令和７年度　特別研究報告書

外干し環境における洗濯物乾燥時間の推定精度向上に関する研究

龍谷大学　先端理工学部　知能情報メディア課程

Y220159　中川　大空

指導教員　三好　力　教授

研究梗概

本研究は、外干し環境における洗濯物の乾燥時間をより高精度に推定するためのシステムを構築することを目的とし、OpenWeatherMapAPIから取得した気温、湿度、風速、風向のリアルタイム気象データを用い、蒸発モデルに基づく乾燥時間推定式を構築した。特にベランダの方位と風向の関係に基づき実効風速を算出する補正を導入し、既存サービスでは考慮されていない微気象的条件をモデルに組み込んだ。当システムと既存システムの予測値を実測乾燥時間と比較し本研究の有用性を証明するための実験を行った。

目次

第１章　はじめに

1.1 研究背景

近年、共働き世帯の増加や生活スタイルの多様化により、家事の効率化が求められている。洗濯においても、乾燥機付き洗濯機やコインランドリーなどの利用が普及している一方で、電気代の高騰や衣類の痛み、自然乾燥による仕上がりの良さなどの理由から、外干しを選択する家庭も依然として多い。

また、テレビや天気予報アプリなどでは「洗濯指数」「部屋干し推奨」といった簡易的な洗濯物情報が日常的に提供されており、外干しに対する需要が一定以上存在することを示している。

しかし、これらの情報は気温や湿度などの一般的な条件に基づくものであり、地域の微気象やベランダの向き、風の影響などの局所的条件を十分に反映していない。そのため、実際の乾燥時間との誤差が生じることが課題となっている。

したがって、実際の気象条件や環境要素をより細かく反映した外干し乾燥時間の推定システムを開発することは、日常生活における利便性の向上に加え、エネルギーの有効活用や持続可能な生活にも寄与すると考えられる。

図 1　洗濯指数　（日本気象協会より）

1.2 研究目的

洗濯物の乾燥時間は気温、湿度、風速などの気象要因に大きく左右される。従来の乾燥時間推定システムでは、地域ごとの気象データの解像度や風向・日照条件の反映が不十分であり、実際の環境との誤差が生じている。

本研究では現在気象 API として広く利用されている OpenWheatherMap API から取得した公開情報データとベランダ環境情報を組み合わせ、より現実的な外干し乾燥時間を求めるシステムの改善を目的とする。

第 2 章　既存技術

2.1　ウェザーニュース

図 2　「お洗濯物情報」　（ウェザーニュースより）

洗濯物の乾燥時間予測に関する既存技術として、ウェザーニューズ社が提供する気象情報サービス「ウェザーニュース」が挙げられる。同サービスでは、気温、湿度、降水量、風速などの基本的な気象データに加え、お洗濯情報という形で洗濯物の乾燥時間を提供している。

しかしながら、ウェザーニュースの洗濯情報は、ユーザの設置環境（ベランダの方向、日照の有無など）を反映した細粒度の予測を行うものではない。そのため、同じ地域においても実際の乾燥環境との差が生じる可能性がある。

以上より、既存技術では気象条件に基づいた一般的な指標提示が中心であり、個別環境に適応した乾燥時間予測の高精度化には依然として改善の余地がある。

2.2　はれほす

図 3　はれほす

同様の洗濯物乾燥支援サービスとして「はれほす」というスマートフォンアプリが存在する。本アプリは、気象データと過去の乾燥実績をもとに洗濯物の乾きやすさをスコアとして提示し、乾燥のしやすさを視覚的に把握できるインタフェースを備え、日照や気温の推移といった情報を直感的に理解できる点が特徴である。

しかしながら、本サービスでは風速や風向といった乾燥に大きな影響を及ぼす気象パラメータの反映が十分でなく、個別環境条件を適切に反映した高精度な乾燥予測には至っていない。

さらに、本アプリは洗濯物を干す開始時間を午前８時に固定しており、この時間に基づいた予測情報のみを提供している。このため、実際に干し始める時間が早朝や夜間など午前８時から大きくずれる場合、提示されている結果が実環境の乾燥状況を適切に反映できていない可能性が高い。

