Intelligent Reasoning Systems
Final Project Report

Project Name : MyTrail

Group Number: 5
Group Member:

Zheng Jiecheng, A0290838R
Zhao Ziyang, A0285923U
Jin Ziping, A0263234L
Zhao Yuan, A0263565W

Menu

1. Introduction

MyTrail is an application for exploring cities, recording leisurely walks or outdoor walking tracks. This application will generate routes based on some ideas proposed by users through intelligent reasoning and unique optimization models, and quickly display them in the built-in Google Maps. Users can then select any route created by the intelligent system and start exploring the surrounding area. Since each location is generated in real time through the Google API, MyTrail can avoid outdated locations. In addition, user route selection and feedback are also data for training the inference model, which means that more and more user data can make the model smarter.

2. Business Case

2.1. Background

Outdoor walking and running apps are widely adopted as health awareness grows. In places like Singapore—where dense urban fabric interleaves with parks and waterfronts—people frequently want novel, intent-aligned routes rather than static lists of attractions. Existing fitness apps focus on tracking and performance; lifestyle apps list venues but rarely transform free-form user intent into a closed-loop, non-crossing path.

2.2. Market Context

- **AI-assisted experiences**: Consumers increasingly expect intelligent, intent-aware planning.
- **Experiential fitness**: Demand is shifting from pure performance to novelty and discovery.
- **Geospatial integration**: Real-time POIs and environmental context are becoming standard in lifestyle apps.

2.3. User Pain Points

There are many apps on the market that offer content for exploring the surrounding area, but there is a gap in the field of combining route planning and generating routes based on user input. In particular, there are very few apps that help users explore the surrounding environment and combine intelligent systems to assist in decision-making. Users need to simplify the process of exploring and starting outdoor activities.

- Lack of generative, exploration-oriented route creation.
- No intelligent ranking when multiple candidate routes exist.
- Systems rarely understand user intent; they force fixed patterns or manual filters.

2.4. Competitive Landscape

There are two main categories of competing products on the market: sports apps and exploration apps:

- **Strava, Keep, etc.:** Excellent at recording workouts, less focused on **prompt-to-route** exploration.
- **Meituan-like lifestyle apps:** Great for nearby POIs, but **not** for composing POIs into a **coherent walking loop** tuned to a user's intent

MyTrail implements the workflow of user input-route generation, simplifies the user's operation complexity, and provides the function of converting user input into content that the system can understand, sorting and optimizing routes. Each time the user will get a different route, which may include parks, commercial areas, beaches, etc.

2.5. Target Users

MyTrail has a lightweight mobile client that is currently in service and operation in Singapore. Its target user groups are: new residents or tourists in Singapore, people who want to explore the city in Singapore, and those who enjoy outdoor activities.

- **Initial users:** New residents and visitors in Singapore who enjoy casual outdoor exploration. Satisfy the user needs of random exploration and store "blind box opening"
- **B2B expansion:** Promote POIs and solve **long-tail recommendations** by exposing lesser-known attractions via intent-matched routes.

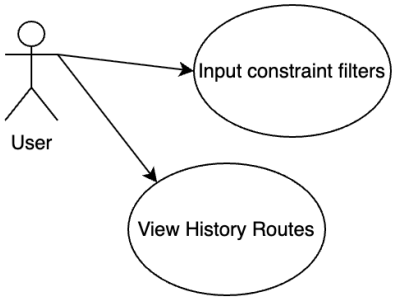
After some research, we learned that users have strong interest in the application areas of the MyTrail App and are optimistic about its integration with artificial intelligence systems. Users hope to have a seamless experience of inputting text and getting solutions.

3. System Design

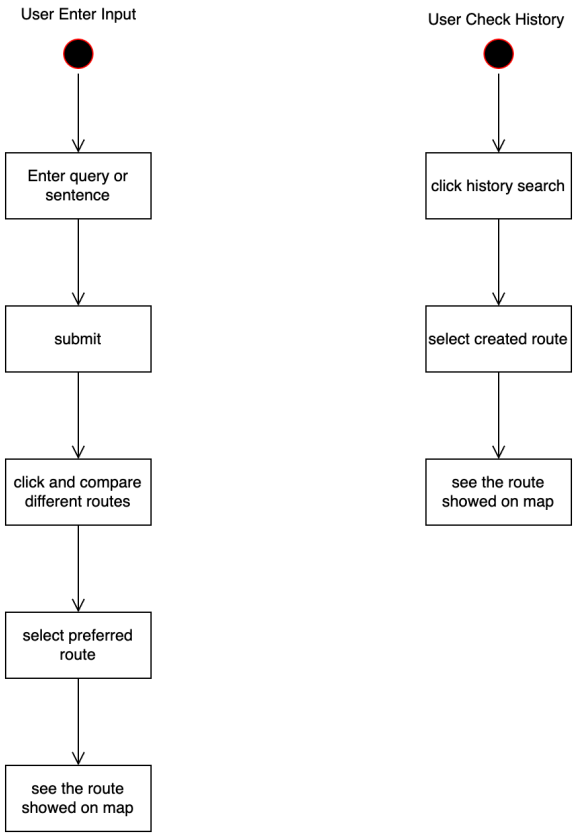
3.1. User Experience

1. Users can directly view nearby surrounding information in the embedded Google map, and can enter prompts in the search box
2. Backend returns several diverse, closed-loop routes with duration, distance, and POI highlights
3. Users tap `start` button to see turn-level details; the app records the chosen route for learning.
4. The last 10 searches are available locally for quick replay

