HikeMatch: An Explainable, Context-Aware Hybrid Recommender System for Hiking Trails

Yuancan Stefan Li
stefan.li@tum.de
Technical University of Munich
Munich, Germany

Ruben Kolbusa
ruben.kolbusa@tum.de
Technical University of Munich
Munich, Germany

Caspar-Antoine Maximilian Pagel
caspar.pagel@tum.de
Technical University of Munich
Munich, Germany

ABSTRACT

This work presents HikeMatch, a context-aware hybrid recommender system for hiking trails that integrates three complementary recommendation strategies: content-based filtering on engineered trail features, geospatial indexing for proximity-aware retrieval, and semantic similarity derived from hierarchical topic modeling. By incorporating real-time contextual data, including weather conditions, user fitness level, and trip purpose, the system delivers recommendations that align with both user preferences and real-world constraints.

Evaluation demonstrates that the system achieves high catalog coverage (99.5%) while maintaining geographic coherence, with core recommendation logic executing in sub-millisecond time. This lightweight architecture enables deployment on modest cloud infrastructure without sacrificing recommendation quality. Informal user testing reveals that explainability features enhance user trust and support the discovery of trails beyond the primary recommendation list. These results suggest that effective context-aware trail recommendation is possible through integration of complementary strategies, without relying on black-box deep learning models whose recommendations are difficult to trace back to concrete features.

KEYWORDS

recommender systems, hybrid recommendation, context-aware recommendation, explainability, hiking trails, topic modeling, geospatial indexing

ACM Reference Format:
Yuancan Stefan Li, Ruben Kolbusa, and Caspar-Antoine Maximilian Pagel. 2026. HikeMatch: An Explainable, Context-Aware Hybrid Recommender System for Hiking Trails. In . ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION AND MOTIVATION

Outdoor recreation, particularly hiking, has experienced significant growth in recent years, with a majority of users turning to digital platforms to discover new trails. [1]. A key objective when designing such recommendation systems is the integration of external contextual data, such as weather condition, time-of-day, or geographic features [2]. However, this brings with it the challenge of generating recommendations from various types of data, like semantic descriptions, coordinates, and numerical features, each requiring a different approach to generate recommendations. To handle this complexity, many recommender systems are starting to integrate large language models (LLMs) as a key component [3] [4]. While this can lead to high-quality results, AI explainability is often still insufficient in practice [5].

Therefore, we present a context-aware hybrid recommender system for hiking routes that addresses three key challenges: (1) integrating real-time contextual data such as weather conditions and seasonality into the recommendation process, (2) combining multiple recommendation strategies to capture different aspects of trail similarity, and (3) providing transparent explanations that help users understand why specific routes are recommended.

Our system employs a hybrid architecture that fuses content-based filtering with geospatial indexing and semantic similarity analysis. Content-based filtering operates on engineered features including trail difficulty, elevation gain, distance, and terrain characteristics. A geospatial index enables efficient retrieval of nearby trails based on user location. Semantic similarity, derived through topic modeling on trail descriptions, captures thematic relationships between hikes that numerical features alone cannot express, with the aim to represent the experiential qualities of each hike.

Central to our approach is the integration of contextual awareness. Users specify their current situation through parameters including location, available time, fitness level, trip purpose, and weather preferences. Additionally, the system can incorporate live weather data from external APIs or allow users to specify hypothetical conditions. For example, a user planning a photography trip receives different recommendations than one seeking a high-intensity training hike, even when starting from the same location.

We ensure explainability through the ability to trace from final recommendations to explicit features and phrases in the data that had an impact on the result. This transparency serves both practical and trust-building purposes: users can evaluate whether the system's reasoning aligns with their own preferences and adjust their inputs accordingly.

2 METHODOLOGY

2.1 Dataset

Our dataset is based on approximately 1,160 hiking routes from the Pacific Northwest region of the United States, retrieved from

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

the OregonHikers [6] page in xml format. Each hike entry contains structured metadata including geographic coordinates (latitude, longitude), trail distance in kilometers, elevation gain in meters, maximum altitude, and a difficulty rating encoded as an integer (1=Easy, 2=Moderate, 3=Hard). Additionally, each hike includes boolean flags indicating seasonal accessibility (spring, summer, fall, winter) and the presence of notable features such as lakes, waterfalls, viewpoints, wildlife areas, old-growth forests, wildflowers, beaches, and peaks. Each entry also contains a free-text description that describes the trail course, scenery, and hiking experience.

We have preprocessed this data using regular expressions to extract the features from the xml file and stored them in CSV format. Further preprocessing included the conversion from miles to kilometers and foot to meters.

2.2 Feature Engineering

Feature engineering transforms raw trail attributes into a unified representation suitable for similarity computation and scoring. Our approach addresses three challenges: normalizing numerical scales, encoding categorical attributes, and deriving implicit features from raw data.

