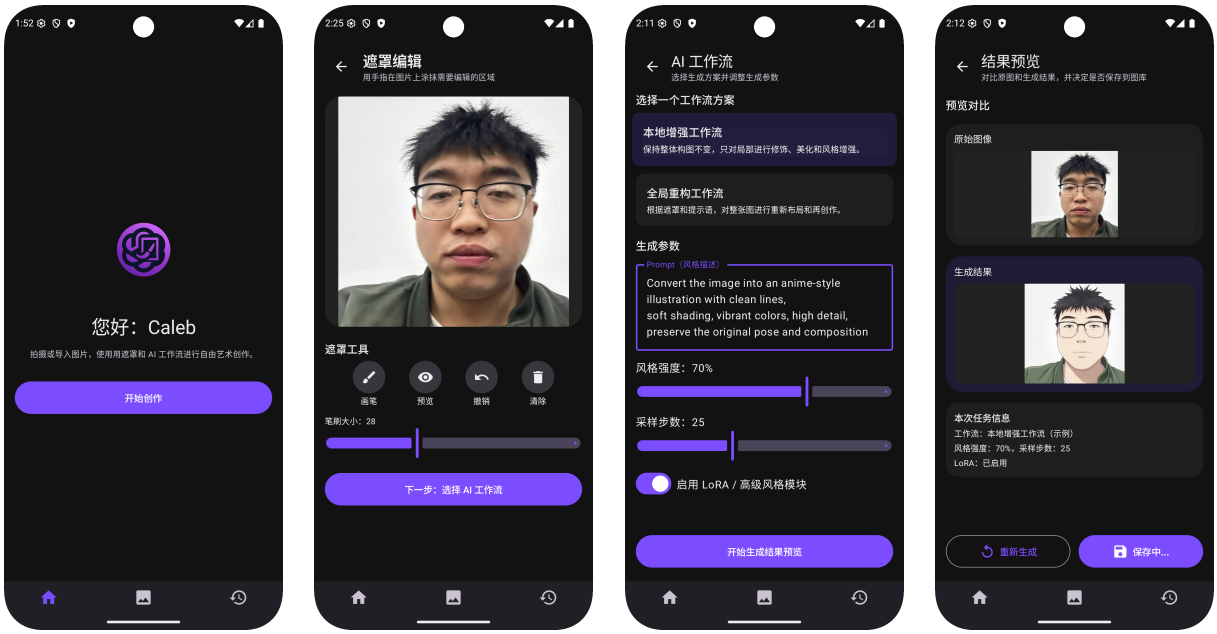


Figure 1: Overview of the LocalCanvas mobile application workflow, from initial image input and mask editing to AI workflow configuration and result generation.

2 Requirements Analysis

This section provides a detailed analysis of the requirements of the LocalCanvas application. The requirements are derived from realistic usage scenarios observed in mobile AI image editing contexts and are further refined through an examination of the project goals and system constraints. By distinguishing between user needs, functional requirements, and non-functional requirements, this section establishes a clear and traceable foundation for the subsequent system design and implementation.

2.1 User Needs Analysis

LocalCanvas is primarily designed for mobile users who intend to perform AI-assisted image creation and editing in everyday scenarios. Compared with professional desktop users, mobile users typically operate in fragmented time periods and expect interactions to be direct, intuitive, and responsive. Based on these assumptions, several core user needs can be identified.

First, users expect the application to support simple and efficient image input, allowing them to select images directly from the local gallery without complex preprocessing steps. In addition, a key expectation is the ability to precisely specify which regions of an image should be affected by AI generation. For this reason, touch-based masking is considered essential, as it aligns naturally with mobile interaction habits. Furthermore, users generally prefer configurable yet understandable AI generation settings, where meaningful control can be achieved without requiring technical expertise. Finally, users value the ability to preview results, compare outputs, and revisit previously generated images,

especially when AI generation involves noticeable latency or multiple attempts.

These user needs are summarized in Table 1, which serves as an abstract representation of user expectations rather than an exhaustive behavioral model.

Table 1: User Needs of the LocalCanvas Application

ID	User Need Description
UN-1	The user wants to select or import images easily on a mobile device.
UN-2	The user wants to precisely specify editing regions using touch gestures.
UN-3	The user wants to configure AI generation settings in an intuitive manner.
UN-4	The user wants to preview and compare generated results conveniently.
UN-5	The user wants to save and revisit generated images at any time.
UN-6	The user wants the application to remain usable under unstable network conditions.

2.2 Functional Requirements

Functional requirements translate user needs into concrete system capabilities. They describe the specific behaviors that the system must exhibit in order to support the intended usage scenarios. In the context of LocalCanvas, functional requirements focus on image handling, interaction support, AI service integration, and task management.

At a minimum, the system must allow users to select images from the local device and present them in an editable format. Once an image is loaded, the system should provide interactive masking tools that enable users to define regions of interest through touch input. Based on the defined mask and user-specified parameters, the system is expected to construct structured requests and invoke external AI generation services. Since AI generation may take a noticeable amount of time, the system must manage these operations asynchronously and provide appropriate feedback to users. After generation, the system should display the results clearly and store them locally to support later access and comparison. In addition, the system must be able to handle failures, such as network interruptions or service delays, without causing data loss or application crashes.

Table 2 summarizes the main functional requirements identified for the LocalCanvas application.

Table 2: Functional Requirements

Category	Requirement Description
Image Input	The system shall allow users to select images from the local gallery.
Mask Editing	The system shall support touch-based region masking and editing.
AI Generation	The system shall generate images based on user-defined masks and parameters.
Task Management	The system shall manage asynchronous AI generation tasks.
Result Display	The system shall display original and generated images for comparison.
Data Persistence	The system shall store generated results and task history locally.
Error Handling	The system shall handle network delays and generation failures gracefully.

2.3 Traceability Matrix

The following table outlines the traceability between user needs (UN) and functional requirements (FR). Each user need is mapped to the corresponding functional requirements to ensure that all user needs are addressed by the system.

Table 3: Traceability Matrix: User Needs to Functional Requirements

User Need (UN)	**Mapped Functional Requirements (FR)**	**Description**
UN-1: The user wants to generate an image using a given prompt and a starting image.	FR-1: The system shall accept user input for a prompt and an image.	This requirement addresses the basic functionality of input handling for the image generation task.
UN-2: The user wants to modify the generated image by applying masks.	FR-3: The system shall allow users to draw masks on the image.	This requirement ensures that users can interact with the image and apply masks for editing.
UN-3: The user wants to see a response time that is quick and responsive when interacting with the system.	FR-4: The system shall generate the final image within 15 seconds of submission.	This functional requirement ensures that the performance criteria related to system responsiveness are met.
UN-4: The user needs the system to be compatible across different devices.	FR-5: The system shall support Android 8.0 and above.	This requirement ensures that the app is compatible with a wide range of devices.
UN-5: The user needs to know that their personal data is secure.	FR-6: The system shall ensure that data is encrypted during transmission and only stored temporarily.	This functional requirement addresses data privacy and security concerns.

2.4 Non-Functional Requirements

In addition to functional correctness, the quality attributes of the system play a critical role in determining its overall usability and reliability. Given that LocalCanvas operates on mobile devices, non-functional requirements related to performance, usability, stability, and compatibility are particularly important.

From a performance perspective, the system should provide smooth and responsive interaction during mask editing and image preview. Touch operations should not introduce noticeable lag, and visual feedback should remain consistent even when processing

large images. In terms of reliability, the application should remain stable during long-running AI generation tasks and be capable of recovering gracefully from interruptions caused by network instability, backgrounding, or configuration changes. Compatibility is another important consideration: the application should function correctly across multiple Android versions and device configurations, minimizing behavior differences caused by hardware diversity. Taken together, these non-functional requirements ensure that the system not only works as intended, but also delivers a practical and dependable user experience in real-world mobile environments.

The non-functional requirements of the application focus on performance, reliability, compatibility, and security to ensure a stable and user-friendly experience.

Category	Requirement	Description and Target
Performance	UI Responsiveness	The mask drawing interface should respond to user input with a rendering latency of less than 100 ms to ensure smooth and continuous interaction.
Performance	Image Generation Latency	The average response time for AI-based image generation should be within 15 seconds under stable network conditions.
Reliability	Task Recovery	The system should allow users to retry failed image generation tasks caused by network interruptions without restarting the application.
Reliability	System Stability	The application should maintain a crash rate below 1% during normal usage based on local testing and cloud-based testing results.
Compatibility	Android Version Support	The application should support Android 8.0 (API level 26) and above to cover the majority of active Android devices.
Compatibility	Screen Adaptability	The user interface should adapt correctly to different screen sizes and resolutions without layout distortion.
Security and Privacy	Data Transmission	Only necessary data, including user-provided images, masks, and text prompts, should be transmitted to the external AI service.
Security and Privacy	Local Data Storage	User data should not be permanently stored on the server side and should only be temporarily cached locally during the application session.
Security and Privacy	User Consent	The application should clearly inform users about data usage and obtain explicit user consent before uploading any content to external AI services.

3 Overall System Design

This section presents the overall design of the LocalCanvas application from a system-level perspective. The design emphasizes modularity, clarity of responsibility, and robustness

under mobile constraints. By organizing the application into distinct layers and well-defined functional modules, the system aims to support complex AI-driven workflows while maintaining maintainability and extensibility.