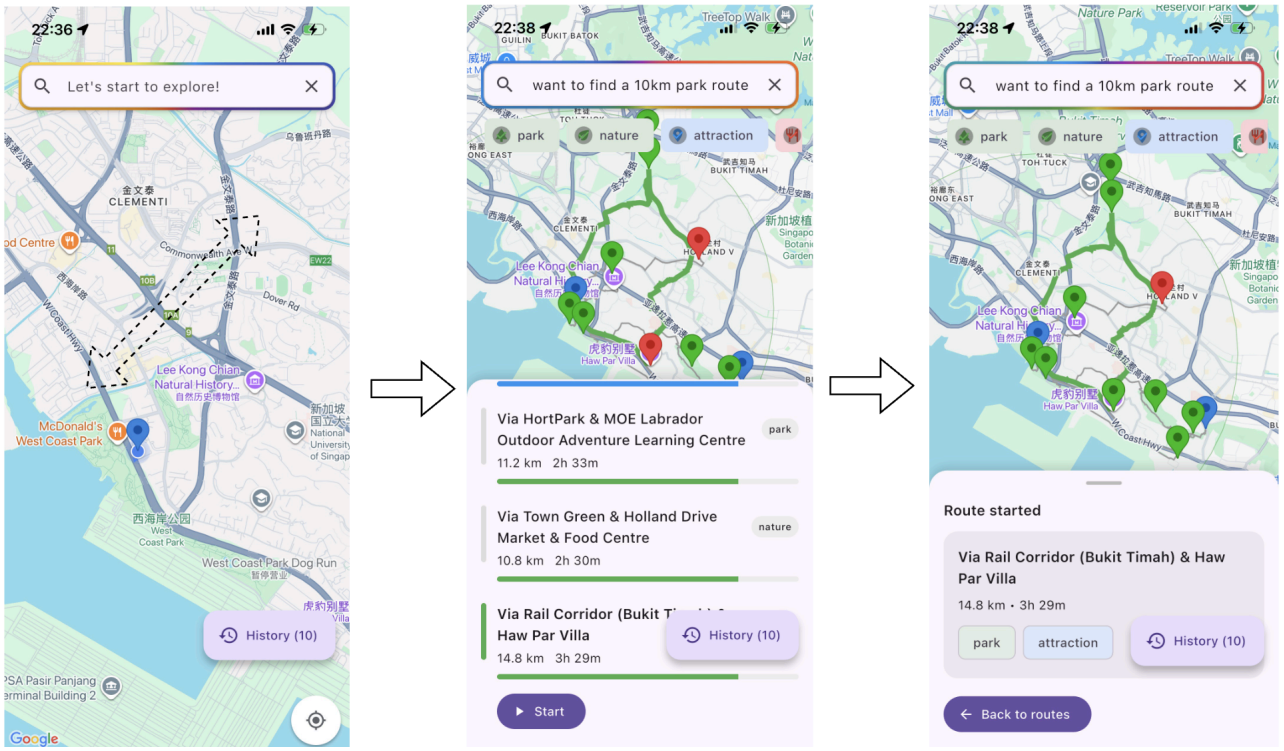
Picture 3.1: Use Case



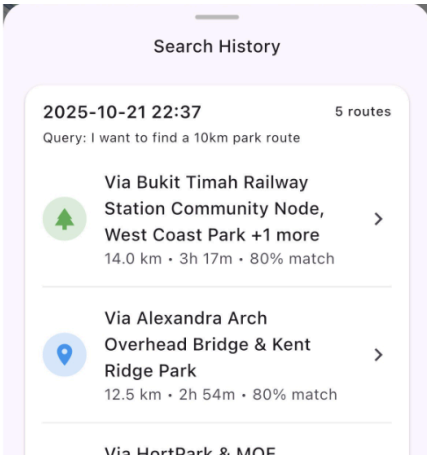
Picture 3.2: Activity Diagram



Picture 3.3: Screenshots from MyTrail App（Find Routes）



Picture 3.3: Screenshots from MyTrail App（View History）



3.2. Architecture

MyTrail is a full-stack project with a complete front-end and back-end platform and microservice deployment server.

- **Mobile Client (Flutter)**: Cross-platform; embedded Google Map (zoom, locate, draw polylines).
- **Backend (FastAPI)**: REST services; controllers delegate to NLP and route services. FastAPI services with clear controller/service boundaries and typed pydantic DTOs
- **ML Microservice (Flask/FastAPI)**: Decoupled NLU and ranking models for hot-swaps and independent scaling. Flask/FastAPI loading model and tokenizer configs; consistent tokenization/alignment with training.
- **Data Stores**: MongoDB for route logs, user choices, and training artifacts.

3.3. Machine Learning & Optimization Algorithm

This project has two machine learning and two optimization algorithms:

3.3.1. Natural-Language Understanding (NLU)

Goal: Convert a short prompt into a **normalized JSON** specification consumable by the route generator.

Example Input Prompt:

```
{
  "text": "I want to find a park route."
}
```

Target JSON:

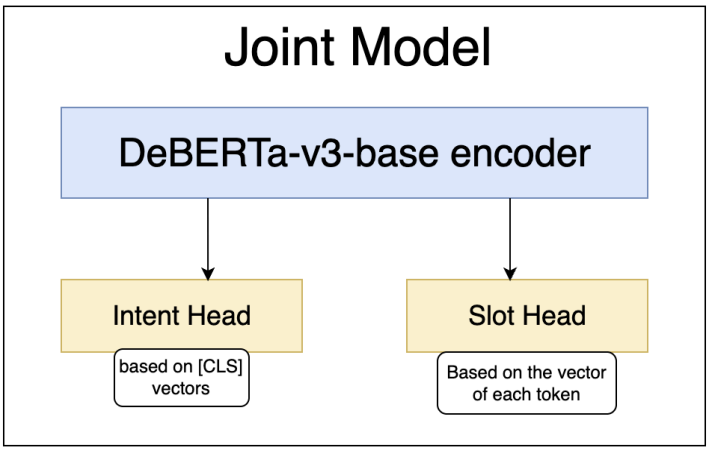
```
{
    "intent": {
        "confidence": 0.6356122493743896,
        "label": "route_find"
    },
    "slots": [
        {"label": "O","word": "I"},
        {"label": "O","word": "want"},
        {"label": "O","word": "to"},
        {"label": "O","word": "find"},
        {"label": "O","word": "a"},
        {"label": "B-CAT_INC","word": "park"},
        {"label": "O","word": "route"}
    ]
}
```


Model Architecture:

We adopt a Joint Intent–Slot model with a shared encoder (DeBERTa-v3-base) and two heads:

- Intent Head: single-label classification over a small ontology (e.g., route_type, theme intent).
- Slot Head: token-level BIO tagging for spans such as distance ranges, time windows, location hints, and POI categories.