Numerical Features. The dataset contains numerical attributes with different scales: distance (0.8–50+ km), elevation gain (20–1500+ m), and altitude (30–3157 m). We retain these features in their original units for interpretability in the user interface, while applying context-dependent transformations during the scoring phase. This design choice preserves the semantic meaning of raw values for display purposes while enabling normalized comparisons internally.

Categorical Feature Encoding. Difficulty ratings are encoded as ordinal integers (1=Easy, 2=Moderate, 3=Hard), enabling distance-based matching where adjacent difficulty levels can receive partial credit during scoring. Trail features—including waterfall, lake, viewpoint, old-growth forest, wildlife, beach, mountain peak, meadow, river, and forest—are represented as binary flags, forming a 10-dimensional one-hot feature vector per trail. Seasonal accessibility is similarly encoded as four binary flags indicating suitability for spring, summer, fall, and winter hiking.

Derived Features. We compute estimated hiking duration from distance and elevation using a simplified Naismith's rule:

$$t_{\text{est}} = \frac{d}{v_h} + \frac{e}{v_e} \tag{1}$$

where d is distance in kilometers, e is elevation gain in meters, $v_h = 4$ km/h is the assumed horizontal hiking speed, and $v_e = 400$ m/h is the vertical ascent rate. This derived feature enables duration-based filtering without requiring explicit time annotations in the source data.

Trip Purpose Prototypes. To align recommendations with user intent, we define purpose-specific feature prototypes derived from domain knowledge. Each trip purpose (scenery, fitness, family, wildlife, nature) is associated with a weight vector over the numerical features, representing the expected feature distribution for that activity type. For example, the *scenery* prototype assigns high weights to viewpoints (0.94), rivers (0.90), and forests (0.91), while the *fitness* prototype emphasizes mountain peaks (0.86) and elevation gains.

Score Component Capping. To prevent any single factor from dominating recommendations, individual score components are capped at predefined maxima before fusion: difficulty match at 30 points, duration match at 25 points, and feature match at 45 points. All component scores are subsequently normalized to a $[0, 100]$ scale by computing the ratio of achieved points to maximum possible points.

2.3 System Architecture

The system follows a client-server architecture deployed within a Docker container and is hosted on Railway for production. It consists of a react-based frontend running via Nginx and a FastAPI backend that exposes RESTful API, as shown in Figure 1.

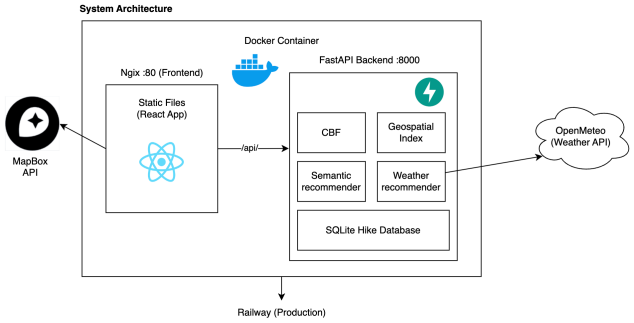

Figure 1: Dockerized web app architecture: React frontend, FastAPI backend with CBF, GSI, and semantic/weather recommenders, SQLite database, and deployed on Railway.

Frontend. The frontend is a single-page application built with React and TypeScript, styled with Tailwind CSS, and using Mapbox GL for map visualization. It captures user preferences, displays recommendations, and provides interactive features such as the trail connection graph and comparison view. All data is fetched from the backend via HTTP requests to the `/api` endpoints.

Backend. The FastAPI backend implements four recommendation components that are combined to produce the final rankings:

(1) **Content-based filtering:** Hikes are represented as normalized feature vectors encoding difficulty, distance, elevation, duration, and boolean trail attributes. A cosine similarity matrix is precomputed over all hikes, enabling fast retrieval of similar trails. This component also supports context-based queries by converting user preferences into a feature vector (see Section 2.4).

(2) **Geospatial indexing:** Hike coordinates are projected into three-dimensional Cartesian space and indexed using a KD-tree. This allows efficient radius-based and nearest-neighbor queries, returning hikes within a specified distance from the user's location.

(3) **Semantic similarity:** Trail descriptions are analyzed to extract experiential qualities that numerical features cannot capture. Using hierarchical Non-negative Matrix Factorization (NMF) on adjectives extracted from descriptions, each

hike is represented as a 60-dimensional topic vector. Similarity between hikes is computed via cosine similarity over these vectors (see Section 3.2 for details).

(4) **Weather-based recommendation:** Hike suitability is evaluated based on current or simulated weather conditions. Weather data is retrieved from the Open-Meteo API or from predefined seasonal presets. An XGBoost model predicts a suitability score for each hike based on combined trail and weather features, which is used to rerank recommendations.