3.1 System Architecture Overview

The architecture of LocalCanvas follows a layered design pattern commonly adopted in modern Android applications. At a high level, the system can be divided into three main layers: the presentation layer, the data and logic layer, and the external service layer. Each layer encapsulates a specific set of responsibilities, reducing coupling and improving overall system stability.

The presentation layer is responsible for user interaction and visual rendering. It is implemented using Jetpack Compose and handles screen layout, navigation, and real-time interaction feedback, such as touch-based masking and progress indicators. The data and logic layer serves as the core of the application, coordinating user actions, managing application state, and orchestrating asynchronous tasks. The external service layer includes AI generation services and platform-level components such as local storage and media access.

Figure 2 illustrates the high-level system architecture of LocalCanvas and the relationships between its main components.

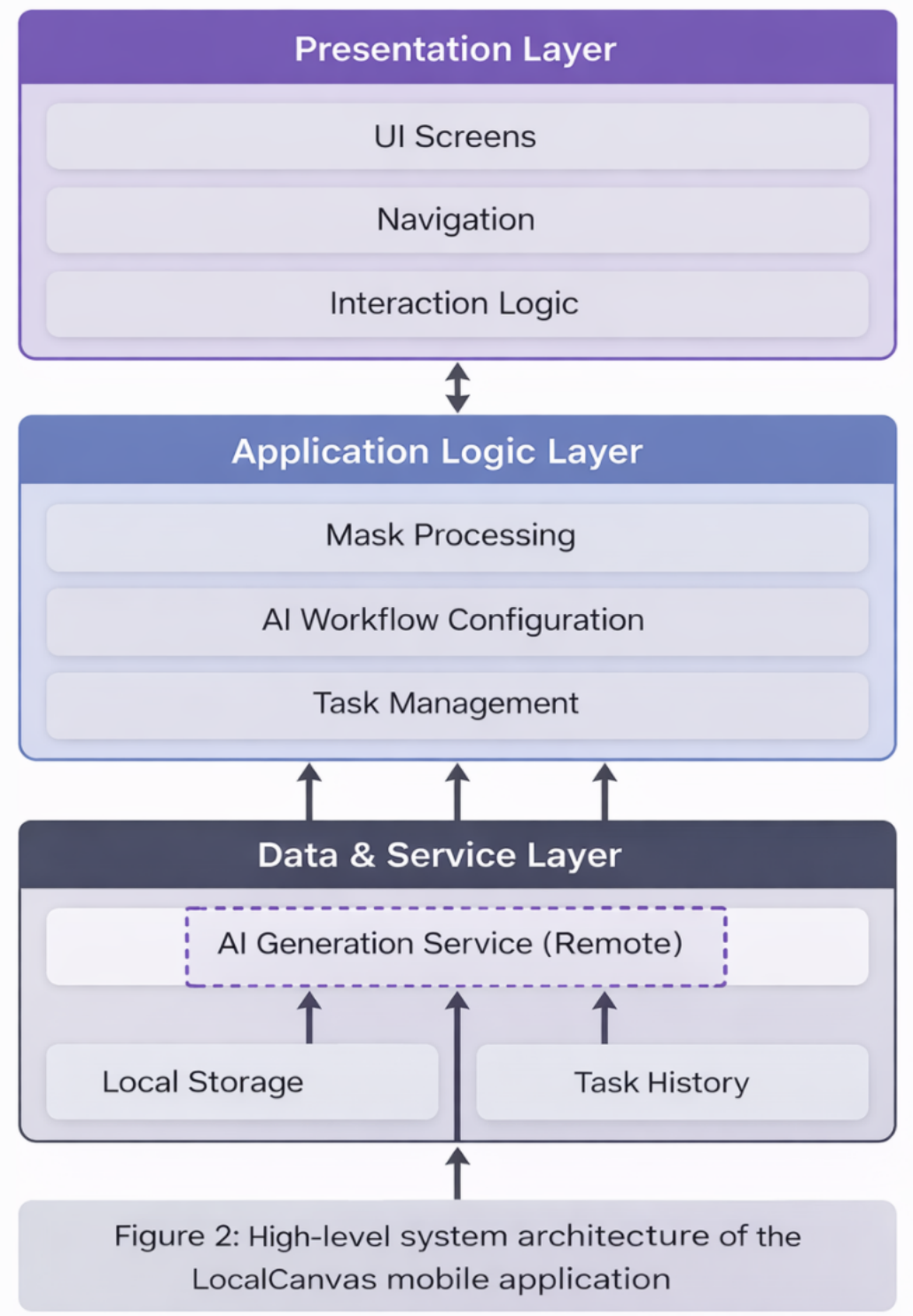


Figure 2: High-level system architecture of the LocalCanvas application.

This layered architecture allows the application to evolve independently at different levels. For example, changes in AI service providers or storage mechanisms can be accommodated with minimal impact on the user interface, while UI refinements can be introduced without altering core task management logic.

3.2 Remote AI Service Communication

The system communicates with the remote AI generation service using a structured request format. Two common methods are used for data transmission: 1. **JSON Format**: The request body includes the prompt, image, and optional mask in Base64-encoded format. 2. **Multipart Format**: Used for file uploads, where images and masks are sent as separate files.

```
POST /v1/images/generations
Content-Type: multipart/form-data
Authorization: Bearer <API_KEY>

--boundary
Content-Disposition: form-data; name="image"; filename="input.png"
Content-Type: image/png

<binary-data-of-image>

--boundary
Content-Disposition: form-data; name="mask"; filename="mask.png"
Content-Type: image/png

<binary-data-of-mask>

--boundary
Content-Disposition: form-data; name="prompt"
A futuristic cityscape at sunset
```

3.3 Application Workflow Design

From the user's perspective, LocalCanvas is designed around a clear and linear creative workflow. The workflow begins with image input and proceeds through region selection, AI configuration, generation execution, and result management. Each step is explicitly represented in the interface, reducing ambiguity and helping users maintain a sense of control throughout the process.

When a user initiates a new task, an image is either captured or imported from the local gallery. The user then enters the mask editing stage, where touch gestures are used to define regions of interest. After confirming the mask, the user configures the AI workflow by selecting generation modes and adjusting parameters. The system then submits the task for AI generation and provides feedback on task progress. Once generation is complete, results can be previewed, saved, or regenerated.

Figure 3 presents the end-to-end application workflow, highlighting the sequential relationship between major interaction stages.

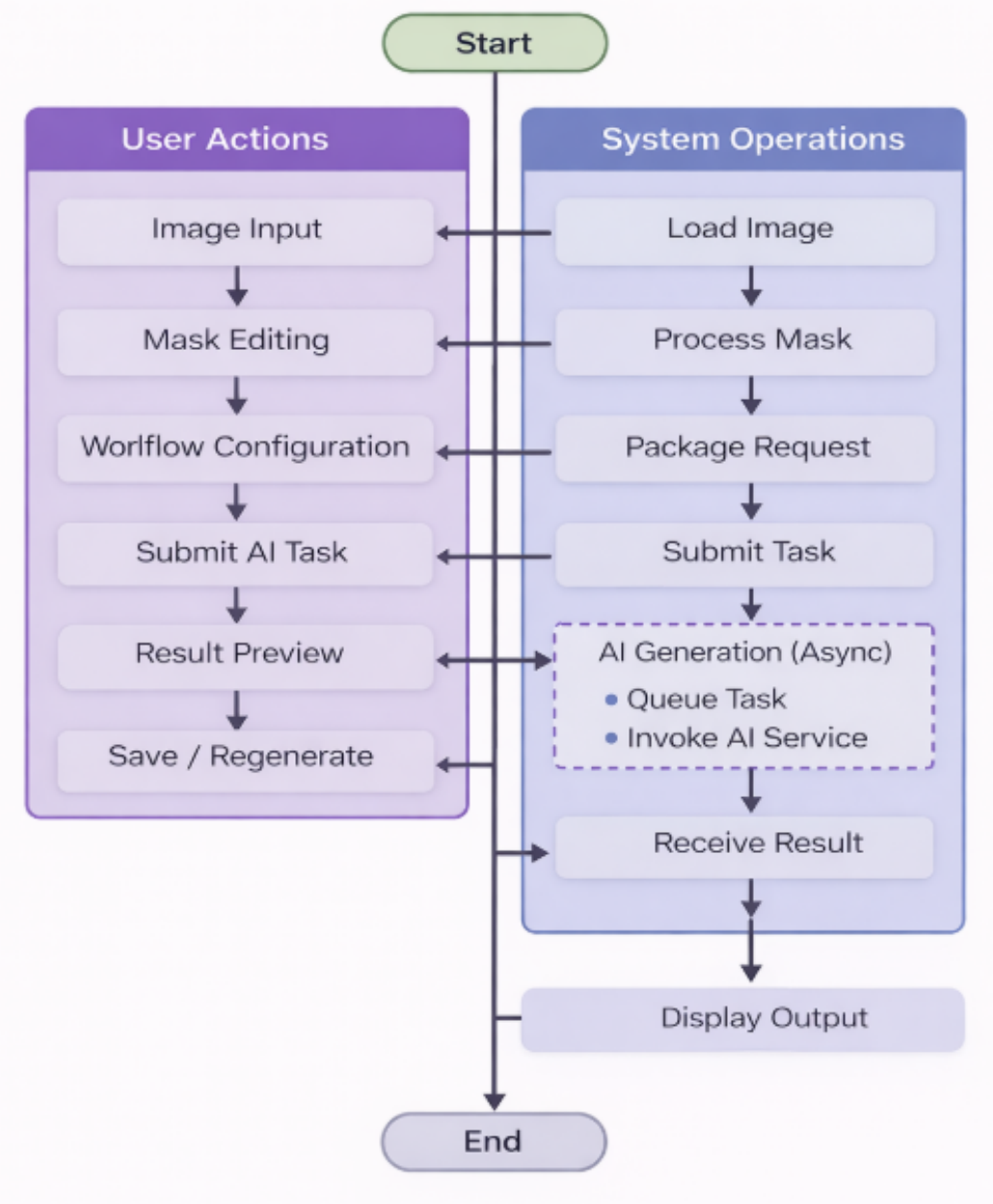


Figure 3: End-to-end workflow of image generation in the LocalCanvas application.