Picture 3.4: Model Structure



Loss

$$L = CE(intent) + \lambda \cdot CE(BIO\ slots)$$

with λ balancing slot precision vs. intent accuracy.

Training Setup:

- Data: ~200 cold-start prompts bootstrapped from an LLM, then lightly curated. Future: continuously augment with user prompts.
- Split: 90/10 (train/test).
- Optimization: AdamW; smaller LR for encoder, larger for heads.
- Metrics: Intent accuracy; slot F1 via seqeval; latency and throughput (p50/p95).

Picture 3.5: Evaluation Result

```
Final evaluation on data/test.jsonl:
Task   | Metric           |  Value
-------+------------------+-------
intent | accuracy         | 1.0000
intent | precision_macro  | 1.0000
intent | recall_macro     | 1.0000
intent | f1_macro         | 1.0000
intent | precision_micro  | 1.0000
intent | recall_micro     | 1.0000
intent | f1_micro         | 1.0000
slot   | precision        | 0.9048
slot   | recall           | 0.7917
slot   | f1               | 0.8444
```

Post-Processing & Normalization:

- Convert BIO tags to spans (bio_to_spans), then map to schema keys.
- Normalize units (e.g., minutes → 24h time, miles → km).
- Resolve synonyms to a whitelisted taxonomy (e.g., “cafés” → cafe).

The deployed server finally returns an example:


```
{
    "avoid_categories": [],
    "distance_km": null,
    "duration_min": 30,
    "elevation_gain_min_m": null,
    "include_categories": [
        "park"
    ],
    "pet_friendly": false,
    "radius_km": 5.0,
    "route_type": "loop",
    "time_window": null
}
```


3.3.2. Routes Ranking using Linear Regression

Basic Steps:

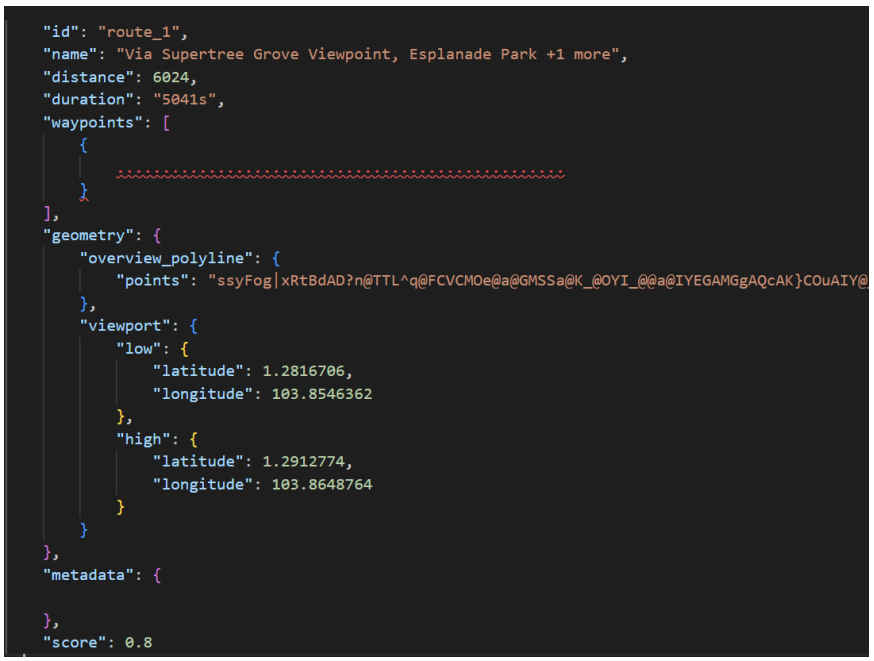
We use the ruled-based scoring + LR model to predict and score candidate routes, and then recommend them based on the score. The LR model uses scikit-learn's Linear Regression model, with waypoint data as X and the score as the label for training. We also customize the feature extractor to extract the 'waypoint' data information and build the train & infer pipeline.

sklearn Pipeline: `RouteFeatureExtractor → DictVectorizer → LinearRegression`

Dataset:

Synthetic dataset using LLM and manual annotation

Picture 3.6: Data Format



Details:

Artificially set the weight of each parameter and score each route first

```
# Weights: rating 0.15, length 0.15, diversity 0.10, scenic 0.40, loop 0.20
score = (
    0.15 * rating_score +
    0.15 * length_score +
    0.10 * diversity_score +
    0.40 * scenic_bonus +
    0.20 * loop_bonus
)
```

Evaluation:

The model was evaluated using MAE and R2 to determine the performance of the model.

Picture 3.7: Evaluation Result

```
Validation: MAE=0.0650 | R2=0.7411 | n=14
Saved model pipeline to: out/ranking_LR_model.pkl
```

Deployment:

Use FastAPI for rapid deployment, upload the collected routes data in json format, and output the predicted score results. By inputting route information, the model will predict the score of the input route and finally output the 'predicted_score' for subsequent ranking score extraction.

3.3.3. Optimization Methods

Stage 1: Polar-Angle Seeding (Non-crossing initial loop):

- Compute the polar angle of each waypoint around the user's start location.

- Sort waypoints clockwise to build a closed loop that empirically reduces edge crossings and backtracking.
- This yields a strong TSP seed under urban geometry.