The final recommendation score is computed by fusing the outputs of the content-based, geospatial, and semantic components using weighted aggregation, followed by reranking based on weather suitability and preference alignment (see Section 2.5).

Data Layer. Trail data is stored in a SQLite database initialized from a preprocessed CSV file. The database stores all hike attributes defined in the data model and provides fast access for the recommendation components. This lightweight approach avoids external database dependencies while supporting the full feature set.

2.4 Context Modeling

The recommender system models user context using a structured *RecommendationContext* object. It captures three categories of contextual data: *location constraints*, *environmental conditions*, and *user preferences*. The model is implemented as a Pydantic schema. Table 1 summarizes the fields and their rationale.

Table 1: RecommendationContext (Pydantic schema) fields and rationale (backend/src/recommender/models.py:65–82).

Category	Field(s)	Type	Purpose
Location	latitude, longitude	float	User's geographic position
Location	max_radius_km	float (1–500)	Maximum search radius
Environment	weather_condition	Enum	Current weather or preset scenario
Environment	daytime	Enum	Morning / Afternoon / Evening / Anytime
Environment	season	Enum	Summer / Spring / Fall / Winter
Preferences	fitness_level	Enum	Beginner / Intermediate / Advanced
Preferences	duration	Enum	Short (1–2h), Medium (3–5h), Long (5+h)
Preferences	trip_purpose	Enum	General / Scenery / Fitness / Family / Wildlife

The `weather_condition` field supports both *live mode* (fetching real-time weather data via an API) and *preset scenarios*. This design enables systematic testing and demonstrations across diverse context configurations without depending on external weather services.

Context propagates through the recommendation pipeline as follows:

(1) **Construction:** The user interface constructs a context object from the selected conditions and sends it to the backend.

(2) **Hard filtering:** The system applies season and duration constraints at the database level. Duration preferences are converted to distance and elevation constraints using Naismith's rule. The search space is restricted to hikes suitable for the target season whose distance and elevation lie within a ±50% range of the computed estimates.

(3) **Feature-vector conversion:** The context is transformed into a feature vector that maps the user context into the hike vector space, enabling similarity comparisons between context and hikes.

(4) **Geospatial filtering:** The context latitude and longitude are used for proximity-based retrieval of hike candidates.

2.5 Score Fusion and Reranking

The system combines the outputs of the three components (content-based similarity, geospatial proximity, and semantic similarity) into a single recommendation score. Each component produces a normalized score between 0 and 1, which are combined using a weighted sum:

$$s_{\text{final}} = w_c \cdot s_{\text{content}} + w_g \cdot s_{\text{geo}} + w_s \cdot s_{\text{semantic}} \quad (2)$$

with weights $w_c = 0.6$, $w_g = 0.2$, and $w_s = 0.2$. Content-based similarity is assigned the highest weight, as it most directly captures the trail attributes that are of greatest importance to users, whereas geospatial and semantic similarity are assigned lesser weights. After computing the fused scores, the system applies two reranking steps:

(1) **Weather reranking:** When weather data is available, hikes with poor conditions (e.g., heavy rain, extreme temperatures) are moved down in the ranking, even if they scored well on other factors.

(2) **Preference alignment:** Hikes that deviate too far from the user's difficulty or duration preferences are penalized, so that a trail rated "Hard" does not appear at the top for a user who selected "Easy".

This approach keeps the ranking process transparent: trails are recommended based mainly on their features, location, and thematic fit, giving users a clear understanding of the recommendation, while also taking into account that weather and preference mismatch have affected the initial recommendations.

2.6 Explainability

A key goal of HikeMatch is to help users understand why a trail was recommended. Instead of showing only a ranked list, the system provides explanations that connect recommendations to specific trail attributes and user preferences. **Tracing Recommendations to Features.** Each component contributes explanations that users can inspect:

- **Content-based:** The system identifies which features match the user's preferences (e.g., Has waterfall," Matches your difficulty level") and flags potential issues (e.g., Longer than your preferred duration").
- **Geospatial:** Distance is shown in kilometers with simple labels like very close," nearby," or moderate distance."

- **Semantic:** Because topic vectors are built from adjectives in trail descriptions, high similarity between two trails can be traced back to shared descriptive terms like scenic," peaceful," or "challenging."

Qualitative Explanations. In early versions, we displayed numerical scores for each component (e.g., "Content: 78, Location: 85"). During user testing, we found that these numbers were misleading, because users assumed small differences were meaningful when they were not. We therefore replaced numerical scores with qualitative explanations organized into three categories:

- **Why this trail:** Positive factors like matching features, suitable difficulty, and alignment with trip purpose.
- **Location:** How far the trail is from the user, shown in kilometers.
- **Things to consider:** Potential drawbacks, such as the trail being harder or longer than requested.

The Connection Graph. The trail connection graph also supports explainability. By showing which trails are semantically related, it helps users see why certain hikes cluster together and discover alternatives with similar characteristics. This makes the reasoning of the semantic recommender visible without requiring users to understand topic modeling.

