令和７年度　特別研究報告書

移動経路と降水予報を統合した動的降水確率予測手法

龍谷大学　先端理工学部　知能情報メディア課程

Y220163　長井翔平

指導教員 三好 力 教授

研究梗概

近年、経路探索アプリと高解像度降水ナウキャストは広く普及しているが、徒歩や自転車等の短距離の移動での目的ではそれぞれ独立して利用されていることが多い。そのため、ユーザーは「経路」と「天候」の情報を自ら頭の中で統合し、移動中に雨に遭遇するリスクを主観的に判断する必要があった。

本研究では、この問題を解決するため、短距離移動を目的とした経路探索と降水予報を動的に統合するシステムを提案する。本システムは、ユーザーが設定した経路と移動速度に基づき、経路上の各地点への到着予測時刻を算出する。そして、その未来時刻における各地点の降水予報を API 連携により取得し、地図上に可視化する。

目次

第 1 章　はじめに

1.1　研究背景

1.1.1　気候変動という社会的背景

近年、気候変動の影響により、日本国内における局地的な大雨（いわゆるゲリラ豪雨）のリスクが高まっている。気象庁の統計[1]によると、全国のアメダスで観測された 1 時間降水量 50mm 以上の短時間強雨の年間発生回数は、統計期間（1976 年～2024 年）において明らかな増加傾向を示している。特に、最近 10 年間の平均発生回数は、統計開始当初の 10 年間と比較して約 1.5 倍に達している。このような突発的かつ激しい気象変化は、自動車のような雨風を凌ぐ手段を持たない徒歩や自転車の移動者にとって深刻な脅威であり、正確な回避行動を取るための情報は死活的に重要である。

1.1.2　既存技術の現状と高度化

一方、情報技術の分野では、スマートフォンやナビゲーションアプリの普及により、移動支援技術の利用が拡大している。特に気象予測においては、気象レーダーを基にした高解像度降水ナウキャストが重要な役割を果たしている。これは、全国 20 箇所に設置された気象ドップラーレーダーに加え、自治体が保有する雨量計データ、ウィンドプロファイラやラジオゾンデの高層観測データ、国土交通省レーダ雨量計のデータ等を統合的に解析するシステムである。これにより、250m 解像度という高精細な降水分布を 30 分先まで高精度に予測することが可能となっており[2]、すでに多くの天気予報サービスでリアルタイムの意思決定支援情報として活用されている。

1.1.3　解決すべき課題：情報の分断と認知的負荷　しかし、既存のシステムには重大な欠点がある。それは「経路情報（空間）」と「気象情報（時間）」がそれぞれ独立して提供されている点である。 ナビゲーションアプリは最短経路を提示するが、その未来の天候は不明であり、天気予報アプリは雨の有無を伝えるが、それがユーザーの移動経路上のどこで発生するかまでは示さない。一部の地図アプリでは雨雲レーダーを重ねて表示する機能も存在するが、これはあくまで現時点または特定の未来時刻の「面」の情報を表示するに留まる。 そのため、ユーザーは「自分が移動する速度」と「雨雲が移動する速度」という 2 つの動的な要素を自らの頭の中で統合し、将来のリスクをシミュレーションする必要がある。このような情報の統合処理はユーザーに高い認知的負荷を強いるものであり、結果としてリスク判断を主観的かつ曖昧なものにさせているのが現状である。

1.1.4　本研究の目的

本研究では、これらの問題を解決するため、短距離移動を目的とした経路探索と降水予報を動的に統合するシステムを提案する。本システムは、ユーザーの移動経路と通過時刻に合わせて降雨リスクをシステム側で自動的に計算・診断するものである。これにより、徒歩や自転車での移動における意思決定の精度向上と、真にリアルタイム性の高い天気予報

の実現を図ることを目的とし、実際に出発地と目的地を設定して API 連携による可視化を行うことで、その有効性の検証を行う。

第 2 章　既存技術

意思決定の精度向上、リアルタイム性の高い天気予報の実現のためいくつかの技術が既存する。

2.1　ピンポイント天気予報 / 高解像度降水ナウキャスト

天気予報は、ユーザーの意思決定を支援する最も基本的な既存技術である。Yahoo!天気やウェザーニュースなどのサービスでは、アメダスや気象レーダーの観測データを基にした高精度な天気予報が提供されている。特に、高解像度降水ナウキャスト（雨雲レーダー）は、雨雲の分布と動きをリアルタイムに近い形で視覚的に表示し、数十分先の雨を予測する。

これらの技術は、特定の「地点」における天候を高精度で予測することに特化している。しかし、その予測はあくまで静的な地点を対象としており、ユーザー自身の「移動」という動的な要素を直接考慮していない。そのため、出発地と目的地が晴れ予報であっても、その経路の途中で雨雲に遭遇するリスクをユーザー自身が推測する必要があり、その判断は主観に依存するという課題がある。

全国の天気予報（７日先まで）

2025年10月11日23時発表

日付	今夜 11日(土)	明日 12日(日)	明後日 13日(月)	14日(火)	15日(水)	16日(木)	17日(金)	18日(土)
釧路	曇	曇	晴時々曇	晴時々曇	曇時々晴	曇時々晴	曇	曇一時雨
降水確率(%)	-/-/-/0	0/10/20/20	20	20	30	30	40	50
信頼度	-	-	-	A	A	B	B	C
最低/最高(℃)	- / -	6 / 15	7 / 16	6 / 15	7 / 17	7 / 17	7 / 16	10 / 16

図 1　「全国の天気予報」（気象庁ホームページより）

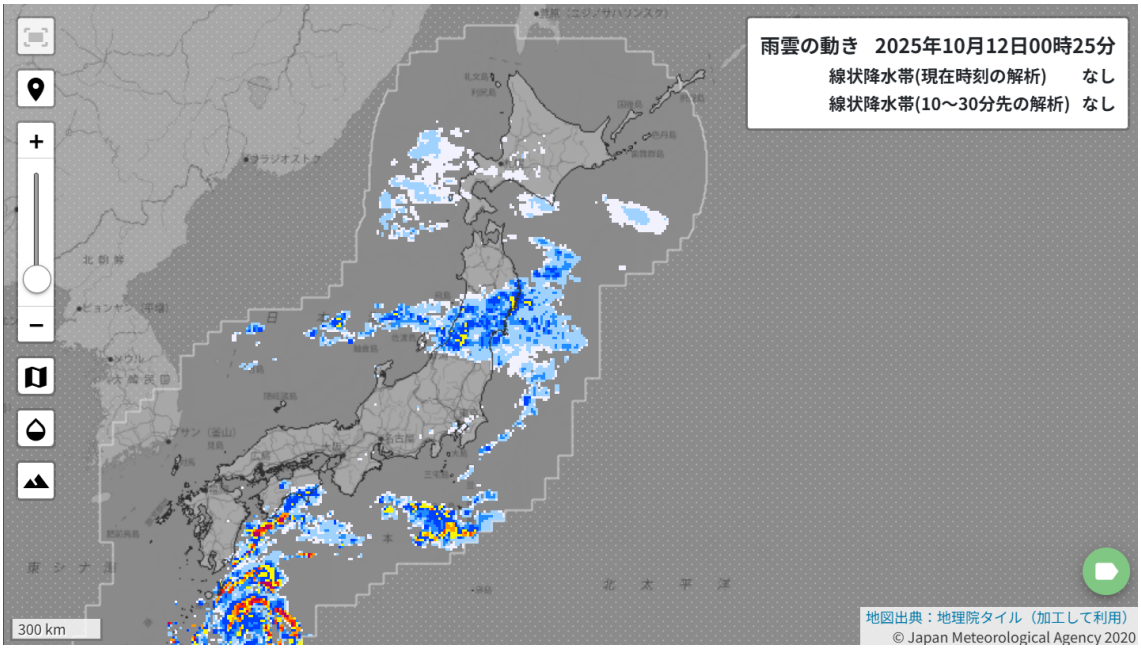


図２　「雨雲の動き」（気象庁ホームページより）

2.2　Yahoo!マップ/Yahoo!カーナビ

本研究と近しいものとして Yahoo!マップ/ Yahoo!カーナビがある。Yahoo!マップ/ Yahoo!カーナビでは 2 地点の経路探索機能に加え、雨雲レーダーを画面に表示する機能がある。経路探索と雨雲レーダーを組み合わせたシステムではあるが、降水の有無はユーザーの判断に左右される。

その他に似たシステムとしてレイヤーを重ねることで実現できる Google マップや NAVITIME がある。[3]

2.3　トラックカーナビ

トラックカーナビは高精度な到着予想時刻表示、車高・車幅・車長・大型車規制などと車両別に走行実績が多い道路を考慮したルート検索、複数の訪問先の最適訪問順の自動計算、大型車が駐車可能な施設検索など、トラックドライバー向けに特化した日本初のカーナビアプリである。トラックドライバー向けに特化した機能の一つにルート検索時に運行ルート上に 30mm/h 以上の大雨が予想される場合に、そのエリアを回避するルートを提案・ナビゲーションする機能[4]がある。この機能は経路探索と降水量を考慮したシステムであり動的であるが、本研究の目的である徒歩、自転車での移動の改善には想定されているエリアが広範囲かつ徒歩、自転車で雨雲を避けることは現実的でない。

第 3 章　提案手法

3.1　既存技術の改善案

ユーザーの意思決定精度を向上させる方法として、「情報の自動解析」と「リスクの定量的可視化」を行うことで、既存技術の課題を解決できると考える。

Yahoo!マップのような既存技術は、経路と雨雲を重ねて表示するが、ユーザーがその情報を解釈し、リスクを判断する必要がある。本研究では、ユーザーの移動経路上における動的な降雨リスクを提示するため、複数の既存技術を統合した Web アプリケーションシステムを提案する。本システムは、ユーザーから出発地と目的地の指定を受け付け、経路探索と気象予報 API を連携させることで、これまでユーザーの主観的な解釈に委ねられていた時空間的なリスクシミュレーションを自動化する。