This workflow-oriented design ensures that users are guided step by step, while still allowing flexibility for iteration and refinement. By structuring the creative process explicitly, the system reduces cognitive load and supports efficient task completion, even in fragmented mobile usage scenarios.

3.4 Task State Machine Design

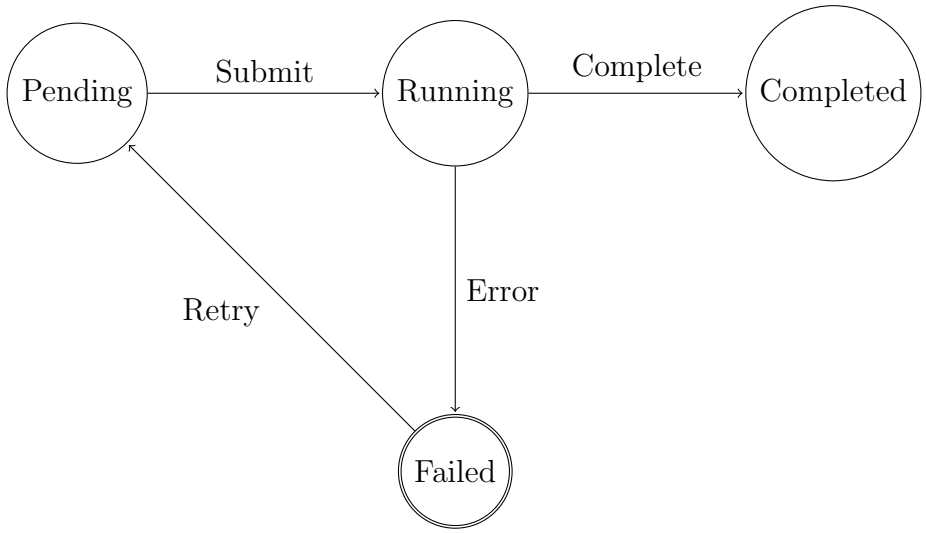
The system uses a state machine to manage the lifecycle of image generation tasks. The task transitions between four states: Pending, Running, Completed, and Failed.

State Transitions:

- **Pending → Running**: Task submitted for processing.

- **Running → Completed**: Task completed successfully.
- **Running → Failed**: Task failed due to timeout or network error.
- **Failed → Pending**: Retry task after user approval or automatic retry.

State Transition Diagram:

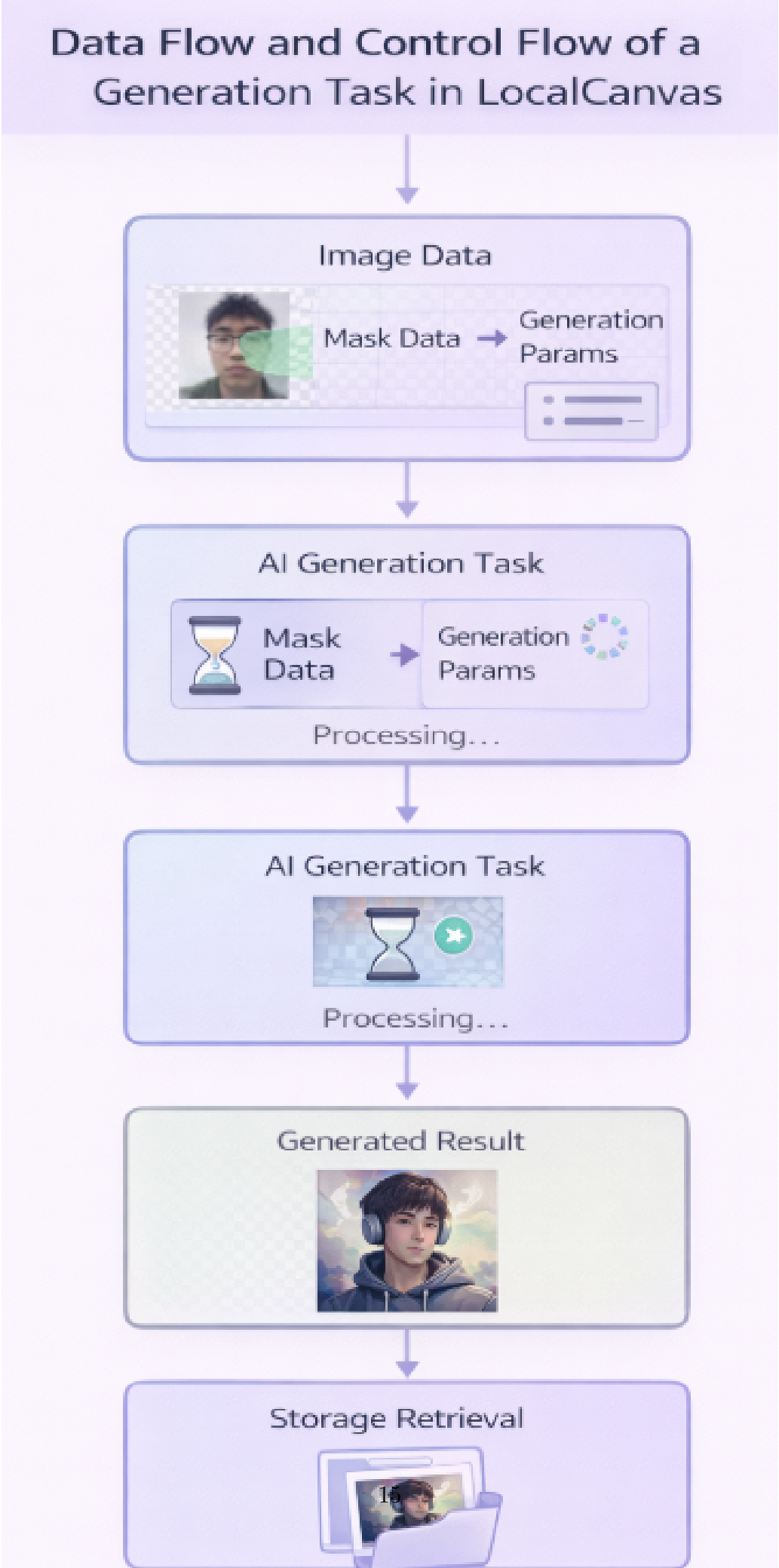


3.5 Data Flow and Control Flow

In addition to user-visible workflows, the internal data flow and control flow play a critical role in ensuring system reliability. Data in LocalCanvas primarily consists of image resources, mask data, generation parameters, task metadata, and generated results. These data elements flow through the system in a controlled and traceable manner.

When an image is selected, it is first loaded into memory for display and interaction. Mask data generated through user input is stored separately and later combined with generation parameters to form a structured AI request. During AI generation, task metadata such as status and progress are continuously updated and persisted locally. Upon completion, the generated image and associated parameters are stored and made available for later access.

Figure 4 illustrates the data flow and control flow within the system, showing how user actions, asynchronous tasks, and persistent storage interact during the lifecycle of a generation task.


Data Flow and Control Flow of a Generation Task in LocalCanvas
Image Data
Mask Data
Generation Params
AI Generation Task
Mask Data
Generation Params
Processing...
AI Generation Task
Processing...
Generated Result
Storage Retrieval

By clearly separating data responsibilities and maintaining explicit control flow, the system can handle interruptions, retries, and recovery scenarios more effectively. This design is particularly important for mobile environments, where network conditions and application lifecycle events are inherently unpredictable.

4 Module Design and Implementation

This section describes the design and implementation of the main functional modules in the LocalCanvas application. Each module is designed with a clear responsibility and aligns closely with the overall workflow introduced in Section 3. Together, these modules form a cohesive and extensible system that supports the complete image generation lifecycle on mobile devices.

4.1 Home and Navigation Module

The Home and Navigation module serves as the entry point of the application and is responsible for guiding users into the core creative workflow. Upon launching the application, users are presented with a clean and focused home screen that emphasizes task initiation rather than configuration complexity. From this screen, users can start a new creation task by capturing an image, importing an existing image, or accessing previously generated content.