2.7 User Interface

We developed an interactive web application using React with TypeScript, styled with Tailwind CSS, and featuring Mapbox GL for geographic visualization. The interface follows a four-page structure that guides users from preference selection through recommendation exploration.

Onboarding. Users begin by specifying their hiking preferences: trip purpose (scenery, fitness, family, or nature), difficulty level, expected duration, and optional trail features such as waterfalls, lakes, or viewpoints.

Exploration Interface. The main page uses a three-column layout .The left panel displays current preferences, live weather data

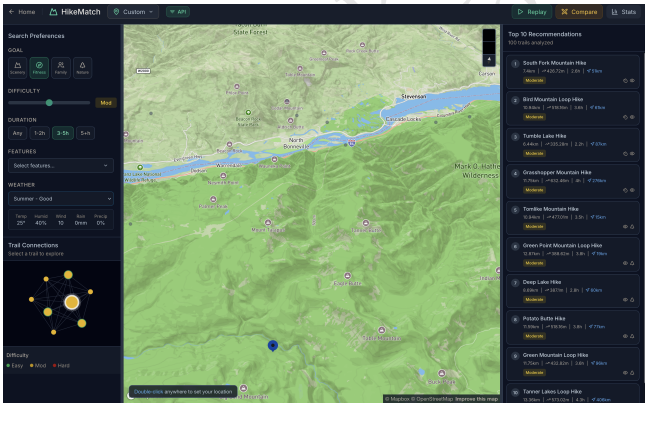

Figure 2: Exploration page with a three-column layout

from the Open-Meteo API, and a trail connection graph. The center column shows an interactive 3D terrain map with trail markers colored by difficulty. The right panel lists the top recommendations with key statistics including distance, elevation gain, and estimated duration.

Selecting a trail opens a detail modal with full trail information and an *explainability panel.* This panel shows why the trail was recommended (matching features, difficulty fit, trip purpose alignment), the distance from the user's location, and potential considerations such as duration exceeding preferences or seasonal limitations. This replaces earlier designs that showed numerical percentage scores, which user testing revealed to be misleading.

The *trail connection graph* visualizes semantic relationships between trails using a force-directed layout. Users can click connected nodes to discover trails that share thematic characteristics with their current selection, supporting exploration beyond the primary recommendation list.

Comparison Interface. The comparison page allows users to

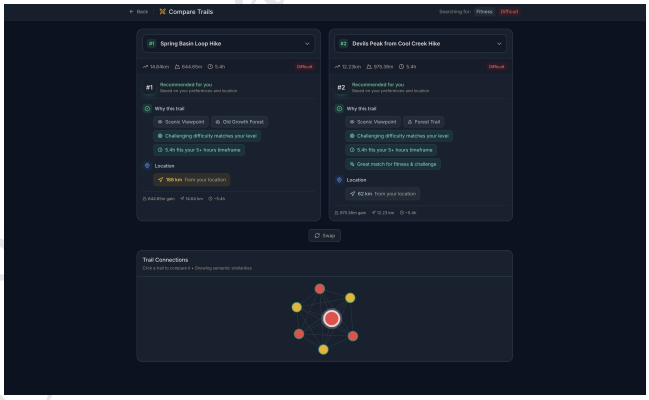

Figure 3: Compare Page. Two hikes compared side by side, with a mini-graph to explore others

evaluate two trails side by side. Each trail is displayed in a card showing key metrics and the full explainability panel. Users can swap trails using a dropdown selector or by clicking nodes in the mini graph below, which shows trails semantically related to the current selection. This design helps users understand the trade-offs between options—for example, one trail may be closer while another better matches their feature preferences.

Statistics Dashboard. An analytics page aggregates metrics across all recommendations, displaying total and average distance, elevation, and duration, as well as difficulty distribution and feature coverage. This overview helps users assess whether the recommendation set aligns with their expectations.

Recommendation Journey. When recommendations are first generated, a short animation sequence visualizes the system's process: analyzing preferences, searching nearby trails, computing similarities, and revealing results. This animation reinforces transparency by showing users that their inputs drive the recommendations. Users can skip the animation if preferred.

2.8 Deployment

The system is packaged as a single Docker container using a multi-stage build process. In the first stage, the frontend is compiled into

static assets, whereas the second stage sets up the FastAPI backend application.

Nginx acts as a reverse proxy: it serves the static frontend assets directly and forwards API requests (prefixed with `/api/`) to the backend running on an internal port. This architecture simplifies deployment by exposing only a single external port.

The resulting container is deployed to Railway [7], a platform-as-a-service (PaaS) provider. Railway automatically detects the `Dockerfile`, builds the image, and exposes port `80` to the public internet. Environment variables (e.g., API keys and authentication tokens) are configured via Railway's dashboard and injected at runtime.