3.2　システム概要

本システムのバックエンドには Python を採用した。Python は豊富な科学技術計算ライブラリを有しており、地理空間情報の処理（Geopy, OSMnx）[5][6]やデータ解析に適しているためである。また、Web フレームワークには軽量かつ拡張性の高い Flask[7]を採用し、将来的な機能拡張や API 連携の柔軟性を確保した。全体の処理フローは以下の通りである。(図 5 参照)

1.ユーザーが Web ブラウザ経由で対象都市（京都・沖縄・新潟等）を選択し、出発地と目的地の住所を入力する。

図 3　初期入力画面

2.Geopy ライブラリを用いて、入力された住所を緯度経度の座標情報に変換する。この際、選択された都市の地理的範囲（Bounding Box）を検索条件に加えることで、同名の他地域への誤認識を防ぐ。

3.空間解析として、OSMnx ライブラリを用いて選択された都市の OpenStreetMap 道路ネットワークデータを取得し、最短経路を算出する。

4.時間解析として、算出された経路長とユーザーが選択した移動速度（徒歩・自転車）に基づき、経路上に設定された各サンプル地点への到着予測時刻を計算する。

5.時空間統合解析として、各サンプル地点の緯度経度と到着予測時刻をキーとし、OpenWeatherMap API に気象データをリクエストする。

6.取得した降水予報データを基に、Folium ライブラリを用いて経路やリスク地点を地図上に描画し、最終的な結果を Web ブラウザに表示する。

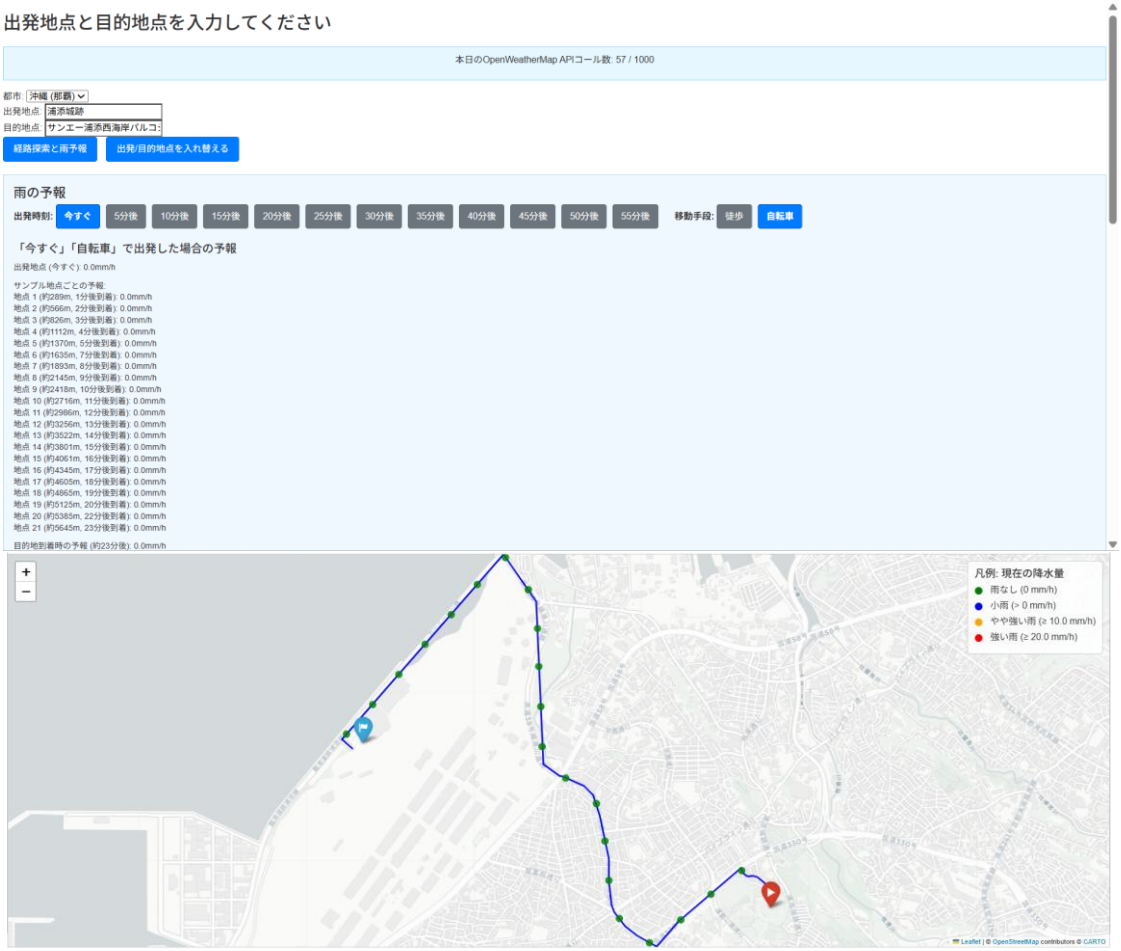


図 4　最終結果の表示

3.2.1　空間解析：経路計算とサンプリング

本システムにおける空間解析は、経路の計算と、その経路上の分析点（出発地、目的地、サンプル地点）の設置という 2 段階で構成される。

まず、ユーザーによって指定された出発地と目的地の住所を Geopy ライブラリで緯度経度に変換し、OSMnx で構築した対象都市の道路ネットワークグラフ上の最寄りノードを特定する。次に、NetworkX ライブラリのダイクストラ法に基づき、2 ノード間の最短経路（距離ベース）を算出する。

次に、算出された経路上を始点からたどる。高解像度降水ナウキャストが 250m メッシュであるが、本研究で使用する OpenWeatherMap API は特定のメッシュは定義されておらず緯度・経度を設定し予測値を返す。そのため、無料プランにおける 1 日のコール回数制

限や実際の雨雲レーダーのメッシュサイズを考慮し、経路に沿って移動した際の始点からの直線距離が 250m 間隔となる地点の座標を、道路形状（エッジ）上の線形補間により算出し、新たなサンプル地点として設定する。これにより、交差点の少ない長い一本道やカーブの多い道路においても、等間隔かつ正確なサンプリングが可能となる。

3.2.2　時空間統合解析：動的予報取得

本手法の核心は、空間（場所）と時間（未来）の情報を統合して、未知の情報を生成する点にある。

まず、3.2.1 で設定された各サンプル地点までの経路距離を geopy.distance.geodesic ライブラリから算出する。その距離とユーザーの移動速度（例: 自転車 250m/分）から、各サンプル地点への到着が何分後になるかという到着予測時刻を計算する。

次にユーザーが「経路探索」ボタンを押すと出発地・目的地・および全てのサンプル地点の分の API リクエストを一斉に送信し、コール回数節約のため 60 分後までの minutely 予報データを全て事前に取得する。

次に、各サンプル地点までの経路距離と、それらの予報データを紐づけた統合データセット（all_forecast_data）を構築し、Web ブラウザ（クライアント）に渡す。

ブラウザ側では、ユーザーが出発時刻や移動モードを変更するたびに、この統合データセットを参照する。JavaScript が、選択された条件（出発遅延時間、移動速度）から各地点への到着予測時刻を再計算し、データセット内から対応する時刻の降水量を引用して表示を更新する。これにより、ユーザーはサーバーとの通信を待つことなく、リアルタイムに予測結果を切り替えることが可能となる。

3.2.3　予測結果の定量的可視化

ユーザーの直感的な理解を促し、意思決定を支援するため、解析結果を定量的に可視化して提示する。具体的には、Folium ライブラリを用いて生成した地図上に、以下の情報を描画する。

最短経路（青い線）：ユーザーが進むべきルート。

サンプル地点（赤い点）：降水予報を取得した分析ポイント。各マーカーの色は、その地点における「現時点」の降水強度に応じて動的に変更される。緑は降雨なし、青は小雨、オレンジは中程度の雨、赤は激しい雨を示し、ユーザーは地図を見るだけで、経路全体の現在の降雨状況を直感的に一目で把握することができる。

予測情報（ポップアップ）：各サンプル地点をクリックすると、「経路距離：約 Xm」「現在の降水量: Z mm/h」といった具体的な数値情報が表示される。

テキストによる可視化として、地図の上または下に専用の情報表示エリアを設ける。このエリアには、出発時刻や移動モードの選択に応じて、以下の具体的な予測情報が一覧形式でリアルタイムに更新・表示される。
これにより、ユーザーは「いつ、どこで、どれくらいの雨に遭う可能性があるか」を一目で把握でき、出発時刻を調整する、傘を用意するといった具体的な行動へと繋げることが可能となる。
出発地点の予測: 「出発地点（X 分後): Y mm/h」

サンプル地点ごとの予測:
地点 1 (経路距離: A m, 到着: B 分後): C mm/h
地点 2 (経路距離: D m, 到着: E 分後): F mm/h　等

目的地点の予測: 「目的地到着時（総移動時間: G 分後): H mm/h」

これにより、ユーザーは「いつ、どこで、どれくらいの雨に遭う可能性があるか」を地図で視覚的に捉えつつ、テキスト情報で具体的な数値を確認できる。この 2 つの情報を組み合わせることで、出発時刻を調整する、傘を用意するといった、より精度の高い行動判断が可能となる。

