AI Powered Weather Insurance System on a Blockchain

1st Ben Taylor
School of Computing
Newcastle University
Newcastle, UK
b.taylor9@newcastle.ac.uk

2nd Eliot Young
School of Computing
Newcastle University
Newcastle, UK
e.j.young2@newcastle.ac.uk

3rd Nikodem Jandernal
School of Computing
Newcastle University
Newcastle, UK
n.jandernal2@newcastle.ac.uk

4th Jeny Wightman
School of Computing
Newcastle University
Newcastle, UK
j.wightman3@newcastle.ac.uk

5th Lucy Garden
School of Computing
Newcastle University
Newcastle, UK
l.garden2@newcastle.ac.uk

6th Sakai Newman
School of Computing
Newcastle University
Newcastle, UK
s.newman2@newcastle.ac.uk

7th Wesley Smith
School of Computing
Newcastle University
Newcastle, UK
w.smith7@newcastle.ac.uk

8th Cosmin Irimescu
School of Computing
Newcastle University
Newcastle, UK
c.irimescu2@newcastle.ac.uk

***Abstract*—We developed an innovative semi-automated insurance system which uses machine learning models such as a Convolutional Neural Network (CNN) image-based assessment model to classify damages to users' properties and a Local Outlier Factor (LOF) model for weather anomaly detection. We also integrated blockchain for added security involving transactions. The CNN model achieves an accuracy of 82% classifying damage severity and LOF model classified multiple recent natural disasters correctly. Traditional insurance payouts typically take 48 hours whereas our system aims to achieve this in under 5 minutes.**

***Index Terms*—Artificial Intelligence (AI), Blockchain, Insurance Systems, Machine Learning**

I. Introduction

The increasing frequency and severity of natural disasters has highlighted a need for fast, fair and transparent home insurance solutions. While traditional home insurance companies often suffer from lengthy claim processing times, excessive paperwork, and overly specific policies leaving homeowners vulnerable, our project took a fresh approach offering coverages for all natural disasters. Using innovative artificial intelligence (AI) models and blockchain technology we can increase the efficiency and scope of our coverage to create a new, innovative, automated weather-based insurance system.

Our system leverages CNN for image-based damage assessments and weather sensor data to reduce the time taken for insurance claims to be paid and increase efficiency in the overall system. Using blockchain-powered smart contracts we can guarantee tamper-proof claim processing, reducing need for human intervention and minimising the risk of fraudulent claims. Our project explored technologies at the forefront of AI's involvement in insurance and cryptocurrency payments proving how this technology could innovate the world of insurance.

II. Aim

The overall aim of the project was to innovate the world of insurance with automation. We achieve this through leveraging both blockchain for secure payments and machine learning to verify submitted claims.

III. Objectives

- Research and create two machine learning models to automate the claim verification process. One classifies extreme weather and is evaluated by testing its predictions of recent natural disasters. The other classifies severity of damage found in proof images submitted alongside claims and should achieve accuracy of at least 80%.
- Create a website frontend, allowing users to create accounts, log in, and log claims on properties they own. It should also allow admin users to manage and review claims.
- Create a secure database system to store relevant user information necessary to the functionality of the application, evaluating it against attacks such as SQL injection, and where important values are encrypted.
- Facilitate insurance policy payments and claim payouts through escrow accounts, ensuring they pass rigorous unit tests and are verified by Dafny.
- Implement secure RBAC within smart contracts to prevent unauthorised access, tested against common attacks such as brute force.
- Log transactions and events on the blockchain to create a transparent and immutable audit log.
- Create a series of equations for consistent premiums, payouts and generating a disaster risk value, as well as finding a way to consistently convert Fiat values to Ethereum amounts.
- Produce a secure voting system to require a majority of reviewers to agree on the classification of the claim submission.

IV. Rationale

As the world grows increasingly reliant on AI, it is reasonable to assume that insurance companies will eventually leverage AI models to automate payout claims and to automatically determine the severity of property damage. A recent study into AI's role in the insurance industry discusses improvements in areas such as underwriting, claim processing, customer experience and fraud detection [1]. Considering the increase in natural disasters worldwide, there is a corresponding increase in the number of insurance claims made regarding property damage to homes caused by natural disasters. However, due to the unpredictable nature of natural disasters, many home insurance companies only offer basic policies, creating a big gap in the market to provide more comprehensive cover. To resolve this issue, we've decided to explore this market and see if we can fill this gap using advanced AI models and blockchain.

V. Literature Review

The extreme weather identification model is designed to solve an unsupervised outlier detection problem, looking into data points that do not fit into the established normal weather patterns without predefined labels. Alghushairy et al [2] presented the Local Outlier Factor as an effective density-based method that detects anomalies based on deviations from surrounding neighbourhoods. LOF is ideal for this project as it excels in detecting localised anomalies within dynamic datasets, making it best suited for extreme weather detection. We also explored autoencoders as an alternative approach to outlier detection, Gupta et al [3] discussed their application in anomaly detection by training neural networks to reconstruct normal data, identifying anomalies based on local density deviations. However, autoencoders require significant training data and computational power, whereas LOF is more efficient in real-time detection making it better suited for this situation.

Nidula Elgiriyewithana et al [4] propose a comprehensive, openly accessible global weather dataset, which updates daily containing a multitude of metrics (40+), and collects data from 248 sensors across the globe. This dataset appeared to be the most appropriate for our needs and was used in this project. An alternative is the weather forecast accuracy dataset proposed by Sujay Kapadnis et al [5]. Which solely contains US weather data collected over a period of 16 months, although it lacks daily updates and therefore only contains historical data from a given period. The National Weather Service dataset [6] is produced by the US government and is also restricted to data from that country. Although it is a feature-rich dataset, restrictions around accessibility and a lack of worldwide support limit its effectiveness for our project. A final potential alternative is the Demografy's Consumer Demographics Prediction API [7]. However, this is not a viable option for our project due to it being restricted behind a paywall.

Optimising gas consumption in smart contracts is essential for efficiency, cost reduction, and scalability. Khanzadeh et al. [8] detail methods to reduce gas consumption when writing smart contracts and deploying them on the Ethereum blockchain [8]. Techniques discussed in the paper, such as struct and state variable packing, reduce memory consumption by rearranging member variables to fit within fixed-size memory blocks, significantly decreasing the number of blocks needed to store data. Moreover, Khanzadeh et al. recommend using smaller integer data types where necessary, such as uint8 over uint256, to reduce the amount of storage and computation required, leading to lower gas costs and improved contract efficiency.