Navigation within the application is implemented using a bottom navigation structure combined with contextual screen transitions. This design ensures that core functions such as creation, gallery browsing, and task history remain easily accessible while preserving a clear separation between different stages of the workflow.

Figure 5 illustrates the main home interface and navigation structure of the application.

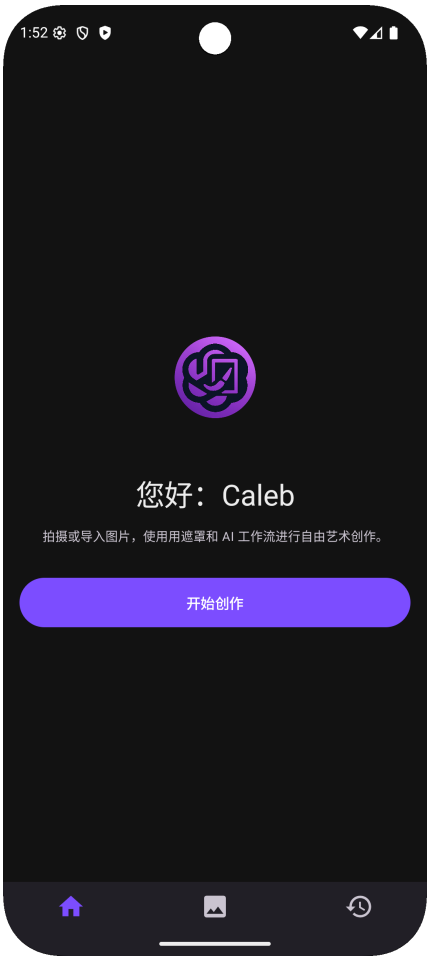


Figure 5: Home screen and navigation structure of the LocalCanvas application.

4.2 Mask Editing Module

The Mask Editing module is one of the core components of LocalCanvas and plays a crucial role in enabling controllable AI generation. This module allows users to define specific regions of an image through direct touch interaction, thereby explicitly indicating where AI processing should be applied.

The interface supports freehand drawing with adjustable brush size and transparency, as well as operations such as undo, clear, and preview. These features are designed to accommodate both coarse selections and fine-grained adjustments. By providing immediate visual feedback during the masking process, the system helps users verify their intent before proceeding to AI generation.

Figure 6 shows the mask editing interface, highlighting the touch-based interaction design and real-time visual feedback.

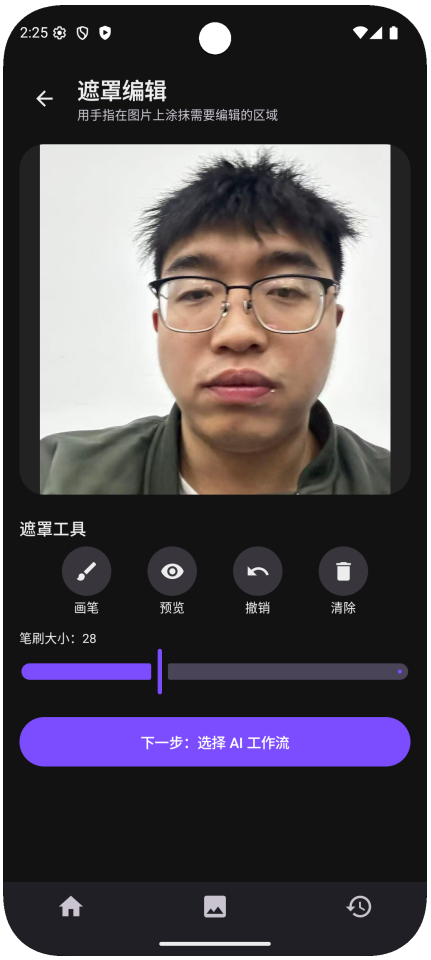


Figure 6: Mask editing module with touch-based region selection.

4.3 AI Workflow and Generation Module

After defining the mask, users enter the AI Workflow and Generation module, where generation parameters are configured and tasks are submitted for processing. This module is designed to balance flexibility and usability by exposing essential parameters while avoiding unnecessary technical complexity.

Users can select different generation modes, adjust style strength, and edit prompt text through a structured interface. All configuration options are presented in a visual and interpretable manner, allowing users to understand the impact of their choices. Once confirmed, the system packages the mask data and parameters into a structured request and initiates an asynchronous AI generation task.

Figure 7 illustrates the AI workflow configuration interface and parameter settings.

Figure 7: AI workflow configuration and generation module.

4.4 Failure Handling and Retry Strategy

Given that network connectivity can fluctuate, the application employs a retry strategy for failed tasks. The system will retry the request up to 3 times if a failure occurs due to a network issue or timeout.

Pseudo-code for Retry Logic:

```
fun generateImageWithRetry(request: ImageGenerationRequest): ImageGenerationResponse
    var retries = 0
    var result: ImageGenerationResponse? = null
    while (retries < 3) {
        try {
            result = aiService.generateImage(request)
            return result
        } catch (e: NetworkException) {
            retries++
            if (retries >= 3) {
                throw ImageGenerationFailedException("Max retry attempts reached")
            }
```

```
                Thread.sleep(2000)  // Wait before retrying
            }
        }
        return result!!
    }
```

4.5 Result Display and Gallery Module

The Result Display and Gallery module is responsible for presenting generated outputs and supporting iterative refinement. After a generation task completes, the system displays the original image alongside the generated result, enabling direct visual comparison. This side-by-side presentation helps users quickly evaluate the effectiveness of the applied mask and generation parameters.

Users can choose to save the generated image, regenerate results with modified parameters, or return to previous steps. Saved results are organized within the gallery, where they can be browsed and revisited later. Each result is associated with its corresponding parameters, supporting traceability and reproducibility.

Figure 8 shows the result display interface and gallery view.

Figure 8: Result display and gallery module for managing generated images.

4.6 Task Management and History Module

The Task Management and History module provides transparency and robustness for long-running AI generation tasks. Since generation may take a noticeable amount of time, the system maintains a task list that records task status, progress, and outcomes. This design allows users to monitor ongoing tasks and revisit completed or failed tasks without losing context.

Task metadata, including generation parameters and timestamps, are persisted locally. This enables the application to recover task information after interruptions and supports weak-network usage scenarios. By exposing task history to users, the system enhances trust and reinforces the sense of control over the creative process.

Figure 9 illustrates the task history interface and status tracking mechanism.

Figure 9: Task management and history module with status tracking.

4.7 Application Logo and Visual Identity

In addition to functional modules, visual identity plays an important role in communicating the design philosophy of LocalCanvas. The application logo is designed as a symbolic representation of the system's core concept: combining AI-driven generation with human-controlled creativity on mobile devices.

The logo integrates visual elements inspired by a mobile device frame and a brush-like gesture, emphasizing that the application places creative control directly in the user's hands. At the same time, abstract geometric patterns are used to suggest the presence of an intelligent AI engine operating beneath the interface. This combination reflects the project's intention to balance automation and user agency rather than replacing human input entirely.

Figure 10 presents the LocalCanvas application logo and its visual composition.

Figure 10: LocalCanvas application logo and visual identity design.

5 User Interface and Interaction Design

This section focuses on the design of the user interface and interaction mechanisms in the LocalCanvas application. Given the complexity of AI-assisted image generation, the interface is designed to reduce cognitive load, guide users through the workflow, and provide continuous visual feedback. The overall design emphasizes clarity, consistency, and direct manipulation, which are particularly important in mobile usage contexts.

5.1 UI Design Principles

The user interface of LocalCanvas follows several core design principles. First, simplicity is prioritized to ensure that users can quickly understand the available actions without extensive learning. Non-essential controls are minimized, and advanced options are revealed progressively only when needed. Second, consistency is maintained across screens through uniform layout patterns, icon styles, and color schemes, allowing users to transfer knowledge from one part of the application to another. Finally, feedback and visibility are emphasized so that users are always aware of the current system state, especially during long-running AI generation tasks.

In addition, the interface adopts a dark-themed visual style, which not only aligns with contemporary mobile design trends but also reduces visual fatigue during prolonged use. The dark background further enhances the contrast of images and masks, making visual evaluation more comfortable and accurate.

Figure 11 presents representative interface elements that reflect these design principles.

