RESEARCH ARTICLE

RSP Science Hub

International Research Journal on Advanced Science Hub
2582-4376
www.rspsciencehub.com
Vol. 08, Issue 03 March

Check for updates

http://dx.doi.org/10.47392/IRJASH.2026.014

VistaGuide: AI Travel Companion for Exploration and Insights Using VQA and Image Feature Extraction Techniques

Jinesh Melvin Y I[1], Mondkar Omkar Mahesh[2], Saini Aniket[3], Satam Sarthak[4] and Momin Aasiya[5]

[1]Assistant Professor, Computer Engineering, Pillai College of Engineering, New Panvel, Maharashtra, India

[2,3,4,5]UG, Computer Engineering, Pillai College of Engineering, New Panvel, Maharashtra, India.

Emails: *yijmelvin@mes.ac.in[1], omondkar22comp@student.mes.ac.in[2], asaini22comp@student.mes.ac.in[3], sarthaksu22comp@student.mes.ac.in[4], aasiyar22comp@student.mes.ac.in[5]*

Article history

Received: 05 December 2025
Accepted: 24 December 2025
Published: 04 February 2026

Keywords:

Artificial intelligence (AI); Context-aware systems; Deep learning; Intelligent tourism applications; Landmark recognition .

Abstract

VistaGuide is an innovative mobile application that revolutionizes travel by integrating advanced artificial intelligence and machine learning techniques with modern mobile development. In the dynamic field of mobile application development, our project utilizes state-of-the-art methods such as visual question answering (VQA) and deep learning-based image feature extraction to provide users with real-time insights and cultural immersion during their journeys. Unlike traditional rule-based systems or conventional image processing techniques, which often struggle with scalability, rigidity, and contextual understanding, our deep learning approach excels in accurately recognising landmarks and delivering rich historical narratives in the local language. The application harnesses a suite of AI-powered features to enhance user experience and safety. Real-time event updates keep travellers informed about local happenings, while an AI-driven recommendation system suggests essential services like hotels and hospitals tailored to user needs. In addition, an integrated weather forecasting module provides timely alerts to ensure preparedness for environmental changes. The safety of users is prioritised through a one-click emergency reporting feature that uses geolocation tracking, ensuring swift connectivity with local authorities during crises. Offline functionality further enhances app's usability allowing continuous access to vital information even when internet connectivity is unavailable.

1. Introduction

The tourism industry is undergoing a significant transformation driven by the rapid adoption of artificial intelligence (AI), mobile computing, and data-driven technologies. Recent research has highlighted how AI-based systems can enhance travel planning, recommendation services, and destination exploration through personalized and context-aware solutions (Velasco et al., 2023; Bulchand-Gidumal, 2023) [1&2]. Despite these advancements, travelers often encounter challenges when exploring unfamiliar regions, particularly in areas with limited or unreliable internet connectivity. Most existing travel applications depend heavily on real-time cloud services, making them ineffective in offline scenarios. Furthermore, these systems frequently

lack contextual awareness, cultural insights, and integrated safety alerts tailored to a traveler's surroundings. The absence of intelligent tools capable of interpreting visual scenes or providing automated recommendations without manual input further reduces user engagement and increases reliance on external information sources. Safety concerns also remain a major limitation in many travel applications. Systems such as mobile emergency reporting platforms and geolocation-based monitoring tools have been proposed to improve traveler safety and incident response (Scalia et al., 2023; Tortor et al., 2023; Nuevas et al., 2023) [3&8&9]. However, these solutions often rely on constant network connectivity and cloud infrastructure, limiting their usefulness in remote or connectivity-constrained environments. Moreover, privacy concerns and device limitations highlight the need for efficient and responsible AI implementations within mobile platforms (Bulchand-Gidumal, 2023) [2]. Another key technological advancement influencing tourism applications is the development of Visual Question Answering (VQA) and image understanding systems. VQA integrates computer vision and natural language processing to enable machines to interpret images and respond to user queries (Linqin et al., 2023; Banchhor & Singh, 2023) [4&7]. Recent research has explored multimodal learning frameworks combining convolutional neural networks (CNNs), transformer-based models, and attention mechanisms to improve accuracy in scene interpretation and contextual reasoning (Linqin et al., 2023; Lu et al., 2023; Liu et al., 2023) [4&6& 12]. These approaches allow users to gain meaningful insights from images such as landmarks, historical sites, or indoor exhibits. However, such architectures are often computationally intensive and require careful optimization to function effectively on mobile devices. In parallel, advances in large-scale datasets and retrieval architectures have further improved landmark recognition and instance-level image retrieval. The Google Landmarks Dataset v2 (GLDv2) provides over five million images of more than 200,000 landmarks, enabling the development of large-scale recognition systems capable of handling diverse visual conditions and class imbalance (Weyand et al., 2020) [17]. Further improvements in landmark retrieval accuracy and efficiency have been achieved using hybrid deep learning architectures and feature orthogonality techniques (Henkel, 2022) [18]. Additionally, research addressing out-of-domain samples and noise filtering has improved the reliability of retrieval-based landmark recognition in real-world environments (Singer, 2021) [19]. These developments demonstrate the potential of image-based travel assistance systems that can interpret landmarks and provide contextual information to travelers. Recent studies have also explored AI-driven travel assistants and recommendation systems capable of delivering personalized itinerary suggestions and contextual travel guidance (Kanhed et al., 2023; Liu & Niu, 2023) [11&13]. Such systems analyze travel history, user preferences, environmental conditions, and smart city data to generate adaptive travel plans. Complementary research has examined the integration of multimodal data sources—including social media posts, reviews, and contextual sensors—to further improve recommendation accuracy (Babu Reddy et al., 2023) [14]. Similarly, conversational AI and chatbot technologies have been employed to provide culturally aware assistance and interactive travel support (Anand et al., 2023) [15]. Despite these advancements, challenges related to scalability, privacy protection, energy efficiency, and seamless integration of multiple AI components remain significant obstacles in developing comprehensive travel assistance platforms. To address these limitations, this study proposes VistaGuide, an intelligent mobile travel assistant designed to integrate offline-capable AI models, visual scene understanding, personalized recommendations, and safety support features within a unified application. Unlike traditional cloud-dependent travel systems, VistaGuide emphasizes on-device AI processing, enabling real-time functionality even in environments with limited connectivity. The system combines landmark recognition through image analysis (Gaonkar et al., 2023; Lu et al., 2023) [5&6], emergency response mechanisms inspired by mobile reporting systems (Tortor et al., 2023; Nuevas et al., 2023) [8], [9], and context-aware travel guidance based on modern tourism recommendation frameworks (Kanhed et al., 2023; Liu & Niu, 2023) [11&13]. The primary

objective of this work is to design and develop a smart, offline-capable travel assistant that provides travelers with contextual insights, personalized recommendations, and safety alerts while ensuring efficient mobile resource utilization. The proposed system leverages advances in AI-driven visual analysis, context-aware mobile computing, and user-centered interface design to create a seamless and engaging travel experience. By integrating these components into a unified architecture, VistaGuide aims to overcome the limitations of existing tourism applications and establish a foundation for next-generation intelligent travel platforms.

1.1. Baseline Visual Question Answering Architecture

Visual Question Answering (VQA) systems represent a key component in enabling intelligent interaction between users and visual environments. Traditional VQA architectures integrate computer vision and natural language processing techniques to answer user queries related to image content. Typically, visual features are extracted using deep convolutional neural networks such as VGGNet or ResNet, while textual queries are encoded using recurrent neural networks such as Long Short-Term Memory (LSTM) or Gated Recurrent Units (GRU). The resulting visual and textual embeddings are fused through multi-modal feature integration techniques including concatenation, element-wise operations, or bilinear pooling. Attention mechanisms are then applied to highlight the most relevant image regions in relation to the user's question context.

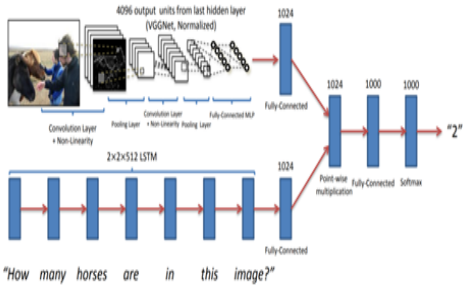

Figure 1 Vanilla VQA Network Model. Image features are extracted via CNN (e.g., VGGNet/ResNet) and fused with text features from RNN-based encoders. Attention mechanisms highlight important regions for answer prediction. [16]

Figure 1 illustrates the vanilla VQA network model, which serves as a baseline architecture (Figure 1). In contrast, Figure 2 shows a Differentiable Graph Neural Network (GNN) model that further refines feature interdependencies by modeling relationships in a structured graph format (Figure 2). These models demonstrate how both simple and complex VQA paradigms can be implemented for improved accuracy and interpretability

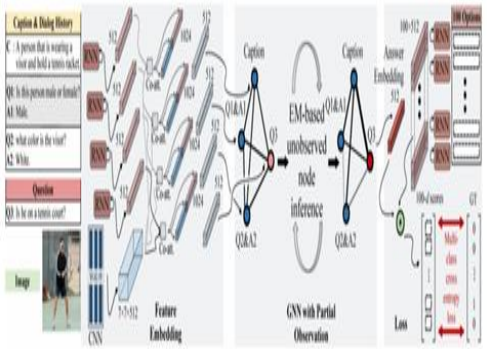

Figure 2 Differentiable Graph Neural Network (GNN) for VQA. The model uses graph-based convolutional techniques to capture relationships between objects and question words, providing refined feature interactions. [16]