Pseudocode:

```
class RouteGenerationService:
    def _optimize_waypoint_order_by_angle(
        if len(waypoints) <= 1:
            return waypoints

        def calculate_bearing(center_lat, center_lng, point_lat, point_lng):
            """Calculate the bearing (0-360 degrees) from the center to a waypoint."""
            lat1, lng1 = math.radians(center_lat), math.radians(center_lng)
            lat2, lng2 = math.radians(point_lat), math.radians(point_lng)

            d_lng = lng2 - lng1

            y = math.sin(d_lng) * math.cos(lat2)
            x = math.cos(lat1) * math.sin(lat2) - math.sin(lat1) * math.cos(
                lat2
            ) * math.cos(d_lng)

            bearing = math.atan2(y, x)
            bearing = math.degrees(bearing)
            bearing = (bearing + 360) % 360  # Normalize to 0-360 degrees

            return bearing

        # Compute the bearing for each waypoint
        waypoints_with_angle = []
        for wp in waypoints:
            lat = wp["location"]["lat"]
            lng = wp["location"]["lng"]
            angle = calculate_bearing(center[0], center[1], lat, lng)
            waypoints_with_angle.append((wp, angle))

        # Sort by bearing (clockwise)
        waypoints_with_angle.sort(key=lambda x: x[1])

        optimized = [wp[0] for wp in waypoints_with_angle]
```

Stage 2: 2-Opt Refinement (TSP improvement):

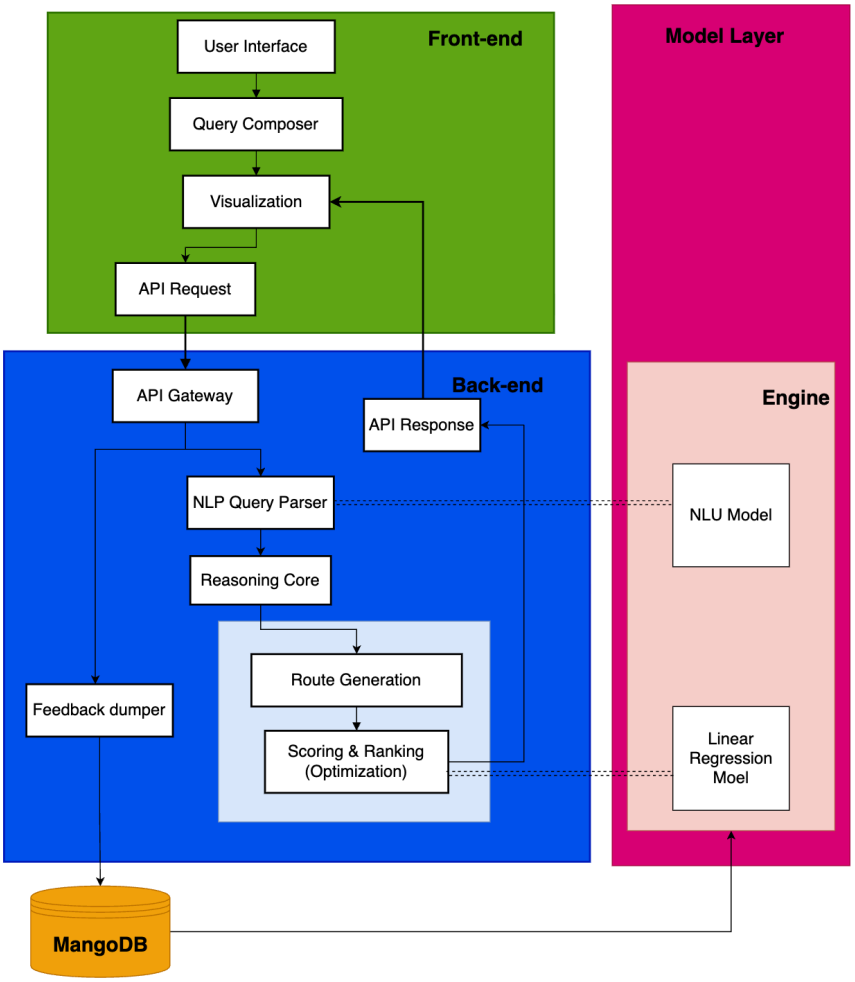
Iteratively consider edges *(i,i+1)* and *(j,j+1)*. If swapping them shortens the tour, **reverse** the segment *(i+1..j)*

- Typical complexity per pass: O(n^2). For our waypoint counts (small n), latency stays low.
- Distance is computed with haversine or (preferably) Directions API polyline distances to reflect actual walkability

Pseudocode:

```python
def optimize_waypoint_order_by_two_opt(waypoints: List[Dict]) -> List[Dict]:
    """
    Apply a 2-opt style heuristic to remove line intersections from an open path.

    The algorithm repeatedly searches for intersecting edges and reverses the
    intermediate segment until no intersections remain. This prioritises removing
    crossings over minimising total distance.
    """
    if len(waypoints) < 4:
        # Fewer than four points cannot produce intersecting non-adjacent edges.
        return waypoints

    optimized = waypoints[:]
    improved = True

    while improved:
        improved = False

        for i in range(len(optimized) - 3):
            for j in range(i + 2, len(optimized) - 1):
                p1 = _extract_point(optimized[i])
                p2 = _extract_point(optimized[i + 1])
                p3 = _extract_point(optimized[j])
                p4 = _extract_point(optimized[j + 1])

                if _segments_intersect(p1, p2, p3, p4):
                    # Reverse the segment between i+1 and j (inclusive) to remove
                    optimized[i + 1 : j + 1] = reversed(optimized[i + 1 : j + 1])
                    improved = True
                    break

            if improved:
                break

    return optimized
```

3.4. End-to-End Workflow



1. Prompt + GPS posted from app (Flutter).
2. Controller routes to NLU (or LLM fallback) → normalized schema.
3. Route Service pulls POIs via Google APIs → builds waypoint sets.
4. Optimization: Polar-angle seeding → 2-opt refinement → feasible loops.
5. Ranking Service scores candidates → top-N returned.
6. Frontend draws polylines, metadata (distance, ETA, POIs).
7. Telemetry logs route choices for future training.
8. Local cache stores up to 10 recent sessions for instant replay.