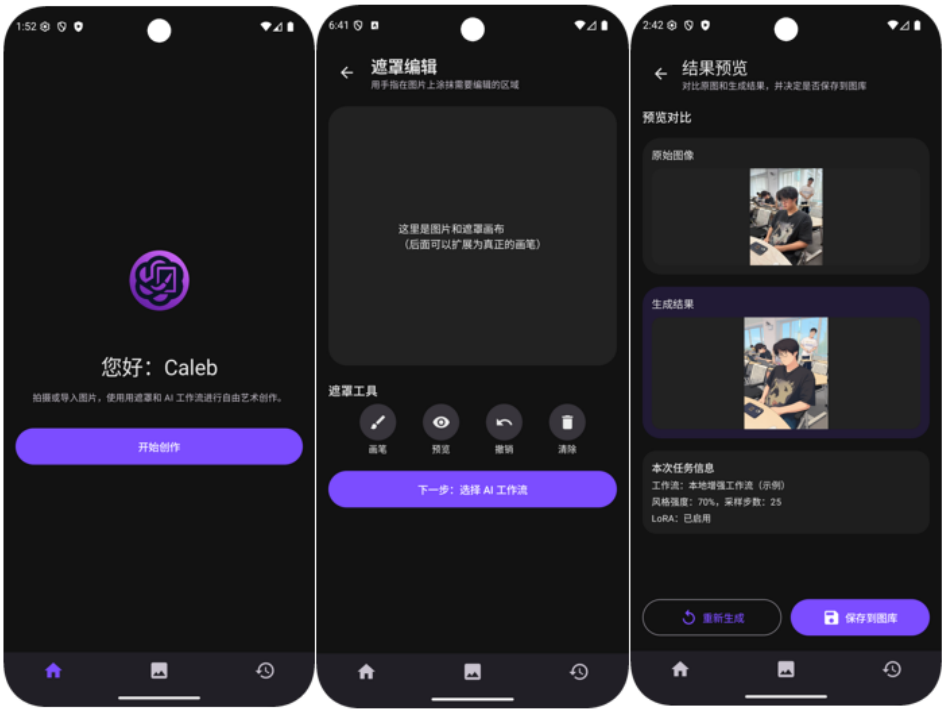


Figure 11: Representative interface elements illustrating the UI design principles of LocalCanvas.

5.2 Screen Layout and Navigation Flow

The screen layout of LocalCanvas is organized around a clear hierarchical structure that mirrors the application workflow. Major functional areas, such as creation, gallery, and task history, are separated into distinct screens while remaining accessible through intuitive navigation mechanisms. A bottom navigation bar is used to provide persistent access to high-level functions, reducing the need for deep navigation hierarchies.

Within each screen, content is arranged to follow natural reading and interaction patterns, typically from top to bottom. Primary actions are positioned prominently, while secondary actions are placed in contextual menus or secondary panels. This layout strategy helps prevent accidental operations and keeps the user's attention focused on the current task.

Figure 12 illustrates the overall navigation flow between major screens in the application.

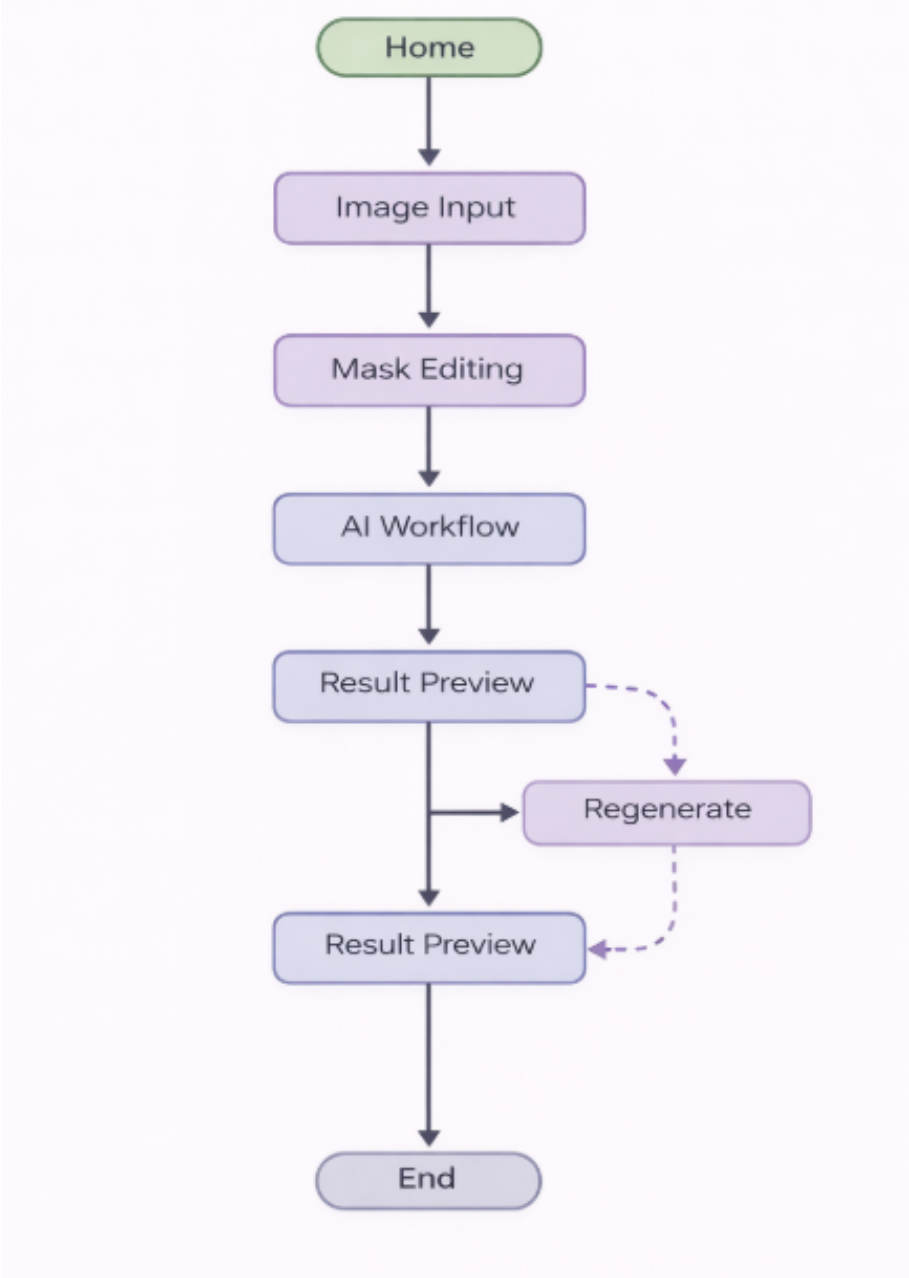


Figure 12: Navigation flow between major screens in the LocalCanvas application.

5.3 Key Interaction Design

Interaction design is a central aspect of LocalCanvas, particularly in modules that require precise user input and continuous feedback. The most representative interaction is touch-based masking, where users directly manipulate the image surface to define regions of interest. This direct manipulation approach aligns closely with human intuition and avoids the abstraction typically associated with text-based input.

During mask editing, the system provides immediate visual feedback by overlaying the mask on the image in real time. Users can adjust brush size, undo recent actions, and preview the mask before confirming. Similarly, during AI generation, progress indicators and status messages are displayed to inform users of ongoing operations, reducing

uncertainty and perceived waiting time.

Figure 13 shows an example of key interaction sequences, including mask drawing and generation feedback.

Figure 13: Key interaction examples, including touch-based masking and generation feedback.

Through these interaction mechanisms, LocalCanvas aims to create a balanced experience in which advanced AI capabilities remain accessible and controllable. By integrating interaction design closely with system feedback, the application supports efficient and confident user operation even in complex creative tasks.

6 Key Technologies and Engineering Challenges

This section introduces the key technologies adopted in the LocalCanvas application and discusses the main engineering challenges encountered during development. As a mobile AI-based image creation system, LocalCanvas must address not only functional correctness but also performance, robustness, and usability under real-world mobile constraints. The selected technologies and design decisions reflect a balance between practicality and extensibility.

6.1 Technology Stack Overview

LocalCanvas is developed primarily using Kotlin on the Android platform, leveraging modern Android development frameworks to improve maintainability and user experience. The user interface is implemented using Jetpack Compose, which enables a declarative approach to UI construction and simplifies state-driven rendering. This choice allows the interface to respond naturally to changes in application state, such as task progress updates and result availability.

Navigation between screens is managed using the Navigation Compose framework, ensuring a clear separation of navigation logic from UI components. Asynchronous operations, including AI generation requests and file I/O, are handled using Kotlin Coroutines, which provide a lightweight and structured mechanism for concurrency. This approach helps prevent UI blocking while maintaining readable and manageable code.

For data handling, the application adopts a repository-based design pattern to abstract interactions with external AI services and local storage. Image resources and generated results are persisted using Android's storage APIs, ensuring compatibility across different Android versions. Together, these technologies form a cohesive stack that supports responsive interaction and robust task management.

6.2 Image Processing and Mask Consistency

One of the most critical technical challenges in LocalCanvas lies in maintaining consistency between user-drawn masks and the underlying image data. On mobile devices, images may be displayed at resolutions that differ significantly from their original pixel dimensions. As a result, touch coordinates captured during mask editing cannot be directly applied to the original image without transformation.

To address this issue, the system establishes a coordinate mapping mechanism that translates touch input from screen space to image pixel space. Mask data is stored independently of display resolution and later recombined with the original image during AI request construction. This design ensures that the mask accurately reflects the user's intent regardless of device resolution or screen orientation.

Figure 14 illustrates the relationship between screen coordinates, image coordinates, and mask data.