RBAC is an essential requirement for ensuring smart contract security on the Ethereum blockchain. Halima Mhamdi et al [9] describe methods for implementing RBAC, particularly by msg.sender to authenticate agents and ensure secure interactions within a blockchain network. Additionally, they highlight the use of require statements to enforce access control, ensuring that only authorised roles can execute specific functions within the contract, thereby preventing unauthorised access and potential security vulnerabilities. The National Institute of Standards and Technologies (NIST) [10] defines RBAC as a standard model for managing access permissions based on user roles. RBAC improves security by enforcing the least privilege and separation of duties, to defend against unauthorised access to confidential systems.

Ensuring the accurate classification of insurance claims is critical to the feasibility of our system. Given the costs associated with payouts, minimising fraudulent claims is essential. Submitted images play a key role in the verification process, providing evidence that property damage has occurred.

The MEDIC [11] [12] [13] [14] [15] dataset was selected as it contains images mostly taken from smartphones, replicating the most likely devices used to submit claims. While others primarily use satellite imagery. Abdelghani Dahou et al [16] proposed utilising this dataset to detect misinformation in Arabic social media posts. They use Multi-Task Learning with a transformer-based model to extract contextual features. These extracted features are then fed into an advanced feature selection model that uses a modified Nutcracker Optimisation Algorithm. A. M. Mustafa et al [17] leverage pre-trained models such as VGGNet19 for disaster classification. Then interprets the results with XAI techniques. Achieving results of 76.93% on the MEDIC dataset. This paper primarily focuses on transparency and trustworthiness. Overall, this dataset hasn't been used for damage classification and is more commonly used to classify social media behaviour.

Pretrained models are adapted to a variety of problems as they provide complexity and sheer size that cannot be replicated easily. K. Simonyan et al [18] proposed a very deep convolutional neural network that achieved 71.3% Top 1 accuracy on the ImageNet validation dataset [19]. Achieving this through increasing depth using small 3x3 convolutional filters. Tan et al[[20] achieved 84.3% accuracy on the ImageNet validation set. To achieve this, they proposed EfficientNet, a model that introduces a scaling method by using a compound coefficient to uniformly scale the network's depth, width, and resolution. They first applied this scaling method to models such as MobileNet [21] and ResNet [22]. This process informed the creation of a baseline network, which was then scaled to create the EfficientNet models. Finally, the best-performing model available through Keras [23] is ConvNeXtXLarge which was proposed by Z. Liu et al [24]. They achieved a Top 1 accuracy of 86.% on the same dataset. The authors achieved this by modernising the ResNet [22] architecture towards that of a Vision Transformer design [25] by modifying both macro and micro-level architectures. For instance, they adjusted layer compute ratios, introduced a “patchify” stem and substituted elements such as ReLU with more modern techniques such as GELU [26].

F. Nex et al [27] propose three convolutional neural networks (CNN) pre-trained to classify building damage from UAV or satellite imagery. This top-down appearance provides benefits in automation, however, any damage to the side of properties or flood damage cannot be assessed. The remaining research typically focuses on a smaller subsection of damage assessment. Such as S. Mangalathu et al [28] who explores using machine learning for classifying earthquake-induced building damage. Training from data collected during the 2014 South Napa earthquake. The paper evaluates techniques such as decision trees, k-nearest neighbours and decision trees. Limitations almost entirely stem from the dataset used which reflects why we believe the upside of the MEDIC dataset is worth the downsides.

Using user-inputted data raises the risk of data poisoning. J. Lin et al. [29] highlights two significant threats faced by machine learning models: adversarial attacks and data poisoning. This paper explores methods used to exploit these attacks, the potential impact this may have on machine learning systems, and the challenges developers face to defend against them. Overall, the authors provide insights into the vulnerabilities of machine learning models and emphasise the need for robust countermeasures to protect against such threats.

SQL injection is a significant security risk that exploits vulnerabilities in the connection between a web server’s front end and its database, potentially leading to unauthorised data access and manipulation, as discussed by Girdhar Gopal [30]. A key mitigation strategy is comprehensive input sanitisation, which presents malicious queries. Parveen Sultana et al [31] highlight client-side validation as a preventative measure, restricting harmful characters before submission.

VI. Threat Model

A. Adversaries

One potential class of adversary are fraudsters who wish to manipulate claims to force payouts to themselves for financial profit. Another, more sophisticated, class of attacker are cybercriminals attempting to perform attacks at scale that affect the overall system and not just the individual. Corrupt employees can also become insiders working either alone or alongside outside attackers against the system.

B. Potential Threats

The machine learning models can be manipulated to produce outputs that are favourable for an attacker. This can be achieved for example through data poisoning. The servers hosting the project can also be attacked to cause local outages. Attackers may try to exploit logical vulnerabilities within the smart contracts. Insiders may bypass security using their increased permissions to compromise the integrity of the system. There is also the potential to exploit vulnerabilities in the smart contracts.

C. Adversaries’ Capabilities

Adversaries can deploy data poisoning to attack the machine learning models, where the data the models train on is injected with fake datapoints to alter the predictions they give. Any unencrypted data sent across the network can also be intercepted by an attacker and read by them. Financial transactions on the blockchain are immutable – they cannot be undone. Therefore, following an attack that results in a financial transaction, it is incredibly difficult for us to recover lost funds. Adversaries may also attempt to corrupt or recruit high privilege employees to bypass security mechanisms.

VII. Security Policy

- Remove any outliers in machine learning training datasets to fight data poisoning that includes extreme values.
- Ensure sensitive data in the database (e.g. passwords) are stored encrypted or hashed so that—even with database access—they cannot be read.
- Implement formal verification of smart contracts in Dafny to ensure input validation.
- Multi-reviewer-based secure voting system for user submitted images which enter the training dataset to limit data poisoning.
- As smart contracts are public, RBAC is used to prevent unauthorised access or modification.
- Redundant backup of codebase (GitHub) to restore application following attacks.
- Implement a blockchain-based audit log to immutably track actions throughout the system, helping to prevent the impact of insider attacks through the threat of detection.

VIII. Disaster Recognition Model

The damage classification model automates assessing damage severity in user submitted images when making a claim. Damage is classified into 3 tiers: little or none, mild, and severe. Automatic payouts are only issued when the model is highly confident in its prediction; while lower damage classifications require manual review by our reviewers to verify the images contain damage. Through this method of claim assessment, we reduce the risk of fraudulent claims successfully being paid out, ensuring the financial stability of the insurance system. During the planning stage, we decided to prioritise false acceptance rate (FAR) as this aligned with our goal of reducing false payouts. False acceptance comes with fraught financial implications while false rejections only cause minor inconvenience while the claim gets manually reviewed.