以上より本アプリは洗濯物乾燥に関する意思決定を支援するうえで有用な情報を提供し

ているものの、乾燥時間の数値化、風向・風速を含む詳細なパラメータの活用、干し始め時刻の柔軟入力といった面において課題が残る。これらは、ユーザの具体的な行動判断をより精密に支援するために改善が期待される点である。

第３章　提案手法

3.1　概要

本研究では、OpenWheatherMap API から取得した気象データを用いて、蒸発速度式に基づく洗濯物の乾燥時間推定システムを構築する。

また、既存技術では考慮されていないベランダの方位及び風向との関係を反映できるように蒸発係数の補正を行い、外干し環境における実際の環境条件に近づけるよう改良を行った。

提案システムの有効性を確認するために、実測データとの比較及び既存システムとの誤差分析を行った

3.2　OpenWheatherMap API

OpenWheatherMap API は、世界各地の気象情報を提供する Web API サービスである。緯度・経度を指定することで、地域ごとの気温、湿度、風速、風向、雲量などを JSON 形式で取得することができる。

本研究では、乾燥時間推定に必要な以下の４つのパラメータを利用した

取得項目	意味	単位	利用目的
気温(temp)	大気の温度	℃	飽和水蒸気圧算出
湿度(humidity)	相対湿度	%	大気中水蒸気圧算出
風速(wind speed)	地上風の速さ	m/s	蒸発係数補正
風向(wind deg)	風の向き	°	ベランダ方位との比較

3.3　乾燥時間推定の基本アプローチ

洗濯物の乾燥は、含水量の減少とともに進行する。本研究では、乾燥に必要な総水分量と、時間当たりの蒸発量の比率によって乾燥時間を求める近似モデルを使用する。

乾燥時間 t =　初期水分量/(蒸発量/時間)

3.4　乾燥時間算出式

洗濯物からの水分蒸発は、次式であらわされる。

$$E = k * (Yw - Ya) \quad (1)$$

ここで、

・E:単位時間当たりの蒸発量(kg/m²*s)
・k:蒸発係数(風速・風向に依存)
・Yw:衣類表面の飽和水蒸気圧(気温による)
・Ya:大気中の水蒸気圧(相対湿度による)

式(1)における蒸発係数 k は、衣類表面における水蒸気の移動抵抗を表し、主に周囲の風速の影響を受ける。一般に風速が大きいほど衣類表面の境界層が薄くなり、水蒸気の拡散が促進されるため、蒸発量は増加する。簡易的には次式のように風速 v を用いて表すことができる。

$$k = k_0(1 + av) \quad (2)$$

ここで、k_0は基準蒸発係数、a は風速補正係数である。

外干し環境においては、風速だけでなく、風向とベランダ方位の関係が洗濯物表面に到達する風の強さに大きな影響を与える。ベランダが建物に囲まれている場合、風が正面から流入する場合と側面あるいは背面から流れる場合とでは、衣類表面に形成される境界層の状態が異なり、蒸発効率も変化する。

そこで本研究では、ベランダ方位角を θb、気象データから得られる風向角を θw とし、両者の差 $\Delta\theta$ を次式で定義する。

$$\Delta\theta = |\theta w - \theta b| \quad (3)$$

この角度差を用いて、洗濯物に実際に寄与する実効風速 veff を次式であらわす。

$$veff = v * \cos(\Delta\theta) \quad (4)$$

これにより、風が真正面からあたる場合は最大効果を発揮し、横風の場合は影響が半減し、背面方向ではほぼ効果がない状態を表現することができる。

次に、衣類表面の飽和水蒸気圧 Yw は、気温 T に対してテトン式により次式で近似される。

$$Yw = 610.78\exp(17.27T/(T + 237.3)) \quad (5)$$

また、大気中の市場気圧 Ya は相対湿度 RH を用いて次式で与えられる。

$$Ya = RH/100 * Yw \qquad (6)$$

さらに、衣類表面に含まれる水分量を W、衣類表面積を A とすると、乾燥時間 T は蒸発量との関係系から次式で求められる。

$$T = W/EA \qquad (7)$$

3.4.1 時間変化を考慮した蒸発モデル

前節で考慮した実効風速 veff を用いることで、風速及び風向が蒸発に与える影響を考慮できる。しかし、式(7)は気象条件が時間的に一定であることを仮定しており、実際の外干し環境における乾燥過程を十分に表現できない。