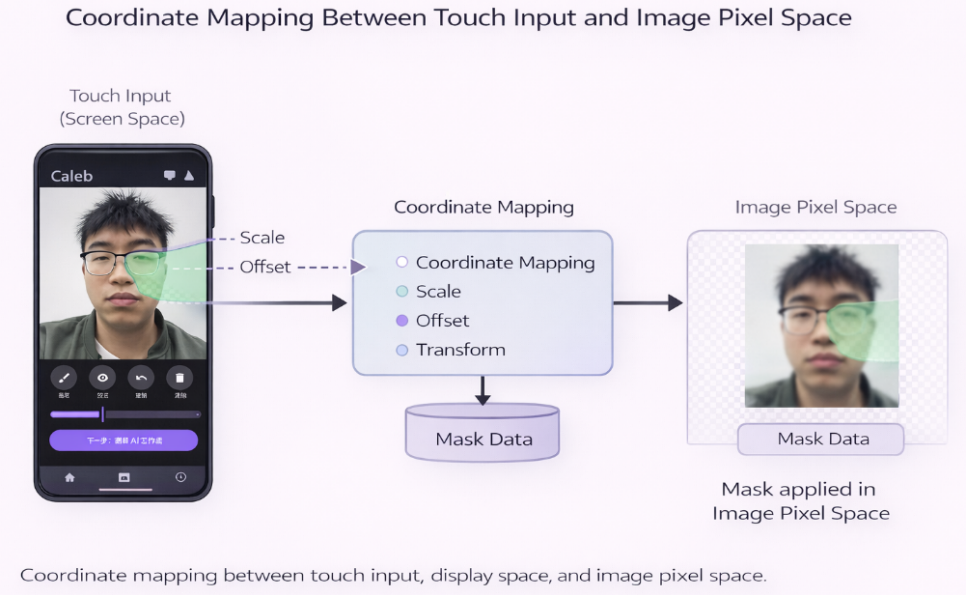


Figure 14: Coordinate mapping between touch input, display space, and image pixel space.

6.3 Asynchronous Task Handling and Network Latency

AI image generation is inherently time-consuming and often dependent on remote services. This introduces challenges related to network latency, unpredictable response times, and potential task failures. In a mobile context, these issues are further complicated by application lifecycle events such as backgrounding or configuration changes.

LocalCanvas addresses these challenges by treating AI generation as an asynchronous task with explicit lifecycle management. Task states, including pending, running, completed, and failed, are tracked and persisted locally. The user interface reflects these states through progress indicators and status messages, helping users remain informed during long-running operations. By decoupling task execution from the UI layer, the system avoids blocking interactions and supports recovery after interruptions.

Figure 15 illustrates the asynchronous task handling mechanism and its interaction with the UI and storage components.

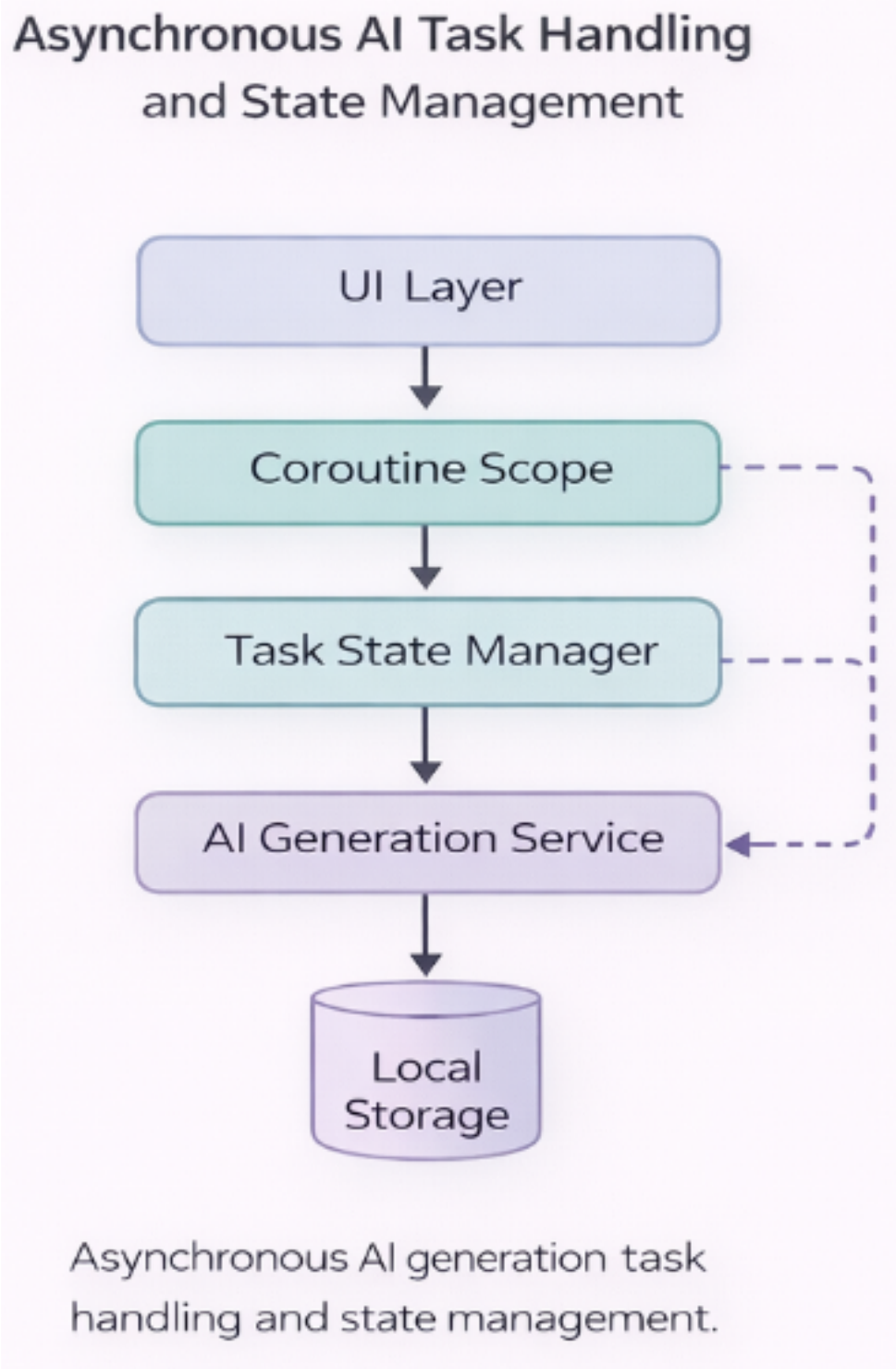


Figure 15: Asynchronous AI generation task handling and state management.

6.4 Data Storage, Recovery, and Compatibility

Another important engineering consideration is reliable data storage and recovery. Generated images, masks, and task metadata must be preserved across application restarts and system interruptions. At the same time, the storage mechanism must comply with Android's evolving storage policies and permission models.

To meet these requirements, LocalCanvas separates transient data from persistent re-

sources. Temporary data, such as intermediate mask representations and in-memory image buffers, are carefully managed to reduce memory pressure during runtime. Persistent data, including generated images and task metadata, are stored using platform-supported storage mechanisms to ensure long-term accessibility and compatibility across Android versions.

Special attention is paid to recovery scenarios. Task metadata is persisted incrementally so that, in the event of application termination or unexpected interruption, the system can restore task states and present them accurately to the user. This approach enhances reliability and prevents data loss, particularly in weak-network or resource-constrained environments. In addition, compatibility considerations are incorporated into storage design to accommodate differences in file access behavior across Android devices and system versions.

Figure 16 illustrates the relationship between transient data, persistent storage, and recovery mechanisms within the application.