A. Model architecture

To achieve accurate results, we decided on CNN [32] for the model architecture. As they are well suited to image classification problems due to their ability to automatically extract relevant spatial and structural patterns such as cracks or dents. CNNs learn hierarchical features directly from raw images as opposed to traditional machine learning techniques that would require manual feature extraction to improve efficiency.

To simplify the design and implementation process we leveraged transfer learning from a pre-trained model. We decided on the ConvNeXtXLarge [24] model developed by Meta Research [32] (formerly Facebook). Transfer learning allows us to take advantage of a complex model pre-trained on the ImageNet-21k [19] dataset and then fine-tuned on the ImageNet 1k [19] dataset.

The complexity of both the training dataset and the model itself enables the model to identify essential image patterns, such as edges, textures, and object structures. To adapt this general model to specialise in damage classification we add several additional layers on top of this. Due to the size of the pretrained model we freeze all its layers and only the added ones are trainable. One of these layers was a convolutional block attention module (CBAM) [33]. This enhances feature selection by applying attention mechanisms to both channel and spatial dimensions.

B. Fine-tuning

We fine-tune this model on the MEDIC [11] [12] [13] [14] [15] dataset. This was chosen as it contains images posted on social media at times relevant for natural disasters, which increased the likelihood of images taken on common smartphones and cameras. This would better replicate the type of images users making claims would submit. Whereas most other damage datasets contained satellite or UAV (Unmanned Aerial Vehicle) imagery.

C. Retraining

To ensure the constant improvement of our model's classification performance, we introduce images submitted from manually reviewed claims into the training dataset. This approach allows the model to adapt to the ever-changing appearance of household damage as construction techniques and materials change. This aims to prevent adversaries such as cybercriminals from using data poisoning to skew the model classification abilities.

IX. Extreme Weather Detection Model

The aim of the Extreme Weather Detection Model is to give an estimate of historical weather-related risk at a given location on Earth. This will allow scaling prices and payouts based on how likely extreme weather is for a given location.

A. Dataset

The dataset chosen for the model is a worldwide weather repository for various cities around the world, providing labels such as wind speed, temperature, precipitation levels, humidity level and gust speed which can be used to predict any extreme weather events. It also provides information such as latitude, longitude and time of recording. This information will be used to determine if there were any extreme weather events that have occurred for the past 21 days at the nearest location of which the claim was made.

As the dataset features daily updates, we import this through an application program interface (API) from Kaggle [4]. This allows the model to be retrained daily ensuring it up to date with the latest weather data. In case of local failures, the model will automatically be re-trained, and the dataset will be re-downloaded, making the model more resilient in the process.

B. Model

As previously stated, the architecture used was a Local Outlier Factor (LOF) model, which is an architecture designed for unsupervised anomaly detection. The implementation used in the project is provided by scikit-learn [34]. The dataset used is large enough that clusters may begin to appear around common types of weather (e.g. cloudy, light rain, sunny), while there are considerably less for extreme weather events. The LOF model, given a dataset, detects these clusters and classifies datapoints as either within them or without them. We can then use these classifications to determine if weather at a given location and time in the dataset was extreme.

The model itself outputs either –1 or 1, for an outlier or inlier respectively. For these values to be used to represent a disaster risk score between 0 and 1, multiple weather events need to be considered at once. To do this, the array of the model's predictions over the given period is incremented by one (to the range 0-2) and then summed before being divided by the number of events and then halved. This gives model-predicted extreme weather incidence between 0 and 1 for a group of weather events.

This system is used to help calculate the risk associated with a given property by checking extreme weather in its area over the last year. The higher the value output by the model, the more expensive insurance is for a given property. This model is also used to flag insurance claims for manual review without running the more expensive disaster recognition model if no extreme weather is predicted over the last 21 days.

X. Manual Review of Machine Learning models

The goal of the machine learning models was to allow clear and obvious damage caused by natural disasters to achieve an almost instantaneous payout. However, sometimes the models underperform, or additional manual verification is required. We do this through a secure voting system that prevents insider attacks and data poisoning.

Firstly, the image associated with the claim is stored in the database with the id of the claim that it relates to. Within the claim, the decision of the severity level of the damage is also stored. Each claim contains the predicted damage score obtained through the machine learning model.

After this, each claim that cannot be automatically approved gets sent to multiple reviewers. Each reviewer manually reviews the associated images and description to assess the level of damage present. To ensure this is done while maintaining confidentiality and integrity, Shamir's secret sharing is used to require a minimum of 3 out of 5 reviewers agreeing on the classification. Once this decision is made the status and score gets updated.

XI. Blockchain

A. Access control

Access control in the deployed smart contracts is enforced through a security policy that defines read and write permissions for employees and administrators, while restricting policyholders to read-only access. RBAC is implemented using inline sender verification and conditional checks to ensure proper authentication. Users are limited to actions such as paying their premiums, renewing their policies, accessing their own claims and policy details. These restrictions ensure that users can only interact with their own data, maintaining security and preventing unauthorised access.

B. Audit log

We have implemented blockchain-based audit logging into our project, to satisfy the security dimension. This ensures that necessary policy data, claim data and transactions are securely recorded. Smart contracts manage creation and logging of policy data and claim data, while tracking all transactions made into and out the smart contracts.

We chose a blockchain-based approach due to the inherent security properties of the blockchain. All records stored on the blockchain are permanent and immutable, which makes it an ideal solution for audit logging as it prevents fraudulent alterations or cover ups and maintains integrity of the log. Additionally, a blockchain based approach inherently produces an append-only audit log. Utilising the blockchain also enhances trust and transparency, as all records on the blockchain are publicly verifiable, which strengthens compliance by providing an indisputable log. Furthermore, the decentralised nature of the blockchain eliminates single points of failure. This prevents corruption or deletion of the audit log, which prevents adversaries from hiding malicious activities and ensuring full availability of an accurate audit log. We implemented a minimal data logging approach, to reduce the exposure of personal information, as blockchain records are publicly accessible. Each record is therefore logged with a unique identifier, allowing for traceability without exposing personal information.

C. Gas efficiency

To improve efficiency and reduce transaction costs, various gas optimisation techniques were applied across the five smart contracts.

In the policy contract, storage was optimised through struct packing and the use of smaller data types to reduce memory overhead. The admin's address was set as immutable to lower storage access costs [8]. Function modifiers were replaced with inline conditional checks to reduce execution overhead. Events were optimised by indexing key parameters and using compact data types for premium and payout values.

The escrow and payment contracts optimised storage writes by deleting data instead of resetting values to zero. Inline RBAC checks replaced explicit validation functions, reducing gas consumption. Event logs were further optimised with indexed parameters and smaller data types for more efficient queries [8].