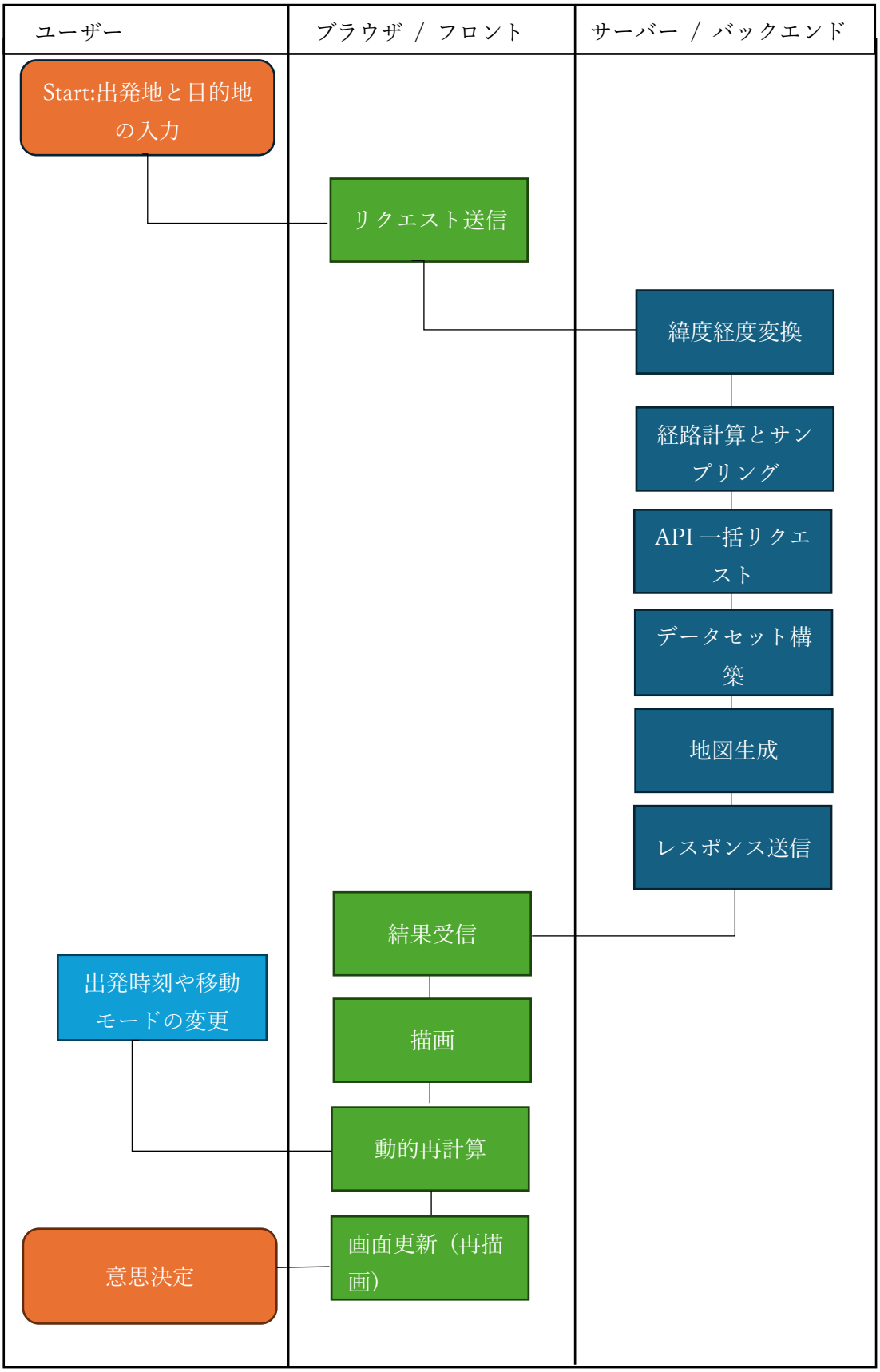

ユーザー
ブラウザ / フロント
サーバー / バックエンド
Start:出発地と目的地の入力
リクエスト送信
緯度経度変換
経路計算とサンプリング
API 一括リクエスト
データセット構築
地図生成
レスポンス送信
結果受信
描画
出発時刻や移動モードの変更
動的再計算
画面更新（再描画）
意思決定

図 5　処理フロー

第 4 章　実験

本章では実験目的、実験概要、実験結果について記述する。

4.1　実験目的

本研究のシステムとしての正誤性、意思決定の精度向上を測ることを目的とする。

4.2　実験概要

正誤性、意意思決定支援の有効性を測るため実験を 2 つ行う。

4.2.1　正誤性の実験概要

本システムの降水量予測の有効性を検証するため、気象庁が提供する「高解像度降水ナウキャスト」との比較実験を行う。気象庁が提供する「高解像度降水ナウキャスト」は、気象ドップラーレーダーによる広域的な観測データに対し、地上の雨量計（アメダス等）による局所的かつ高精度な観測値を組み合わせることで補正を行った解析雨量データである。

通常、雨量計のみのデータでは観測地点間の空白地帯が生じるため、本研究のようにユーザーが任意の経路を連続的に移動するケースにおいては、経路上の正確な降水状況を捉えきれない可能性がある。一方で、気象レーダーのみでは地形による遮蔽や距離減衰の影響を受け、雨量の定量的な精度に課題が残る。

高解像度降水ナウキャストは、これら双方の利点を統合し、250m メッシュという極めて細かい空間分解能で面的な降水分布を再現している。これにより、観測地点が存在しない道路上であっても、極めて現実に近い降水強度を推定することが可能となる。したがって、移動経路上の局所的なリスク判定を行う本実験において、現在利用可能なデータの中で最も信頼性が高く、比較対象として最適であると判断した。[2]

実験方法

本システムが出力した、京都市内の 3 つの異なる（金閣寺エリア、八坂神社エリア、北山エリア）経路上の各地点および到着予測時刻における降水量と、それに対応する時空間の高解像度降水ナウキャストの実況値を目視にて照合する。

評価にあたっては、単なる数値的な誤差よりも、ユーザーが直感的に把握する「雨の強さ」のレベルが一致しているかを重視する。そのため、本研究では気象庁の雨雲レーダー表示で用いられている降水量の図 6 凡例を評価基準として採用し、予測値と実況値が同一の区分（色）に含まれる場合を「正確」と判定する。

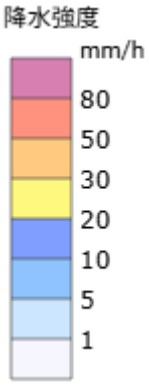


図 6 凡例

4.2.2　意思決定支援の有効性検証概要

本システムがユーザーの意思決定に与える効果を定量的に評価するため、大学生および成人を含む被験者 10 名を対象に、比較実験および統計的検定を行う。

実験方法

実験は Google フォームを用いたアンケート形式で実施する。比較対象として、既存のナビゲーションサービス（Yahoo!マップ）の雨雲レーダー表示を再現した画面と、本システムが生成した予測画面の 2 種類を用意する。

被験者に対し、それぞれの画面から「経路上の降雨リスク」を判断するタスクを課し、以下の 2 項目について 5 段階のリッカート尺度で主観評価を求める。

1.判断の難易度**:** リスクを判断することがどれほど容易であったか（1:非常に難しい ～ 5:非常に簡単）

2.判断の体感時間**:** 意思決定にどれほどの時間を要したと感じたか（1:非常に時間がかかった ～ 5:瞬時に判断できた）

得られた回答データについて、提案手法群と既存手法群の平均値の差を対応のない t 検定を用いて分析し、本システムによる意思決定の精度向上および認知負荷の軽減効果を検証する。

第 5 章　結果

5.1　正誤性の実験結果

京都市内の 3 つの異なるルート（金閣寺エリア、八坂神社エリア、北山エリア）において、提案システムによる予測値と、気象庁高解像度降水ナウキャストによる実況値（正解データ）を比較した結果を以下に示す。

5.1.1　全体的な傾向 図 7（散布図）に示す通り、全体として実況値が 0〜5mm/h 程度の弱い降雨においては、本システムの予測値は概ね実況と一致し、高い正解率を示した。しかし、実況値が 20mm/h を超える強い降雨（図中の右側の領域）においては、本システムの予測値が実況値を大幅に下回る過小評価の傾向が顕著に見られた。

5.1.2　地域別の特性 場所ごとの詳細な比較結果（図 8）を以下に述べる。

・金閣寺-嵐山ルート: 比較的高い精度で推移した。特に 8 月 27 日の事例では、降水量の増減トレンドを概ね捉えることができていた。

・八坂神社-知恩院ルート: 小雨の検知は正確であったが、8 月 25 日のような突発的な激しい雨（実況 80mm/h 超）に対しては、システム上の予測が 10mm/h 程度に留まり、大きな乖離が生じた。

・北山駅-下鴨神社ルート: 最も精度が低い結果となった。特に山間部に近い北山周辺では、実況では「激しい雨（30〜50mm/h）」が観測されているにもかかわらず、システムは「降雨なし」または「小雨」と予測するケースが複数回確認された

5.1.3　評価基準に基づく判定 4.2.1 で定義した評価基準（気象庁の凡例に基づく階級一致）を用いた結果、全サンプル地点における雨の強さの階級一致率は約 35.3%となり、特に夏季の局地的な豪雨に際しては、ユーザーにリスクを過小に伝えてしまう課題が残る結果となった。

表 1　正誤性の実験結果表

金閣寺-嵐山					
2025/08/17 13~（自転車）	経路距離(サンプル地点)	時間	本システム	雨雲レーダー実況	評価
	現在地	0	0	0~1	正確
	8	10	0	0~1	正確
	16	20	0.1	0~1	正確
	目的地	30	0.2	0~1	正確
16~（自転車）	現在地	0	0.6	30~40	不正確
	8	10	0.6	1~5	不正確
	16	20	2.4	1~5	正確
	目的地	30	5	1~5	正確