そこで本研究では、蒸発量を時間の関数として扱い、乾燥家庭を数値積分により評価するモデルへ拡張する。

時間 t における単位面積当たりの蒸発速度 E(t)は次式であらわされる。

$$E(t) = k_0(1 + aveff(t))(Yw(t) - Ya(t)) \qquad (8)$$

3.4.2 乾燥時間の数値的算出

衣類表面に含まれる水分量を W(t)、衣類表面積を A とすると、微小時間Δt における水分量の変化は次式で与えられる。

$$W(t + \Delta t) = W(t) - E(t)A\Delta t \qquad (9)$$

本研究では、気象予報データが 3 時間間隔で提供されることを考慮し、Δt = 3 時間として離散的な時間積分を行った。初期含水量 W(0)が 0 以下となる時刻を乾燥完了時刻と定義し、その時刻までの経過時間を推定乾燥時間とした。

3.5　システムの概要

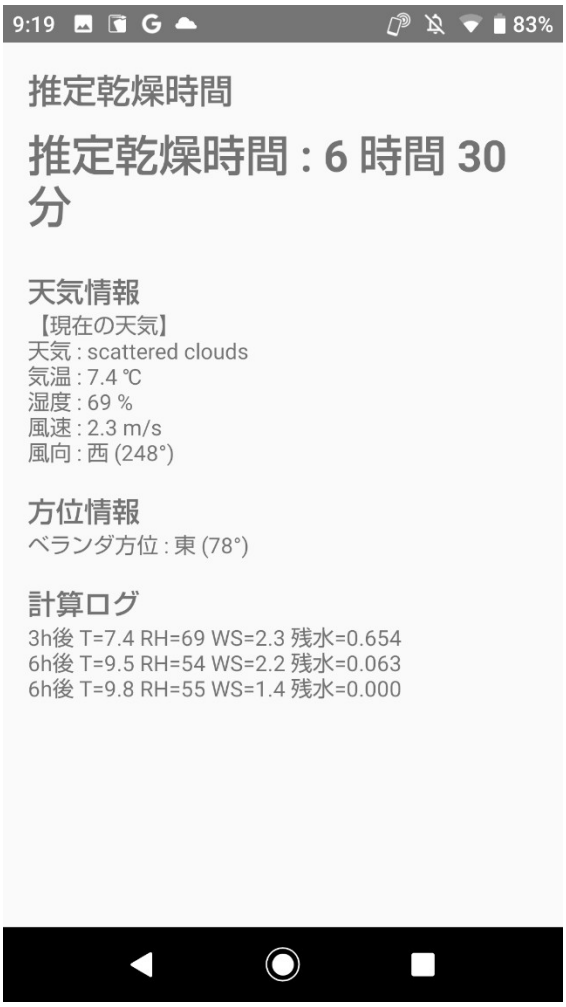

図 4　測定結果表示画面

本研究で提案する乾燥時間予測システムは、蒸発量モデルに基づいて洗濯物の乾燥に要する時間を推定するものであり、Java を用いて実装し、Android Studio を用いて Android アプリとして開発した。

本システムは、気象データ取得部、乾燥時間算出部、結果提示部の 3 つの機能から構成される。

気象データ取得部では、OpenWeatherMap API を利用し、端末の位置情報から現在地を特定したうえで、現在時刻及び将来の気象予報データ(気温、湿度、風速、風向)を取得する。これにより、単一時点の気象条件だけでなく、時間経過に伴う気象変化を考慮した乾燥予測を可能としている。

乾燥時間算出部では、第 3 章で示した蒸発量モデルを用いて、取得した気象データを入力して蒸発速度を時間ごとに算出する。さらに、洗濯物に含まれる水分量の時間変化を数値積分により評価し、水分量が 0 に到達するまでの経過時間を乾燥時間として推定する。