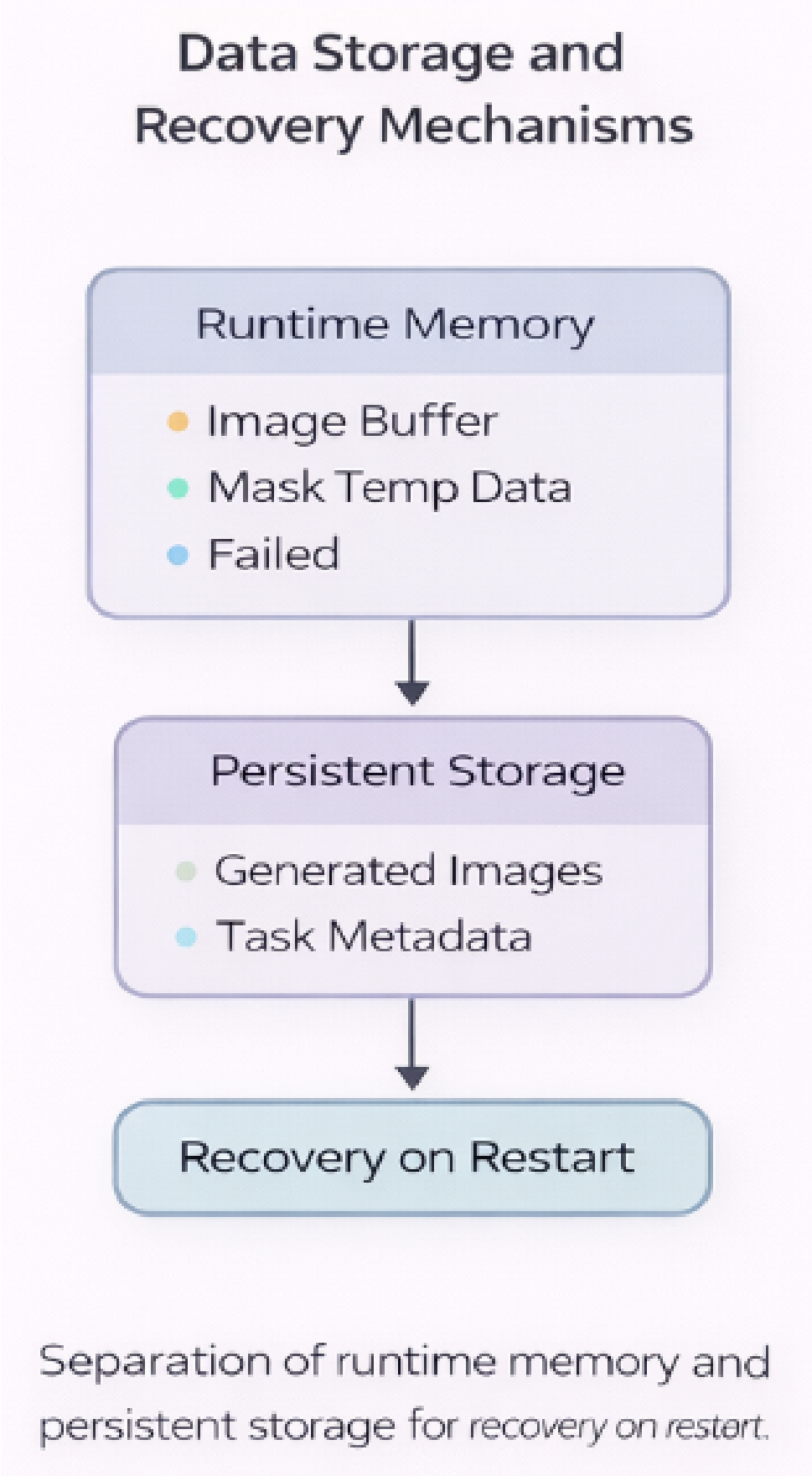


Figure 16: Data storage and recovery mechanisms in the LocalCanvas application.

6.5 Engineering Trade-offs and Design Considerations

Throughout the development of LocalCanvas, several engineering trade-offs were carefully evaluated. For instance, increasing image resolution can improve generation quality but also raises memory consumption and processing time. Similarly, exposing more AI configuration parameters enhances flexibility but may increase user cognitive load. Based on these considerations, the system adopts balanced defaults while allowing optional customization where appropriate.

Another important trade-off involves responsiveness versus robustness. While aggressive background execution could reduce perceived waiting time, it may conflict with mobile system constraints and battery usage policies. Consequently, LocalCanvas prioritizes predictable and stable behavior over maximum throughput. By making these design choices explicit, the system maintains a consistent user experience while remaining adaptable to future extensions.

Overall, the technologies and solutions discussed in this section reflect the practical challenges of deploying AI-driven image generation on mobile devices. Rather than relying on idealized assumptions, the design emphasizes resilience, clarity, and user trust, which are essential for real-world application usage.

6.6 Core Implementation Highlights (Code-Level Design)

This subsection presents representative code-level implementations that demonstrate how the core technologies described above are realized in practice. Rather than providing complete source files, selected code snippets are shown to highlight key design decisions and implementation strategies adopted in the application.

6.6.1 Remote AI Image Generation Service

The application integrates a remote AI image generation service through a dedicated API layer built with Retrofit. All image generation requests are encapsulated in a unified request model, enabling flexible configuration of prompts, input images, optional masks, and output resolution.

```
@POST("v1/images/generations")
suspend fun generateImage(
    @Header("Authorization") token: String,
    @Body request: ImageGenerationRequest
): ImageGenerationResponse

data class ImageGenerationRequest(
    val model: String,
    val prompt: String,
    val image: String,
    val mask: String?,
```

```
    val size: String
)
```

This design ensures a clear separation between network communication and user interface logic, while allowing future extensions such as model replacement or parameter tuning with minimal refactoring.

6.6.2 Repository Pattern for Asynchronous Task Management

To improve maintainability and scalability, the application adopts a repository pattern to manage AI-related requests. The repository abstracts the underlying network implementation and provides a unified interface for asynchronous image generation tasks.

```
class SiliconFlowRepository(private val api: SiliconFlowApi) {

    suspend fun generateImage(
        prompt: String,
        imageBase64: String,
        maskBase64: String?
    ): ImageGenerationResponse {
        val request = ImageGenerationRequest(
            model = "stable-diffusion",
            prompt = prompt,
            image = imageBase64,
            mask = maskBase64,
            size = "1024x1024"
        )
        return api.generateImage("Bearer $API_KEY", request)
    }
}
```

By isolating AI service interactions within the repository layer, the overall system architecture becomes more modular and easier to test or extend.

6.6.3 Image Encoding and Mask Preprocessing

Before transmission to the remote AI service, user-provided images and masks are converted into Base64-encoded strings to ensure compatibility with the service interface. This preprocessing step guarantees consistent data formatting across different devices and Android versions.

```
fun bitmapToBase64(bitmap: Bitmap): String {
    val outputStream = ByteArrayOutputStream()
    bitmap.compress(Bitmap.CompressFormat.PNG, 100, outputStream)
```

```
    return Base64.encodeToString(
        outputStream.toByteArray(),
        Base64.NO_WRAP
    )
}
```

This approach minimizes transmission errors and simplifies the integration between the mobile client and the cloud-based AI service.

6.6.4 Interactive Mask Editing with Jetpack Compose

The mask editing module is implemented using Jetpack Compose and Canvas APIs, enabling users to interactively draw and refine mask regions directly on the image. Real-time rendering ensures immediate visual feedback during user interaction.

```
Canvas(modifier = Modifier.fillMaxSize()) {
    drawPath(
        path = currentPath,
        color = Color.White,
        style = Stroke(width = brushSize)
    )
}
```

This implementation provides intuitive and precise mask control, which is essential for achieving high-quality AI-based image editing results.

7 Testing and User Experience Evaluation

This section evaluates the LocalCanvas application from both a technical and user-centered perspective. Testing focuses on functional correctness, performance stability, and cross-device compatibility, while user experience evaluation examines usability, responsiveness, and overall interaction quality. Together, these evaluations provide evidence that the system operates reliably in realistic usage environments.

7.1 Functional and Performance Testing

Functional testing is conducted to verify that the core features of LocalCanvas operate as intended across the complete workflow. Key functionalities tested include image selection, mask editing, AI workflow configuration, asynchronous task execution, result display, and task history management. Each functional module is first validated independently and is then examined as part of an integrated end-to-end process to ensure consistent behavior across module boundaries.

Performance testing focuses on system responsiveness and resource usage during critical interactions, such as touch-based mask drawing and image preview, as well as during long-running AI generation tasks. Particular attention is paid to memory consumption when handling high-resolution images and to UI responsiveness under concurrent background operations. The evaluation results indicate that the application maintains stable performance under typical usage conditions, with no noticeable lag during interactive operations and acceptable responsiveness during background processing.

Table 4 summarizes the main functional and performance testing items, along with their corresponding outcomes.

Table 4: Summary of Functional and Performance Testing

Test Item	**Test Description**	**Result**
Image Import	Import images from local gallery	Passed
Mask Editing	Touch-based region drawing and preview	Passed
AI Generation	Task submission and result retrieval	Passed
Task Management	Task state tracking and recovery	Passed
Result Saving	Save generated images locally	Passed
Performance	UI responsiveness under load	Acceptable

In addition to tabular verification, an aggregated visual summary is used to illustrate the overall testing outcomes across multiple dimensions, including functional correctness, compatibility, performance, and stability. This visual representation helps provide a concise overview of system reliability under diverse testing conditions.

Figure 17 presents a consolidated view of the testing results, highlighting that the application achieves a high pass rate across tested categories and demonstrates consistent behavior on different devices and configurations.

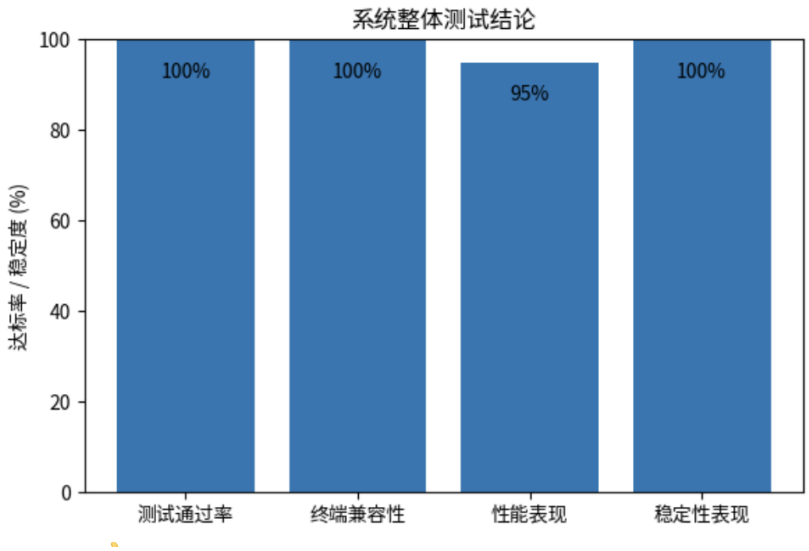


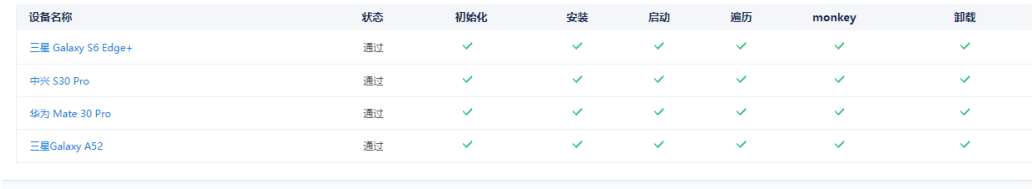
Figure 17: Overall summary of functional, compatibility, performance, and stability testing results for the LocalCanvas application.