2025/8/25 (自転車)	現在地	0	2.7	50~80	不正確
	8	10	7.1	40~50	不正確
	16	20	8.6	40~50	不正確
	目的地	30	8.6	40~50	不正確
2025/8/27 (自転車)	現在地	0	5	5~10	正確
	8	10	6	30~40	不正確
	16	20	6.5	5~10	正確
	目的地	30	6.5	5~10	正確
八坂神社-知恩院					
2025/08/17 13~ (徒歩)					
	現在地	0	0.2	1~5	不正確
	目的地	10	0.1	10~20	不正確
16~ (徒歩)	現在地	0	0	1~5	不正確
	目的地	10	0	1~5	不正確
2025/8/25 (徒歩)	現在地	0	10	80~	不正確
	目的地	10	8.9	80~	不正確
2025/8/27 (徒歩)	現在地	0	3	0	不正確
	目的地	10	4.9	1~5	正確
北山駅-下鴨神社					
2025/08/17 13~ (自転車)					
	現在地	0	0	30~40	不正確
	目的地	10	0	50~80	不正確
16~ (徒歩)	現在地	0	0.1	1~5	不正確
	2	10	0.1	30~40	不正確
	5	20	0.2	50~80	不正確
	目的地	30	0.2	50~80	不正確
2025/8/25 (自転車)	現在地	0	0.7	80~	不正確
	目的地	10	0.7	30~40	不正確
2025/8/27 (自転車)	現在地	0	5	1~5	正確
	目的地	10	6.5	5~10	正確

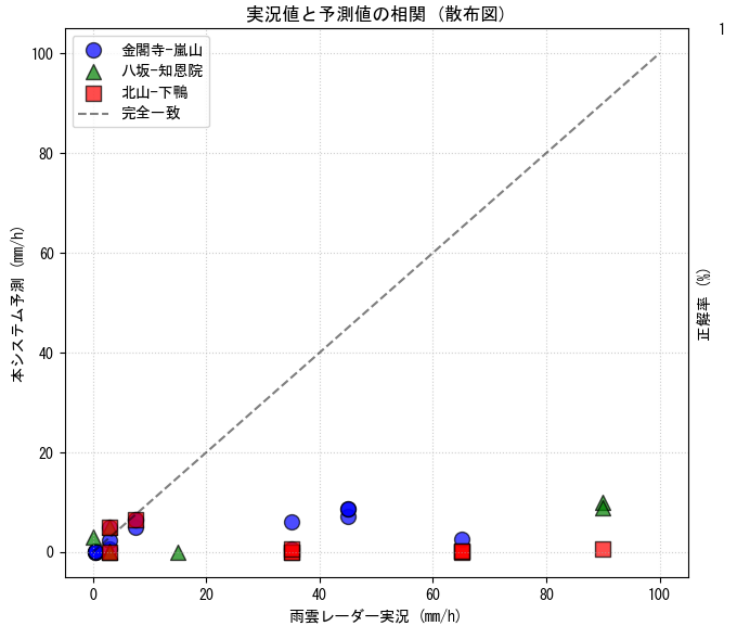


図 7　実況地と予測値の相関（散布図）

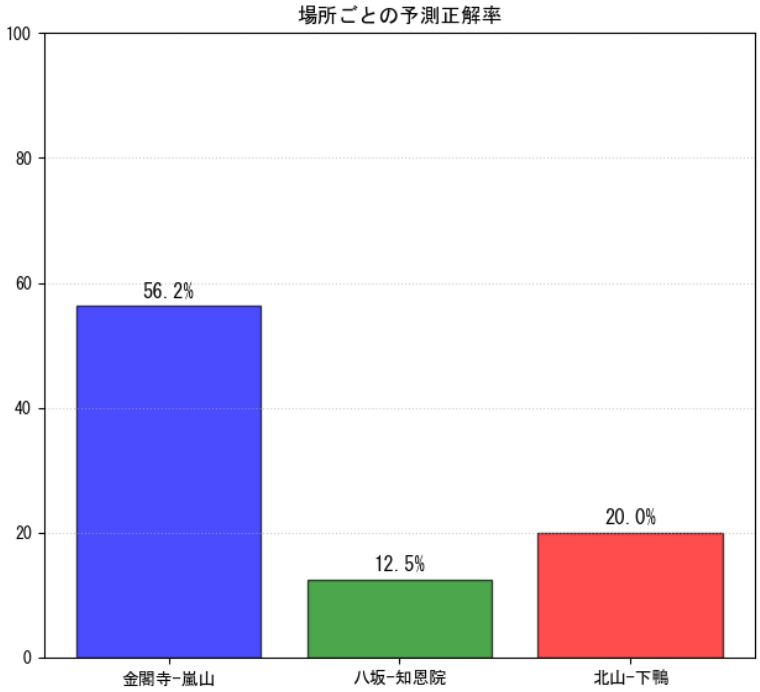


図 8 場所ごとの予測正解率

5.2　意思決定支援の有効性検証結果

回答データの精査を行ったところ、1 名の参加者（参加者 ID: 7）において、一方の条件に対し全ての項目で最低評価（1 点）をつける等の極端な回答傾向（床効果）が確認され

た。 本研究では、これが回答態度や特定のバイアスによる外れ値であると判断し、分析対象から除外した。したがって、以下の分析は残る 9 名のデータを用いて行った。
t 検定の結果：条件 A および条件 B に対する評価の平均値を算出し、対応のある t 検定（片側検定）を行った。

5.2.1　判断の難易度検証結果

統計的な有意差は認められなかったものの（t(8) = 0.37, n.s.)、記述統計量においては条件 A (M = 4.22）が条件 B ($M = 4.19）をわずかに上回った。しかしその差は極めて小さく、両条件の判断難易度は同程度であると考えられる。

表 2 条件 A と条件 B における評価得点の比較（N=9）

条件	平均値（Mean）	標準偏差（SD）	t 値	p 値
条件 A	4.22	0.81	0.37	n.s.
条件 B	4.19	0.84		

5.2.2　判断の体感時間検証結果

先の分析と同様の基準により 1 名の参加者を除外し、残る 9 名のデータを用いて対応のある t 検定を行った。 その結果、有意差は認められなかったものの、条件 A (M = 3.93, SD = 1.14）は、条件 B (M = 3.78, SD = 0.53）をわずかに上回った。しかしその差は極めて小さく、両条件の判断難易度は同程度であると考えられる。

表 3 条件 A と条件 B における評価得点の比較（N=9）

条件	平均値（Mean）	標準偏差（SD）	t 値	p 値
条件 A	4.07	1.11	1.17	n.s.
条件 B	3.92	0.77		

第 6 章　考察

6.1　予測誤差の要因分析

実験結果において、特に「北山駅-下鴨神社」ルートや、8 月 25 日のような突発的な豪雨事例において、本システムの予測値が実況値（気象庁データ）と比較して著しく過小評価となる傾向が見られた。この主要な要因は、本システムがデータソースとして利用している OpenWeatherMap API の特性と限界 にあると考えられる。

第一に、更新頻度と即時性の差異である。比較対象とした気象庁の高解像度降水ナウキャストは、国内の高密度なレーダー網（XRAIN 等）を活用し、数分単位の急激な変化を捉えている。対して、OpenWeatherMap は広域気象モデルに基づいており、その空間解像度は数 km 四方程度と推測される。一方、日本の気象庁が提供する高解像度降水ナウキャストは 250m メッシュという極めて細かい粒度で観測を行っている。特に京都のような三方を山に囲まれた盆地地形においては、局地的な地形性降雨が発生しやすく、グローバルモデルではこれらを十分に捉えきれなかったことが、予測精度の乖離を生んだ最大の要因と考えられる。

第二に、地形効果の考慮である。特に誤差が大きかった北山エリアは山間部に隣接しており、地形性降雨が発生しやすい。国内の観測網に特化した気象庁データと比較し、グローバル API ではこうした微細な地形効果による降水量の増幅を十分に反映しきれなかったと推察される。

以上のことから、本システムのロジック（時空間統合）自体は有効に機能しているものの、最終的な出力精度は入力データソース（API）の品質に強く依存することが明らかとなった。今後の展望として、より高解像度な国内特化型 API への切り替えや、複数の気象ソースを統合することで、予測精度の底上げが可能であると考えられる。

6.2　意思決定支援に関する検証結果の考察

6.2.1　統計的検出力の不足

第一の要因は、アンケートのサンプルサイズ（被験者数）である。本実験では 10 名を対象としたが、これは一般的なユーザビリティテストと比較しても小規模である。 t 検定において有意差を検出するためには、ある程度のサンプルサイズが必要となる（統計的検出力）。今回の実験では、提案手法の結果としてわずかに数値に差があったものの少人数のデータのばらつきに埋もれてしまい、統計的な確証を得るには至らなかった可能性が高い。今後、被験者数を数十名規模に拡大することで、有意差が顕在化する可能性がある。

6.2.2　既存技術の完成度とシナリオの選定

第二の要因は、比較対象とした既存技術（Yahoo!マップ）が既に高いユーザビリティを有している点である。 単純な降雨シナリオ（例：広範囲の雨雲が明らかに経路を覆っている場合）においては、既存の雨雲レーダーの視覚情報だけでも十分に判断が容易であり、提