結果提示部では、推定された乾燥時間を「○時間○分」という直感的な形式で表示するとともに、計算過程に用いられた気象条件や風向補正の影響をテキストとして提示する。これにより、ユーザは単に数値結果を得るだけでなく、「なぜその乾燥時間になったのか」を理解したうえで洗濯物の干し方や取り込み時間を判断できる。

また、本システムはスマートフォンに内蔵された方位センサを利用してベランダの方位を取得し、風向との相対関係を考慮する点に特徴がある。これにより、設置型センサを必要とせず、ユーザの負担を増やすことなく実用的な乾燥時間予測を実現している。

第４章　実験

4.1　実験目的

本研究で提案する洗濯物乾燥時間予測システムの有用性を検証するために、既存の乾燥時間情報提供サービスによる予測結果と比較を行い、実際の乾燥時間に対する誤差を評価することを目的とする。特に、本研究の特徴であるベランダの方位と風向の関係性と現在地に置けるより詳細な気象情報の取得に基づく乾燥時間の数値予測が、既存サービスよりも精度向上に寄与することを明らかにする。

4.2　実験方法

4.2.1　実験環境

・実験場所：ベランダ(東向き)
・使用物品：フェイスタオル(パイル生地)
・測定装置：Android スマートフォン
　・本研究で開発したアプリを搭載
　・Java で実装し、Android Studio により開発
・取得データ：
　・現在地の気温・湿度・風速・風向(OpenWeatherMap API)
　・ベランダ方位角(スマートフォン内蔵センサ)
・記録方法：
　・アプリに表示される推定乾燥時間及び計算ログ
　・スクリーンショット及びメモアプリによる時刻記録

4.2.2　実験手順

1. 試料準備
　市販のフェイスタオル（パイル生地、綿 100%）を家庭用洗濯機で通常洗濯した後、標準コースによる脱水を行い、実験用試料とした。洗濯条件（洗剤量、水量、脱水時間）はすべて同一とした。
2. 干しはじめ条件の設定
　実験は晴天または曇天の日中に実施し、あらかじめ定めた干し始め時刻（午前 8 時 00 分～午前 9 時 00 分）に、実験場所であるベランダにフェイスタオルを 1 枚のみ設置した。タオルは洗濯ばさみを用いて縦向きに吊り下げ、風通しを妨げない状態とした。設置完了時刻を乾燥開始時刻として記録した。
3. 乾燥時間予測の取得

タオル設置直後に、本研究で提案する乾燥時間予測アプリをスマートフォン上で起動し、現在地の気象情報および推定乾燥時間を取得した。同時に、比較対象とする他の乾燥時間予測手法についても同様に予測時間を取得し、それぞれの画面をスクリーンショットとして保存した。

4. 乾燥状態の確認方法

乾燥開始後は、30 分間隔でフェイスタオルを手で触れ、表面および内部の手触りを確認した。特に、冷たさが感じられなくなり、手で軽く握った際に湿り気が感じられない状態を乾燥判定の基準とした。

なお、各予測システムにおいて示された乾燥時間のうち、最も短い予測時間の 30 分前からは、確認間隔を 10 分ごとに短縮した。

5. 実測乾燥時間の決定

フェイスタオルが上記の乾燥判定基準を満たした時刻を乾燥完了時刻として記録し、乾燥開始時刻との差を実測乾燥時間と定義した。

6. 予測誤差の算出

以下の式で各予測値との誤差を算出する。

$$\text{誤差} = |\text{実測乾燥時間} - \text{予測乾燥時間}|$$

7. 再試行と精度評価

以上の手順を異なる日・異なる気象条件下で複数回実施し、各予測手法について平均誤差を算出することで、乾燥時間予測精度の比較評価を行った。

4.3 実験結果

以下の表が実測時間と提案システム、スマートニュースの予測時間の比較表となる実測乾燥時間と提案システムの誤差を誤差 A、実測乾燥時間とスマートニュースの誤差を誤差 B とする。

