HumanMade: A Platform for Verified Human Creations

Author: Luca Sarif-Kattan
Supervisor: Ligang He
Year of study: 2024
Keywords: AI, Generative AI, Human, Creations, Blockchain, Web App, Node.js, Neural Network

Abstract: With the dawn of generative AI, the landscape of human creativity and ingenuity is increasingly encroached upon. It is clear to most experts that this trend will continue in some fashion [1], and there is a high chance that the entire creative space will be heavily dominated by AI in the near future. Due to the anthropocentric worldviews many of us hold [2] and to retain the pride of the human identity, having a platform which supports and incentivises human creation seems like it will be an in-demand product, and a beneficial necessity. HumanMade seeks to do this whilst avoiding the flawed approach of directly detecting AI-generated content, by:

- Giving humans an easy way to document and upload their Creations
- Ensuring Creations are tamper-proof and traceable to the original creator
- Enabling and empowering the community of users to decide what they think is human-made and deserves their attention

Acknowledgements: I would like to thank myself for my amazing ideas, skilled programming and genius-level problem solving. I would like to thank Ligang He for a frictionless and enjoyable supervisor experience.

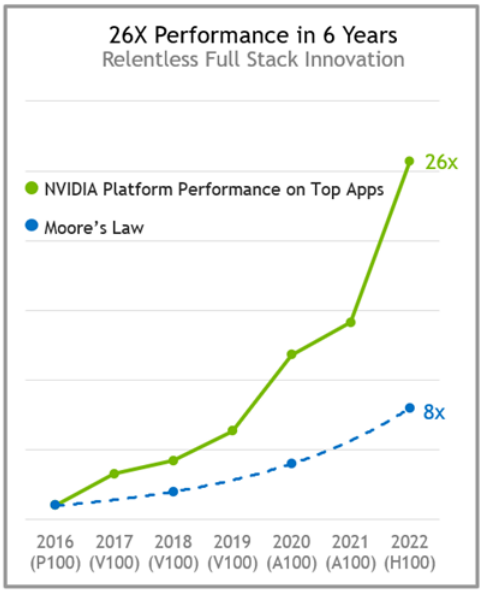
Contents

1 Motivators

1.1 The Generative AI Explosion

We are currently (March of 2024) in the midst of a generative AI (GAI) arms race. ChatGPT managed to reach 100 million users in 2 months [3]. Incredible advancements are being made weekly. More broadly, experts are predicting High-Level Machine Intelligence by 2059 [1], and this is an exponentially decaying timeline, down 8 years from the previous time the survey was given. Here are some of the main reasons for this current AI explosion:

- **Advancements in hardware.** The architectures which a lot of these models are based off of were discovered a while ago, (for example, the seminal Attention Is All You Need paper was released in 2017 [4]), but recently we have witnessed huge improvements in the AI accelerated hardware needed to scale these models. Examples include Google's development of the TPU [5], and Nvidia's development of the Ampere architecture line of GPUs. GAI models require large-scale compute for the training phase.



[6]

Figure 1: A comparison of Nvidia hardware progress compared to Moore's law. The A100 GPU was used to train OpenAI's GPT-4. [7]

- **Lack of regulations and restrictions.** There are currently no restrictions on the development of such technologies in the US, and the EU has introduced limited restrictions in the new AI act [8]. There are obvious geopolitical incentives to allow the acceleration of GAI technologies, and open letters calling to pause giant AI experiments have failed to bring action [9].
- **Profit incentives.** Such technologies are applicable across a huge spectrum of high-level tasks. There was 14 billion dollars in funding in 2023 towards GAI technologies [10].

1.2 A Look at the SOTA

A big part of the motivation for such a project comes from the unbelievable progress and ability of current GAI models, and how we might attempt to improve their detection. Therefore, in

this section we take a brief look into the current state-of-the-art in GAI. At the moment there are two dominant model types:

- Transformer models, used for text generation
- Diffusion models, used for image and video generation

Transformers were initially designed for NLP tasks, but have proven to be incredibly versatile and scalable [11]. Performance has increased linearly with model size, even as model sizes have increased to trillions of parameters.