案システムによる「自動解析」の優位性が発揮されにくい状況（天井効果）にあったと考えられる。 本来であれば、雨雲の動きが複雑で目視判断が困難なシナリオを用意すべきであったが、前節（6.1）で述べた通り、本システムは現時点で複雑な気象条件下での予測精度に課題を残している。そのため、実験シナリオを比較的単純なものに限定せざるを得ず、結果として既存技術との明確な差別化が困難になったと推察される。 システムの予測精度が向上すれば、より判断難易度の高いシナリオでの検証が可能となり、提案手法の真価である「認知負荷の軽減効果」がより明確に示されると期待できる。

6.2.3　実験環境と UI の不整合

第三の要因は、提示した UI と閲覧デバイスの不整合である。 本システムは PC の Web ブラウザでの閲覧を前提に、地図と詳細なテキスト情報を並列表示する UI 設計を行っている。しかし、本実験では Google フォームを用いて実施したため、多くの被験者がスマートフォン等の小型画面で閲覧した可能性がある。 その場合、PC 向けに設計された詳細なテキスト情報や地図が縮小表示され、かえって視認性が低下し、情報の読み取りにストレス（認知負荷）を与えた可能性がある。本来の「情報を噛み砕いて伝える」というメリットが、デバイスの制約によって相殺されてしまったと考えられ、今後はレスポンシブ対応やスマートフォンアプリ化による UI の最適化が求められる。

6.3　社会実装に向けた課題と展望

今後の展望として、単なるリスクの提示にとどまらず、能動的な『回避ルート提案機能』の実装が挙げられる。現在のシステムは最短経路上の雨量を表示するのみだが、例えば『到着が 5 分遅れても雨に濡れないルート』や『雨雲が通り過ぎるまで雨宿りすべきカフェの提案』など、行動変容を直接促す機能を追加することで、システムの有用性は飛躍的に向上すると考えられる。また、本研究では考慮しなかった『屋根のあるアーケード街』や『地下通路』の情報を道路ネットワークデータに属性として付与することで、より実用的な雨天時ナビゲーションが実現可能となる。

第 7 章 まとめ

本研究では、近年の気候変動による局地的な降雨リスクの増大と、既存の移動支援技術における情報の分断という課題に着目し、短距離移動を目的とした経路探索と降水予報を動的に統合する Web アプリケーションシステムを開発した。

7.1 研究の総括

本システムは、ユーザーの移動経路と速度に基づき、経路上の各地点への到着予測時刻を算出することで、静的な地点予報では捉えきれない「移動中の降雨リスク」を可視化することを可能にした。また、地図上の色分け表示と詳細なテキスト情報を組み合わせることで、専門的な気象情報をユーザーが直感的に理解できる形へと自動解析・翻訳する新たな情報提示手法を提案した。

7.2 検証結果の要約

開発したシステムの有効性を検証するため、予測精度の検証（正誤性）と意思決定支援の有効性検証の 2 つの実験を行った。

第一に、予測精度の検証においては、気象庁の高解像度降水ナウキャストを正解データとして比較を行った。その結果、京都市内の平地部（金閣寺エリア等）や穏やかな降雨状況下においては、本システムは概ね正確な予測を提供し、実用性を有することが確認された。一方で、山間部に隣接する地域（北山エリア）や、突発的な激しい豪雨（ゲリラ豪雨）に対しては、利用したグローバル気象 API の空間解像度や更新頻度の特性により、降水量を過小評価する傾向や時間的なズレが生じることが明らかとなった。

第二に、意思決定支援の有効性検証においては、被験者を対象とした比較実験を行った。結果として統計的な有意差の検出には至らなかったものの、提案手法を用いたグループにおいて、判断の容易さや体感時間の短縮に関する肯定的な傾向が見られた。これにより、本システムが目指す「情報の自動解析による認知負荷の軽減」というコンセプト自体の有効性が示唆された。

7.3 結論

以上の結果より、本研究で提案した「経路探索と気象情報の動的統合」というアプローチは、従来の独立した情報提供では解決できなかった、移動者の主観的なリスク判断を客観的かつ定量的に支援する上で、極めて有効な手段であると結論付ける。

現時点ではデータソースの精度や UI 設計に改善の余地を残すが、これらは技術的な実装レベルの課題であり、本研究が示した「動的なリスク可視化」という概念の価値を損なうものではない。今後、より高精度な国内特化型気象データの統合や、スマートフォンに最適化されたインターフェースの実装、さらには能動的な回避ルート提案機能への拡張を行

うことで、本システムは実社会における安全で快適な移動を支える基盤技術となり得ると確信する。

謝辞

本研究を遂行するにあたり、終始懇切丁寧な御指導と御鞭撻を賜りました、三好教授に、心より感謝の意を表します。先生の多角的な視点からのご助言と、研究に対する真摯な姿勢のご教授なくしては、本論文を完成させることはできませんでした。
また、在学中、共に研究生活を過ごし、温かい励ましを頂きました三好研の諸先輩方、ならびに同期、後輩の皆様に深く感謝いたします。
加えて、本研究で使用した気象データや地図データの利用にあたり、貴重なオープンデータを提供してくださった気象庁、OpenStreetMap コミュニティ、および OpenWeatherMap の開発者の方々に敬意を表します。
最後に、大学生活の長きにわたり、精神的および経済的に私を支え、学業に専念できる環境を整えてくれた家族に、この場を借りて深く感謝いたします。

参考文献

[1] 気象庁: 大雨や猛暑日など（極端現象）のこれまでの変化.
https://www.data.jma.go.jp/cpdinfo/extreme/extreme_p.html
[2] 高解像度降水ナウキャスト
https://www.jma.go.jp/jma/kishou/know/kurashi/highres_nowcast.html
[3] Yahoo!カーナビに「雨雲レーダー」機能ができました
https://note.com/yahoo_carnavi/n/na414de288cda
[4] 『トラックカーナビ』、気象予報を考慮して迂回する「気象防災ルート検索」提供開始
https://corporate.navitime.co.jp/topics/pr/202409/09_5791.html
[5] Geopy Contributors: Geopy Documentation.
https://geopy.readthedocs.io/
[6] 【Python】OSMnx で車両走行道路の最短経路探索＆可視化をしてみる
https://qiita.com/moshi/items/e383491664d028cd2166
[7] allets Projects: Flask Documentation.
https://flask.palletsprojects.com/