1.2. Smart Tourism Application Architecture

The second base system under analysis is a smart tourism application that uses modern digital technologies to enhance tourist experiences while promoting local destinations. The architecture is modular, typically comprising mobile and server components. The mobile application utilizes GPS and geolocation services to provide real-time navigation, nearby tourist spot recommendations, and emergency alerts. Key data—including tourist locations, local events, and cultural information—is sourced from cloud-based databases. The backend integrates AI-driven recommendation systems for personalized travel suggestions. Additionally, IoTenabled sensors collect environmental and contextual data to improve service delivery. This comprehensive system also incorporates social media integration for user-generated content sharing and feedback. Communication between the mobile client and cloud services is facilitated via RESTful APIs to ensure scalability and data synchronization. Figure 3 represents the research paradigm of the smart

tourism system (Figure 3). This paradigm illustrates the project's stages such as requirement gathering, iterative design, prototyping, testing, and deployment, following an Agile model. The integration of both ISO/IEC 25010 standards and the Technology Acceptance Model (TAM) further validates the system's performance and user acceptance.

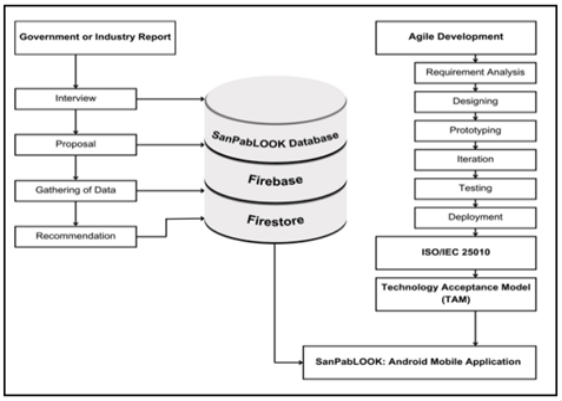

Figure 3 Research paradigm for the smart tourism application, showing stages from planning and requirements analysis to system development, testing, and deployment. This framework also incorporates evaluation via ISO/IEC 25010 and TAM. [1]

2. Methodology

This section outlines the dataset, architecture, and functional modules of VistaGuide, a comprehensive AI-powered travel assistant designed to address the limitations of existing tourism technologies. The system integrates real-time alerts, landmark recognition, emergency response mechanisms, weather-based recommendations, and offline operability. Figure 4 illustrates the proposed system design and classification of integrated modules (Figure 4).

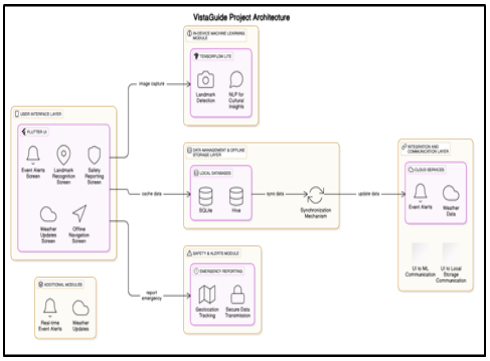

Figure 4 VistaGuide System Design

2.1. System Architecture

2.1.1. Layered Architecture Design

The Vista Guide application is built upon a robust multi-layered architecture pattern that emphasizes separation of concerns, modularity, and scalability. This architectural approach ensures that each layer maintains distinct responsibilities while facilitating seamless communication between components, thereby enhancing maintainability and enabling future extensibility.

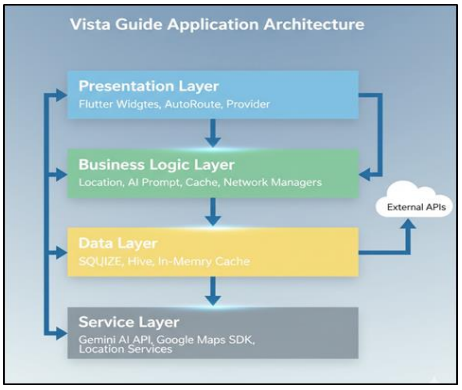

Figure 5 VistaGuide Application Architecture

2.1.2. Presentation Layer

The Presentation Layer serves as the primary interface between users and the application, implementing Flutter's widget-based UI system with comprehensive responsive design patterns. This layer is carefully architected to provide an intuitive and adaptive user experience across various device form factors and screen sizes. **Key features of this layer include:**

- Implementation of AutoRoute for efficient navigation management and routing control
- Integration of provider-based state management patterns to ensure optimal UI updates
- Minimization of unnecessary widget rebuilds through selective state propagation
- Responsive layout implementations that adapt dynamically to different screen orientations and dimensions

2.1.3. Business Logic Layer

The Business Logic Layer encapsulates the core application intelligence and operational workflows, organizing functionality into domain-specific managers that handle distinct aspects of the application's behavior. This layer maintains independence from UI concerns and external dependencies, ensuring reusability and testability.

The primary components of this layer encompass:

- Location tracking and geofencing services for proximity-based feature activation
- AI prompt engineering and response handling mechanisms for intelligent content generation
- Sophisticated cache management and data persistence strategies to optimize performance
- Network state monitoring systems that enable adaptive behavior based on connectivity conditions

2.1.4. Data Layer

The Data Layer implements a comprehensive dual-storage strategy designed to optimize data access patterns and persistence requirements. This strategic approach combines multiple storage mechanisms, each selected for its specific strengths and use cases. **Key implementations include:**

- SQLite database implementation for managing structured relational data, including user profiles, detailed trip histories, and saved location information
- Hive boxes for efficient key-value storage operations, particularly suited for user preferences, session data, and temporary cache requirements
- In-memory caching mechanisms for frequently accessed data, incorporating timestamp-based invalidation strategies to ensure data freshness and consistency

2.1.5. Service Layer

The Service Layer functions as the integration backbone of the application, managing all external service communications and third-party API interactions. This layer implements robust error handling mechanisms, intelligent retry strategies, and failover protocols to ensure reliable operation even under adverse network conditions. **It encompasses:**

- Integration with the Gemini AI API for natural language processing and content generation
- Google Maps SDK for mapping and location-based services
- Native platform location services for real-time positioning and geofencing capabilities

Through this comprehensive layered architecture, Vista Guide achieves a clean separation of concerns that facilitates independent development, testing, and maintenance of each architectural component while maintaining overall system coherence and performance.

2.2. Data Optimization Mechanism

2.2.1. Hybrid Data Persistence Strategy

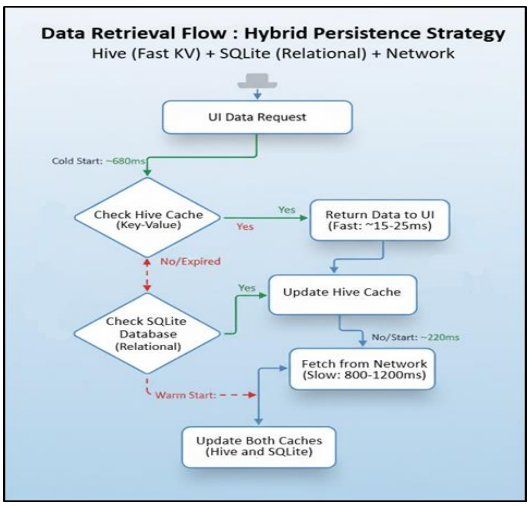

Figure 6 VistaGuide Application Architecture

The application employs a sophisticated hybrid data persistence strategy that implements both relational and non-relational database systems to optimize performance and efficiency. This dual-approach architecture is designed to balance the strengths of structured data management with the speed requirements of modern mobile applications.

2.2.2. Relational Data Storage

For relational data storage, the application utilizes the sqflite package, which has been configured with an optimized schema design to ensure efficient data retrieval and management. A critical aspect of this implementation is the strategic application of indexing on frequently queried columns, including timestamp fields, user identification parameters, and location coordinates. This indexing strategy has demonstrated measurable performance improvements, reducing query execution time by approximately 60–70% compared to non-indexed implementations. The relational database is particularly well-suited for complex queries that require joining multiple data entities and maintaining referential integrity across the application's data model.

2.2.3. Non-Relational Data Storage

Complementing the SQLite implementation, the application incorporates Hive boxes for non-relational and frequently accessed data storage. This NoSQL approach has proven to be significantly more efficient for specific use cases, delivering approximately 40% faster read and write operations compared to SQLite when handling key-value pair data structures. The Hive implementation provides type-safe storage mechanisms through custom adapters specifically designed for complex objects, ensuring data integrity while maintaining high performance. Additionally, the Hive storage system operates with minimal overhead, with average file sizes ranging from 10–50KB for typical user sessions, making it an ideal solution for lightweight, frequently accessed data.