Other Feature:

Capturing user behavior, storing route selection data, and returning it to the backend:

When a user clicks the Start button after completing a route search, the mobile screen displays detailed route information. The app also returns the user's selection to the backend. For example, if a user queries a total of five routes, the user-selected route will be marked as 1, and the others will be marked as 0. This data will be stored in the backend MongoDB database.

4. Data Collection

Machine learning model training data uses a small amount of manual annotation and LLM to generate cold start data.

Map data: Google Map API

5. Results and Evaluation

5.1. For NLU Model

Metrics

Intent accuracy.

Slot F1 (seqeval, entity-level; report micro/macro).

Exact-match rate of the final JSON schema.

Latency p50/p95 (ms) per request.

5.2. For Linear Regression Model

Metrics

MAE points

5.3. For Route Optimization

Among the routes generated multiple times, (visual inspection) >80% of the routes meet the requirements

Frontend App is operational, the backend server is successfully deployed, and the machine learning model is successfully deployed via Flask/FastAPI.

Latency: From prompt to routes < 10 s end-to-end on MVP infrastructure.

NLU Intent Accuracy (Test): 1.0

NLU Slot F1(Test): 0.8444

Linear Regression MAE: 0.065

6. Findings and Discussion

6.1. Findings

- **Obtaining nearby information**: Maps and nearby POIs are needed to find information about the user's surroundings. The Google Maps API provides many APIs for directly finding locations based on latitude, longitude, and range information.

- **Drawing routes between locations**: The Google Maps API provides the "encoded polyline" method to convert routes into code, allowing the front-end to draw routes directly on Google Maps.
- Appropriate use of **pre-trained models** allows for rapid training and achieves the desired results. The DeBERTa-v3-base model in the Microsoft library is compact and easy to train, making it convenient for building joint intent slot models and can be used as an encoder.

6.2. Discussion

- When designing the Joint Intent Slot Model, what should be the structural design? Should the two models share a common encoder and be trained together?

  Discussion Results: **Shared encoder** (DeBERTa-v3-base) for joint intent+slots kept latency low (single forward pass) and improved mutual regularization.

  Using DeBERTa-v3-base as a pre-trained model can transform user input into the same semantic/syntactic signals, and sharing a common encoder enables mutual regularization. Furthermore, a single encoder reduces latency, memory usage, and deployment requirements, meeting the requirements of an MVP project. Low online latency: Intents and slots can be obtained with a single forward inference, reducing latency and avoiding a degraded user experience.

- Model or Framework Selection for the Joint Model: Comparing DeBERTa-v3-base and distilroberta-base, the latter performs inferior to DeBERTa. The BiLSTM-CRF framework is not very effective for this model.
- **Rule-based scoring** was interpretable but brittle; a **simple linear model** captured preferences better while staying explainable.

  The initial idea is to directly use manual judgment to score routes. By designing weights for different features, the output scores for different routes are then ranked, forming a Rule-Based approach for knowledge elicitation. (Advantages: Concise, highly interpretable, and can be quickly launched)

  Ideas for Improvement: Use linear regression to score the model and machine learning algorithms for data mining, thereby delivering more accurate results to users.

- **Data scarcity** was the main risk; integrating **user choice logs** and **LLM-assisted augmentation** mitigated cold start.

  Insufficient data for model training can easily lead to overfitting during training. Initially, use the LLM model to create a certain amount of cold start data, and manually inspect and annotate the small amount of data. The project incorporates a function that collects user usage habits to form training data. After accumulating a sufficient number of users, continuous training and deployment will be conducted to improve the model.

- **2-opt after polar seeding** was a pragmatic trade-off: excellent for small waypoint counts and robust in non-convex urban layouts.

  How should the generated waypoints be connected to form a path? How can the path minimize backtracking and internal route intersections? Using polar angle

sorting and the 2-opt method can ensure that each waypoint is aligned with the given path.

7. Future Work

- **Containerization & Cloud**: Dockerize services; autoscale NLU/ranking
- Use user data to **retrain models** and **improve performance**.
- **Personalization**: Lightweight user embeddings; contextual bandits for online learning.
- Replace superior machine learning models and improve optimization algorithms.
- **Optimize** app interactions, track user paths in real time, and provide data such as altitude and speed.
- Develop payment and **community** features to unlock revenue opportunities, like POI partnerships, optional premium features, social route sharing.

8. Conclusion

In the MyTrail project, team members collaborated to develop and research tasks across various modules. Initial success was achieved through research and analysis of industry and market needs, and the finalization of the project's content and direction. Throughout the project, team members conducted in-depth research on front-end and back-end frameworks, machine learning model development, and parameter tuning. They continuously optimized and explored new solutions based on project needs, not only meeting the requirements of the course project but also focusing on market-oriented operations.

MyTrail demonstrates that ML-first interpretation of open-ended prompts, coupled with principled route optimization and learning-to-rank, can turn “I want to…” into concrete, map-ready walking loops in seconds. Beyond fulfilling course requirements, the system sketches a credible path to production—decoupled microservices, model hot-swaps, and a clear feedback loop—to keep improving as the user base grows

The MyTrail App has opened up a new track for exploring cities and outdoor activities. We encourage more users to participate in the outdoors in their leisure time, to explore areas that they have never been able to visit before, to experience life and refresh their impression of the city.

Appendix A: Project Proposal

1. Introduction

MyTrails is a mobile application designed for newcomers and residents in Singapore who enjoy running, hiking, or brisk walking but may not be familiar with the city's diverse outdoor trails. Unlike traditional navigation apps that focus on optimal or predefined paths, MyTrails emphasizes exploration and variety. The system automatically generates randomized yet safe and feasible routes from the user's current location, tailored to distance and duration preferences.

The generated routes are then ranked according to personalized filters such as terrain type, slope intensity, crowd density, and environment (urban streets vs. parks). By encouraging users to explore unfamiliar yet accessible areas, MyTrails aims to combine fitness, adventure, and urban discovery in one experience.