付録

ソースコード

tannsaku.py

```
from flask import Flask, request, render_template
import osmnx as ox
import networkx as nx
import folium
import pickle
import os
from geopy.distance import geodesic
import time
import requests
import json
from datetime import datetime, timezone

# リクエスト間の遅延
time.sleep(1)
from geopy.geocoders import Nominatim

app = Flask(__name__)

CITY_MAP = {
    "kyoto": {
        "query": "Kyoto, Japan",
        "graph_file": "kyoto_graph.pkl"
    },
    "okinawa": {
        "query": "Okinawa Island, Japan",
        "graph_file": "okinawa_main_island_graph.pkl"
    }
}

OPENWEATHER_API_KEY =
os.getenv("OPENWEATHER_API_KEY")
SAMPLE_DISTANCE_THRESHOLD_M = 250 # 直線距離での
閾値
API_CALL_COUNT_FILE = "api_call_count.json"
MAX_API_CALLS_PER_DAY = 1000

WALK_SPEED_MPM = 80
BIKE_SPEED_MPM = 250

if not OPENWEATHER_API_KEY:
    raise ValueError("OPENWEATHER_API_KEY 環境変数が
設定されていません。")

def load_graph(city_choice="kyoto"):
    if city_choice not in CITY_MAP:
        city_choice = "kyoto"
    config = CITY_MAP[city_choice]
    graph_file_path = config["graph_file"]

    if os.path.exists(graph_file_path):
        print(f"--- [DEBUG] Loading graph from file:
{graph_file_path} ---")
        with open(graph_file_path, "rb") as f:
            graph = pickle.load(f)
    else:
        print(f"--- [DEBUG] '{graph_file_path}' not found.
Downloading map for '{config['query']}'... ---")
        graph = ox.graph_from_place(config["query"],
network_type="drive")
        with open(graph_file_path, "wb") as f:
            pickle.dump(graph, f)
        print("--- [DEBUG] Download complete. ---")
    return graph

def get_api_call_status():
    if not os.path.exists(API_CALL_COUNT_FILE):
        return {"date":
datetime.now(timezone.utc).strftime("%Y-%m-%d"), "count": 0}
    with open(API_CALL_COUNT_FILE, "r") as f:
        data = json.load(f)
    today_str =
datetime.now(timezone.utc).strftime("%Y-%m-%d")
    if data["date"] != today_str:
        data = {"date": today_str, "count": 0}
    return data

def update_api_call_count(current_count):
    today_str =
datetime.now(timezone.utc).strftime("%Y-%m-%d")
    data = {"date": today_str, "count": current_count + 1}
    with open(API_CALL_COUNT_FILE, "w") as f:
        json.dump(data, f)
    return data["count"]

def get_weather_forecast(lat, lon, api_key):
    api_status = get_api_call_status()
    if api_status["count"] >= MAX_API_CALLS_PER_DAY:
        print("API コール数の上限に達しました。")
        return {"error": "API_CALL_LIMIT_EXCEEDED",
"minutely": None}
    url =
f"https://api.openweathermap.org/data/3.0/onecall?lat={lat}&lo
n={lon}&exclude=current,daily,alerts&appid={api_key}&units=
metric"
    print(f"--- [DEBUG] Sending API Request to: {url} ---")
    try:
        response = requests.get(url, timeout=5)
        response.raise_for_status()
        data = response.json()
        update_api_call_count(api_status["count"])
        if 'minutely' not in data:
            data['minutely'] = []
        return data
    except requests.exceptions.RequestException as e:
        print(f"OpenWeatherMap API へのリクエスト中にエラ
ーが発生しました: {e}")
        return {"error": "API_REQUEST_FAILED", "minutely":
None}


def sample_points_by_radius(graph, route, sample_radius_m):
    if not route or len(route) < 2:
        return [], 0

    sampled_points_info = []

    # スタート地点の座標
    start_node = graph.nodes[route[0]]
    last_sample_coords = (start_node['y'], start_node['x']) #
(Lat, Lon)

    current_route_dist_accumulated = 0.0 # 出発地からの「道の
り」の累積

    for i in range(len(route) - 1):
```

```
            u = route[i]
            v = route[i+1]

            # エッジデータの取得
            edge_data = graph.get_edge_data(u, v)
            if edge_data:
                edge_length = edge_data[0].get('length', 0)
            else:
                edge_length = 0


            #チェック距離
            check_step_m = 10.0
            num_steps = int(edge_length // check_step_m) + 1

            u_node = graph.nodes[u]
            v_node = graph.nodes[v]

            for j in range(1, num_steps + 1):
                # エッジ上の位置を比率(0.0〜1.0)で計算
                ratio = min(1.0, (j * check_step_m) / edge_length)
if edge_length > 0 else 1.0
                if j == num_steps: ratio = 1.0 # 最後は確実に終点
まで

                # 線形補間で現在の座標を算出
                curr_lat = u_node['y'] + (v_node['y'] - u_node['y'])
* ratio
                curr_lon = u_node['x'] + (v_node['x'] - u_node['x'])
* ratio
                curr_coords = (curr_lat, curr_lon)

                # 直線距離で判定
                straight_dist = geodesic(last_sample_coords,
curr_coords).meters

                if straight_dist >= sample_radius_m:
                    # 直線距離が 250m を超えたので、サンプリン
グ地点

                    # UI 表示用に「道のり」も計算して保存
                    dist_on_this_edge = edge_length * ratio
                    total_route_dist =
current_route_dist_accumulated + dist_on_this_edge

                    sampled_points_info.append({
                        "lat": curr_lat,
                        "lon": curr_lon,
                        "route_distance_m": total_route_dist, # 表
示は「道のり」
                    })

                    # 基準点を更新
                    last_sample_coords = curr_coords

            # 次のエッジへ進む前に、累積距離を更新
            current_route_dist_accumulated += edge_length

    try:
        total_route_length = nx.shortest_path_length(graph,
route[0], route[-1], weight='length')
    except nx.NetworkXNoPath:
        total_route_length = 0

    return sampled_points_info, total_route_length


@app.route("/", methods=["GET", "POST"])
def index():
    map_html = None
    api_call_message = None
    all_forecast_data = {}
    selected_city = "kyoto"
    if request.method == "GET":
        selected_city = request.args.get("city_selection",
"kyoto")

    current_api_status = get_api_call_status()
    api_call_message = f"本日の OpenWeatherMap API コール
数: {current_api_status['count']} /
{MAX_API_CALLS_PER_DAY}"

    if request.method == "POST":
        selected_city = request.form.get("city_selection",
"kyoto")
        print(f"--- [DEBUG] Selected city: {selected_city} ---")

        geolocator = Nominatim(user_agent="my_application",
timeout=10)
        start_address = request.form["start_address"]
        end_address = request.form["end_address"]

        viewbox = None
        try:
            city_query = CITY_MAP.get(selected_city,
CITY_MAP["kyoto"])["query"]
            gdf = ox.geocode_to_gdf(city_query)
            bounds = gdf.total_bounds

            viewbox = [
                (float(bounds[1]), float(bounds[0])),
                (float(bounds[3]), float(bounds[2]))
            ]

            print(f"--- [DEBUG] Geocoding within viewbox:
{viewbox} ---")
        except Exception as e:
            print(f"--- [WARN] Could not get viewbox for
{city_query}: {e} ---")
            viewbox = None

        try:
            use_bounded = viewbox is not None

            start_location = geolocator.geocode(start_address,
viewbox=viewbox, bounded=use_bounded)
            if not start_location:
                print("--- [DEBUG] Start address not found in
viewbox. Retrying without viewbox... ---")
                start_location =
geolocator.geocode(start_address)

            if not start_location:
                return render_template("index.html",
error_message1="出発地点の住所が見つかりませんでした。",
api_call_message=api_call_message,
all_forecast_data=all_forecast_data, selected_city=selected_city)
            start_lat, start_lon = start_location.latitude,
start_location.longitude
```

```
            end_location = geolocator.geocode(end_address,
viewbox=viewbox, bounded=use_bounded)
            if not end_location:
                print("--- [DEBUG] End address not found in
viewbox. Retrying without viewbox... ---")
                end_location =
geolocator.geocode(end_address)

            if not end_location:
                return render_template("index.html",
error_message2="目的地点の住所が見つかりませんでした。",
api_call_message=api_call_message,
all_forecast_data=all_forecast_data, selected_city=selected_city)
            end_lat, end_lon = end_location.latitude,
end_location.longitude

            print(f"--- [DEBUG] Start Coords: ({start_lat},
{start_lon}) ---")
            print(f"--- [DEBUG] End Coords: ({end_lat},
{end_lon}) ---")

        except Exception as e:
            print(f"--- [ERROR] Caught exception during
geocoding: {e} ---")
            return render_template("index.html",
error_message=f"住所の検索中にエラーが発生しました: {e}",
api_call_message=api_call_message,
all_forecast_data=all_forecast_data, selected_city=selected_city)

        distance = geodesic((start_lat, start_lon), (end_lat,
end_lon)).meters

        if distance > 7000:
            error_message = "距離が 7km を超えています（推
奨は 7km 以内）。長距離だと API コールが多発する場合があり
ます。"
            return render_template("index.html",
error_message=error_message,
api_call_message=api_call_message,
all_forecast_data=all_forecast_data, selected_city=selected_city)

        graph = load_graph(selected_city)

        start_node = ox.distance.nearest_nodes(graph, start_lon,
start_lat)
        end_node = ox.distance.nearest_nodes(graph, end_lon,
end_lat)

        try:
            route = nx.shortest_path(graph, start_node,
end_node, weight='length')
        except nx.NetworkXNoPath:
            return render_template("index.html",
error_message="指定された 2 地点間の経路が見つかりませんで
した。", api_call_message=api_call_message,
all_forecast_data=all_forecast_data, selected_city=selected_city)

        sampled_points_info, total_route_length =
sample_points_by_radius(
            graph, route,
SAMPLE_DISTANCE_THRESHOLD_M
        )

        weather_data_start = get_weather_forecast(start_lat,
start_lon, OPENWEATHER_API_KEY)
        time.sleep(0.1)

        sampled_points_forecasts = []
        for i, point_info in enumerate(sampled_points_info):
            lat = point_info["lat"]
            lon = point_info["lon"]
            route_dist_m = point_info["route_distance_m"]
            weather_data = get_weather_forecast(lat, lon,
OPENWEATHER_API_KEY)
            sampled_points_info[i]["minutely"] =
weather_data.get('minutely', [])
            sampled_points_forecasts.append({
                "distance_m": route_dist_m,
                "minutely": weather_data.get('minutely')
            })
            time.sleep(0.1)

        weather_data_end = get_weather_forecast(end_lat,
end_lon, OPENWEATHER_API_KEY)
        time.sleep(0.1)

        all_forecast_data = {
            "start_location": {"minutely":
weather_data_start.get('minutely')},
            "end_location": {"minutely":
weather_data_end.get('minutely')},
            "sampled_points": sampled_points_forecasts,
            "total_route_length_m": total_route_length
        }

        route_map = folium.Map(
            location=[start_lat, start_lon],
            zoom_start=13,
            tiles="CartoDB positron"
        )

        route_coords = [(graph.nodes[node]['y'],
graph.nodes[node]['x']) for node in route]
        folium.PolyLine(route_coords, color="blue", weight=2.5,
opacity=1).add_to(route_map)

        for point_info in sampled_points_info:
            lat = point_info["lat"]
            lon = point_info["lon"]
            route_dist = point_info["route_distance_m"]
            minutely_data = point_info.get("minutely", [])
            precipitation_now =
minutely_data[0].get("precipitation", 0) if minutely_data else 0
            popup_text = f"経路距離: 約{route_dist:.0f}m<br>
現在の降水量: {precipitation_now:.1f}mm/h"

            # 降水量色分け
            if precipitation_now >= 20.0:
                marker_color = "red"
            elif precipitation_now >= 10.0:
                marker_color = "orange"
            elif precipitation_now > 0:
                marker_color = "blue"
            else:
                marker_color = "green"

            folium.CircleMarker(
                location=(lat, lon),
                radius=5,
                color=marker_color,
```

```
                fill=True,
                fill_color=marker_color,
                popup=popup_text,
                weight=1,
                fill_opacity=0.8,
                opacity=1
            ).add_to(route_map)

        folium.Marker(
            [start_lat, start_lon],
            popup="出発地点",
            icon=folium.Icon(color='red', icon='play')
        ).add_to(route_map)

        folium.Marker(
            [end_lat, end_lon],
            popup="目的地点",
            icon=folium.Icon(color='blue', icon='flag')
        ).add_to(route_map)

        if route_coords:
            route_map.fit_bounds(folium.PolyLine(route_coor
ds).get_bounds())

        map_html_path = "templates/route_map.html"
        route_map.save(map_html_path)
        with open(map_html_path, "r", encoding="utf-8") as f:
            map_html = f.read()

        current_api_status = get_api_call_status()
        api_call_message = f"本日の OpenWeatherMap API コー
ル数: {current_api_status['count']} /
{MAX_API_CALLS_PER_DAY}"

        return render_template("index.html",
                               map_html=map_html,
                               api_call_message=api_call_me
ssage,
                                                   all_forecast_d
ata=all_forecast_data,
                               selected_city=selected_city)

    return render_template("index.html",
                           map_html=map_html,
                           api_call_message=api_call_messag
e,
                           all_forecast_data=all_forecast_data
,
                           selected_city=selected_city)

if __name__ == "__main__":
    app.run(debug=True)
```