Moreover, in the claims contract, struct packing improved storage efficiency, and marking the external policy interface as immutable reduced access costs. Indexed event parameters were used to enhance query performance.

The payout contract consolidated redundant functions into a single process, to streamline payouts and lower transaction costs.

These optimisations significantly improve gas efficiency, reducing execution costs while maintaining functionality and security.

XII. Calculations

One of the core functions of our weather-based insurance system is implemented through a series of interconnected equations designed to perform critical calculations. The equations act as a mathematical framework for consistently ensuring transparency and reproducibility, aligning with the industry standard outlined in "How Insurance Works" by the Association of British Insurers [35]. They perform key tasks such as converting Fiat (GBP, USD, ect) into Ethereum and Wei using real-time exchange formulas, calculating geodesic (shortest 3D curve across the earth's surface) distances and normalising them into risk scores. As well as determining insurance premiums and claim payouts through a damage assessment. This approach allows us to automate these calculations in a decentralised environment and adapt established insurance methodologies to ensure that risk and premium computations remain reliable and aligned with current market conditions.

A. Cryptocurrency Conversion

Due to the nature of this project, we needed to find a way to convert fiat currency amounts into Ethereum, more specifically Wei, Ethereum's smallest unit. To do so we use an external API, CryptoCompare [36] to dynamically retrieve real-time exchange rates, which ensures conversions remain accurate and reflect current market conditions. This conversion is the base of all transactions in this project, integrating traditional financial metrics with our decentralised blockchain-based premium and payout computations, ensuring all financial calculations are performed with a high standard of precision and security.

B. Disaster Risk Assessment

To help show users a need for this type of insurance we provide them with a risk score upon completing their sign-up; a normalised value based on two primary components, proximity to risk, and frequency in that area. We use OpenStreetMap Nomination API [37] to convert a user's postcode into geographical coordinates, which are there compared to an extensive list of natural disaster hotspots, such as tectonic plate boundaries. The distance between these two coordinates is calculated to generate a proximity score. Then from the weather detection machine learning model, we compute a disaster frequency score, presenting the likelihood of extreme weather events in the area. The final score is calculated by averaging the two values and providing a normalised number.

C. Premium

When calculating the value a user would pay, we had to consider real circumstances and explore pre-existing insurance policies for market research. We discovered that most home insurance policies don't cover all-natural disasters, or 'acts of god' as defined by The Association of British Insurers to be "an event that is not the fault of any individual" [38], which aligns with our initial project objectives. This shows the importance of accurately determining the premiums for our system, as traditional insurance policies leave many circumstances unaddressed. We decided on using a base premium that can be adjusted based on risk score. For the initial baseline value, we decided on £70 which, at the time of writing converts to around 48456655521635904 Wei. This baseline cost ensures low-risk properties have a minimum premium amount. The insured value is determined by multiplying the house's current value by a set insurance coverage percentage which is 80%. This replicates real-world home insurance policies that also insure around 80% of the total property value. The risk-based premium is then calculated by applying a premium rate of 0.2% to the insured portion of the property value, which falls within the industry standard range. This component reflects the variable part of the premium that adjusts according to the risk assessment value connected to the property. The final sum is generated by adding the fixed base premium to the risk-based premium giving us a total monthly premium.

D. Final Payout

To ensure fair and consistent payout amounts across the system we run a calculation using a set house insurance coverage of 80%. The final payout is determined by multiplying the home coverage value by the normalised value of the damage score. Which is a value set to 0 for little or no damage, 1 for mild damage or 2 for severe damage which is computed by the image assessment machine learning model. Once the final payout is confirmed it is processed through the smart contracts.

XIII. Website Design and Architecture

The visual website design shown in Appendix Figure 2 follows key user experience and user interaction principals to ensure a clear and intuitive movement through the webpage [39].

Our system took a simple two-server approach with independent Django and React applications. On one hand, the Django server offers RESTful APIs and manages core backend logic and database interactions; on the other hand, the React application is an independent client that makes requests to Django. We choose these technologies because our team was already familiar with them, and this experience has been an advantage to our development process.

The Django backend is constructed on top of Django version 5.1.7 and Django rest framework. Data is stored using SQLite3, which presently contains user account information, claims data, and corresponding operation logs. Furthermore, RBAC is implemented in the backend where each user has a specific role (such as Admin, Reviewer, or User) that determines the endpoints they can access. Additionally, we utilise the Django-cors-headers package to facilitate secure communication between our React frontend, which is hosted on a different port, and Django's REST endpoints.

The front end is built on JavaScript (ES6) using React's component-based design. Communication with Django is done mostly through a series of RESTful endpoints. After a successful login by a user, the session state is managed on the client side by the frontend by storing the user's role and id number, allowing customisation of the user interface depending on the logged in users profile. This also makes it easy to add more administrative controls or claims validation functionality in the future.

1) Data Transmission: Whenever the user submits or updates a claim, React sends an HTTP request to a specific Django endpoint. If this operation is successful, Django saves the data to SQLite3 and then sends back JSON responses. The front end then updates the data on display in real-time.

2) Security: While the system can be expanded with JWT tokens or more sophisticated authentication mechanisms, our present strategy depends on Django's native session management supplemented with CORS headers to facilitate cross-origin requests. This is straightforward and efficient for our development setup.

The architecture is modelled in a two-server setup, thus maintaining flexibility throughout the development phase as well as the deployment process. Each server can be operated separately, enabling separate testing and scalability as required. Besides, we advocate for the incorporation of other modules of the project by attaching them to the Django server through other endpoints, thereby maintaining a simple and user-friendly front end. Briefly, our two-tier system with React and Django provides a good separation of responsibilities, enables effective collaboration, and serves as a sound foundation for future developments in security, reliability, and further integrations

A. Implementation

We divided our implementation work between the React front end and the Django backend, with each service being constructed and operated separately. This decoupling allowed us to rapidly iterate on a per-piece basis, roles to be cleanly defined and helped us steer clear of the potential dependencies that are introduced when frontend code relates to backend logic. In the following sections, we outline the specific implementation steps and key components of each layer.

B. React Frontend

In the React layer, we utilised ‘Create React App’ to initialise our development environment, thereby establishing a recognisable directory structure that describes various concerns neatly. The project is structured into:

1) Components: Individual pieces of the user interface, i.e., login forms, claim submission forms, and dashboards. These were organised as functional components, which utilise React Hooks for managing local state.

2) Pages: Custom routes (e.g., ‘/login’, ‘/dashboard’, and ‘/claims’) that compose several components to present a cohesive view. We use React Router for navigation, thus providing a seamless single-page application experience.

3) Services: Wrapped pieces designed to simplify the task of issuing HTTP requests to our Django API. Services encapsulate the raw ‘fetch’ or ‘axios’ procedures and handle things such as error handling and setting request headers.