表 1 を見ると、晴天条件においては、提案システムの誤差 A は概ね 30 分以内に収まる場合が多く、実測乾燥時間に比較的近い値を示していることが分かる。一方、スマートニュースによる予測では、誤差 B が 1 時間以上となる日も複数確認され、実測値との差が大きくなる傾向が見られた。また、曇天条件においては、全体的に乾燥時間が長くなる傾向が確認され、提案システムおよびスマートニュースのいずれにおいても誤差が増加する傾向が見られた。しかしながら、提案システムの誤差 A は、誤差 B と比較して小さい場合が多く、特に 12 月 3 日および 12 月 7 日以降の曇天条件においても、実測乾燥時間との差が相対的に抑えられていることが確認できる。これらの結果から、提案システムは、気象条件に加えて風向とベランダ方位を考慮することで、既存の乾燥情報サービスと比較して、実際の乾燥状況に近い乾燥時間を推定できていると考えられる。

表 1　各システムにおける予測精度評価表

日付	天候	干しはじめ	実測乾燥時間	提案システム	スマートニュース	誤差 A	誤差 B
11/20	晴れ	8:00	6h10m	6h40m	5h50m	30m	20m
11/21	晴れ	8:00	6h50m	6h40m	5h20m	10m	1h30m
11/23	晴れ	8:10	6h40m	6h10m	5h20m	30m	1h20m
11/24	曇り	8:00	7h10m	6h20m	5h40m	50m	1h30m
11/29	晴れ	8:00	3h50m	4h10m	4h40m	20m	50m
11/30	晴れ	8:40	5h00m	5h20m	5h20m	20m	20m
12/1	晴れ	8:10	5h30m	5h10m	6h20m	20m	50m
12/3	曇り	8:00	8h00m	7h00m	6h00m	1h00m	2h00m
12/7	曇り	9:10	6h50m	6h20m	5h40m	30m	50m
12/8	曇り	8:00	6h20m	7h00m	6h10m	40m	10m
12/9	曇り	8:00	8h10m	7h40m	6h40m	30m	1h30m

第５章　考察

5.1　提案システムの有効性に関する考察

本研究では、OpenWeatherMap API から取得したデータとベランダ方位及び風向の関係性を考慮した蒸発モデルに基づき、外干し環境における乾燥時間推定システムを構築した。

表 1 に示す実験結果より、提案システムの予測値は実測乾燥時間との誤差がおおむね 10~40 分程度に収まる結果が多く確認された。

この結果から、現在地の気象データを用い、時間変化を考慮した数値積分モデルを適用することで、外干し乾燥時間を実用的な精度で推定できると言える。

特に、晴天日における実験結果(11/20、11/21、11/29 など)では、提案システムの誤差 A が既存サービス(スマートニュース)の誤差 B より小さい傾向が確認された。

このことから、風向とベランダ方位の関係を考慮した実効風速 veff による補正が、蒸発量の見積もり精度向上に寄与していると言える。

5.2　風向・ベランダ方位補正の効果

本研究の特徴である風向とベランダ方位の関係を考慮した補正は、式(4)で定義した実効風速 veff を通じて蒸発係数に反映されている。

表 1 において、晴天かつ風の影響を受けやすい条件下では、提案システムの誤差が比較的小さくなる傾向が見られた。

これらの結果から、風が洗濯物に正面から当たる場合とそうでない場合の差をモデルに組み込むことにより、従来の一律的な乾燥指標よりも実環境に近い予測が可能になったと言える。

一方で、曇天時（12/3、12/9 など）には、提案システムにおいても誤差が 1 時間程度生じるケースが確認された。

このことから、風向補正のみでは説明できない要因が乾燥速度に影響していることが示唆される。

5.3　誤差要因に関する考察

提案システムにおいて誤差が生じた主な要因として、蒸発係数 k の算出において基準係数 k_0および補正係数 a を一定値とした簡易モデルを採用した点が挙げられる。