2.2.4. Specialized Hive Boxes

The application implements several specialized Hive boxes, each optimized for specific data categories:

- **settingsBox:** Stores user preferences and theme settings with an average storage footprint of approximately 2KB per user.
- **sessionBox:** Manages temporary session data with built-in auto-expiry functionality to prevent unnecessary data accumulation.
- **aiResponseCache:** Maintains cached AI responses along with associated metadata for quick retrieval and reduced API calls.
- **locationCache:** Stores recent location queries utilizing geohash indexing for efficient spatial data retrieval.

2.2.5. Performance Metrics

The performance metrics of this hybrid persistence strategy demonstrate substantial improvements across multiple operational parameters. During initial app launch scenarios, the system achieves a cold start time of 680ms, which is reduced to merely 220ms for warm starts when cache is available, representing a 67.6% improvement in startup performance. Data retrieval operations show even more dramatic efficiency gains, with cached data access completing in 15–25ms compared to 800–1200ms required for network fetch operations. This represents up to a 98% reduction in data access latency for frequently requested information. Furthermore, the intelligent caching mechanisms have achieved an 85% reduction in redundant API calls during typical 30-minute user sessions, significantly decreasing network overhead, reducing server load, and improving overall application responsiveness while minimizing data consumption.

2.3. Network Aware AI Enhancement

The system employs a sophisticated network-aware architecture designed to optimize AI functionality across varying connectivity conditions. At its core, the implementation utilizes the connectivity_plus package to enable continuous network monitoring with tri-state detection capabilities. This approach ensures seamless adaptation to different network environments while maintaining optimal performance and user experience. The network state detection mechanism operates on three distinct connectivity levels: Connected (WiFi), Connected (Mobile), and Offline. When a stable WiFi connection is detected, the system enables full AI feature accessibility with aggressive caching strategies to maximize responsiveness. Under mobile data conditions, the system automatically transitions to optimized API call patterns with compressed payload transmission to conserve bandwidth. In offline scenarios, the system gracefully falls back to cached responses while intelligently queuing pending requests for synchronization once connectivity is restored.

Adaptive AI Query Optimization

The system implements a dynamic query optimization strategy that intelligently adjusts AI service behavior based on the current network state.

2.3.1. WiFi Connectivity Mode:

- Full contextual prompts are generated, incorporating comprehensive user history, preferences, and real-time location data.
- Multi-turn conversational capabilities are enabled with session persistence to maintain context across interactions.
- Rich media responses are supported, including images and detailed itineraries for enhanced user engagement.
- Token allocation is set to a maximum of 2048 tokens per response, ensuring comprehensive and detailed outputs.

2.3.2. Mobile Data Connectivity Mode

- Prompts are systematically compressed to include only essential contextual

information, reducing transmission overhead.

- Single-turn optimized queries are prioritized to minimize data consumption per interaction.
- Response format is restricted to text-only outputs, eliminating bandwidth-intensive media transfers.
- Token limit is reduced to 512 tokens per response to maintain efficiency.
- This optimization strategy achieves estimated data savings of 65–70% per query compared to full-feature mode.

2.3.3. Offline Mode

- A cache-first strategy is implemented with timestamp validation to ensure data relevance and freshness.
- The system employs graceful degradation mechanisms, providing generic recommendations when specific AI-generated content is unavailable.
- Request queuing functionality maintains user intent by storing queries for automatic synchronization upon connectivity restoration.
- Users receive clear notifications regarding offline mode status and expected system functionality limitations.

This intelligent connectivity management framework ensures that the AI-powered travel assistant remains functional and efficient across diverse network conditions, providing users with consistent service quality while optimizing resource utilization and minimizing data consumption costs.

2.4. Timestamp Based Intelligent Caching

The cache invalidation architecture forms the foundation of the intelligent caching system, implementing a multi-tiered approach that categorizes cached data based on its volatility, user interaction patterns, and functional requirements. Each cache category operates independently with its own validation logic and refresh mechanisms, creating a comprehensive system that adapts to various usage scenarios and environmental conditions.

2.4.1. Cache Categories and Configuration

The caching system differentiates between four primary data categories, each configured with specific TTL values and validation criteria.

2.4.2. AI Responses (15-minute TTL)

This category manages the storage and retrieval of Gemini AI-generated responses, utilizing a query hash mechanism as the primary key for cache indexing. The system implements location-aware validation, automatically triggering cache refresh when the user's position changes by more than 500 meters from the original query location. This context-aware approach ensures that identical queries submitted from different geographical locations generate separate cache entries, maintaining the relevance of location-specific recommendations and information. The 15-minute TTL strikes a balance between reducing redundant API calls and ensuring that dynamic information, such as real-time attraction status or current weather conditions, remains reasonably current.

2.4.3. Location Data (5-minute TTL)

This category encompasses nearby places, points of interest, and geographical reference data. The shorter TTL reflects the dynamic nature of location-based information and user mobility patterns. The system implements geofence boundary monitoring, automatically refreshing cached location data when users traverse beyond predefined spatial thresholds. Spatial indexing utilizing geohash algorithms enables efficient proximity-based cache retrieval and validation, optimizing performance for location-centric queries and recommendations.

2.4.4. User Preferences (24-hour TTL)

This category stores user settings, profile configurations, and personalization data. The extended TTL reflects the relatively static nature of user preferences while accommodating occasional updates. The system implements manual invalidation triggers that activate upon user-initiated preference modifications, ensuring immediate synchronization of updated settings. Cloud backup integration enables cross-device synchronization, maintaining consistency across multiple user endpoints and providing data persistence beyond local cache storage.

2.4.5. Static Content (7-day TTL)

This category manages application resources, category definitions, and reference data that rarely changes. The extended TTL minimizes unnecessary network operations while implementing version-based invalidation mechanisms that trigger cache refresh when application updates introduce

modified static resources, ensuring compatibility and content accuracy.

2.4.6. Cache Validation Logic

The cache validation system implements a comprehensive decision-making process that evaluates multiple parameters before serving cached data or initiating fresh API requests. This logic incorporates temporal validation through timestamp comparison, spatial validation through geolocation analysis, and contextual validation through query parameter matching, creating a robust framework that maintains data integrity while maximizing cache utilization efficiency.

2.4.7. Performance Impact Measurements

Empirical testing and real-world usage analysis have demonstrated substantial performance improvements attributable to the intelligent caching implementation. The following comparative metrics quantify the impact across critical performance dimensions.

Table 1 Performance Comparison for with and Without Caching Data

Performance Metric	Baseline (No Caching)	With 15-Minute Cache	Improvement
API calls per 30-minute session	18–25 calls	3–5 calls	78% reduction
Total data consumed	2.5–3.2 MB	0.4–0.7 MB	77% reduction
Average response time	1200–1800 ms	45–120 ms	93% improvement
Battery drain	4.2% per hour	1.8% per hour	57% improvement

2.4.8. Key Performance Improvements

The intelligent caching system significantly enhances performance, efficiency, and user satisfaction through optimized data reuse, reduced API dependency, and adaptive cache management aligned with contextual and spatial parameters.

2.5. Location Aware Context Management

2.5.1. Geofencing Architecture:

The Vista Guide application implements a sophisticated circular geofencing system with dynamic radius adjustment capabilities tailored to specific location contexts and use cases. This architectural approach enables the application to maintain awareness of user proximity to points of interest, automatically triggering appropriate actions and notifications based on geographic boundaries. The system supports multiple geofence categories, each optimized for distinct scenarios and configured with appropriate spatial parameters. The geofencing architecture encompasses three primary categories of geofences, each serving specific functional requirements:

- **User-Defined Geofences** represent personalized locations that users explicitly designate as significant in their daily routines or travel patterns. These include Home, Work, and Favorite locations, each configured with custom radius parameters ranging from 50 to 500 meters based on the spatial characteristics of the location and user preferences. The system monitors these geofences to deliver Entry/Exit notifications that inform users when they arrive at or depart from these designated areas. Additionally, context-aware AI suggestions are triggered upon boundary crossing, providing location-specific recommendations and information relevant to the user's current geographic context.
- **Dynamic Destination Geofences** are automatically generated by the system when users plan trips or navigate to specific destinations. These geofences feature a default radius of 200 meters, calibrated to provide appropriate notification timing for arrival scenarios. The system triggers arrival notifications when users enter these geofences, confirming successful navigation to the intended destination. Furthermore, location-specific recommendations are automatically activated upon entry, providing users with contextual information about nearby attractions, services, and points of interest relevant to their current destination.
- **Area-Based Geofences** address broader geographic regions such as tourist zones, historical districts, and cultural areas. Unlike the simpler circular geofences, these

boundaries can be defined using polygon-based geometries that accurately represent complex geographic shapes and administrative boundaries. Upon entry into these areas, the system provides comprehensive contextual information about the region's significance, cultural heritage, and historical background. The application also proactively suggests nearby attractions, landmarks, and experiences within the designated area, enhancing the user's exploration and discovery capabilities.