When the user submits a new insurance claim through the interface, the related form data (i.e., text inputs, file uploads) is sent to our backend through a POST request. We use CORS (via ‘django-cors-headers’ on the server side) to allow cross-origin requests from the React dev server. Any response from the server is then displayed in real-time to guide the user’s next action.

C. Django Backend

On the backend, we kept the typical Django project structure. We had one Django app, named ‘core’, that managed all the business logic for registering a user, logging in, and filing claims:

1) Models: We created models such as ‘User’ (extending Django’s built-in user model to support custom roles) and ‘Claim’ (to track user submissions, claim status, and timestamps). Since we’re using SQLite3, there are virtually no setup and maintenance costs.

2) Views & URLs: Code for a particular endpoint such as is specified by class-based or function-based views. Authentication and permissions are checked here, typically by a combination of Django’s built-in authentication and DRF permission classes. This ensures that only users with appropriate access can update or retrieve some data. To facilitate role-based access, we utilise our own ”user profile” fields. Upon an incoming request, the view checks the role of the request’s user object. If a request tries to reach an endpoint meant for an Admin while the user is in a customer role, we return a 403 (Forbidden). This process ensures that even if an attacker manipulates the frontend, the backend is actively enforcing rigorous access control.

XIV. Scalability

To enhance scalability, the models need to be able to run efficiently to meet hypothetically increasing demand. Computationally heavy functions such as both the machine learning models, and the corresponding equations are run on separate threads to prevent causing the entire website to wait upon the completion of the algorithm.

To ensure the computationally heavy models are not repeatedly loaded upon claim submission, the program processes claims once every 5 minutes. Upon a claim submission, a check is performed to see when the models were previously run. If 5 minutes have passed, all new claims are processed in bulk to prevent the repeated creation of memory intensive threads.

Both the weather detection model and its corresponding dataset are stored locally, and these local copies are fetched in favour of creating new ones when the model is run. The time since the last time new copies were downloaded/created is tracked, and if it has been longer than a day since the most recent datapoint in the dataset, new copies are fetched. Of note, this decreases security as compromised datasets or models could be uploaded, and these would be run on the backend unless the dataset was greater than a day old. However, is decreases time taken to run a prediction by a significant amount, greatly enhancing scalability.

The blockchain is inherently scalable, due to it being decentralised and holding no single point of failure, it ensures fault tolerance for the smart contract system. Additionally, our smart contracts were developed to be highly efficient with minimal gas usage. We have reduced the complexity of on-chain operations and using appropriate data types within logging and computations. This ensured that the smart contracts required less computational resources to execute, which results in faster processing times and lower transaction fees.

RBAC within the smart contracts contribute heavily to the scalability of the project, mainly by preventing unnecessary blockchain writes as only authorised users can perform writing operations on the blockchain, reducing congestion and transaction load.

XV. Threat Mitigation, Security & Resilience

As noted in the threat model, the weather detection model is highly vulnerable to data poisoning, as the dataset used is publicly available and is updated regularly. To mitigate this, we remove any obvious outliers in the weather data which give readings which are highly unlikely to happen (for example any reading that exceeds the global maximum or minimum) by implementing manual outlier detection. This will detect any outlier readings and check if is higher or lower than the highest or lowest recorded instance for a specific weather data reading. If there is a reading which is higher than the highest or lowest recorded instance for a particular weather reading, the row is removed from the final dataset. The relevant weather data is then normalised so that it is easier for the model to detect any outliers which indicate for a possibility for extreme weather.

Since the dataset is updated daily & model re-trained daily, the system is naturally resilient to local failures. Any local outage can be remedied by simply re-initialising the model and redownloading the dataset. The newest iteration of the dataset and trained model are also stored locally, so in the case of the dataset download malfunctioning the local backup can be used instead for near-identical performance.

XVI. Evaluation

A. Image model

TABLE I
Model performance over multiple metrics

Classification report	Precision	Recall	F-1 Score	Support
Little or none	0.87	0.95	0.91	10252
Mild	0.36	0.12	0.18	1527
Severe	0.75	0.75	0.75	3909
Accuracy	-	-	0.82	15688
Macro average	0.66	0.61	0.61	15688
Weighted average	0.79	0.82	0.80	15688

TABLE II
FAR and FRR

Performance	Percentage
False Acceptance Rate	20%
False Rejection Rate	28%

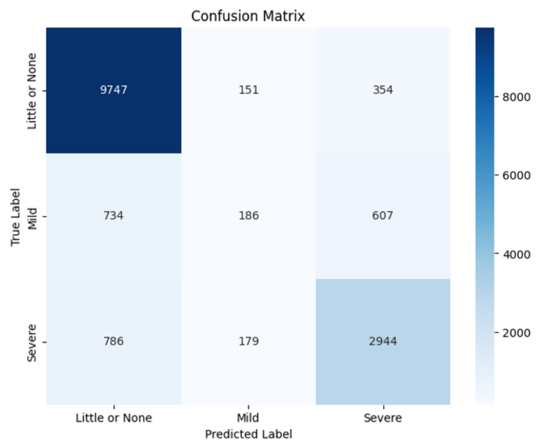


Fig. 1. Confusion Matrix

Overall, we selected a wide range of metrics to display a comprehensive overview of the model's performance. This includes both the overall performance and separated into per-class performance. Doing this allows for fine-tuning: ensuring the false acceptance rate is reduced to be as low as possible even at the slight expense of the false rejection rate increasing.

False acceptance is defined as when an image that shows 'little or no' damage is instead classified to be either 'mild' or 'severe', fooling the model into approving the claim for automatic payout. False rejection occurs when an image that should be classified as 'mild' or 'severe' gets classified as 'little or none'. Causing the model to reject the claim.

A false acceptance and rejection rate of 0.20 and 0.28 respectively show that our model has achieved its goal of focussing more on false acceptance as this minimises false payouts. However, the overall performance could massively be improved given more resources and a dataset containing images more specific to the exact nature of the project.

The Image-based CNN damage classification model achieves an accuracy of 82% on the MEDIC test dataset, with strong performance in classifying both the 'little or none' and 'severe' categories with an F1 score (harmonic mean of precision and accuracy) of 0.91 and 0.75 respectively. However, the model struggles significantly with images that fall under the 'mild' category, achieving a poor recall score of 0.12 and an F1 score of 0.18. The confusion matrix backs up this claim by showing the misclassification of 'mild' cases as either 'little or none' (734 instances) or 'severe' (607 instances), as seein in Figure 1. The main cause of these issues is class imbalance – some classes contain many more training samples than others. Potential fixes could include SMOTE [39/E24], a class rebalancing algorithm, or under sampling.