この仮定より、衣類の形状や揺れ、ベランダ周辺で発生する乱流の影響を十分に反映できていないと言える。

また、本研究では乾燥判定を人の感触によって行っているため、完全乾燥と判断するタイミングに個人差や時間分解能の限界が含まれている。

このことから、実測乾燥時間そのものにもある程度の不確かさが含まれている可能性があると言える。

5.4　日射影響とモデルの限界

本研究では、気温・湿度・風速・風向を用いた蒸発モデルを構築したが、日射量を直接モデルに取り込んでいない。

特に晴天時の外干しにおいては、直達日射による衣類表面温度の上昇が蒸発速度を大きく左右する。

曇天時において誤差が比較的大きくなる傾向が見られたことから、日射の有無や強度を考慮しないモデルには限界があると言える。

以上の考察を総合すると、本研究で提案した乾燥時間推定システムは、気温・湿度・風速・風向といった取得容易な気象データに加え、ベランダ方位というユーザ固有の環境条件を考慮することで、外干し環境における乾燥時間を実用的な精度で推定できる可能性を示したと言える。

特に、風向とベランダ方位の関係を反映した実効風速 veff による補正は、晴天時や風の影響を受けやすい条件下において有効に機能し、従来の一律的な乾燥指標では捉えきれない環境差を表現できた点は、本研究の有効性を示す重要な成果である。

一方で、日射量を直接考慮していないことや、蒸発係数を簡易的に一定値としている点、乾燥判定に主観的要素が含まれる点など、本モデルにはいくつかの制約も存在する。これらの制約により、特に曇天時や気象条件が短時間で変化する状況においては、推定誤差が大きくなる傾向が確認された。

以上より、提案システムは外干し乾燥時間推定において一定の有効性を有する一方で、モデルの高度化および入力データの拡充によって、さらなる精度向上の余地があることが明らかとなった。

第 6 章　まとめ

本研究では、OpenWeatherMap API から取得した気象データと、ベランダ方位および風向の関係を考慮した蒸発モデルに基づき、外干し環境における乾燥時間推定システムを提案・実装した。

提案手法では、設置型センサを用いず、一般的に取得可能な気象情報とスマートフォン等で把握できる環境条件を用いることで、ユーザ固有の干し環境を反映した乾燥時間推定を実現している点に特徴がある。

実験結果から、提案システムは既存サービスが提供する一律的な乾燥指標と比較して、実測値に近い予測を示す場合が多く、特に晴天時において有効性が確認された。

今後の課題として、乾燥判定方法の定量化が挙げられる。本研究では人の感触による乾燥判定を用いたが、衣類重量の変化を測定することで、より客観的で再現性の高い評価が可能になると考えられる。

また、日射量や衣類表面温度を取得し、蒸発モデルに直接組み込むことで、特に晴天時および曇天時における誤差の低減が期待できる。さらに、衣類の種類や材質ごとに異なるパラメータを導入することで、より実環境に即したモデルへの拡張も可能である。

加えて、本研究では OpenWeatherMap API の無料プランを利用したため、取得可能な気象予報データは 3 時間間隔に限定されていた。この制約により、数値積分では $\Delta t = 3$ 時間として乾燥過程を評価しており、短時間で変化する気象条件を十分に反映できていない可能性がある。

一方で、同 API の有料プランを利用することで、1 時間間隔の気象予報データを取得することが可能である。時間分解能を 3 時間から 1 時間へ向上させることで、蒸発速度 $E(t)$ の時間変化をより詳細に評価でき、乾燥時間推定精度のさらなる向上が期待される。

謝辞

本研究を行うにあたりご指導ご鞭撻をいただいた三好教授に心から感謝いたします。また、研究に対する助言や議論をしていただいた三好研究所の皆様に心より感謝いたします。

参考文献

[1] “Weather API”
https://openweathermap.org/api?utm_source=chatgpt.com
[2] “One Call API 3.0”
https://openweathermap.org/api/one-call-3?utm_source=chatgpt.com
[3] 1 km メッシュの高解像度で「洗濯指数」を API 提供
https://jp.weathernews.com/news/39951?utm_source=chatgpt.com
[4] 洗濯物の乾きやすさを予測！地球にやさしい洗濯アプリ「はれほす」
https://linkwith-sdgs.com/sdgsaction/3695-2/?utm_source=chatgpt.com
[5]日本気象協会
https://tenki.jp/indexes/cloth_dried/