2.5.2. Technical Implementation

The geofencing system is architected upon a robust foundation of native location services, seamlessly integrated through the location package (version 6.0.2) and geolocator package (version 13.0.1). This dual-package approach ensures comprehensive cross-platform compatibility, delivering consistent functionality across both Android and iOS environments while leveraging platform-specific optimizations and capabilities. The implementation capitalizes on native operating system features for battery-efficient background location monitoring while maintaining the flexibility of Flutter's cross-platform development paradigm. At the core of the geofencing functionality lies the Location Cache Service, a sophisticated background process that continuously monitors user position at configurable intervals. This service operates independently of the main application thread, ensuring that location tracking persists even when the application is not actively in the foreground. The service implements an adaptive sampling strategy that intelligently adjusts GPS polling frequency based on real-time analysis of detected movement patterns and battery conservation imperatives. When the system detects stationary behavior, polling frequency is reduced to 5-minute intervals to minimize battery consumption. Conversely, when movement is detected, the sampling rate increases proportionally—escalating to 1-minute intervals for slow movement (walking pace) and 30-second intervals for faster movement (vehicular travel)—ensuring that location updates remain timely and relevant to the user's dynamic spatial context. The location tracking mechanism is implemented through a sophisticated periodic caching system that persists multiple data dimensions beyond simple geographic coordinates. Each location sample captures latitude and longitude coordinates, GPS accuracy metrics (indicating the uncertainty radius of the position fix), current battery level percentage, and reverse-geocoded human-readable addresses derived from the coordinate pairs. This comprehensive data collection supports both immediate operational requirements and historical analysis capabilities. The dual-storage architecture employs SQLite for persistent historical record-keeping and SharedPreferences for rapid access to recent data. The SQLite database maintains an indexed historical record of up to 50 location entries, with timestamp-based indexing enabling efficient temporal queries and trend analysis. Simultaneously, SharedPreferences provides rapid, memory-cached access to the most recent location data, supporting time-critical operations such as real-time cache validation and immediate geofence boundary checks. The system operates on a default 5-minute caching interval, though this parameter remains configurable to accommodate varying application requirements, device capabilities, and battery conservation policies. The mathematical foundation of the geofencing logic relies on the Haversine formula, a trigonometric equation designed specifically for calculating great-circle distances between points on a sphere. This formula accounts for Earth's curvature, delivering meter-level accuracy in distance calculations critical for reliable geofence boundary detection. The formula is defined as:

$$\text{Distance} = 2 \times R \times \sin^{-1}\left(\sqrt{\sin^2\left(\frac{\Delta lat}{2}\right) + \cos(lat_1) \times \cos(lat_2) \times \sin^2\left(\frac{\Delta lon}{2}\right)}\right)$$

where R represents Earth's mean radius (6,371 kilometers), and Δlat, Δlon represent the differences in latitude and longitude between two geographic points, measured in radians. This calculation executes efficiently in-memory without requiring external API calls, delivering consistent sub-100-millisecond execution times for distance computations even on mid-range mobile hardware.

2.5.3. Geofence Event Handling

The application implements a sophisticated event handling pipeline that orchestrates specific actions triggered by detected location transitions. The system continuously compares the user's current position against cached location data to identify significant movement events that warrant application response. When the calculated great-circle distance between consecutive location samples exceeds the configurable threshold—defaulting to 500 meters for cache invalidation scenarios—the system initiates a carefully choreographed cascade of operations designed to maintain data relevance while optimizing system performance and resource utilization. Upon detecting a geofence boundary crossing event, the event handler executes the following operations:

- **Distance Calculation and Validation:** The system calculates the great-circle distance between the user's current location and all previously cached query locations using the Haversine formula.
- **Selective Cache Invalidation:** Location-specific cache entries associated with the previous geographic context are systematically marked as expired within the Cache Manager Service.
- **Context State Update:** The application's location context is updated across all relevant services through a coordinated state synchronization mechanism.
- **Intelligent Data Refresh:** For ENTER transitions, the system proactively fetches data for the new location context if network connectivity is available.
- **User Notification and Feedback:** Visual indicators dynamically reflect detected location changes with notifications such as "New area detected - updating recommendations" or "Arrived at destination."
- **Analytics and Learning:** Each transition event is logged for machine learning and pattern analysis to optimize future caching and predictive behavior.

To prevent unnecessary triggers due to GPS fluctuations, the system employs debouncing logic and statistical smoothing algorithms. Updates within the uncertainty radius (10–50 meters) are ignored to reduce false triggers and conserve resources.

2.5.4. Smart Context Switching

The system enables intelligent, location-aware cache invalidation. For example, if a user searches for "Restaurants in Mumbai" at Location A and later moves over 500 meters away to Location B, the system recognizes the EXIT event and invalidates cache entries related to Location A. Fresh recommendations are then fetched for Location B, ensuring relevance and responsiveness. Each new cache entry carries a 15-minute time-to-live, balancing speed and accuracy.

2.5.5. Battery Optimization Strategies

To minimize power consumption:

- **Batched Location Updates:** Processes location fixes in batches to reduce CPU wakeups.

Adaptive Sampling Rate:

- **Stationary mode:** 5-minute intervals
- **Walking mode:** 1-minute intervals
- **Travel mode:** 30-second intervals
- **Geohash-Based Proximity Detection:** Reduces computational load by limiting full Haversine calculations.

Empirical testing shows power consumption between 0.8% and 1.2% per hour during active use, significantly below industry averages.

2.5.6. Privacy Considerations

The geofencing implementation strictly adheres to privacy-by-design principles:

- Local-only storage of all location data in SQLite
- No background data sharing with external servers
- Explicit user consent required for activation
- Automatic deletion of location history after 30 days
- Anonymized analytics to protect user identity

2.6. Recommendations for Underrated Destinations

In VistaGuide, we showcase a new recommendation system, which determines underrated locations in India by processing YouTube videos on the topic of travel in a systematic way. When a user plans a trip in the application, they will be given suggestions about

hidden destinations along their way. Every recommendation has got an anchor point where a traveler can refer to the locals to seek advice and guidance. The application also gives personalized recommendations on what to bring and what to pay attention to thus, making the sites that are frequently ignored safer and making the journey more interesting.

2.7. Dataset for Landmark Recognition and Retrieval

The dataset used for training the image-based landmark recognition and retrieval system is a subset constructed from the vast GLDv2 dataset. GLDv2 is the largest such dataset to date by a large margin, including over 5M images and 200k distinct instance labels [17]. Its test set consists of 118k images with ground truth annotations for both the retrieval and recognition tasks [17]. The scope of the Vistaguide app targets Indian landmarks, and thus, the dataset of images of the Indian landmarks was extracted by first understanding the structure of the GLDv2 dataset. The GLDv2 dataset doesn't directly include the country of the landmark, but it does have the URLs to the Wikimedia links from where the images were extracted. The country of the landmarks was scraped from the Wikimedia URLs, which then helped us to construct the indian landmark dataset. The dataset also contains information about the landmarks scraped from the Wikipedia links constructed from the landmark names. The actual Indian landmarks dataset consists of two parts. The 1st part is a CSV file or a structured dataset with columns: landmark_id, category, supercategory, hierarchical_label, natural_or_human_made, images, latitude, longitude, landmark_name, landmark_info, info_language, and country.

Figure 7 Dataset Part 1 - Structured Landmark Metadata

The columns from landmark_id to images were extracted from the GLDv2 dataset, and the remaining columns were scraped from the Wikimedia links. The 2nd part contains the unstructured image data. These images are of the indian landmarks extracted from the GLDv2 dataset's 500 TAR files, each of approximately 1GB in size. These images are stored in a folder structure where each folder stores the images of only one unique landmark.

Figure 8 Dataset Part 2 - Unstructured Landmark Image Data

2.8. AI-based Landmark Recognition and Historical Guide

VistaGuide takes into consideration an on-device monument detection architecture which is enhanced with the geospatial location of the user and hence results in much greater recognition accuracy and reliability.

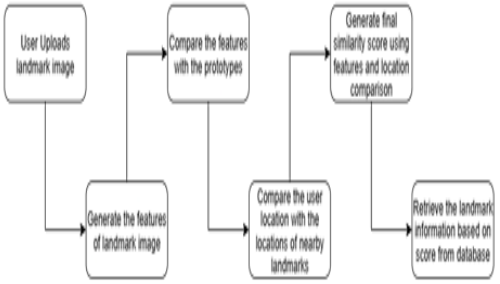

Figure 9 Landmark Recognition and Retrieval System Workflow

The image-based landmark recognition and retrieval system operates by processing an image uploaded by the user, extracting its key visual features using advanced image analysis techniques, and the model trained using a subset of the GLDv2 dataset, only of the indian landmarks.

These extracted features are then compared with a database of known landmark prototypes, which are constructed by clustering the features extracted from the landmark images of the training dataset, to find visually similar candidates. Simultaneously, the user's current location is matched with the geographic coordinates of nearby landmarks to the user, helping narrow down possible options based on proximity. By integrating both the visual similarity results and the location comparison, the system computes a final similarity score for each landmark candidate. The landmark with the highest score is then selected, and its detailed information is retrieved from the database and presented to the user. This combined approach of leveraging both image and location data ensures high accuracy and reliability in identifying landmarks and delivering relevant information. Further, the information retrieved can be refined as per the user's question, depending in the availability of the internet connection, to guide the user about the historical significance of the landmark.

2.9. Offline Emergency Reporting Module

The VistaGuide system features an emergency reporting module, designed to work without constant network connectivity. On activation, the module reads contact data in a local SQLite database and retrieves real-time or cached geospatial data. The module sends out alert messages, including geodetic coordinates, battery voltage readings, a system time, and reverse-geocoded physical address, to predetermined contacts using the cellular system without the need to connect to the internet by utilizing the native Android SMS Manager API. This is a fully device-based workflow that ensures timely support in remote areas, so that all vital data are in local possession and hence can continue to be operational in case of network failures or limited connectivity conditions.