The strength of this model is its ability to handle non-relevant images due to the abundance of these types of images within the dataset. On top of this, the overall performance of the model is at a good but not great level providing a sufficient accuracy score for this to be beneficial to the overall insurance process.

Weaknesses include the lack of context surrounding the image. While predictions on the actual images are accurate, there is no way of knowing where or when the submitted image was taken.

B. Extreme Weather Detection Model

For the extreme weather model, testing is innately difficult because the model is unsupervised - there is no ground truth to compare outputs to. Visualisations were created for each of the model's dimensions to demonstrate what patterns it classifies as extreme. In addition, isolated tests were run for specific, known, recent natural disasters to determine if the model classified them as extreme weather points or not.

TABLE III
Example natural disasters and their model scores

Natural Disaster	Model Score (0-1, 1 = Most Extreme, 4d.p)
Missouri Hurricane (March 2025)	**0.6364**
Florida Tornado (March 2025)	**0.1905**
Control (London, March 2025)	**0.0909**

Example natural disasters (and a control) tested alongside their model scores over a timeframe of the last 21 days. The control by far has the lowest score, indicating a low false positive rate. However, not all extreme weather is consistently *given a score less than 0.5, indicating a tendency towards false negatives.*

C. Smart Contracts

As the smart contracts are integral to the project, it is imperative that they work as intended. Therefore, we developed unit tests for all the smart contracts, to check that the functions within them behave correctly with given valid or invalid inputs. This helps validate core functionalities and prevents logical failures, as an undetected bug could produce vulnerabilities within our system, especially as smart contract code is readable by everyone.

After the first iteration of running the unit tests, we found that nearly all smart contract functions behaved as intended with only a few discrepancies. For example, there was a discrepancy between the values in a "claim" struct during generation against the values emitted in the event responsible for logging the claim details; this was then amended within the responsible smart contract.

We further evaluated the security of the smart contracts using formal verification, this was implemented as it was a requirement within our security policy. While unit testing was an effective method of evaluating whether functions behaved as expected, it was only focused on intended use cases and did not cover every possible edge case or unintended interaction within the smart contracts. We therefore incorporated Dafny's formal verification system, which applies mathematical proofs to evaluate all possible states and logic flows. This approach allowed us to detect vulnerabilities within our smart contracts that our unit tests do not account for, such as logical inconsistencies or edge cases.

After implementing this approach, we found many potential logical inconsistencies and edge cases that could enable a vulnerability within our smart contracts. For example, we discovered scenarios where interdependent functions could fail unexpectedly, leading to unintended errors within the function responsible for paying out claims from Escrow.

D. Website Evaluation

To evaluate the web design, we used Google Lighthouse and ran analytics on the user landing home page and the user's dashboard. The websites home page, Appendix Figure 2 received a Google Lighthouse performance score of 95 (view appendix for full review report) and a 100-accessibility rating. The user dashboard, Appendix Figure 3, received an overall performance of 78, with an accessibility score of 80. While the scores are not equal, they are suitable results for our project.

E. Evaluation Summary

We successfully met all our original aims and objectives set out in the planning stage of the project. As each objective was evaluable with a clear goal to outline successful completion, we broke down the project into smaller more manageable sections. Doing so streamlined the overall development process.

XVII. Limitations & Future Work

The weather detection model uses a simplistic architecture in Local Outlier Factor. Due to the time constraints of the project, other potential architectures such as autoencoders were not even attempted. It is possible alternatives would yield better results, and this should be explored in future work. Furthermore, the detection model only considers weather events in a vacuum, instead of considering weather over time. Autoencoders would be a great way to explore the idea of considering time-series in predictions further, due to their innate excellence in this niche [3], making them a natural next step.

The current premium and payout calculations would need more detailed research to establish more specific premium values that consider specific factors, such as rebuild costs in certain areas prone to tornados or wildfires. Which would add another value to said equations due to time constraints and limited resources this was out of scope for the current scale of this project.

Failure to implement time-locking mechanisms was a limitation of the blockchain implementation. In the high-level design, it was proposed that time-locking would be incorporated to prevent large, rapid withdrawals, which are often associated with fraudulent activity. However, due to time constraints and testing challenges, this feature could not be implemented or thoroughly evaluated. Future iterations of the system should prioritise the inclusion of time-locking to enhance security and mitigate potential risks.

To improve the damage classification model's performance, we could implement separate models for each type of disaster. Currently, the model is trained on images caused by a variety of different disasters making features found differ massively. This was unrealistic to implement as it would require a model to classify the category of the disaster and then one for each natural disaster severity classifier. Doing so would have added too much training time and complexity to this project.

For the image classification model, images from reviewed claims are used to retrain the model. Some consequences of doing this include adversaries being able to perform data poisoning attacks. We solve this with RBAC and employee voting, however, attacks that add adversarial hidden noise to images would be undetected.

Collecting our own natural disaster image dataset that doesn't contain the same quantity of noise could most likely improve results as the images would solely focus on the actual damage, as opposed to random watermarks and noise in the current dataset.

Also experimenting with other pre-trained models such as the ConvNeXtXLargeV2 [41]. This improves performance through adding additional components and improving the overall model architecture. Unfortunately, this model isn't available within TensorFlow-Keras [23] currently making this implementation too time-consuming.

XVIII. Conclusion

Our AI-powered weather insurance system integrates machine learning and blockchain to semi-automate secure claims verification. The combination of a CNN-based damage classification model and an LOF-based weather anomaly detection model ensures a robust verification mechanism. Overall, the system's ability to automate the claims approval process provides a significant improvement in terms of processing timeframe when compared to traditional insurance that requires manual review. Blockchain enhances security and privacy through streamlining the payment system with smart contracts. A secure append-only tamperproof audit log is also stored on the blockchain. The immutable nature of blockchain ensures user trust through transparency and immutable transaction records.

While the damage classification model achieves a high level of performance, challenges remain. Particularly regarding false acceptance, due to the financial risk associated with incorrectly paying out claims. While this limitation does not prevent the core functionality of the model, it could potentially impact the feasibility of the system as a profitable application. The weather detection model also demonstrates promising results; however, limitations remain regarding the location of sensors. Some issues regarding user distance to sensors could impact the relevance of the reported weather assessment. Blockchain successfully enhances the payment system however, limitations regarding it still require an admin to approve payout limit the true automation of the smart contracts.