index..html

```
<!DOCTYPE html>
<html>
<head>
    <title>経路探索と雨の予報</title>
    <style>
        body { font-family: sans-serif; }
        .map-container {
            width: 100%;
            height: 600px;
            border: 1px solid #ccc;
            margin-top: 20px;
        }
        .weather-info-box {
            background-color: #f0f8ff;
            border: 1px solid #add8e6;
            padding: 15px;
            margin-top: 20px;
        }
        .forecast-content {
            white-space: pre-wrap;
            margin-top: 10px;
            font-size: 0.9em;
            line-height: 1.4;
        }
        .api-status {
            background-color: #e6f7ff;
            border: 1px solid #91d5ff;
            padding: 10px;
            margin-bottom: 15px;
            text-align: center;
        }
        .button-group button {
            margin-right: 10px;
            padding: 8px 15px;
            border: 1px solid #007bff;
            background-color: #007bff;
            color: white;
            border-radius: 4px;
            cursor: pointer;
        }
        .button-group button:hover {
            background-color: #0056b3;
        }
        .controls-container {
            display: flex;
            gap: 20px;
            margin-bottom: 15px;
            align-items: center;
            flex-wrap: wrap;
        }
        .control-group button {
            margin-right: 5px;
            margin-bottom: 5px;
            padding: 8px 12px;
            border: 1px solid #6c757d;
            background-color: #6c757d;
            color: white;
            border-radius: 4px;
            cursor: pointer;
        }
        .control-group button:hover {
            opacity: 0.8;
        }
        .control-group button.active {
            background-color: #007bff;
            border-color: #0056b3;
            font-weight: bold;
        }

        .map-container {
            position: relative; /* 子要素を絶対位置で配置する
ための基準 */
        }

        .map-legend {
```

```
            position: absolute;     /* 親要素(.map-container)を基準に配置 */
            top: 10px;              /* 上から 10px */
            right: 10px;            /* 右から 10px */
            background-color: rgba(255, 255, 255, 0.85); /* 少し透けた白 */
            padding: 10px;
            border: 1px solid #ccc;
            border-radius: 5px;
            z-index: 1000;          /* 地図タイルより手前に表示 */
            font-size: 14px;
        }
        .map-legend h3 {
            margin-top: 0;
            margin-bottom: 5px;
            font-size: 16px;
        }
        .map-legend div {
            margin-bottom: 4px;
        }
        .map-legend .legend-color-dot {
            display: inline-block;
            width: 12px;
            height: 12px;
            border-radius: 50%; /* 円にする */
            margin-right: 8px;
            vertical-align: middle;
        }

    </style>
</head>
<body>
    <h1>出発地点と目的地点を入力してください</h1>

    {% if api_call_message %}
        <div class="api-status">
            <p>{{ api_call_message }}</p>
        </div>
    {% endif %}

    {% if error_message1 %}<p style="color: red;">{{ error_message1 }}</p>{% endif %}
    {% if error_message2 %}<p style="color: red;">{{ error_message2 }}</p>{% endif %}
    {% if error_message %}<p style="color: red;">{{ error_message }}</p>{% endif %}

    <form method="POST" id="routeForm">
        <label>
            都市:
            <select name="city_selection">
                <option value="kyoto" {% if selected_city == 'kyoto' %}selected{% endif %}>京都</option>
                <option value="okinawa" {% if selected_city == 'okinawa' %}selected{% endif %}>沖縄（那覇）</option>
                <option value="niigata" {% if selected_city == 'niigata' %}selected{% endif %}>新潟</option>
            </select>
        </label><br>
        <label>出発地点: <input type="text" name="start_address" id="start_address" required value="{{ request.form.get('start_address', '') }}"></label><br>
        <label>目的地点: <input type="text" name="end_address" id="end_address" required value="{{ request.form.get('end_address', '') }}"></label><br>
        <div class="button-group">
            <button type="submit">経路探索と雨予報</button>
            <button type="button" onclick="swapAddresses()">出発/目的地点を入れ替える</button>
        </div>
    </form>

    {% if all_forecast_data and all_forecast_data.total_route_length_m %}
        <div class="weather-info-box">
            <h2>雨の予報</h2>

            <div class="controls-container">
                <div id="departure-time-controls" class="control-group">
                    <strong>出発時刻:</strong>
                    <button type="button" onclick="updateForecastDisplay(0, this)" class="active">今すぐ</button>
                    <button type="button" onclick="updateForecastDisplay(5, this)">5 分後</button>
                    <button type="button" onclick="updateForecastDisplay(10, this)">10 分後</button>
                    <button type="button" onclick="updateForecastDisplay(15, this)">15 分後</button>
                    <button type="button" onclick="updateForecastDisplay(20, this)">20 分後</button>
                    <button type="button" onclick="updateForecastDisplay(25, this)">25 分後</button>
                    <button type="button" onclick="updateForecastDisplay(30, this)">30 分後</button>
                    <button type="button" onclick="updateForecastDisplay(35, this)">35 分後</button>
                    <button type="button" onclick="updateForecastDisplay(40, this)">40 分後</button>
                    <button type="button" onclick="updateForecastDisplay(45, this)">45 分後</button>
                    <button type="button" onclick="updateForecastDisplay(50, this)">50 分後</button>
                    <button type="button" onclick="updateForecastDisplay(55, this)">55 分後</button>
                </div>
                <div id="travel-mode-controls" class="control-group">
                    <strong>移動手段:</strong>
                    <button type="button" onclick="updateForecastDisplay(null, this)" class="active" data-mode="walk">徒歩</button>
                    <button type="button" onclick="updateForecastDisplay(null, this)" data-mode="bike">自転車</button>
                </div>
            </div>

            <div id="forecast-results-container">
                <h3 id="forecast-title">「今すぐ」「徒歩」で出発した場合の予報</h3>
                <div id="start-point-forecast" class="forecast-content"></div>
```

```
                    <div id="sampled-points-forecast"
class="forecast-content"></div>
                    <div id="destination-forecast" class="forecast-
content"></div>
                </div>
            </div>
        {% endif %}

        {% if map_html %}
            <div class="map-container">
                {% if map_html %}
            <div class="map-container">
                {{ map_html | safe }}

                <div class="map-legend">
                    <h3>凡例: 現在の降水量</h3>
                    <div>
                        <span class="legend-color-dot"
style="background-color: green;"></span>
                        雨なし（0 mm/h）
                    </div>
                    <div>
                        <span class="legend-color-dot"
style="background-color: blue;"></span>
                        小雨（&gt; 0 mm/h）
                    </div>
                    <div>
                        <span class="legend-color-dot"
style="background-color: orange;"></span>
                        やや強い雨（≥ 10.0 mm/h）
                    </div>
                    <div>
                        <span class="legend-color-dot"
style="background-color: red;"></span>
                        強い雨（≥ 20.0 mm/h）
                    </div>
                </div>
                </div>
        {% endif %}
                {{ map_html | safe }}
            </div>
        {% endif %}

        <script>


            // Python から渡された全ての予報・経路データ
            const allForecastData = {{ all_forecast_data | tojson |
safe }};

            // 定数
            const WALK_SPEED_MPM = 80;
            const BIKE_SPEED_MPM = 250;

            // グローバル変数で現在の選択状態を保持
            let currentDepartureOffset = 0;
            let currentTravelMode = 'walk';


             * ページ読み込み完了時に初期表示を実行
             */
            document.addEventListener('DOMContentLoaded', ()
=> {
                // データが存在する場合のみ初期表示を実行
                if (allForecastData &&
allForecastData.total_route_length_m) {
                    updateForecastDisplay();
                }
            });


             * 出発/目的地点を入れ替える関数
             */
            function swapAddresses() {
                const startInput =
document.getElementById('start_address');
                const endInput =
document.getElementById('end_address');
                const temp = startInput.value;
                startInput.value = endInput.value;
                endInput.value = temp;
                document.getElementById('routeForm').submit();
            }

            /**
             * 指定された分数の予報データを取得するヘルパー関
数
             * @param {Array} minutelyData - OpenWeatherMap の
minutely データ配列
             * @param {number} targetMinute - 目的の分数（e.g.,
15)
             * @returns {string} - 降水量テキスト（e.g.,
"0.5mm/h")
             */
            function getPrecipitationAt(minutelyData,
targetMinute) {
                targetMinute = Math.round(targetMinute);
                if (!minutelyData) return "データ取得失敗";
                if (targetMinute >= minutelyData.length) return
"60 分を超えるため予測不可";
                if (targetMinute < 0) return "---";

                const precipitation =
minutelyData[targetMinute]?.precipitation || 0;
                return `${precipitation.toFixed(1)}mm/h`;
            }

            /**
             * 画面上の予報表示を全面的に更新するメイン関数
             * @param {number|null} newOffset - 新しい出発遅延
時間（分）。null の場合は変更しない。
             * @param {HTMLElement|null} clickedButton - クリ
ックされたボタン要素
             */
            function updateForecastDisplay(newOffset = null,
clickedButton = null) {
                if (newOffset !== null) {
                    currentDepartureOffset = newOffset;
                }
                if (clickedButton && clickedButton.dataset.mode) {
                    currentTravelMode =
clickedButton.dataset.mode;
                }

                // --- UI（アクティブボタン）の更新 ---
                document.querySelectorAll('#departure-time-
controls button').forEach(btn => {
                    btn.classList.toggle('active',
parseInt(btn.textContent.match(/¥d+/)?.[0] || '0') ===
```

```
currentDepartureOffset || (btn.textContent === '今すぐ' &&
currentDepartureOffset === 0));
                });
                document.querySelectorAll('#travel-mode-controls
button').forEach(btn => {
                        btn.classList.toggle('active', btn.dataset.mode
=== currentTravelMode);
                });

                // --- 予報計算と表示 ---
                const speed = currentTravelMode === 'walk' ?
WALK_SPEED_MPM : BIKE_SPEED_MPM;
                const travelModeText = currentTravelMode ===
'walk' ? '徒歩' : '自転車';
                const departureTimeText = currentDepartureOffset
=== 0 ? '今すぐ' : `${currentDepartureOffset}分後`;

                document.getElementById('forecast-
title').innerText = `「${departureTimeText}」
「${travelModeText}」で出発した場合の予報`;

                console.clear(); // 毎回コンソールをクリアして見
やすくする
                console.log(`--- [デバッグ] 計算開始 ---`);
                console.log(`出発遅延: ${currentDepartureOffset}
分`);
                console.log(`移動手段: ${travelModeText} (速度:
${speed} m/分)`);
                console.log(`--------------------------`);

                // 1. 出発地点の予報
                const startForecastText = `出発地点
(${departureTimeText}):
${getPrecipitationAt(allForecastData.start_location.minutely,
currentDepartureOffset)}`;
                document.getElementById('start-point-
forecast').innerText = startForecastText;

                // 2. 各サンプル地点の予報
                let sampledPointsHtml = "サンプル地点ごとの予
報:¥n";
                allForecastData.sampled_points.forEach((point,
index) => {
                        const timeToReach = point.distance_m / speed;

</html>
```

```
                        const arrivalTime = currentDepartureOffset +
timeToReach;
                        const precip =
getPrecipitationAt(point.minutely, arrivalTime);

                        console.log(`▼ 地点 ${index + 1}`);
                        console.log(`  距離:
${Math.round(point.distance_m)} m`);
                        console.log(`  移動時間:
${timeToReach.toFixed(1)} 分`);
                        console.log(`  到着時刻（出発から):
${arrivalTime.toFixed(1)} 分後`);
                        console.log(`  -> 降水量: ${precip}`);
                        sampledPointsHtml += `地点 ${index + 1} (約
${Math.round(point.distance_m)}m,
${Math.round(timeToReach)}分後到着): ${precip}¥n`;
                });
                document.getElementById('sampled-points-
forecast').innerText = sampledPointsHtml;

                // 3. 目的地点の予報
                const totalTimeToReach =
allForecastData.total_route_length_m / speed;
                const finalArrivalTime = currentDepartureOffset +
totalTimeToReach;
                const finalPrecip =
getPrecipitationAt(allForecastData.end_location.minutely,
finalArrivalTime);

                console.log(`▼ 目的地点`);
                console.log(`  総距離:
${Math.round(allForecastData.total_route_length_m)} m`);
                console.log(`  総移動時間:
${totalTimeToReach.toFixed(1)} 分`);
                console.log(`  最終到着時刻（出発から):
${finalArrivalTime.toFixed(1)} 分後`);
                console.log(`  -> 降水量: ${finalPrecip}`);
                console.log(`--------------------------`);
                const destinationForecastText = `目的地到着時の予
報（約${Math.round(totalTimeToReach)}分後): ${finalPrecip}`;
                document.getElementById('destination-
forecast').innerText = destinationForecastText;
        }
    </script>
</body>
```