To prevent unauthorized load on the system, all recommendation endpoints require token-based authentication. Clients must include a valid Bearer token in the `Authorization` header. Valid tokens are configured via the `AUTH_TOKENS` environment variable. Requests with missing or invalid tokens are rejected by the backend server with `401 Unauthorized`.

3 EXPERIMENTS

3.1 Generating Context Vectors

To retrieve hikes for a given context, the system must project the *RecommendationContext* into the same vector space used by the content-based recommender. This section describes how incoming context attributes are transformed into a comparable feature vector.

Several context attributes map directly to individual feature dimensions:

- **Fitness level** → difficulty encoding (`Beginner`=1, `Intermediate`=2, `Advanced`=3)
- **Latitude/longitude** → geographic coordinates
- **Season flags** → set to 0 in the context vector, since seasonal compatibility is enforced via hard filtering prior to similarity computation

However, the `trip_purpose` attribute (`General`, `Scenery`, `Fitness`, `Family`, `Wildlife`) represents an abstract user intent that cannot be mapped directly to a single scalar feature. To address this, we employ a prototype-based approach.

Step 1: LLM-based labeling. We use GPT-4o-mini [8] to annotate all hikes in the dataset with purpose suitability scores (0.0–1.0) for each category. The LLM receives structured hike metadata (e.g., distance, elevation gain, altitude, and boolean terrain features). We explicitly exclude textual descriptions, since they are not part of the content-based vector representation.

Step 2: Prototype computation. For each trip purpose category, we compute a prototype vector as the mean feature vector across all hikes with high suitability scores (≥ 0.7) for that category. These prototypes capture the characteristic feature distribution associated with each purpose type.

For example, the `Family` prototype is characterized by low values for `distance_km` (5.4 km) and `elevation_gain_m` (97 m), whereas the `Fitness` prototype exhibits substantially higher values (19.8 km distance and 861 m elevation gain).

Step 3: Context vector assembly. When converting a *RecommendationContext* to a feature vector (`recommender.py:111–152`), the system first retrieves the prototype vector associated with the selected `trip_purpose`. It then constructs a *virtual hike vector* by merging this prototype with the directly-mapped context attributes.

This method enables cosine similarity computation between user context and actual hike vectors, ranking trails by how well they match the user's intended trip profile.

3.2 Semantic Recommender and Topic Discovery

The system requires a semantic understanding of hike descriptions in order to recommend trails with similar experiential qualities beyond what the existing numerical features can capture. We explored two approaches: (1) a knowledge graph based on extracted keywords, and (2) a topic model based on Non-negative Matrix Factorization (NMF). This section outlines both approaches and explains the final design choice in favor of the topic model.

Approach 1: Global Knowledge Graph. Our initial approach constructed a bipartite graph connecting hikes to semantic concepts extracted from their descriptions. To this end, we implemented two extraction methods:

(1) **TF–IDF with POS filtering:** Using spaCy for part-of-speech tagging, we extracted noun and adjective phrases (1–2 words) from descriptions and ranked them by TF–IDF scores computed across the corpus. The top 15 concepts per hike were selected as that hike's topics.

(2) **YAKE (Yet Another Keyword Extractor):** A statistical keyword extraction method that considers term frequency, position, and context without requiring corpus-level statistics. [9] YAKE scores are inverted (lower is better), so we normalized them via $1/(1 + \text{score})$.

The graph $G = (V, E)$ is bipartite with vertex set $V = H \cup C$, where:

- H is the set of hike nodes, one for each hike in the dataset
- C is the set of concept nodes, containing all keywords extracted across all hike descriptions

Edges $E \subseteq H \times C$ connect each hike to its associated concepts, with edge weights corresponding to the concept's relevance score for that hike.

Edges E connect each hike to its associated concepts. This enables performing hops from one hike to another via shared concepts.

Recommendations are computed using PageRank. Given a seed hike, the algorithm propagates probability mass through shared concepts, thereby measuring semantic relatedness:

$$\texttt{scores} = \texttt{nx.pagerank}\big(G,\ \texttt{personalization} = \{\texttt{seed\_node} : 1.0\}\big),$$

where `seed_node` is the node name corresponding to the input hike for which similar hikes are to be retrieved.

In particular this approach allows for explainability, since a recommendations can be traced back as a path to the seed node, and allows to aggregate the exact topics that connect the hikes along the way.

While conceptually elegant, the knowledge graph approach revealed drawbacks when applied to our dataset:

(1) **Sparse connectivity:** Some hikes share only few or no concepts with others, resulting in disconnected subgraphs and,

in some cases, fully isolated hikes. These hikes therefore receive poor recommendation coverage.