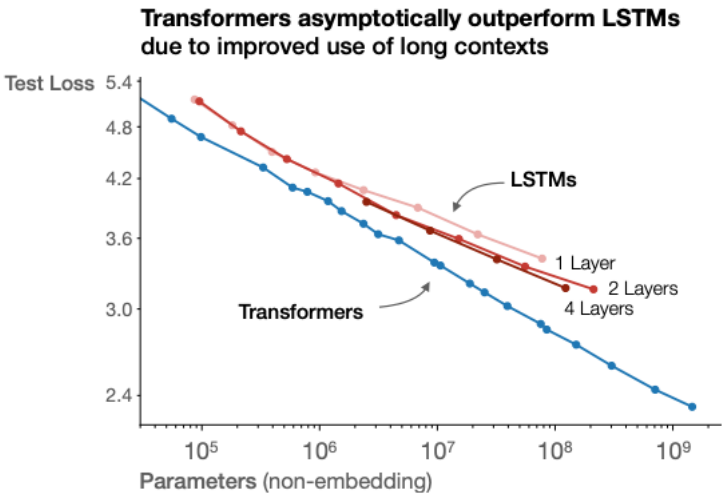


Figure 2: Test loss (performance) against size (parameter count) of a model

SOTA for Transformers is currently Claude Opus, recently released by Anthropic. A few stats include Opus achieving 86.8% on the MMLU (Undergraduate level multiple choice test) and 84.9% on the HumanEval code benchmark [12].

Diffusion models gradually transform noise into a coherent image, based on learned data distributions. The SOTA is inherently more subjective in this area, but Midjourney version 5 is widely regarded as a front-runner in image generation. Midjourney can generate extremely detailed images from highly specific prompts.

Figure 3: Generated using Midjourney with the prompt: 'A scholarly turtle wearing glasses and a graduation cap, sitting in front of a stack of books. He has a wise and contemplative expression, as if lost in philosophical thought'

SOTA in video generation is clearly Sora, a recently released model from OpenAI which took the industry by surprise. Essentially working by diffusion but on multiple frames, Sora can also take in highly specific prompts and output detailed and realistic videos.

Figure 4: A frame from a Sora video, generated with the prompt: 'A stylish woman walks down a Tokyo street filled with warm glowing neon and animated city signage. She wears a black leather jacket, a long red dress, and black boots, and carries a black purse. She wears sunglasses and red lipstick. She walks confidently and casually. The street is damp and reflective, creating a mirror effect of the colorful lights. Many pedestrians walk about.'

1.3 Preserving Human Creativity

The implication is that soon, we will be inundated with synthetic media and art. Whilst it is important to preserve human creations for philosophical reasons like maintaining the human identity, creative spirit and pride, there are two clearer motivators for building HumanMade:

1. There will be free-market demand for authentic, human made creativity
2. There is demand for GAI detection and tracking

Expanding on the first - recent research (perhaps unsurprisingly) shows a clear bias in humans towards human made art [2]. In this study, participants in different groups were presented with the same piece of art, but labelled as either human made or AI made. These labels directly effected a participants' preference to buy the art, via biased perceptions of the creativity and awe they felt towards it.

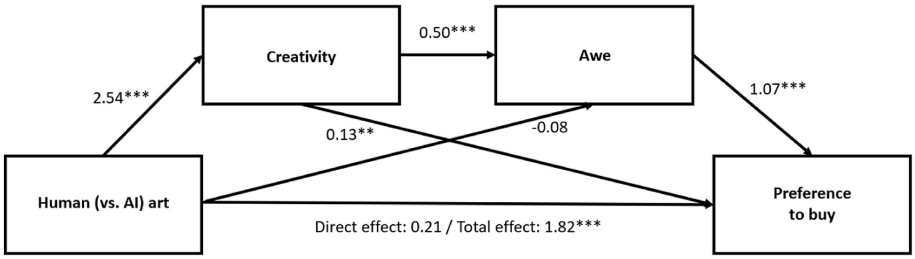


Figure 5: Indirect effect of human (vs. AI) art on preference to buy via perceived creativity and experienced awe.