References

[1] A. Durant, F. Mcclure, M. Karunakaran, L. Anderson, and G. Nakato, "Artificial Intelligence Is Transforming the Insurance Industry, Introducing Innovative Methods That Revolutionise the Buying Process for Customers ," Researchgate, Nov. 23, 2022. https://www.researchgate.net/publication/386050902_ARTIFICIAL_INTELLIGENCE_IS_TRANSFORMING_THE_INSURANCE_INDUSTRY_INTRODUCING_INNOVATIVE_METHODS_THAT_REVOLUTIONIZE_THE_BUYING_PROCESS_FOR_CUSTOMERS

[2] O. Alghushairy, R. Alsini, T. Soule, and X. Ma, "A Review of Local Outlier Factor Algorithms for Outlier Detection in Big Data Streams," Big Data and Cognitive Computing, vol. 5, no. 1, p. 1, Dec. 2020. doi: https://doi.org/10.3390/bdcc5010001.

[3] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau, "Autoencoder-based network anomaly detection," IEEE Xplore, 2018. https://ieeexplore.ieee.org/abstract/document/8363930.

[4] N. Elgiriyewithana, "World Weather Repository (Daily Updating)," Doi.org, 2023. https://doi.org/10.34740/KAGGLE/DSV/11086825 (accessed Mar. 20, 2025).

[5] S. Kapadnis, "Weather Forecast Accuracy," Kaggle.com, 2024. https://www.kaggle.com/datasets/sujaykapadnis/weather-forecast-accuracy?select=cities.csv (accessed Mar. 20, 2025).

[6] National Weather Service, "National Weather Service," Weather.gov, 2019. https://www.weather.gov.

[7] E. Caterino, "Demografy's Consumer Demographics Prediction API," Datarade.ai, 2024. doi: https://datarade.ai/data-products/demografy-s-consumer-demographics-prediction-api-demografy.

[8] S. Khanzadeh, N. Samreen, and M. H. Alalfi, "Optimizing Gas Consumption in Ethereum Smart Contracts: Best Practices and Techniques," IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C), pp. 300–309, Oct. 2023. doi: 10.1109/qrs-c60940.2023.00056.

[9] H. Mhamdi, B. O. Soufiene, A. Zouinkhi, O. Ali, and H. Sakli, "Trust-Based smart contract for automated agent to agent communication," Computational Intelligence and Neuroscience, vol. 2022, pp. 1–11, Sep. 2022. doi: 10.1155/2022/5136865.

[10] Nist.rip. (2017). NIST Computer Security Division - Automated Combinatorial Testing for Software (ACTS). https://csrc.nist.rip/groups/SNS/rbac (accessed Mar. 19, 2025).

[11] F. Alam, T. Alam, M. A. Hasan, A. Hasnat, M. Imran, and F. Ofli, "MEDIC: A Multi-Task Learning Dataset for Disaster Image Classification," Neural Computing and Applications, vol. 35, no. 3, pp. 2609–2632, 2023.

[12] F. Alam, F. Ofli, M. Imran, T. Alam, and U. Qazi, "Deep Learning Benchmarks and Datasets for Social Media Image Classification for Disaster Response," in 2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), 2020.

[13] F. Alam, F. Ofli, and M. Imran, "CrisisMMD: Multimodal Twitter Datasets from Natural Disasters," in Proceedings of the 12th International AAAI Conference on Web and Social Media (ICWSM), Stanford, California, USA, 2018.

[14] H. Mozannar, Y. Rizk, and M. Awad, "Damage Identification in Social Media Posts Using Multimodal Deep Learning," in Proceedings of ISCRAM, May 2018, pp. 529–543.

[15] D. T. Nguyen, F. Ofli, M. Imran, and P. Mitra, "Damage Assessment from Social Media Imagery Data During Disasters," in Proceedings of ASONAM, Aug. 2017, pp. 1–8.

[16] A. Dahou et al., "Optimizing fake news detection for Arabic context: A multitask learning approach with transformers and an enhanced Nutcracker Optimization Algorithm," Knowledge-Based Systems, vol. 280, pp. 111023–111023, Nov. 2023. doi: https://doi.org/10.1016/j.knosys.2023.111023.

[17] A. M. Mustafa, R. Agha, L. Ghazalat, and T. Sha'ban, "Natural disasters detection using explainable deep learning," Intelligent Systems with Applications, vol. 23, p. 200430, Sep. 2024. doi: https://doi.org/10.1016/j.iswa.2024.200430.

[18] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Apr. 2015. https://arxiv.org/pdf/1409.1556.

[19] "ImageNet," https://www.image-net.org/about.php.

[20] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," Sep. 2020. https://arxiv.org/pdf/1905.11946.

[21] A. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Apr. 2017. https://arxiv.org/pdf/1704.04861.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Dec. 2015. https://arxiv.org/pdf/1512.03385.

[23] Keras, "Home - Keras Documentation," Keras.io, 2019. https://keras.io/.

[24] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," arXiv:2201.03545 [cs], Jan. 2022. https://arxiv.org/abs/2201.03545.

[25] A. Dosovitskiy et al., "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale," Jun. 2021. https://arxiv.org/pdf/2010.11929.

[26] D. Hendrycks and K. Gimpel, "GAUSSIAN ERROR LINEAR UNITS (GELUS)." Available: https://arxiv.org/pdf/1606.08415.

[27] F. Nex, D. Duarte, F. G. Tonolo, and N. Kerle, "Structural Building Damage Detection with Deep Learning: Assessment of a State-of-the-Art CNN in Operational Conditions," Remote Sensing, vol. 11, no. 23, p. 2765, Nov. 2019, doi: https://doi.org/10.3390/rs11232765.

[28] S. Mangalathu, H. Sun, C. C. Nweke, Z. Yi, and H. V. Burton, "Classifying earthquake damage to buildings using machine learning," Earthquake Spectra, vol. 36, no. 1, pp. 183–208, Jan. 2020, doi: https://doi.org/10.1177/8755293019878137.

[29] J. Lin, L. Dang, M. Rahouti, and K. Xiong, "ML Attack Models: Adversarial Attacks and Data Poisoning Attacks," arXiv (Cornell University), Dec. 2021, doi: https://doi.org/10.48550/arxiv.2112.02797.

[30] ResearchGate. (n.d.). (PDF) CASE STUDY OF SQL INJECTION ATTACKS. [online] Available at: https://www.researchgate.net/publication/309404360_CASE_STUDY_OF_SQL_INJECTION_ATTACKS.

[31] Parveen Sultana H, Sharma, N., None Nalini N, Pathak, G. and Pandey, A. (2023). Prevention of SQL Injection Using a Comprehensive Input Sanitization Methodology. doi:https://doi.org/10.3233/atde221269.

[32] "Meta Research," Meta Research. Available: https://research.facebook.com/.