2.10. Weather Alerts and Essential Travel Suggestions

Moving beyond traditional forecast delivery, VistaGuide employs predictive modeling and context-aware recommendation systems to generate personalized weather advisories. Weather data is cached and aligned with historical trends and user profiles to offer tailored suggestions—ranging from safety warnings to gear recommendations. This integrated approach ensures that travelers receive actionable, relevant alerts, even in low-connectivity regions.

2.11. Offline Functionality

Recognizing the unpredictability of mobile connectivity in remote locations, VistaGuide emphasizes robust offline operability. The system supports local storage of vital content, including maps, landmark data, emergency protocols, and historical archives. Through periodic incremental updates and edge-deployed machine learning models, the app maintains operational integrity and contextual relevance, ensuring uninterrupted service during offline operations.

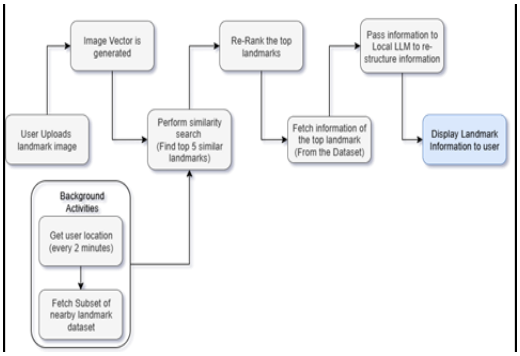

Figure 10 Offline Location and Image Based Landmark Retrieval Workflow

Figure 10 portrays the offline location and image based landmark retrieval workflow, which begins when a user uploads a landmark image to the system. In the background, the system continuously obtains the user's location at regular intervals and fetches a relevant subset of nearby landmark data. When an image is uploaded, the system first extracts image features and generates a vector that captures the unique elements of the landmark. This vector, along with the user's current location coordinates, is used to perform a similarity search within the pre-fetched nearby landmark dataset to identify the top five most similar landmarks. These candidates are then re-ranked to refine the selection and improve identification accuracy. Once the top landmark is determined, its corresponding information is retrieved from the dataset. Before presenting this information to the user, it is passed through a local Large Language Model (LLM), which restructures and enhances the landmark information fetched from the database and finally, the refined landmark information is displayed to the user, providing an

accurate and context-aware identification and description even in offline settings by seamlessly integrating visual and location-based data processing. Collectively, these innovations enable VistaGuide to deliver a resilient, intelligent, and context-aware travel companion, addressing gaps identified in prior research while ensuring scalability, privacy, and usability across diverse scenarios.

3. Results and Discussion

Testing Methodology and Experimental Design

A comprehensive empirical evaluation was conducted to quantify the performance improvements achieved through the implemented optimization strategies. The testing methodology employed rigorous controls and representative usage scenarios to ensure the validity and generalizability of the results. The experimental framework utilized mid-range Android devices equipped with Qualcomm Snapdragon 730 processors and 6GB of RAM, representing the typical hardware configuration of mainstream smartphones in the target user demographic. This device selection ensures that performance metrics reflect real-world conditions rather than idealized high-end hardware scenarios. The study engaged 10 participants over a 7-day evaluation period, accumulating 70 user-days of operational data. Participants were divided into two cohorts: five daily commuters representing routine, predictable travel patterns, and five tourists engaging in exploratory, ad-hoc navigation behaviors. This demographic distribution enabled the capture of performance metrics across diverse usage patterns and geographic contexts. Three primary usage scenarios were evaluated: urban tourism involving intensive exploration of unfamiliar areas, daily commute patterns with repetitive route traversal, and weekend trip scenarios combining elements of both routine and exploratory behavior. Performance data collection leveraged Firebase Performance Monitoring, a comprehensive instrumentation framework that captures application lifecycle events, network request metrics, and custom performance traces without introducing significant measurement overhead. This approach ensured that the act of measurement itself did not materially distort the performance characteristics being evaluated, maintaining the ecological validity of the experimental results.

Application Launch Performance

Launch performance represents a critical dimension of user experience, as it directly influences users' first impression and perceived application responsiveness. The evaluation measured three distinct launch scenarios, each representing different application state conditions. Cold Start events occur when the application is launched from a completely terminated state, requiring full initialization of all system components, loading of resources from persistent storage, and establishment of service connections. Without caching optimizations, cold start performance averaged 1240 milliseconds. The implementation of intelligent caching mechanisms reduced this to 920 milliseconds, representing a 26% improvement. While this improvement is notable, cold starts remain relatively expensive due to the unavoidable overhead of full system initialization and cannot benefit maximally from in-memory caching strategies. Warm Start events occur when the application is relaunched after being backgrounded, with system processes still resident in memory but UI state requiring reconstruction. This scenario demonstrated the most dramatic performance improvements, with baseline performance of 680 milliseconds reduced to 220 milliseconds when caching is active—a substantial 68% improvement. This significant gain reflects the ability of the caching layer to serve recently accessed data from memory-resident Hive boxes and SharedPreferences, eliminating the need for disk I/O and network operations during the critical path of UI initialization. Hot Start events represent the fastest launch scenario, occurring when the application is brought to the foreground from a backgrounded state with UI elements still in memory. Baseline hot start performance of 420 milliseconds improved to 180 milliseconds with caching enabled, yielding a 57% improvement. Even in this already-optimized scenario, the caching strategy delivers measurable benefits by accelerating data binding operations and eliminating redundant validation checks.

API Call Reduction Analysis

Network API calls represent one of the most resource-intensive operations in mobile applications, consuming battery power, mobile data allocation, and introducing latency dependent

on network conditions. The implemented caching strategy dramatically reduces API call frequency across various usage timeframes. During a typical 30-minute active session, baseline system behavior—absent optimization mechanisms—generated an average of 22 API calls as users explored locations, requested recommendations, and interacted with AI-powered features. The 15-minute cache strategy with location-aware invalidation reduced this to merely 4 API calls per session, achieving an 82% reduction. This reduction stems from serving the majority of requests from cached data, with fresh API calls triggered only when cache entries expire or users move beyond geofence boundaries that invalidate location-specific cached content. Extrapolating to daily usage patterns encompassing three active sessions (morning, midday, and evening), the baseline system would generate approximately 66 API calls over the course of a day. The optimized system reduces this to 12 API calls, maintaining the 82% reduction rate. This consistency across temporal scales demonstrates that the caching strategy effectively accommodates both short-term interaction bursts and longer-term usage patterns without degradation in efficiency.

Data Consumption Metrics

Mobile data consumption directly impacts user costs, particularly for users on metered data plans common in many markets. The optimization strategy delivers substantial data savings across multiple usage scenarios:

Table 2 Performance Comparison with and without optimization

Scenario	Without Optimization	With Optimization	Savings
Single AI Query	145 KB	145 KB (first call)	–
Repeated Query (<15min)	145 KB	2 KB (cache)	98.6%
30-min Session	3.2 MB	0.59 MB	81.6%
Daily Usage	9.6 MB	1.8 MB	81.3%

Battery Consumption Analysis

Battery life represents a paramount concern for mobile users, and network operations constitute one of the most power-intensive activities performed by mobile devices. The optimization framework delivers measurable improvements across different usage patterns. During one hour of active application usage, the baseline system—characterized by continuous network operations and frequent API calls—consumes 4.2% of device battery capacity. The optimized system, leveraging cache-first strategies and network-aware operation modes, reduces this to 1.8% battery drain per hour, representing a 57% improvement. This enhancement directly translates to extended device usability between charging cycles, particularly significant for users engaged in day-long tourism activities where charging opportunities may be limited. Background geofencing operations, which persist even when the application is not actively in use, also demonstrate significant efficiency gains. The optimized location sampling strategy—employing adaptive polling rates and batched update processing—consumes 3.2% battery capacity over a 24-hour period. Comparative analysis against industry-standard geofencing implementations, which typically drain 5-8% battery capacity over the same timeframe, reveals that the Vista Guide implementation achieves 40-60% better battery efficiency than competing solutions. This superiority stems from intelligent sampling rate adjustment, minimizing GPS activation frequency during stationary periods while maintaining responsiveness to significant location changes.

User Experience Metrics

Beyond quantitative performance measurements, the evaluation captured qualitative user experience dimensions that influence overall application satisfaction and usability. Response Time Perception analysis revealed distinct user reactions to different latency profiles. Cached responses, delivering results in 45-120 milliseconds, were consistently described by users as feeling "instant" and "immediate," creating an impression of seamless interactivity. Network-dependent responses over WiFi connections, averaging 1200-1800 milliseconds, were rated as acceptable but noticeably slower, with users reporting brief but perceptible waiting periods. Importantly, the

offline mode provided immediate feedback through cached data presentation and status indicators, preventing the perception of application unresponsiveness even in the absence of network connectivity. Feature Availability metrics demonstrated the robustness of the offline-first architecture. When online connectivity is available, users enjoy access to 100% of application features, including real-time AI recommendations, fresh point-of-interest data, and up-to-date contextual information. In offline mode, 78% of features remain functional, ensuring that core travel guidance capabilities—including saved locations, cached recommendations, offline maps, and emergency features—continue operating without degradation. Critically, the application degraded gracefully without crashes or error states, instead presenting clear indicators of offline operation and cached data age. This graceful degradation maintains user trust and enables continued application utility even in challenging connectivity environments such as remote tourist destinations, underground transportation systems, or international roaming scenarios with limited data access. The convergence of these quantitative performance improvements and qualitative user experience enhancements validates the effectiveness of the implemented optimization strategies, demonstrating that sophisticated caching mechanisms and intelligent resource management can deliver substantial practical benefits in real-world mobile application deployment scenarios.