The project's primary goal is to provide users with a novel way to engage with their environment while exercising, supporting both fitness goals and the intrinsic motivation of exploration.

2. Project Background & Market Context

2.1. Background

Outdoor running and walking apps have gained immense popularity worldwide as health awareness increases and urban residents seek convenient exercise solutions. In Singapore, where urban and natural landscapes coexist closely, runners and walkers often look for new routes to keep their routines engaging. However, current tools mainly optimize for shortest or fastest travel paths, or rely on user-generated content without system-level personalization.

2.2. Problem / Challenge Addressed

Mainstream fitness and navigation apps (e.g., Strava, Keep, Google Maps) only partially address the needs of users who seek exploration-oriented experiences. Current gaps include:

- **No randomized or exploratory route generation** – existing apps typically provide optimal routes or showcase fixed community routes but lack the ability to generate diverse, randomized loops aligned with user-defined distance or time constraints.

- **Lack of intelligent ranking across multiple alternatives** – most tools suggest one route or show unorganized options without personalized ranking based on filters like scenery, congestion, or safety.

- **Limited support for slope or elevation awareness** – important for users who either prefer challenging uphill workouts or beginner-friendly flat terrains.

MyTrails addresses these gaps by:

- Generating several candidate routes from user input and randomized exploration algorithms.
- Ranking them using multi-criteria reasoning (e.g., environment, slope, safety) that is explainable to the user.
- Encouraging a spirit of discovery rather than mere utility, making exercise more enjoyable.

2.3. Market Landscape & Research

The global fitness app market is highly competitive and saturated with performance-tracking tools (e.g., Strava, Runkeeper, Nike Run Club). However, few focus on exploration and novelty in route planning. In Singapore, the demand for unique outdoor experiences is growing, particularly among expatriates, fitness enthusiasts, and wellness-conscious residents. MyTrails differentiates itself by combining AI-driven exploration with personalized fitness route recommendation.

The current market for fitness and navigation applications shows dominance by major platforms like Strava, Nike Run Club, Keep, and MapMyRun. These apps emphasize:

- **Performance tracking** (pace, calories, heart rate).
- **Community features** (leaderboards, shared routes).
- **Predefined or crowd-sourced routes** (heatmaps, challenges).

Gaps & Opportunities:

- Few apps allow automatic **randomized route generation**.
- Lack of **customizable filters** for terrain, slope, and environment.
- Demand for **exploration-based fitness** is rising, especially in urban regions where runners seek variety and motivation beyond repetition.

Key Trends:

- Personalization in digital fitness through AI and recommender systems.
- Growth of **experiential fitness** — users valuing fun and novelty as much as performance.
- Integration of **geospatial data** and environmental awareness into lifestyle apps.

Market Potential:

- Singapore serves as an ideal testbed with a mix of parks, waterfronts, and urban neighborhoods.

- Initial target users include:
    - Expats and newcomers unfamiliar with routes.
    - Fitness enthusiasts who want diverse challenges.
    - Wellness-focused users seeking engaging outdoor activities.

3. Project Target & Features

3.1. Target:

We hope to combine intelligent reasoning and optimization algorithms to develop an application that can meet user needs. This application can accurately solve the current pain points and cater to market needs.

3.2. Features:

- **Randomized Route Generation**
    - Automatically generates multiple randomized yet feasible and safe routes based on the user's current location and preferred distance or duration.
    - Encourages exploration of unfamiliar areas instead of always taking the shortest or fastest path.
- **Personalized Filtering and Ranking**
    - Users can set filtering conditions (e.g., preferred waypoints), such as terrain type (urban streets/parks), slope intensity, crowd density, and safety.
    - The system evaluates and ranks candidate routes across multiple criteria and provides explainable recommendations.
- **Combination of Exploration and Fitness**
    - Supports not only exercise but also the discovery of new environments, enhancing the sense of novelty and fun.
    - Integrates fitness, adventure, and urban exploration into one experience.
- **Support for Diverse Scenarios**
    - Suitable for running, brisk walking, and hiking.
    - Covers both urban and natural trails.

4. Project Scope

4.1. Market Scope

MyTrail aims to provide intelligent, personalized route generation for urban outdoor activities such as walking and running. The system is designed to integrate with consumer fitness, travel, and smart city applications, delivering intelligent route generation that leverages rich and dynamic POI (Point of Interest) data for users in cities worldwide.

4.2. Intelligent Systems Focus:

- Intelligent Query Analysis:
  - Uses NLP to extract user preferences such as activity type, distance, duration, and points of interest from input query.
  - Integrates user history and contextual factors (e.g., time, weather, fitness goals) to produce structured constraints for route generation. (future plan)
- Dynamic Route Generation:
  - Generates candidate routes based on real-time user constraints (location, time, preferences) and live POI data.
  - Utilizes advanced multi-criteria scoring algorithms, considering distance, duration, category relevance, and user preferences.
  - Applies diversity optimization and novelty detection to ensure recommendations are varied and engaging.
  - Incorporates user feedback and behavioral data for continuous personalization and learning. (future plan)
- Scalable Architecture:
  - Integrates with global mapping APIs (e.g., Google Maps, OpenStreetMap) for up-to-date data and directions.
  - Supports real-time data pipelines for POI updates and traffic conditions.

4.3. Limitations & Constraints:

- Data Access:
  - Requires reliable access to third-party mapping and POI APIs.
  - Data timeliness issue: Some location data may be outdated.
- Personalization Depth:
  - Effectiveness depends on the availability and quality of user preference and behavioral data.
  - Some specific conditions entered by the user cannot be fully met.

5. Data Collection and Preparation

5.1. Data Sources:

- Global Mapping APIs:
  - Real-time POI (Point of Interest) and route data sourced from providers such as Google Maps, OpenStreetMap, and other commercial mapping platforms.