(2) **Bias towards proper nouns:** Both keyword extraction methods exhibited a strong bias toward proper nouns (e.g., "Deer Valley"), even when filtering with spaCy. This is likely due to many hike descriptions focusing heavily on the course of the route. Such terms primarily contain geographic entities (mountains, valleys, locations) and add limited value to recommendations, since geographic proximity is already captured by the geo-spatial index.

(3) **Lack of abstraction:** Raw keywords were often highly specific to individual hikes and did not provide robust higher-level groupings of experiential qualities.

Approach 2: Hierarchical NMF Topic Discovery. While working on improving the knowledge graph, we came up with an alternative approach, based on topic modeling to learn latent semantic dimensions from the corpus.

Rather than extracting all nouns and adjectives, we focus exclusively on adjectives as carriers of experiential meaning to capture the experiential qualities of hikes. Terms like "scenic", "steep", or "peaceful" are more descriptive of how one might feel during the hike, while nouns like "trail", "forest", or "mountain" pose a redundancy risk given the structured boolean features already present in the dataset. We applied NMF in a hierarchical fashion to capture both broad themes and fine-grained distinctions:

- **Level 1: Broad topics**

```
broad_model = NMF(n_components=10, max_iter=800,
      init='nndsvda')
broad_topic_dist = broad_model.fit_transform(tfidf_matrix)
```

  The 10 broad topics capture high-level experiential categories (e.g., "challenging/steep", "scenic/beautiful", "peaceful/quiet").

- **Level 2: Subtopics**
  For each broad topic, we identified hikes with above-average affinity for that topic in order to model more nuanced categories. We then computed the residual after removing the broad topic's contribution:

```
reconstruction = np.outer(W_broad[:, topic_i],
      H_broad[topic_i, :])
residual = tfidf_subset - reconstruction
residual[residual < 0] = 0 # Maintain non-negativity
```

  Applying NMF to this residual extracts 5 subtopics per broad topic, capturing finer distinctions within each theme. This yields $10 + (10 \times 5) = 60$ topic dimensions per hike.

Thus, each hike is represented as a 60-dimensional vector concatenating broad topic weights and subtopic weights:

$$\text{Hike Vector} = [\text{broad}_0, \text{broad}_1, \ldots, \text{broad}_9, \text{sub}_{0,0}, \text{sub}_{0,1}, \ldots, \text{sub}_{9,4}]$$

This approach improves on many of the drawbacks of our concept graph:

(1) **Coverage:** Topic vectors allow comparing hikes in a dense, continuous similarity space. While outliers may still exist, they are no longer completely isolated. This leads to improved coverage of the search space.

(2) **Vocabulary:** Instead of focusing on a set of 15 concepts per hike, this approach considers the set of all adjectives across the corpus, thus avoiding possible over-fixation on proper nouns.

(3) **Abstraction:** Semantic similarity is no longer dependent on specific keywords, but represented through membership in abstract, discovered categories.

At the same time, the individual scalars in the vector can still be traced back to their corresponding concrete words in the TF–IDF vectors of the hike descriptions, allowing for rigorous explainability of recommendation results.

4 EVALUATION

4.1 Recommendation Quality

We use the following hyperparameters as default configuration for evaluating recommendation quality:

```
class Config:
    # recommender weights
    content_weight: float = 0.6
    geo_weight: float = 0.2
    semantic_weight: float = 0.2
    # topic discovery
    n_broad_topics: int = 10
    n_sub_topics: int = 5
    # evaluation
    sample_size: int = 200
```

The recommendation weights resemble the significance given to the different components during score fusion. The topic discovery values define the number of broad- and sub-topics are used by the semantic recommender. The sample size decides how many random hikes sample to input into the recommender during evaluation.

To quantify recommendation quality without having access to labeled test data, we focus on three core metrics:

(1) **Coverage:** Ratio of the set size of retrieved hikes in recommendation results compared to the size of the set of all hikes. High coverage indicates that the system explores a large portion of the catalog instead of repeatedly recommending a small subset of popular items.

(2) **Geo coherence:** Measures how geographically consistent the recommendations are with respect to the input hike. It is computed as (i) the mean haversine distance (in km) between the queried hike and its recommended hikes, and (ii) the fraction of recommended hikes within a 50 km radius. Lower mean distance and higher within-50 km ratio indicate stronger geographic relevance.

(3) **Diversity:** Measures the average pairwise distance between recommended hikes in feature space (distance, elevation gain, and difficulty). To make this interpretable, we additionally report a normalized value by comparing the system's diversity to a random-selection baseline (system / random), where values closer to 1 indicate diversity comparable to random sampling.

With $top_k = 20$ recommendations per query, the system achieves very high coverage (0.995), indicating that it retrieves almost the full item catalog across different inputs rather than repeatedly recommending the same subset. Geographic coherence shows that almost half of the results are within 50 km (0.473), while the mean distance is 121 km, suggesting that recommendations are a mixture of local and non-local hikes, but all of them are generally reachable

Table 2: Offline evaluation results (random sample size n = 200, top_k = 20).