On the second - tools which attempt to detect GAI are extremely popular (and for obvious reasons). For example GPTZero, a tool which attempts to detect AI generated text, has over 2.5 million users and partnerships with more than 100 organizations [13]. Industry-leading creative software developers Adobe have recently released Content Credentials, which is closer in spirit to what HumanMade is attempting. Both of these tools and more related systems will be further looked into subsequently.

2 Background

2.1 Related systems

We now look to analyse related systems and rate them out of 5 in three different categories - claimed capabilities, accuracy, and ease of use. This will give insight into what the best way to approach the development of HumanMade could be. In general, findings re-iterated the demand for a platform which does more than just attempt direct detection of AI. There were no holistic products with the aim of promoting human creativity.

System	Capabilities	Accuracy	Ease of Use
Adobe Content Credentials	3	2	5

Content credentials [14] is advertised as, first and foremost, a way for creators to attach credit and usage details to their work, and second as a way to be transparent with AI generation. At the former it succeeds, however with the latter, content credentials only indicates the use of generation with regard to Adobe Firefly and other proprietary apps. This does not solve the issue of verifying any piece of AI generated content as specifically human made. Accuracy is poor as the system can easily be cheated - someone could screenshot a piece of digital work and then export it themselves. Ease of use is high as the feature is built right into industry leading apps used for content creation.
GPTZero is a web interface that detects whether your pasted in text is AI generated or not. From testing by pasting in GPT-4 (the current state of the art for chatbots according to

GPTZero	1	4	5

the widely used Huggingface arena leaderboard [15]) generated text, GPTZero performs fairly accurately, predicting 6/7 samples overwhelmingly correctly. GPTZero is a paid service beyond 7 free scans, and does not support any other types of content. Other systems tested perform at or worse than GPTZero.

Sensity	4	4	5

Sensity is a leader in deepfake detection and has a high accuracy according to a recent study [16]. It has API, SDK and UI offerings.

2.2 Issues with Direct Detection

Despite accuracy scores for related systems initially looking positive, research shored up bigger picture concerns when considering a direct detection approach. GAI systems are improving at such a rapid pace that it will likely be impossible to keep up with them.

Sensity mentions on their site, 'As AI technology advances, new and more sophisticated techniques for generating realistic images emerge. Keeping up with these developments requires constant innovation and vigilance.' [17]. Sensity uses mainly ML techniques to identify GAI content [17], and so of course the detection model is only as up-to-date as the data used to train it. When a cutting edge GAI model like Sora drops, content remains undetectable until detection techniques catch up again (assuming it is even possible for them to do so).

Further, during the course of this third year project, Humbot was released [18]. Humbot 'humanizes' AI text to avoid detection by tools like GPTZero and Turnitin. Upon initial testing (7), Humbot works very well, and there are many examples and reviews of functionality on their site.

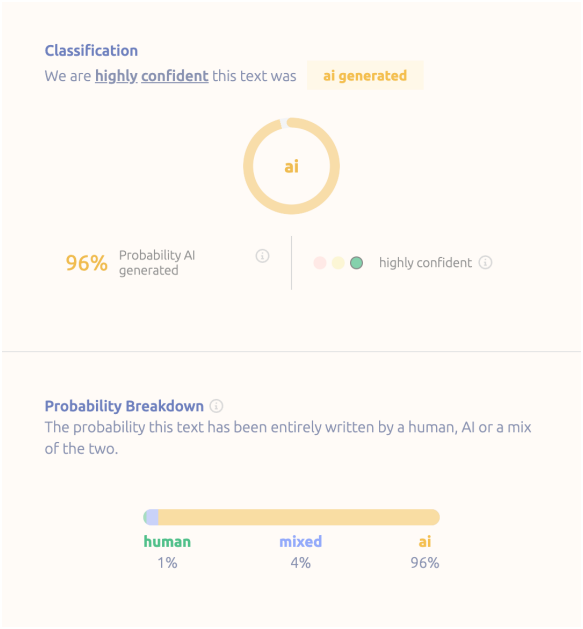


Figure 6: GPTZero classification before using Humbot

Figure 7: Classification after using Humbot

3 Objectives

After adequate research and analysis of the industry, the core objectives for HumanMade were then:

- Creation of an intuitive and modern web app, allowing users to view and support human-made creations
- An easy way for creators to document their project and creative process to the app
- A way to ensure uploaded content is tamper-proof and traceable to a the author

4 Design

4.1 Initial Design Ideas

Initially, to circumvent the extensive issues with direct detection detailed previously, but to still allow for seamless and automatic verification, HumanMade was to work on an 'evidence collection' basis. Evidence relating to the creative process itself would be collected, with the hope that this would give enough of a holistic input to whatever (likely ML) model was to be used to decide whether a creation was human or AI generated. Upon starting the project, there were some large caveats with this approach:

4.1.1 Issues

- Data collection for training would be long & an unproductive use of time. There were no existing datasets for my purpose.
- Data collected would be messy, with few consistent patterns. 'Evidence' could have included screen recordings, screenshots, timelapses, etc. There would not have been many consistent underlying features or patterns to learn to make a fully automated ML approach work well.
- Not actually avoiding the fundamental issues stemming from direct detection. At some point, GAI technologies would probably get so good that they would be able to generate the evidence being used to identify human made creations. This leads to a further philosophical point detailed later, but it was clear a more holistic approach would be required.

4.2 Final Design

The final design relies on some partially automated verification to give users an easier time identifying what might be human made, but the main approach revolves around the key concept of a progression timeline.

- Each progression timeline consists of 'commits', akin to a git branch.
- Each creator-defined commit consists of files, a description, a percentage of completion, and an icon to denote whether AI was used.
- There is a simple interface to make any commit tamper-proof and traceable.
- There is a command line interface for easy publishing to the progress timeline for creatives.

4.2.1 Advantages

This approach had some inherent advantages, addressing the systemic issues we see with direct detection.

- Generative AI has a very obvious 'diffusion' progression. An image of pure noise is slowly transformed into the final output. This would be easy to spot on a progress timeline.
- The approach allows the community to decide the level of AI involvement in a creation they are comfortable with, as this is clearly a gray area. Certain AI tools which simply help a human creator be more effective in creating may not be inherently negative.
- The approach sidesteps the 'arms race' between AI detectors and AI generators by allowing the community to decide for themselves what they think constitutes a human made creation. At a philosophical level, the best 'neural network' one can use to achieve the aims of human made is the human brain.

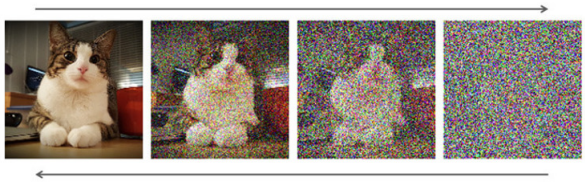


Figure 8: An example of the diffusion progression when generating a photo of a cat [19].

5 Functionality

There are then three main areas of development:

1. **Web App**
   This involves giving users the ability to make new creations, commit to and view their progress timeline for each creation, and a marketplace for finished creations.
2. **Blockchain Interface**
   This involves creating a simple interface to make any commit tamper-proof and traceable by allowing users to add data from their commits to the blockchain.
3. **Command Line Interface**
   This involves the creation of a command line tool that allows users to easily add commits to a progress timeline.

We will look at the completed functionality of each before delving into implementation details.