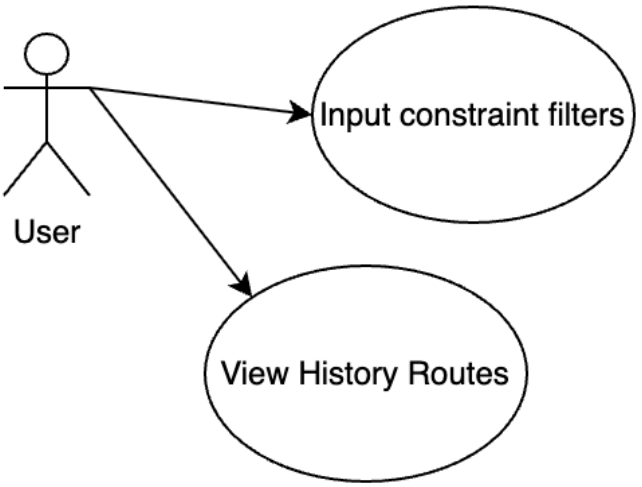
- User-Related Data:
    - User preferences, activity history, and feedback collected through integrated mobile/web applications. (future plan)

6. System Design

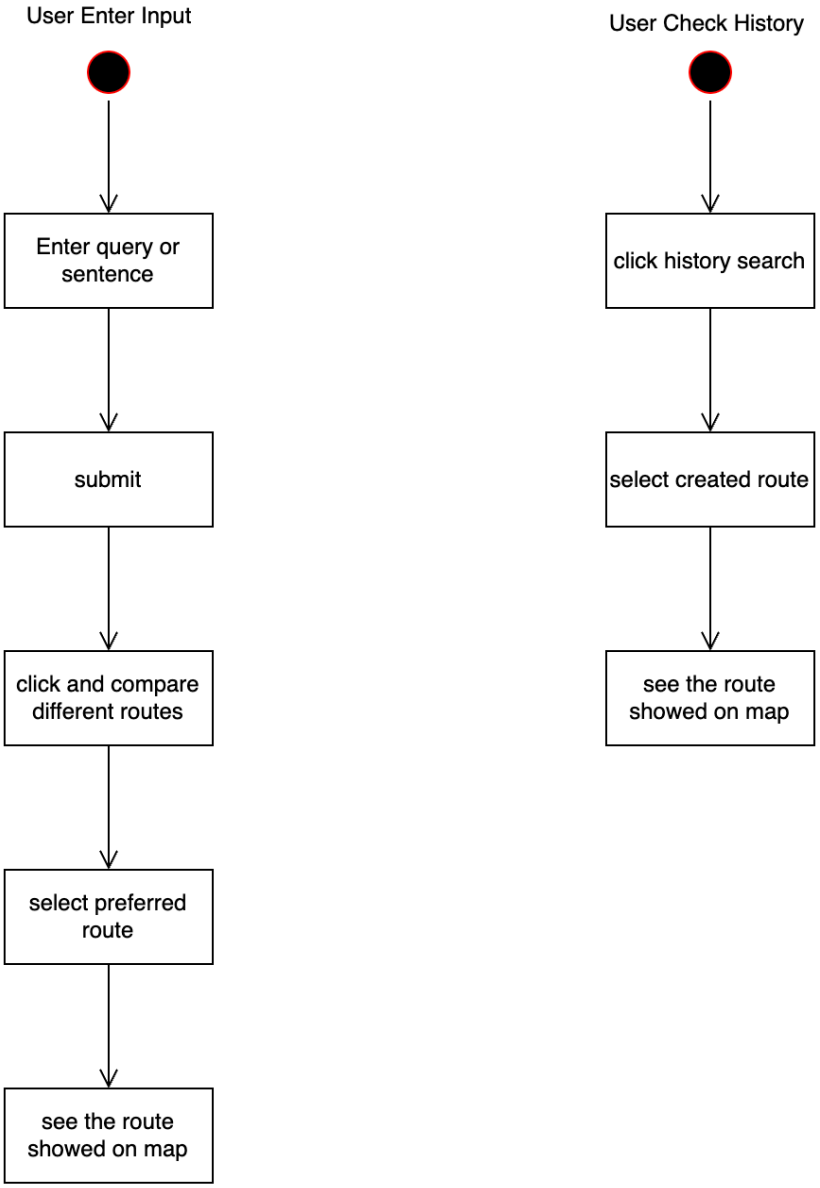
6.1. Architecture Overview:

The MyTrail intelligent route generation system is designed for scalable, real-time urban route recommendations. It integrates external mapping APIs, user data, and advanced reasoning modules to deliver personalized and diverse route suggestions.

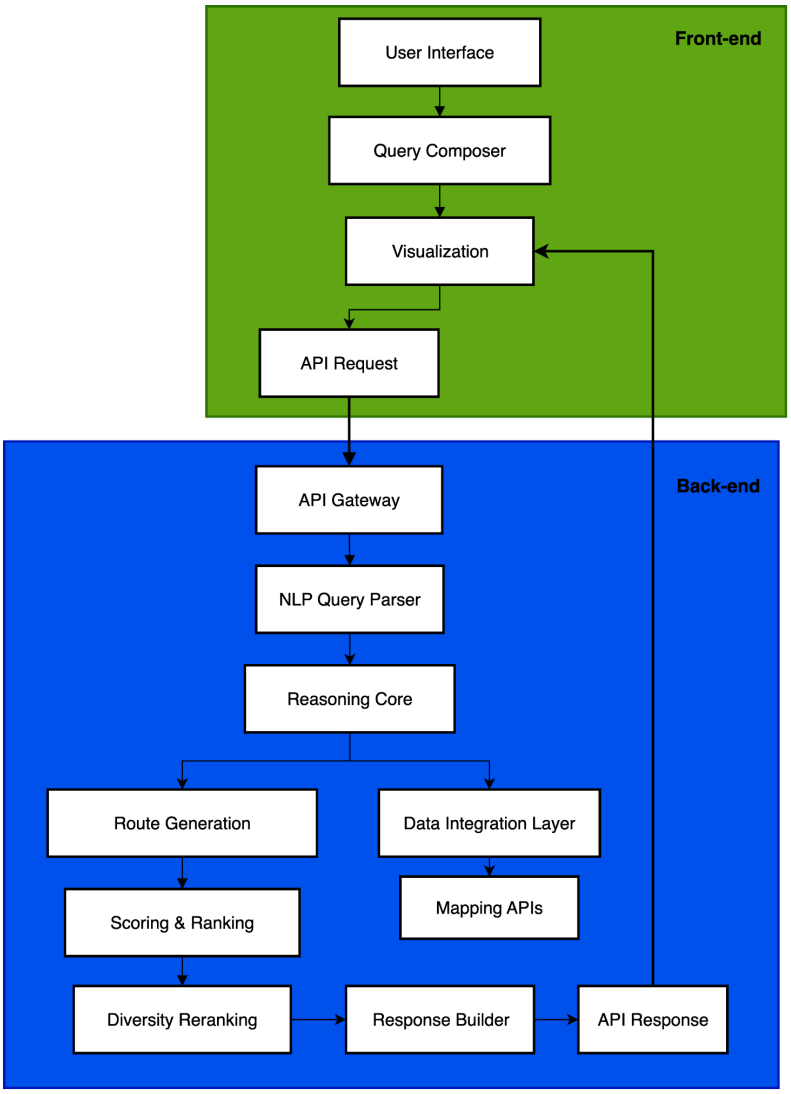
6.2. Use Case:



6.3. Activity Diagram:



6.4. Architecture Flowchart:



6.5. Key Components:

Frontend (Mobile App):

- User Interface:
  - Allows users to input queries (natural language or structured forms), view recommended routes, and provide feedback.

- Query Composer:
  - Guides users to specify preferences (distance, POI types, time, pet-friendliness, etc.).

- Visualization:
  - Displays routes on interactive maps, highlights POIs, and shows route details.

- Feedback Module:

    - Collects user ratings, preferences, and behavioral data for personalization.

Backend:

- API Gateway:
    - Receives requests from frontend, returns formatted responses.

- Query Parser:
    - Interprets user input and extracts structured preferences.

- Data Integration Layer:
    - Connects to mapping APIs for live POI and route data.

- Reasoning Core:
    - Route Generation, Scoring & Ranking, Diversity Reranking.

- Monitoring & Analytics:
    - Tracks system performance and user engagement.

6.6. Reasoning Techniques:

- Query Parsing & Preference Extraction
    - NLP & Rule-Based Extraction: Parse user queries into structured preferences (distance, route type, POI categories, time window, pet-friendliness).

- Route Generation
    - A* and Dijkstra Algorithms: Efficiently generate candidate routes that satisfy user constraints using graph-based pathfinding.

- Multi-Criteria Scoring & Ranking
    - Weighted Feature Scoring: Rank candidate routes by combining distance, duration, POI relevance, and environmental features.

- Diversity Optimization
    - Greedy Diversity Selection / Maximal Marginal Relevance (MMR): Rerank results to ensure variety in geography and POI types.

- Personalization (Future plan)
    - Collaborative Filtering & Reinforcement Learning: Adapt recommendations based on user preferences and behavior for personalized route suggestions.

Appendix B: Course Content Mapping

1. Machine Learning Model:

   NLP Service Module: This module uses an NLU model that is compatible with the Slot and Filler functions in CGS. It uses deBERTa as the underlying model. It then extracts BIO slots through two steps: semantic understanding and slot filtering, and then converts them into the required data.

   Linear Regression Model: MR, RS Supervised Learning, Regression

2. Optimization Algorithms:

   Polar angle sorting and 2-opt optimization：in line with the Reasoning using Optimization Techniques in the RS course

Appendix C: Installation and User Guide

C.1 Overview

The repository is organized as:

- `frontend/` — Flutter mobile app (primary target: iOS).
- `backend/` — FastAPI server exposing REST endpoints.
- `models/nlu_model/` — NLU training code and a lightweight inference service

C.2 Prerequisites

- **OS for iOS dev:** macOS 13+
- **Xcode:** 18+ (with iOS Simulator); CocoaPods 1.13+ (`sudo gem install cocoapods`)
- **Flutter SDK:** 3.x (Dart 3.x). Verify with `flutter --version`
- **Python:** 3.11 with `pip` (recommend using `virtualenv`)
- **Git**
- **Google Cloud keys:**
    - Maps **SDK for iOS** (for the embedded map in Flutter)
    - **Places API** and **Directions API** keys (used by the backend)
- **OpenAI API keys:**

C.3 API KEYs and Address Modification:

1. Clone Github Repository
2. iOS Maps SDK key: path: ios/Runner/Info.plist, find the <key>GMSApiKey</key> and modify
3. Backend API keys：path: app/config/config.py, find and modify google_maps_api_key, openai_api_key, openai_base_url, openai_model,mongo_db(optinal)
4. Model address: nlu_basic_model_url, mongo_uri

C.4 Backend Setup (FastAPI)

Enter the backend folder and enter the terminal, enter the command：(create your python env first):

pip install -r requirements.txt

python -m uvicorn app.main:app --host 192.168.0.207 --port 8000 --reload

C.5 NLU: Train and Serve（Flask）

For **Train**
Enter the models/nlu_model folder and enter the terminal:
pip install -r requirements.txt
python train.py

For **Deployment**
Enter the models/nlu_model folder and enter the terminal:
python app.py --host 0.0.0.0 --port 4000

C.5 Frontend App

Enter the Frontend folder and enter the terminal:
flutter pub get
flutter run

If running on a physical device:

1. Open `ios/Runner.xcworkspace` in Xcode.
2. Set a unique **Bundle Identifier** and your **Apple Team** (signing).
3. Build & run on your device.

C.6 Run Order (Dev)

1. **Start NLU** service (`models/nlu_model/app.py` on port 9000).
2. **Start Backend** (`uvicorn` on port 8000), with `NLU_BASE_URL` pointing to #1.
3. **Run Frontend** (`flutter run`) with `backendBaseUrl` pointing to #2.