[33] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, “CBAM: Convolutional Block Attention Module,” arXiv:1807.06521 [cs], Jul. 2018, Available:https://arxiv.org/abs/1807.06521.
[34] Scikit-Learn, “scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation,” Scikit-learn.org, 2019. https://scikit-learn.org/. [
[35] ABI, “How Insurance Works ABI,” Abi.org.uk, 2019.https://www.abi.org.uk/data-and-resources/tools-and-resources/how-insurance-works/.
[36] “Cryptocurrency Prices, Portfolio, Forum, Rankings,” CryptoCompare. https://www.cryptocompare.com/.
[37] “Nominatim Demo,” nominatim.openstreetmap.org. https://nominatim.openstreetmap.org/ui/search.html.
[38] “Act of God — ABI,” www.abi.org.uk. https://www.abi.org.uk/data-and-resources/tools-and-resources/glossary/act-of-god/.
[39] E. Stevens, “7 fundamental UX design principles all designers should know - UX Design Institute,” UX Design Institute, Sep. 10, 2024. https://www.uxdesigninstitute.com/blog/ux-design-principles/.
[40] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” Journal of Artificial Intelligence Research, vol. 16, no. 16, pp. 321–357, Jun. 2002, doi: https://doi.org/10.1613/jair.953.
[41] S. Woo et al., “ConvNeXt V2: Co-designing and Scaling ConvNets with Masked Autoencoders,” arXiv:2301.00808 [cs], Jan. 2023, Available: https://arxiv.org/abs/2301.00808.
[42] L. Alzubaidi et al., “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” Journal of Big Data, vol. 8, no. 1, Mar. 2021, doi: https://doi.org/10.1186/s40537-021-00444-8.

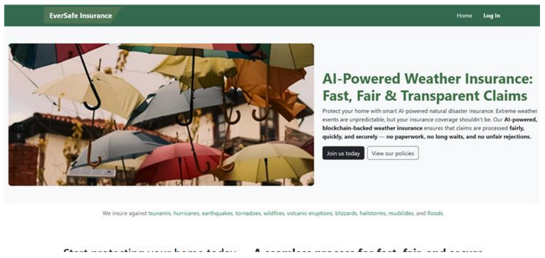
XIX. Appendix

A. Contribution Matrix

TABLE IV
Contribution Matrix

%	Eliot	Ben	Niko	Jenifer	Sakai	Wesley	Lucy	Cosmin
12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5

B. Images


EverSafe Insurance
Home Log In

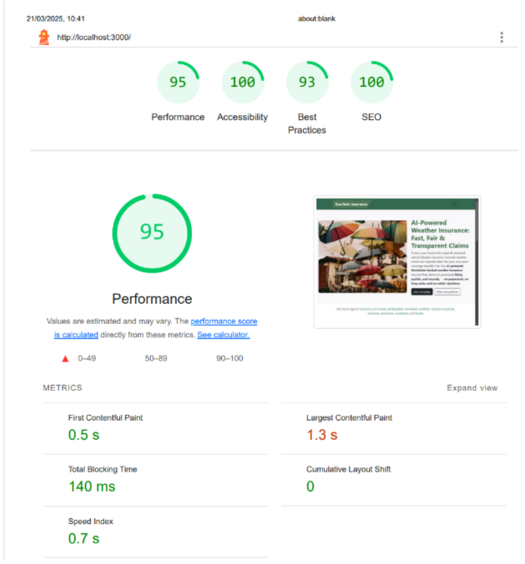
AI-Powered Weather Insurance: Fast, Fair & Transparent Claims

Protect your home with smart AI-powered natural disaster insurance. Extreme weather events are unpredictable, but your insurance coverage shouldn't be. Our **AI-powered, blockchain-backed weather insurance** ensures that claims are processed **fairly, quickly, and securely — no paperwork, no long waits, and no unfair rejections.**

Join us today | View our policies

We insure against tsunamis, hurricanes, earthquakes, tornadoes, wildfires, volcanic eruptions, blizzards, hailstorms, mudslides, and floods.

Fig. 2. EverSafe Insurance home page.


21/03/2025, 10:41 about:blank
http://localhost:3000/

95 Performance | 100 Accessibility | 93 Best Practices | 100 SEO

95
Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 | 50–89 | 90–100

METRICS | Expand view

First Contentful Paint 0.5 s
Largest Contentful Paint 1.3 s
Total Blocking Time 140 ms
Cumulative Layout Shift 0
Speed Index 0.7 s

Fig. 3. Home Page Google Lighthouse review.

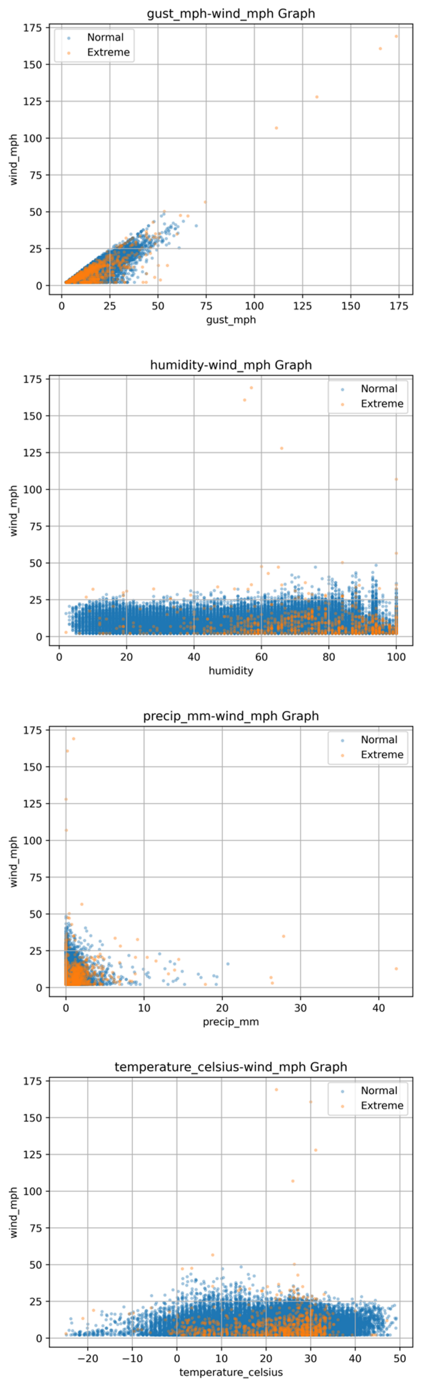


Fig. 4. Scatter plot of wind speed over every other variable, with blue points representing points classified as inliers and orange points classified as outliers. All points with high wind speed are classified as extreme, but other variables are less consistent in their classification, implying wind speed is the primary consideration in outlier classification.