7.2 Cloud Platform Testing Results

To further assess compatibility and stability, LocalCanvas is tested on a third-party cloud testing platform that provides access to multiple Android devices and system versions. This testing environment enables systematic validation across different screen sizes, hardware configurations, and Android releases without relying solely on local physical devices.

Cloud-based testing focuses on application launch behavior, navigation stability, permission handling, and runtime performance. The application is observed to start successfully across tested devices and to maintain consistent behavior during navigation and interaction. No critical crashes are detected during standard operation, and resource usage remains within acceptable limits.

Figure 18 presents representative results from the cloud testing platform, including device coverage and runtime performance indicators.



设备名称	状态	初始化	安装	启动	遍历	monkey	卸载
三星 Galaxy S6 Edge+	通过	✓	✓	✓	✓	✓	✓
中兴 S30 Pro	通过	✓	✓	✓	✓	✓	✓
华为 Mate 30 Pro	通过	✓	✓	✓	✓	✓	✓
三星Galaxy A52	通过	✓	✓	✓	✓	✓	✓

Figure 18: Overview of cloud-based testing results for the LocalCanvas application across multiple Android devices and system configurations.

7.3 User Experience Evaluation and Feedback

User experience evaluation is conducted based on structured and unstructured feedback collected from multiple participants during real usage of the LocalCanvas application. The feedback reflects diverse usage scenarios, ranging from casual image transformation to more advanced prompt-driven creative tasks, and provides valuable insight into both the strengths and limitations of the current system.

Overall, users report a high level of satisfaction with the core functionality and visual output quality of the application. Many participants highlight that the interface appears clean and visually pleasing upon first launch, and that the general operation flow feels smooth and efficient. In particular, the touch-based interaction and result preview mechanism are frequently mentioned as intuitive, enabling users to quickly understand the effect of AI generation and compare results before and after processing. Several users also note that generation speed remains acceptable and does not significantly disrupt the creative process, even when handling relatively complex prompts.

In terms of output quality, feedback consistently emphasizes the natural handling of lighting, composition, and character details in generated images. Users observe that the system demonstrates strong capability in interpreting complex prompts and preserving key visual features from the original image, including facial structure and pose. The diversity of supported styles, ranging from realistic rendering to more stylized or cartoon-like

outputs, is regarded as a major advantage, allowing the application to meet varied creative needs. For common use cases such as avatar generation or stylistic image transformation, the results are generally considered reliable and visually convincing.

Despite these strengths, several usability issues are identified, particularly during initial use. A recurring concern relates to insufficient operational guidance, with some users reporting uncertainty about where to initiate generation or how to access advanced options. In a few cases, first-time users rely on repeated trial and error to discover core functionality, which negatively affects the overall experience. These observations suggest that while the interface is minimalistic, the lack of explicit onboarding or contextual hints may hinder usability for less experienced users.

Specific interaction and layout issues are also reported. Some users encounter difficulties when editing longer prompt texts, such as unexpected triggering of generation or incomplete display of action buttons. Others report that advanced configuration controls, including style modules and sampling parameter sliders, may overlap or become partially inaccessible on certain devices or screen sizes. Although these issues do not prevent task completion in all cases, they introduce friction and reduce confidence during use, especially when users attempt more complex configurations.

A small number of users also note occasional instability when extreme parameter values are selected, such as setting both style strength and sampling steps to high levels. While these cases are relatively rare, they highlight the importance of clearer parameter constraints and feedback to prevent unintended behavior. In addition, several participants express interest in preset prompts and guided templates, which could lower the barrier to entry and reduce dependence on precise manual prompt formulation.

Taken together, the feedback indicates that LocalCanvas succeeds in delivering high-quality AI-generated results through an efficient and visually appealing interface, but would benefit from further refinement in guidance, layout robustness, and parameter handling. These findings directly inform the improvement directions discussed in the following section and provide a grounded basis for future iteration.

Table **??** summarizes representative user feedback collected during the evaluation process, reflecting both positive experiences and practical issues encountered during real usage.

Aspect	Overall Tendency	Representative User Feedback
Usability	Mostly Positive	"The application is easy to use and the overall workflow feels smooth." "The UI design is simple and intuitive, and I can quickly start creating images."
User Interface	Positive with Issues	"The interface looks clean and visually pleasing." "Some buttons are not fully displayed on certain screens, especially when advanced options are enabled."
Prompt Interaction	Mixed	"The system responds well to precise prompts." "When the prompt text becomes too long, some buttons are hidden and editing becomes difficult."
Image Quality	Highly Positive	"The generated images look natural, especially in lighting and facial details." "The AI understands complex prompts well and preserves important visual features."
Performance	Generally Positive	"The generation speed is fast and acceptable for mobile usage." "Most image generation tasks are completed within several seconds to half a minute."
Guidance and Learnability	Needs Improvement	"There is a lack of clear operational guidance for first-time users." "It was not obvious where to start generation or how to access advanced features."
Advanced Configuration	Mixed / Risky	"Setting both style strength and sampling steps to high values sometimes causes unstable behavior." "Advanced controls may overlap or become inaccessible on some devices."
Overall Satisfaction	High	"The application is powerful, efficient, and very suitable for daily creative use." "The before-and-after comparison makes the creative effect very clear."

Overall, the testing and evaluation results suggest that LocalCanvas achieves a solid balance between functionality, visual quality, and usability in typical mobile usage scenarios. The application performs reliably for core tasks and is well received for its intuitive interaction model and output quality. At the same time, the feedback reveals several prac-

tical issues related to onboarding, layout robustness, and advanced parameter handling. These observations indicate that while the system is already suitable for real-world use, targeted refinements could further improve consistency and reduce friction, particularly for first-time and advanced users.

8 Discussion and Future Improvements

8.1 System Limitations

Although LocalCanvas demonstrates stable functionality and a coherent interaction workflow, several limitations are observed during development and evaluation. One notable constraint lies in the dependency on external AI generation services. While this design choice enables access to advanced generation capabilities without on-device model training, it also introduces reliance on network connectivity and service availability. In practice, variations in network quality can still affect task completion time, even though asynchronous handling and progress feedback help mitigate user frustration.

Another limitation relates to the balance between flexibility and usability. To maintain accessibility for non-expert users, the application intentionally limits the number of exposed configuration parameters. As a result, some advanced users may find the degree of customization insufficient for highly specialized creative tasks. This reflects a conscious trade-off rather than a technical deficiency, yet it nonetheless constrains certain usage scenarios. In addition, despite careful memory management, handling high-resolution images on lower-end devices may still lead to increased resource pressure, which places natural bounds on performance consistency across diverse hardware.

8.2 Future Improvements

Based on the identified limitations and user feedback, several directions for future improvement can be considered. One potential extension is the introduction of adaptive configuration modes, where basic and advanced parameter sets are presented according to user preference or experience level. This approach could preserve usability for beginners while offering greater flexibility to experienced users. Another promising direction involves enhancing offline resilience, for example by supporting partial task caching or deferred uploads when network conditions are unstable.

From an interaction perspective, onboarding and contextual guidance could be further refined to assist first-time users in understanding advanced features. Subtle in-app hints or interactive tutorials may reduce the initial learning curve without overwhelming the interface. Finally, future work could explore tighter integration between task history and creative iteration, enabling users to more easily trace, compare, and refine previous results. Taken together, these improvements would strengthen the system's adaptability and extend its applicability to a broader range of creative scenarios.

9 Conclusion

This project presents the design and implementation of **LocalCanvas**, a mobile AI-based visual creation application that emphasizes controllability, usability, and robustness in real-world mobile contexts. By integrating touch-based mask editing with structured AI generation workflows, the system demonstrates how advanced AI capabilities can be made accessible and manageable on resource-constrained devices without sacrificing user agency.

Throughout the development process, particular attention is paid to system architecture, interaction design, and engineering practicality. The layered system design, modular implementation, and asynchronous task handling collectively support a coherent end-to-end workflow, from image input and region specification to result generation and local persistence. Testing results and user feedback further indicate that the application performs reliably across typical usage scenarios and delivers an intuitive user experience.

At the same time, the project also highlights several inherent challenges in deploying AI-driven applications on mobile platforms, including network dependency, performance variability across devices, and the trade-offs between flexibility and simplicity. Rather than treating these issues as purely technical obstacles, the design decisions made in LocalCanvas reflect a pragmatic balance shaped by usability considerations and platform constraints.

Overall, LocalCanvas serves as a practical case study of mobile AI application development, illustrating how thoughtful system design and interaction-oriented engineering can bridge the gap between powerful AI services and everyday user needs. The insights gained from this project may inform future work in mobile creative tools and contribute to broader discussions on human-centered AI application design.