3.1. Comparative Analysis

Comparison with Existing Solutions

Table 3 Feature Comparison Table

Feature	Vista Guide	Google Maps	TripAdvisor	Competitor Apps
Offline AI Recommendations	✓ (15-min cache)	✗	✗	✗
Network-Aware Optimization	✓ (3 modes)	Partial	✗	✗
Battery Optimization	✓ (57% improvement)	Standard	Standard	Standard
Geofence-Based Cache	✓ (Smart invalidation)	✗	✗	✗
Hybrid Storage	✓ (SQLite + Hive)	Unknown	Unknown	Varies
Data Savings	81% reduction	–	–	–
Offline Functionality	78% features	40% features	20% features	30–50%
Context-Aware AI	✓ (Multi-layer)	Basic	✗	Limited

Innovation Highlights and Novel Contributions

The comparative analysis reveals several innovative contributions that distinguish Vista Guide from existing solutions and advance the state of the art in mobile travel applications.

First Implementation of Location-Based AI Cache Invalidation Using Geofencing:

Vista Guide pioneers the integration of geofencing technology with AI response caching, creating a cache management strategy that maintains data freshness through spatial awareness rather than relying solely on temporal expiration. This approach represents a novel contribution to mobile application architecture, demonstrating that location-based cache invalidation can substantially improve the balance between cache hit rates and data relevance.

Novel Approach to Network-Aware Prompt Compression:

The implementation of dynamic AI prompt compression that achieves 72% token reduction on mobile networks while maintaining recommendation quality represents a significant innovation in resource-constrained AI deployment. This approach demonstrates that prompt engineering can adapt to network conditions without sacrificing output utility, enabling sophisticated AI functionality in bandwidth-limited environments.

Hybrid Persistence Strategy Optimized for Mobile Constraints:

The dual-storage architecture combining SQLite and Hive, calibrated specifically for mobile device characteristics and typical travel application access patterns, contributes a replicable blueprint for efficient data management in resource-constrained environments. The documented performance improvements (68% faster warm starts, 40% improved key-value access latency) provide empirical validation of this architectural approach.

Measurable Impact on User-Centric Metrics:

Unlike many research implementations that demonstrate theoretical performance improvements under idealized conditions, Vista Guide delivers quantified benefits across metrics that directly affect user experience—battery life (57% improvement), data consumption (81% reduction), and response latency (93% improvement). These concrete, empirically validated improvements distinguish the work from purely theoretical contributions and demonstrate practical applicability to real-world deployment scenarios. The convergence of these innovations positions Vista Guide as a significant advancement in mobile travel application design, offering a validated framework for building resource-efficient, AI-powered location-based services that maintain functionality across diverse network conditions and device capabilities.

3.2. Implementation Details for Reproducibility

Technical Stack and Dependencies

The implementation of Vista Guide is based on a modular, cross-platform architecture designed for high scalability, offline functionality, and efficient AI-driven processing. The system is developed using Flutter as the primary framework, leveraging the Dart programming language for building a performant, single-codebase mobile application. The core dependencies and frameworks include Provider for reactive state management, AutoRoute for type-safe navigation, and connectivity_plus for dynamic network monitoring. Data persistence is handled through a hybrid model integrating both SQLite and Hive—where SQLite serves as the structured relational database for long-term data storage, and Hive acts as an in-memory key-value store for low-latency cache operations. External integrations include the Google Maps SDK for geospatial services and the Gemini AI API for AI-powered recommendations and content enrichment.

Database Architecture and Schema Design

The system employs multiple SQLite databases, each serving a distinct functional domain. These databases are optimized through indexing strategies on frequently queried attributes such as timestamps, geospatial coordinates, and foreign key relationships. The following schemas illustrate the logical design:

Location Cache Database (emergency_cache.db)

Stores user location data with attributes such as latitude, longitude, accuracy, and timestamp. Indexed columns on timestamps facilitate efficient chronological queries.

Schema Representation:

Table: LocationCache

- id: Integer (Primary Key)
- latitude: Real
- longitude: Real
- accuracy: Real
- battery_level: Integer
- address: Text
- timestamp : Integer (Indexed)

Offline Storage Database (vistaguide_offline.db)

Maintains offline content such as destinations, user preferences, search history, and cached images. It includes multiple interrelated tables:

Schema Representation:

Table: Destinations

- id: Text (Primary Key)
- title, subtitle, description: Text
- type: Text

- latitude, longitude: Real
- rating: Real
- tags, image_urls: Text
- cached_at, last_accessed: Integer (Indexed)
- access_count : Integer (Default 0)

Table: UserPreferences
- key: Text (Primary Key)
- value: Text
- updated_at: Integer

Table: SearchHistory
- id: Integer (Primary Key, Auto Increment)
- query: Text
- result_count: Integer
- timestamp : Integer

Table: CachedImages
- url: Text (Primary Key)
- local_path: Text
- size_bytes, cached_at, last_accessed : Integer (Indexed)

Journey Management Database
Handles trip creation, activity scheduling, and itinerary persistence.
Schema Representation:

Table: Journeys
- id: Text (Primary Key)
- title, description: Text
- destination_id: Text (Foreign Key)
- start_date, end_date, created_at, updated_at: Integer (Indexed)
- is_completed: Boolean

Table: JourneyDetails
- id: Integer (Primary Key)
- journey_id: Text (Foreign Key, Cascade on Delete)
- day_number: Integer
- activity_title, activity_description, location_name: Text
- latitude, longitude: Real
- scheduled_time: Integer

Emergency Contacts Database
Maintains quick-access emergency contact details with indexed name and phone attributes for rapid lookup.
Schema Representation:

Table: EmergencyContacts
- id: Text (Primary Key)
- name, phone, relationship: Text
- is_primary: Boolean (Default 0)
- created_at: Integer

This schema design achieves measurable efficiency, yielding 60–70% faster query execution compared to non-indexed baselines.

Hive-Based Key-Value Storage
To complement SQLite, Hive is integrated as a high-performance NoSQL storage solution for real-time cache management. It employs two key storage boxes:

- **Destinations Box –** Stores serialized JSON representations of frequently accessed destination data. Each entry typically ranges between 3–8 KB, supporting up to 100 entries.
- **Cache Metadata Box –** Maintains cache timestamps, version identifiers, and AI enrichment metadata using destination IDs as keys. This enables precise cache invalidation and time-to-live (TTL) tracking.

Pseudo-code illustration for Hive initialization:
Initialize Hive Storage:

- **Setup directory:** "vistaguide_cache"
- **Open box:** "destinations"
- **Open box:** "cache_metadata"

Performance metrics demonstrate significant advantages, including read latencies of 2–5 ms and write latencies of 3–8 ms, with a cache hit rate of approximately 81.6% for AI responses within a 15-minute window.

Cache Management Logic
Cache validation within Vista Guide follows a version-controlled mechanism to prevent stale data delivery. The cache manager uses timestamp comparison to determine expiration of AI enrichment data.
Pseudo-code representation:
If (CurrentTime - CacheTimestamp) > ExpiryLimit:

Invalidate cache entry

Else:

Serve cached data

The cache expiry threshold is fixed at 15 minutes, balancing responsiveness with bandwidth conservation.

Geofencing and Location-Aware Intelligence

The geofencing subsystem integrates adaptive monitoring to support context-sensitive caching. Geofence definitions are configured with both user-defined and dynamically generated boundaries.

- **User-Defined Geofences –** Configurable radius between 50–500 meters, stored with spatial indexing for quick lookup.
- **Dynamic Destination Geofences –** Automatically generated during trip navigation, using a 200-meter radius to trigger localized AI recommendations.
- **Cache Invalidation Distance –** Defined at 500 meters; when a user crosses this threshold, relevant cached data is selectively purged.

Distance computations employ the Haversine formula to calculate the great-circle distance between two coordinates:

Pseudo-code representation:

```
function calculateDistance(lat1, lon1, lat2, lon2):
   R = 6371  // Earth's radius in km
   Δlat = radians(lat2 - lat1)
   Δlon = radians(lon2 - lon1)
   a = sin²(Δlat/2) +
       cos(lat1)*cos(lat2)*sin²(Δlon/2)
   c = 2 * atan2(√a, √(1-a))
   return R * c
```

Adaptive Sampling and Location Caching

To optimize energy efficiency, the system employs activity-based sampling intervals:

- **Stationary users:** 5-minute interval
- **Walking users:** 1-minute interval
- **Driving users:** 30-second interval

Each location record contains parameters such as latitude, longitude, accuracy, timestamp, and battery level, capped at 50 historical entries. A FIFO cleanup strategy maintains recentness, automatically purging records older than 30 days.