付録

```
package com.example.weatherapp;

import android.Manifest;
import android.annotation.SuppressLint;
import android.content.pm.PackageManager;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.location.Location;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;

import com.google.android.gms.location.*;

import org.json.JSONArray;
import org.json.JSONObject;

import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;

public class MainActivity extends AppCompatActivity {

    private static final int LOCATION_PERMISSION_REQUEST = 100;
    private static final String WEATHER_API_KEY = "自分の API キーを入力";

    private TextView textDryingTime, textWeather, textDirection, textLog;

    private SensorManager sensorManager;
    private float[] accelValues, magnetValues;
    private int balconyDeg = 0;

    private FusedLocationProviderClient fusedLocationClient;
    private LocationCallback locationCallback;

    private final OkHttpClient httpClient = new OkHttpClient();

    // 正規化初期水分量
    private static final double INITIAL_WATER = 1.0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        textDryingTime = findViewById(R.id.textDryingTime);
        textWeather = findViewById(R.id.textWeather);
        textDirection = findViewById(R.id.textDirection);
        textLog = findViewById(R.id.textLog);

        fusedLocationClient =
LocationServices.getFusedLocationProviderClient(this);

        sensorManager = (SensorManager)
getSystemService(SENSOR_SERVICE);
        sensorManager.registerListener(sensorListener,

sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
                SensorManager.SENSOR_DELAY_UI);
        sensorManager.registerListener(sensorListener,

sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
                SensorManager.SENSOR_DELAY_UI);

        requestLocationPermission();
    }

    // ================= 方位センサ =================
    private final SensorEventListener sensorListener = new
SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent event) {
            if (event.sensor.getType() ==
Sensor.TYPE_ACCELEROMETER)
                accelValues = event.values.clone();
            if (event.sensor.getType() ==
Sensor.TYPE_MAGNETIC_FIELD)
                magnetValues = event.values.clone();

            if (accelValues != null && magnetValues != null) {
                float[] R = new float[9];
                float[] I = new float[9];
                if (SensorManager.getRotationMatrix(R, I, accelValues,
magnetValues)) {
                    float[] o = new float[3];
                    SensorManager.getOrientation(R, o);
                    float az = (float) Math.toDegrees(o[0]);
                    if (az < 0) az += 360;
                    balconyDeg = Math.round(az);
                    textDirection.setText("ベランダ方位 : "
                            + degToDirection(balconyDeg) + " (" +
balconyDeg + "°)");
                }
            }
        }

        @Override public void onAccuracyChanged(Sensor sensor, int
accuracy) {}
    };

    // ================= 位置情報 =================
    private void requestLocationPermission() {
        if (ActivityCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION)
                != PackageManager.PERMISSION_GRANTED) {

            ActivityCompat.requestPermissions(this,
                    new
String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                    LOCATION_PERMISSION_REQUEST);
        } else {
            getCurrentLocation();
        }
    }

    @SuppressLint("MissingPermission")
    private void getCurrentLocation() {
        LocationRequest req = LocationRequest.create()
                .setPriority(LocationRequest.PRIORITY_HIGH_ACCUR
ACY)
                .setInterval(5000);

        locationCallback = new LocationCallback() {
            @Override
            public void onLocationResult(LocationResult result) {
                Location loc = result.getLastLocation();
                if (loc != null) {
                    fetchForecast(loc.getLatitude(), loc.getLongitude());

fusedLocationClient.removeLocationUpdates(locationCallback);
                }
            }
        };
        fusedLocationClient.requestLocationUpdates(req,
locationCallback, null);
    }

    // ================= 天気取得 & 10 分刻み乾燥積分
=================
    private void fetchForecast(double lat, double lon) {
        new Thread(() -> {
            try {
                String url = String.format(

"https://api.openweathermap.org/data/2.5/forecast" +
"?lat=%f&lon=%f&appid=%s&units=metric",
                        lat, lon, WEATHER_API_KEY);

                Response res = httpClient.newCall(
                        new