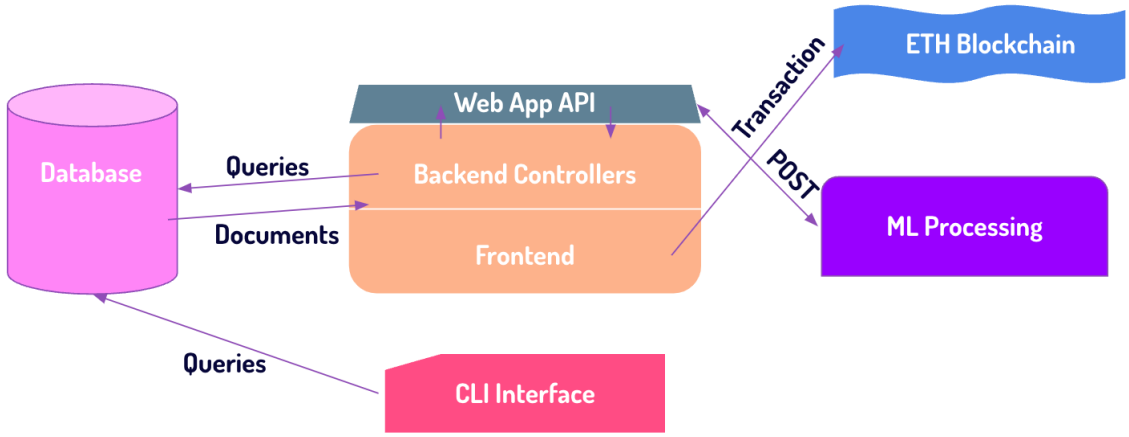


Figure 9: System architecture for HumanMade. [7]

6.2 Web App

6.2.1 Tech Stack

After researching the many available web frameworks, middlewares and backend frameworks, I settled on a modern and scalable tech stack for the web app which would help me develop a reactive and robust app whilst writing clean code.

- **SvelteKit** is a full-stack web framework which eliminates the need for a secondary middleware framework and allows for a simple and modern approach to building fast, reactive web apps. SvelteKit features server-side rendering, hydration, compilation minimisation, and plenty of other modern features to ensure an industry-level product. The framework enforces clean code and object-oriented programming practices.

- **Tailwind** is an open source CSS framework. The UI is built by directly applying predefined styling classes to HTML elements. This allows for quick, consistent, and extensible styling, and works well with the component based methodology of SvelteKit. Tailwind has enamored developers for its customisability, configurability, and responsiveness.

- **Firebase** is a platform owned by Google offering a variety of tools and services to help developers build, improve and grow their apps. We will make use of the **Cloud Firestore** offering from Firebase as the primary database for the web app. Firestore is a NoSQL database that allows for fast querying, dynamic data storage and automatic scalability. Security and analytics are built into the product, and there is a generous free tier to facilitate testing and initial deployment of the project. Firestore also has an excellent modular web API that will play well with a JavaScript frontend. Additionally, Firebase offers **Cloud Storage**, a service for general blob storage of images, audio and video. This will be used for storing files associated with a commit. Again, security and scalability

are integral. Finally, Firebase offers an authentication service which allows for the easy creation and maintenance of user accounts with industry-leading security standards.

6.2.2 General Structure

Roughly, a SvelteKit project consists of:

- a `src/` directory. This contains all the application source code.
- `src/routes` contains all the application's pages and endpoints. SvelteKit uses a filesystem-based routing system, so the folder `src/routes/marketplace` will contain the code for the page route `/marketplace`
- In each route folder, eg. `src/routes/marketplace`, there is a `+page.svelte`, `+layout.svelte`, and potentially a `+page.ts` or `+page.server.ts`.

  `+page.svelte` contains the HTML and any frontend JavaScript for the webpage. `+layout.svelte` contains HTML that can be used to apply a general layout to any pages in the same or nested folders.

  `+page.ts` and `+page.server.ts` are like the 'backend' for a page, and contain load functions (eg. to load data from a database) and form actions (functions which handle POST requests from a form element). There can also be a `+layout.server.ts` which can contain a general load function which applies to any pages in the same or nested folders.
- `src/lib` contains model code reused across the app. Components, utility functions, Svelte stores and type declarations all go here.
- `src/routes/api` is where SvelteKit has you write any POST/GET API endpoints for your application. For example, adding a new folder named `xyz`, containing a file named `+server.ts` with a POST function, creates a new `xyz` endpoint that handles POST requests.