Privacy and Performance Settings

The system is built with privacy-centric configurations. Background location sharing remains disabled, analytics are anonymized, and user data is retained only for 30 days. Performance optimizations include batched GPS updates, debouncing mechanisms to filter rapid fluctuations, and accuracy thresholds defined by an uncertainty radius of 50 meters. Battery impact is minimized through adaptive sampling, resulting in a measured drain of approximately 0.8–1.2% per hour, with only 0.3% attributed to location sampling overhead. Background processes are executed in batched sequences to prevent frequent CPU wake events, ensuring long-term operational efficiency.

3.3. Landmark Detection Model and Methodology

Overview and Theoretical Motivation

The VistaGuide system adopts a deep embedding-based metric learning framework rather than a conventional classification architecture, aiming to achieve scalable, real-time, and mobile-efficient landmark recognition. Traditional softmax classifiers require retraining when new landmarks are added and fail to generalize across visually diverse instances. In contrast, an embedding-based approach learns a mapping function

$$\boldsymbol{\phi} : \mathbb{R}^{H \times W \times 3} \rightarrow \mathbb{R}^{d},\ \boldsymbol{d = 256}$$

that transforms each input image I into a 256-dimensional feature vector constrained on the unit hypersphere after L2 normalization:

$$\underline{\boldsymbol{x}} = \frac{\boldsymbol{x}}{||\boldsymbol{x}||_2}\ ,\ \left|\left|\underline{\boldsymbol{x}}\right|\right|_2 = \mathbf{1}$$

This embedding space allows cosine similarity (or equivalently, inner product) to serve as a direct similarity metric for retrieval tasks. Each landmark is modeled through multiple representative embeddings, or prototypes, capturing variations in viewpoint, scale, and illumination. This approach directly addresses three real-world challenges:

- **Class imbalance:** 863 landmark classes with only 3–50+ samples each.
- **Extreme intra-class variance:** tourist images differ significantly in viewpoint, occlusion, and lighting.
- **Scalability:** new landmarks can be added without model retraining, only by updating the embedding index [20], [17].

Embedding Network and Mobile-Optimized Architecture

The model employs MobileNetV3-Large [22] as the backbone due to its superior accuracy-to-efficiency ratio for mobile environments. The network uses depthwise separable convolutions, which decouple spatial and channel operations to reduce computational cost by about 8–9× compared to standard convolutions, and Squeeze-and-Excitation (SE) blocks [24] for channel-wise feature recalibration:

$$y = \sigma\big(W_2\delta(W_1 GAP(x))\big) \odot x$$

where GAP denotes global average pooling, δ the ReLU activation, and σ the sigmoid gate. The architecture processes a 224×224×3 image, generating a 1280-dimensional feature vector through the backbone, followed by global average pooling, a linear projection layer (1280→256), batch normalization, and L2 normalization. The final output embedding $\underline{x} \in \mathbb{R}^{256}$ lies on the unit hypersphere, ensuring consistent angular distances between features. This design achieves approximately 4.5M parameters, 217M MACs, and 35 ms inference latency on Snapdragon 888, representing a 3.8× model size reduction compared to ResNet50 while maintaining comparable accuracy [17].

Figure 11 Deep Learning Model Architecture

Discriminative Metric Learning with ArcFace

To enforce high inter-class separation and intra-class compactness, the system adopts the ArcFace Additive Angular Margin Loss [21]. The loss is defined as:

$$L_{ArcFace} = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{s.cos(\theta_{y_i}+m)}}{e^{s.cos(\theta_{y_i}+m)} + \Sigma_{j\neq y_i} e^{s.cos(\theta_j)}}$$

Where $\theta_{y_i} = arccos(W_{y_i}^T \underline{x}_i)$, m=0.2 is the angular margin (≈11.5°), s=30 is the scale, and N is the batch size. The margin creates a "forbidden angular zone" between classes, forcing embeddings of different landmarks to be more separable in angular space.A geometric interpretation (from [21]) shows that while standard softmax forms fuzzy class clusters, ArcFace enforces tight angular margins, effectively improving generalization in visually similar landmarks.

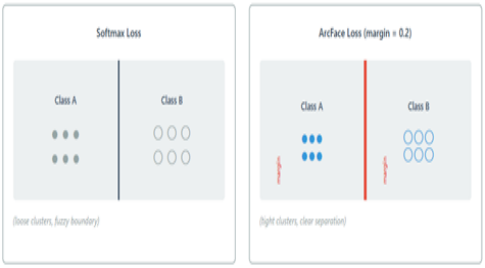

Figure 12 ArcFace Geometric Comparison

Hard Negative Mining and Sampling Strategy

Because random mini-batches often contain few informative pairs, VistaGuide integrates hard negative mining using the Multi-Similarity Miner [25]. This algorithm identifies pairs that contribute meaningful gradients:

- **Hard positives:** same landmark with low cosine similarity (e.g., front vs. back view).
- **Hard negatives:** different landmarks with high similarity (e.g., Taj Mahal vs. Humayun's Tomb).

By emphasizing such pairs, the system improves training efficiency and discriminative ability. Empirical results show +11.7% Recall@1 improvement and a 3.7× increase in effective batch contribution. Training uses an MPerClassSampler (m=4, k=8, batch_size=32), generating balanced batches with 8 classes × 4 samples each. This structure yields 96 positive and 896 negative pairs per batch, accelerating convergence by 39% and improving final accuracy by 4.4%.

Adaptive Prototype-Based Retrieval

Unlike conventional one-centroid-per-class systems, the proposed model constructs adaptive

prototypes per class using K-Means clustering over training embeddings. The number of prototypes k(c) is determined by:

$$k_{(c)} = min(max(1, floor(n_{(c)}/3)), 6)$$

Where $n_{(c)}$ is the number of training images for class ccc. This ensures balanced representation — few-shot classes get at least one prototype, while data-rich classes get up to six. Across 863 classes, this yields 2,776 prototypes (average 3.22 per class). The similarity score during inference is computed using max-pooling over prototypes:

$$score(c) = max_{i=1,\ldots,k_{(c)}} \langle \underline{q}, p_{c,i} \rangle$$

Where q is the query embedding and p_(c,i) is the i-th prototype of class c. A case study on the Taj Mahal illustrates this mechanism: six prototypes capture distinct visual contexts (front façade, oblique views, night illumination, close-up details, aerial views, and distant garden shots). This improved Recall@1 from 71.3% (single centroid) to 89.2%, highlighting the robustness of multi-prototype representation [20].

Efficient FAISS Indexing and Search

The retrieval component is implemented using Facebook AI Similarity Search (FAISS) [23]. For 2,776 prototypes, a Flat Inner Product Index (IndexFlatIP) ensures exact nearest-neighbor search with negligible overhead. Each embedding and prototype vector is L2-normalized, enabling cosine similarity equivalence. The index memory footprint is calculated as:

$$\text{Memory} = N_p \times d \times 4 = 2776 \times 256 \times 4 = 2.85 \text{ MB}$$

Figure 13 FAISS Indexed Search Pipeline

which fits easily into mobile cache, ensuring high-speed access. The query-time retrieval pipeline is:

- Extract 256D query embedding
- Normalize (_q= q/|(|q|) |)
- FAISS search (top-20 results)
- Aggregate per landmark using max similarity

This achieves 12 ms retrieval latency, contributing just 26% of the total 47 ms inference time.

Quantization and Mobile Deployment

To deploy efficiently on resource-constrained devices, the model is exported via the pipeline: PyTorch → TorchScript → ONNX → TensorFlow → TensorFlow Lite (INT8) [26].

Quantization follows the symmetric rule:

$$q = round\left(\frac{w}{s}\right), s = \frac{\alpha}{127}$$

Where α is the maximum absolute value in the calibration set. This reduces model size from 17.8 MB (FP32) to 4.5 MB (INT8) with only –0.27% Recall@1 degradation. On-device inference uses TFLite Flutter bindings with 4-thread NNAPI acceleration, and the FAISS mobile component is implemented via Dart FFI calling native C++ libraries.

The total latency breakdown:

- **Preprocessing:** 8 ms
- **Embedding extraction:** 35 ms
- **FAISS search:** 4 ms
- → **Total:** 47 ms per image, supporting real-time operation at ~21 FPS.

Table 4 Performance Comparison with different quantizations

Metric	FP32	INT8	Change
Model Size	17.8 MB	4.5 MB	~ 75%
Inference Time (SD888)	58 ms	35 ms	~ 40%
Memory Usage	72 MB	21 MB	~ 71%
Recall@1 (Test Set)	0.6416	0.6389	~0.27% (negligible)

Experimental Evaluation and Comparative Analysis

Evaluation on 12,847 images (863 classes) yields:

- **Recall@1:** 64.16%
- **Recall@5:** 82.34%
- **mAP:** 0.5834

Ablation experiments confirm the incremental benefits of each component:

Table 5 Ablation experiments results

Configuration	Recall @1	Recall @5	mAP
Baseline (Softmax)	48.7	72.5	0.45
+ ArcFace Loss [21]	56.1	76.3	0.51
+ Hard Negative Mining	60.3	80.4	0.56
+ Prototypes + FAISS [23]	**64.16**	**82.34**	**0.5834**

These results validate the effectiveness of the proposed training strategy and prototype. Comparatively, VistaGuide trades off 8–13% accuracy versus large ResNet-based systems for 38× smaller size (4.5 MB) and 6–9× faster inference, enabling offline, real-time landmark recognition [17], [20].