Metric	Result
Coverage	0.995
Geo coherence (mean distance)	121.39 km
Geo coherence (within 50 km)	0.473
Diversity (raw feature-space distance)	238.80
Diversity (random baseline)	385.13
Diversity (normalized: system / random)	0.620
Sample size	200

for a day-trip. Diversity is moderate: the normalized diversity score of 0.620 means the system reaches about 62% of the diversity expected from random selection in the used feature space (distance, elevation gain, difficulty), i.e., recommendations are meaningfully varied but still notably more similar than a random baseline.

4.2 System Performance

This section evaluates the runtime performance and deployment resource footprint of the contextual recommender system. Measurements were taken from the production deployment on Railway and from local timing taken on the `/api/recommend_contextual` endpoint.

4.2.1 Backend Pipeline Breakdown. To understand the computational overhead of the recommender itself, we measured the runtime of the individual components of the contextual recommendation pipeline. Specifically, we focused on the execution time of the following functions:

```
CBF.recommend_similar(),
GSI.get_close_items(),
GSI.get_closest_items(), # GSI fallback
Weather_Recommender.recommend(),
HikeRecommender.recommend_contextual() # Score aggregation
```

Table 3 shows the measured functions runtime. The geographic filtering step (KD-tree), content-based retrieval (precomputed similarity matrix), and score aggregation each require less than 1 ms. In contrast, the weather-based recommendation dominates the backend runtime.

Table 3: Offline contextual recommendation runtime breakdown.

Component	Time	% of Total
Geographic filtering (KD-tree)	<1 ms	<1%
Content-based (precomputed matrix)	<1 ms	<1%
Score aggregation	<1 ms	<1%
System overhead	~23 ms	~16%
Weather recommender	~124 ms	~84%
Total backend time	**~147 ms**	**100%**

This confirms that the recommender design is computationally efficient due to (i) precomputation of content-based similarities and (ii) lightweight candidate filtering via spatial indexing. The primary performance bottleneck is the weather recommendation call, which is expected since it is the only function that relies on external api calls to fetch live weather data. Additionally the measurements reveal a system overhead of roughly 23ms.

4.2.2 Deployment Resource Usage (CPU & Memory). The deployment on railway contains both the front- and backend dockerized together. Table 4 shows the observed system load during a period of roughly two hours of normal operation. The system maintains a moderate memory footprint and low CPU usage with short request-level spikes.

Table 4: Deployment resource usage in production (Railway).

Resource	Observed Usage
Memory	~590 MB (frontend, backend, Nginx, cached data)
CPU	Up to 0.5 vCPU during requests; close to 0 at idle

Overall, the resource consumption is compatible with low-cost cloud hosting and indicates that the recommender pipeline itself is not computationally heavy.

4.2.3 Key Findings.

- **Compute efficiency:** core recommendation logic executes in sub-millisecond time for each major stage.
- **Main bottleneck:** external weather enrichment accounts for ~84% of backend runtime.
- **Scalability implication:** the system is expected to scale well with more users, but throughput is constrained by the weather dependency.

4.3 User Testing

After deployment, we conducted informal usability testing with a small group of friends and family (n = 12) to gather feedback on the system. Participants were asked to freely explore the application, set their hiking preferences, and evaluate whether the recommendations matched their expectations. Although not a formal study, this testing provided useful insights that metrics alone could not capture. Users appreciated the score breakdown component, noting that seeing *why* a trail was recommended made them trust the suggestions more. Several participants discovered new trails through the connection graph, confirming that our discovery feature adds value beyond the main recommendation list. We also noticed some unexpected usage patterns. Some users preferred to set hypothetical conditions (e.g., planning a summer hike during winter) rather than using live weather data. This showed us the importance of supporting both real-time and planning-ahead use cases. Others asked for additional filters like dog-friendliness or parking availability, pointing to possible future improvements. The testing process showed us how valuable it is to involve real users early on. While evaluation metrics give objective numbers, they cannot replace watching how people actually use the system. Even these informal sessions revealed small usability issues—like unclear icons and suboptimal defaults—that we were able to fix before the final release.

5 CONCLUSION AND FUTURE WORK

Our evaluation demonstrates that the system achieves near-complete catalog coverage (99.5%) while maintaining meaningful geographic coherence, with nearly half of recommendations falling within 50 km of the query location. The lightweight architecture, with core recommendation logic executing in sub-millisecond time, enables deployment on modest cloud infrastructure. Informal user testing revealed that explainability features enhanced user trust and supported the discovery of trails beyond the primary recommendation list.

Despite these successes, several limitations remain. The system requires more detailed evaluations and hyperparameter optimizations. Furthermore, it currently operates in a stateless manner, unable to learn from user interactions over time. The following subsections outline directions for addressing these limitations and extending the system's capabilities.