t 検定アンケート

移動中の気象情報と意思決定に関するアンケート（移動経路と降水予報を統合した動的降水確率予測手法）

本アンケートは、卒業研究の一環として、気象情報の提示方法がユーザーの行動判断に与える影響を調査するために実施するものです。ご多忙の折とは存じますが、ご協力いただけますと幸いです。

【アンケートの内容】 これから、いくつかの「シチュエーション（移動経路）」と、その時の「気象情報の画面」を画像で提示します。

※青🏴：目的地、赤▷：現在地

【お願いしたいこと】

1. 提示された画像を見て、「もし自分なら、今すぐ出発するか？ それとも待機するか？」といった判断をシミュレーションしてください。
2. その判断を行った際に感じた「分かりやすさ」や「判断にかかった時間（感覚）」について、直感でお答えください。

名前

短文回答

①浦添城跡からサンエー浦添西海岸パルコシティ　(徒歩25分)

※青🏴：目的地、赤▷：現在地

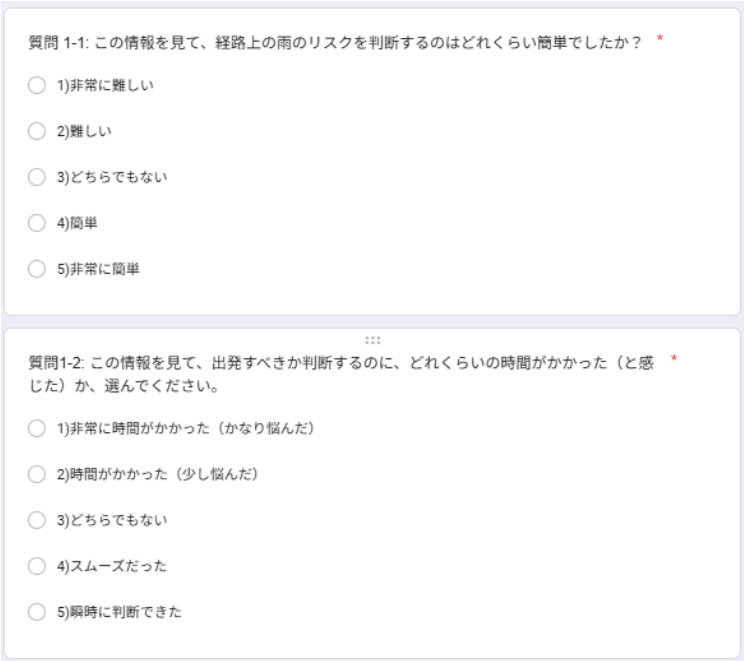
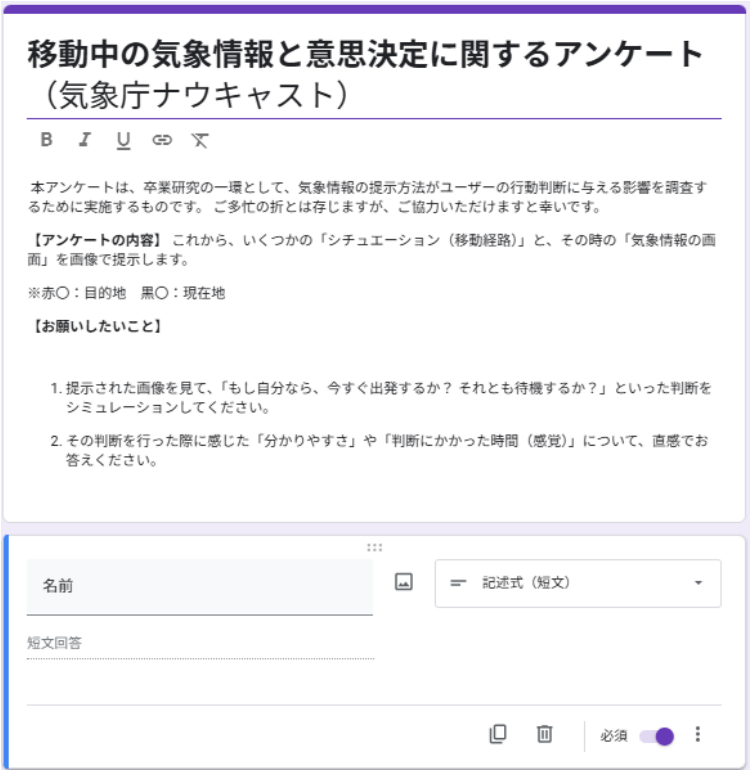


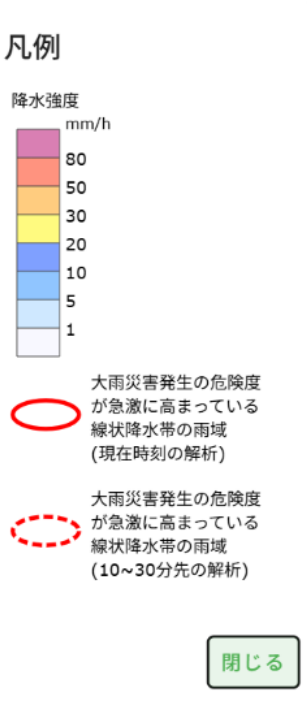
図 9 移動中の気象情報と意思決定に関するアンケート　（移動経路と降水予報を統合した動的降水確率予測手法）

気象庁ホームページより作成

※シナリオ 2,3 についても、提示する画像を変更し、同様の質問項目で実施した。



①浦添城跡からサンエー浦添西海岸パルコシティ （自転車25分）
※赤〇：目的地　黒〇：現在地



閉じる

現時刻


気象庁
雨雲の動き
雨雲の動き　2025年11月17日14時40分
スカイレンタカー
公式サイト初の
限定クーポン

5分後


気象庁
雨雲の動き
雨雲の動き　2025年11月17日14時45分
スカイレンタカー
公式サイト初の
限定クーポン

ｚ

q

図 10 移動中の気象情報と意思決定に関するアンケート　（気象庁ナウキャスト）

気象庁ホームページより作成

※シナリオ 2,3 についても、提示する画像を変更し、同様の質問項目で実施した。