Request.Builder().url(url).build()).execute();

                if (!res.isSuccessful() || res.body() == null)
                    throw new Exception("HTTP " + res.code());

                JSONArray list = new JSONObject(res.body().string())
                        .getJSONArray("list");

                /* ===== 現在の天気（list[0]） ===== */
                JSONObject now = list.getJSONObject(0);
                JSONObject nm = now.getJSONObject("main");
                JSONObject nw = now.getJSONObject("wind");
                JSONObject ndesc =
now.getJSONArray("weather").getJSONObject(0);

                final String nowWeatherText = String.format(
                        "【現在の天気】¥n" +
                                "天気 : %s¥n" +
                                "気温 : %.1f °C¥n" +
                                "湿度 : %d %%¥n" +
                                "風速 : %.1f m/s¥n" +
                                "風向 : %s (%d°)",
                        ndesc.getString("description"),
                        nm.getDouble("temp"),
                        nm.getInt("humidity"),
                        nw.getDouble("speed"),
                        degToDirection(nw.optInt("deg", 0)),
                        nw.optInt("deg", 0)
                );

                /* ===== 乾燥時間計算 ===== */
                double water = INITIAL_WATER;
                double elapsedSec = 0;
                StringBuilder log = new StringBuilder();

                for (int i = 0; i < list.length() && water > 0; i++) {
                    JSONObject o = list.getJSONObject(i);
                    JSONObject m = o.getJSONObject("main");
                    JSONObject w = o.getJSONObject("wind");

                    double temp = m.getDouble("temp");
                    double rh = m.getDouble("humidity");
                    double ws = w.getDouble("speed");
                    int wd = w.optInt("deg", 0);

                    double E = calcEvaporation(temp, rh, ws, wd);

                    for (int j = 0; j < 18 && water > 0; j++) { // 10 分刻
み
                        water -= E * 600;
                        elapsedSec += 600;
                    }

                    log.append(String.format(
```

```
                                        "%dh 後  T=%.1f RH=%.0f WS=%.1f  残水
=%.3f¥n",
                                        (int)(elapsedSec/3600), temp, rh, ws,
                                        Math.max(water, 0)));
                        }

                        int h = (int)(elapsedSec / 3600);
                        int m10 = (int)((elapsedSec % 3600) / 600) * 10;

                        runOnUiThread(() -> {
                            textDryingTime.setText(
                                    "推定乾燥時間 : " + h + "  時間  " + m10 + "
分");
                            textWeather.setText(nowWeatherText);
                            textLog.setText(log.toString());
                        });

                    } catch (Exception e) {
                        Log.e("FORECAST", "error", e);
                        runOnUiThread(() ->
                                textLog.setText("エラー¥n" + e.getMessage()));
                    }
                }).start();
        }

        // ================  蒸発モデル  ================
        private double calcEvaporation(double T, double RH, double ws, int
wd) {
            double es = 611 * Math.exp((17.27 * T) / (T + 237.3));
            double ea = es * RH / 100.0;

            double theta = Math.abs(wd - balconyDeg);
            if (theta > 180) theta = 360 - theta;

            double dir = Math.max(Math.cos(Math.toRadians(theta)), 0.2);
            double veff = Math.min(ws * Math.pow(dir, 0.7), 3.0);

            double k0 = 0.00008;
            return k0 * (es - ea) / 1000.0 * (1 + 0.3 * Math.sqrt(veff));
        }

        private String degToDirection(int deg) {
            String[] d = {"北","北東","東","南東","南","南西","西","北西"};
            return d[(int)Math.round(deg/45.0)%8];
        }

        @Override
        protected void onDestroy() {
            super.onDestroy();
            sensorManager.unregisterListener(sensorListener);
        }

        @Override
        public void onRequestPermissionsResult(
                int requestCode,
                @NonNull String[] permissions,
                @NonNull int[] grantResults) {

            super.onRequestPermissionsResult(
                    requestCode, permissions, grantResults);

            if (requestCode == LOCATION_PERMISSION_REQUEST &&
                    grantResults.length > 0 &&
                    grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                getCurrentLocation();
            }
        }
}
```