5.1 Hyperparameter Optimization

We attempted to optimize hyperparameters to improve recommendation quality by performing a grid search over the geometric mean of all evaluation metrics, aiming to determine the component weights that maximize overall evaluation performance. However, we found that these quantitative metrics do not necessarily correspond to the perceived quality of recommendations during user testing. Due to time constraints, we therefore settled on hyperparameters that best matched user expectations.

This highlights the need for additional evaluation metrics that better capture perceived recommendation quality and can therefore be used to optimize the hyperparameters.

5.2 Data Expansion

The current database is limited to approximately 1,160 trails from Oregon, which constrains its utility for users in other regions. A natural extension would be to incorporate trail data from additional sources like major hiking destinations (Colorado, the Alps, Patagonia).

Expanding the dataset introduces several technical challenges. Trail descriptions from different sources may vary in style, length, and terminology, potentially degrading the quality of the semantic recommender. Hike metadata may differ, requiring further normalization. Additionally, scaling to multiple thousands of trails would likely require more complex indexing structures and potentially approximate nearest-neighbor methods for similarity computation. However, addressing these challenges would significantly broaden the system's applicability.

5.3 Personalization

Implementing persistent user profiles would enable the system to remember past preferences, favorite trails, and completed hikes, reducing input burden and improving recommendation relevance over time.

Measuring user behavior, such as which recommendations users click on, would enable collaborative filtering approaches that utilize patterns across users with similar tastes. This could surface trails that users with comparable hiking histories have enjoyed, complementing the existing content-based and semantic strategies.

6 CONTRIBUTIONS (TEAM 2)

Table 5 summarizes each team member's contributions to the project.

Table 5: Team members and contributions

Team Member	GitHub Handle	Contribution
Yuancan Stefan Li	@Anunuemus	• Weather-based recommender • Documentation • Data preparation
Ruben Kolbusa	@rubenkol	• Database and data preparation • Documentation and Presentation • User Interface • Backend and Frontend harmonization • Context-based recommender
Caspar Pagel	@CasCodes	• Content-based filtering • Geospatial indexing • Topic-based recommender • Graph-based recommender • Evaluation • FastAPI server • Documentation • Cloud deployment

All team members contributed to system design discussions, debugging, weekly presentations, and this final report.

6.1 Disclosure of AI Usage

AI tools were used to assist with grammatical correction and LaTeX formatting of this report.

REFERENCES

[1] A. Schwietering, M. Steinbauer, M. Mangold, *et al.*, "Digitalization of planning and navigating recreational outdoor activities," *German Journal of Exercise and Sport Research*, vol. 54, pp. 107–114, 2024. DOI: 10.1007/s12662-023-00927-1. [Online]. Available: https://doi.org/10.1007/s12662-023-00927-1.

[2] G. Adomavicius and A. Tuzhilin, "Context-aware recommender systems," *AI Magazine*, vol. 32, no. 3, pp. 67–80, 2011. DOI: 10.1609/aimag.v32i3.2364. [Online]. Available: https://doi.org/10.1609/aimag.v32i3.2364.

[3] W. Fan, Z. Zhao, *et al.*, "Recommender systems in the era of large language models (llms)," *IEEE Transactions on Knowledge and Data Engineering*, 2024. DOI: 10.1109/TKDE.2024.3392335. [Online]. Available: https://doi.org/10.1109/TKDE.2024.3392335.

[4] K. Bao, J. Zhang, Y. Zhang, and X. He, "Large language models for recommendation: Past, present, and future," in *Proceedings of the 18th ACM Conference on Recommender Systems (RecSys '24)*, 2024. DOI: 10.1145/3626772.3661383. [Online]. Available: https://doi.org/10.1145/3626772.3661383.

[5] E. M. Renieris, D. Kiron, S. Mills, and A. Kleppe, "Ai explainability: How to avoid rubber-stamping recommendations," *MIT Sloan Management Review*, Jun. 2025, Accessed: 2026-01-31. [Online]. Available: https://sloanreview.mit.edu/article/ai-explainability-how-to-avoid-rubber-stamping-recommendations/.

[6] Oregon Hikers, *Oregonhikers.org*, https://www.oregonhikers.org/, Accessed: 2025-10-27.

[7] Railway, *Railway.com*, https://www.https://railway.com/, Accessed: 2025-12-10.

[8] OpenAI, *Chatgpt (gpt-4o mini)*, https://openai.com/chatgpt, Accessed: 2026-01-23, 2024.

[9] R. Campos, V. Mangaravite, A. Pasquali, A. Jorge, C. Nunes, and A. Jatowt, "Yake! keyword extraction from single documents using multiple local features," *Information Sciences*, vol. 509, pp. 257–289, 2020, ISSN: 0020-0255. DOI: 10.1016/j.ins.2019.09.013. [Online]. Available: https://doi.org/10.1016/j.ins.2019.09.013.