6.2.3 UI/UX

Components: SvelteKit is a component based web framework. Components are custom UI elements you create out of standard HTML elements. This allows for consistent styling, functionality and reusability across the app. Each component consists of:

- a `script` section containing exported component prompts and any interactive functionality
- the HTML for the component

For example, this excerpt from a `Button.svelte` component consists of click, size, submit and icon props.

```
<script lang="ts">
    export let click = () => {};
    export let size = 'md';
    export let submit = true;
    export let icon = ''
</script>
```

and these props are passed to an HTML `<button>` element

```
<button
    on:click={click}
    type={submit ? 'submit' : 'button'}
    ...
```

These props can be given values when the component is actually used in a parent webpage.

```
<Button size="md" icon="fa-brush">Create</Button>
```

Styling: As aforementioned, instead of styling directly with CSS, I used the more modern approach of going with a CSS framework, specifically Tailwind. Tailwind consists of many modular, pre-defined classes which let you style elements rapidly and inside the HTML. This div is styled to be a flexbox, with a specified width and rounded corners.

```
<div class="flex flex-col gap-2 w-6/12 rounded-2xl bg-opacity-80 bg-
   primary p-5">
```

For example, the `flex` class is defined by Tailwind as:

```
.flex {
    display: flex;
}
```

Notice the background is set to be `bg-primary`. Thanks to Tailwind's configurability, we can specify a color palette, font family and font sizes in a `tailwind.config.js`, and Tailwind will automatically incorporate our choices into the generated CSS class options.

```
colors:{
    'primary': '#2E4052',
    'secondary': '#FFC857',
    'tertiary': '#BDD9BF',
    'quaternary': '#412234'
},
```

Icons: Iconography is important for aesthetics and accessibility. Font Awesome is a library that provides high-quality minimalist SVG icons for websites. It exposes these assets via CSS classes, meaning just as with Tailwind we can reference icons directly in the markup of our website, providing consistency and cleanliness across the codebase.

6.2.4 Authentication

Firebase has simple methods for creating and authenticating users with an email and password combination. The authentication flow is as follows:

1. In the event that either method is successful, a result object is returned with a user property containing the user ID, display name and email.
2. We extract and save these into a custom User type object, (and write a user document to the Firestore database if this is a new user), and then set a 'user' cookie value to this object (serialised).
3. We then have a `+layout.server.ts` at the root of `src/routes` containing a load function which returns the user object from the cookie.
4. This load function applies to all webpages, safely and efficiently exposing the user object to all pages which require it.

Using Firebase authentication with Firestore makes the app seamlessly secure. We can write custom security rules in Firestore to restrict user access to certain documents in the database. For example, here we ensure that a creation can only be edited by the original owner of the creation:

```
match /creations/{document=**} {
  allow write: if request.auth.uid == resource.data.uid;
}
```

Firebase authentication handles attaching an auth token to any database requests the user makes, giving Firestore access to the `request.auth` object to use in security rules.

6.2.5 Reactivity & Updates

To ensure a reactive and responsive experience for users, we develop a way to keep the client and server in sync with a clean and elegant approach. We take advantage of Svelte reactive stores and Firestore realtime listeners.

Svelte Stores: These are helpful for managing global state across components. The type of store we use is `writable`, where we store arrays either of type `Creation` or `Product`, for the creation and marketplace tabs respectively. We can read and write values to these arrays, via the `update` and `set` methods of the store, and more importantly we can now subscribe to the store. Then, any components relying on the store values are automatically re-rendered by SvelteKit upon updates to the store. For example, this `each` loop will re-render any of the necessary `CreationDiv` components upon any updates to the `creations` store.

```
{#each $creations as creation (creation.id)}
    <CreationDiv {creation}></CreationDiv>
{/each}
```

Realtime Listeners: These simply subscribe to a collection of documents in the Firestore database, triggering upon any changes to the collection. Firebase returns a collection of `DocumentChanges`, which supplies the document which was updated and the type of update (added, modified or removed).