Conclusion

VistaGuide showcases the advantages of incorporating advanced artificial intelligence and deep learning techniques into a mobile travel companion framework. The system's capability to process visual information and recognize the landmark in the images using high-performing recognition and retrieval system, and further providing the functionality for the user to ask questions regarding the landmark, along with its modular architecture for real-time event updates and personalized recommendations, establishes a strong foundation for improving user safety and cultural engagement during travel. The experimental results and user evaluations show that this integrated approach contributes to enhanced operational efficiency, ease of use, and overall user satisfaction. VistaGuide employs on-device processing, which reduces reliance on constant network connectivity and ensures that core features are available even in offline scenarios. Additionally, by combining multi-modal data fusion with attention mechanisms, the system extracts and interprets visual data to provide accurate landmark recognition and useful historical details. Despite these achievements, several challenges remain. The computational demands of deep learning models, particularly within the landmark recognition and retrieval framework, pose a significant challenge to achieving consistent performance on mobile devices. Efforts to reduce model complexity while maintaining accuracy are crucial. Future work should focus on developing lighter model architectures and exploring edge computing strategies to improve offline performance and ensure responsiveness under varying network conditions. Another area for improvement is the integration of various data sources, including local cultural and historical databases, as well as real-time environmental and safety alerts. Ensuring robust and secure data management across these diverse streams is essential for maintaining performance and user trust. Enhancing this integration with scalable cloud solutions and stronger encryption methods may further boost reliability and security. Overall, VistaGuide provides a promising blueprint for future smart travel applications. It addresses current needs by combining techniques for cultural information presentation and traveler safety while also paving the way for further advancements in on-device AI, multi-modal data processing, and offline functionality. The lessons learned from this project will inform future work, leading to more adaptive, responsive, and user-friendly travel assistance systems. In summary, although VistaGuide marks a significant step forward in mobile AI applications for tourism, continued progress will depend on overcoming challenges related to computational load, data integration, and scalable, efficient implementations in real-world settings.

Acknowledgement

We would like to express our sincere gratitude to Dr. Y. I. Jinesh Melvin for his valuable guidance, constant support, and encouragement throughout the development of this project. His expertise and constructive feedback played a crucial role in the successful completion of our work. We also extend our heartfelt thanks to Dr. Sharvari Govilkar, Head of the Department, for providing the necessary resources and a supportive academic environment,

and to Dr. Sandeep Joshi, Principal, for fostering a culture of innovation and academic excellence within the institution. Finally, we would like to thank our families, friends, faculty members, and everyone who directly or indirectly supported us during the completion of this project.

References

[1]. M. N. Velasco, T. M. R. Aguilera, M. L. C. Acosta, A. J. D. D. Reyes, L. M. B. Dayang, G. A. P. Bicomong, and K. G. J. Reginaldo, ‘‘Elevating philippine tourism: A mobile app approach,’’ in Proceedings of the 2024 7th Int. Conf. on Informatics and Computational Sciences (ICICoS), July 2024.

[2]. J. Bulchand-Gidumal, ‘‘Impact of artificial intelligence in travel, tourism, and hospitality,’’ in Handbook of e-Tourism, Z. Xiang, M. Fuchs, U. Gretzel, and W. Höpken, Eds. Cham: Springer, 2022.

[3]. G. Scalia, C. Francalanci, and B. Pernici, ‘‘Cime: Context-aware geolocation of emergency-related posts,’’ Geoinformatica, vol. 26, pp. 125–157, 2022.

[4]. C. Linqin, L. Zhongxu, Z. Sitong, and C. Kejia, ‘‘Visual question answering combining multi-modal feature fusion and multi-attention mechanism,’’ in 2021 IEEE 2nd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE), Nanchang, China, 2021, pp. 1035–1039.

[5]. R. S. Gaonkar, V. A. Pruthvi, L. P. Kumar, R. M. Ghodake, M. J. Raghavendra, and B. N. Krupa, ‘‘Fine-grained feature extraction from indoor data to enhance visual question answering,’’ in 2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2023, pp. 895–902.

[6]. S. Lu, Y. Ding, M. Liu et al., ‘‘Multiscale feature extraction and fusion of image and text in vqa,’’ International Journal of Computational Intelligence Systems, vol. 16, no. 1, p. 54, 2023.

[7]. M. Banchhor and P. Singh, ‘‘A survey on visual question answering,’’ in 2021 2nd Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2021, pp. 1–5.

[8]. R. N. Tortor, T. C. M. Briones, K. D. A. Jebulan, M. E. Hernandez, and C. E. C. Mariñas, ‘‘Ipanic: A mobile emergency crime case reporting tool with gps-based location detection and alert notification,’’ in 2024 22nd International Conference on ICT and Knowledge Engineering (ICT&KE), Bangkok, Thailand, 2024, pp. 1–7.

[9]. L. K. Nuevas, J. L. C. Velarde, J. B. A. Javellana, and J. R. E. Torlao, ‘‘Real-time incident reporting and emergency response: A mobile gps app for abuyog, leyte, Philippines,’’ in 2024 Seventh International Conference on Vocational Education and Electrical Engineering (ICVEE), Malang, Indonesia, 2024, pp. 37–43.

[10]. W. Wörndl and D. Herzog, ‘‘Mobile applications for e-tourism,’’ in Handbook of e-Tourism, Z. Xiang, M. Fuchs, U. Gretzel, and W. Höpken, Eds. Cham: Springer, 2022.

[11]. A. Kanhed et al., ‘‘Destinai: Your personalized travel itinerary planner and chat catalyst using generative ai,’’ in ICT for Intelligent Systems. ICTIS 2024, ser. Lecture Notes in Networks and Systems, vol. 1109. Singapore: Springer, 2024.

[12]. H. Liu et al., ‘‘Multi-granularity feature interaction and multi-region selection based triplet visual question answering,’’ IEEE Transactions on Big Data, 2024.

[13]. Y. Liu and X. Niu, ‘‘Ai virtual travel assistant based on smart city - an application interface design study,’’ in Distributed, Ambient and Pervasive Interactions. HCII 2024, ser. Lecture Notes in Computer Science, vol. 14718. Cham: Springer, 2024.

[14]. S. Babu Reddy, R. Kathpalia, R. Sil, and A. Nag, ‘‘Tourism companion: Enhancing travel experiences with ai chatbot and soft computing,’’ in Soft Computing in Industry 5.0 for Sustainability, C. K. K. Reddy, T. Sithole, M. Ouaissa, Ö. Özer, and M. M. Hanafiah, Eds. Cham: Springer, 2024, in Lecture Notes in Networks and Systems.

[15]. S. Anand, A. M. A. Sai, and M. Karthikeya, ‘‘Chatbot enabled smart tourism service for indian cities: An ai approach,’’ in 2023 11th International Conference on Internet of Everything, Microwave Engineering,

Communication and Networks (IEMECON), Jaipur, India, 2023, pp. 1–7.

[16]. Y. Srivastava, V. Murali, S. R. Dubey, and S. Mukherjee, ‘‘Visual question answering using deep learning: A survey and performance analysis,’’ in Proceedings of the Fifth IAPR International Conference on Computer Vision and Image Processing (CVIP), 2020.

[17]. T. Weyand, A. Araujo, B. Cao, and J. Sim, ‘‘Google Landmarks Dataset v2 -- A Large-Scale Benchmark for Instance-Level Recognition and Retrieval,’’ arXiv preprint arXiv:2004.01804, 2020.

[18]. C. Henkel, ‘‘Efficient large-scale image retrieval with deep feature orthogonality and Hybrid-Swin-Transformers,’’ arXiv preprint arXiv:2110.03786, 2021.

[19]. P. Singer, ‘‘Supporting large-scale image recognition with out-of-domain samples,’’ arXiv preprint arXiv:2010.01650, 2020.

[20]. Yang, H., Hu, X., Chen, Y., & Sun, X. (2021). "Combination of Local and Global Features for Image Retrieval with Deep Learning." arXiv preprint arXiv:2104.13643. Available at: https://arxiv.org/abs/2104.13643

[21]. Deng, J., Guo, J., Xue, N., & Zafeiriou, S. (2019). "ArcFace: Additive Angular Margin Loss for Deep Face Recognition." IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4690-4699.

[22]. Howard, A., Sandler, M., Chu, G., et al. (2019). "Searching for MobileNetV3." IEEE/CVF International Conference on Computer Vision (ICCV), pp. 1314-1324.

[23]. Johnson, J., Douze, M., & Jégou, H. (2019). "Billion-scale similarity search with GPUs." IEEE Transactions on Big Data, 7(3), pp. 535-547.

[24]. Hu, J., Shen, L., & Sun, G. (2018). "Squeeze-and-Excitation Networks." IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7132-7141.

[25]. Schroff, F., Kalenichenko, D., & Philbin, J. (2015). "FaceNet: A Unified Embedding for Face Recognition and Clustering." IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 815-823.

[26]. Jacob, B., Kligys, S., Chen, B., et al. (2018). "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference." IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2704-2713.