Using these two concepts, we can map changes we see from the listener to changes in the store, allowing us to efficiently update UI every time the database state changes. To do this mapping cleanly, we implement a `Listener` wrapper class for each store, which gives us methods to easily update the store from listener document updates.

```
abstract class Listener<T>{
    abstract update(struct: T): void;
    abstract remove(struct: T): void;
    abstract add(struct: T): void;
    abstract docToType(doc: DocumentSnapshot): T;
}
```

The diagram below illustrates the approach at a high level.

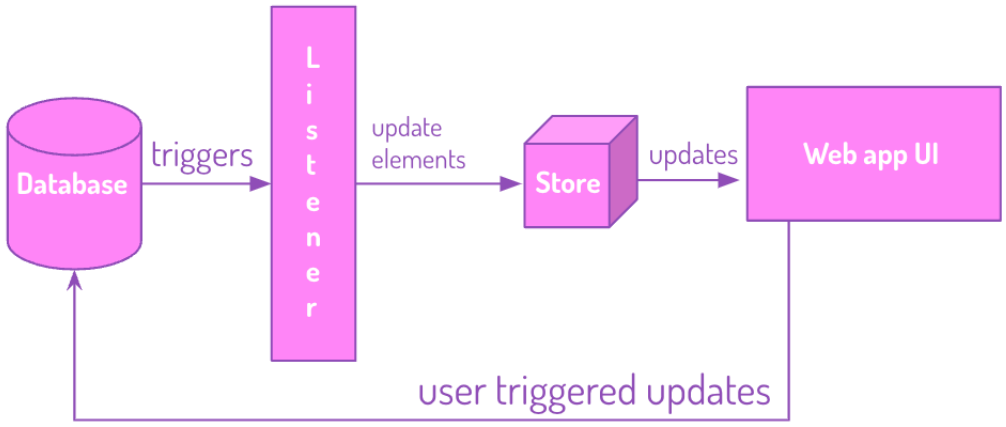


Figure 10: Reactivity system for HumanMade[7]

6.3 Machine Learning Microservice

6.3.1 Tech Stack

1. **TensorFlow** is a free and open-source machine learning library from Google, providing state of the art models and APIs in Python.

2. **Flask** is a micro web framework written in Python, perfect for setting up a simple API for the microservice with which the main web app can interoperate.

This microservice handles computation of the similarity score between images of the same tag across commits, as described in the functionality section.

6.3.2 Firebase Storage

Any images uploaded in a commit are stored in cloud storage, with a `+` separated filename consisting of the image `tag`, `file.type` and the current date to avoid the chance of duplicate filenames.

6.3.3 Data Flow

The data flow when interacting with the ML processing microservice is as follows, and is also detailed in the below diagram.

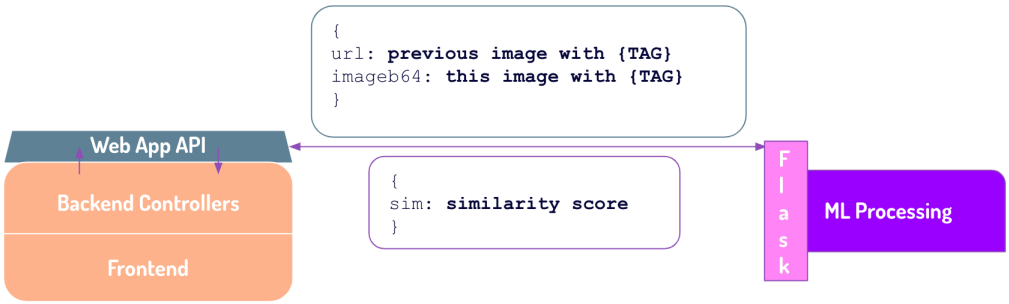


Figure 11: Data flow diagram for the ML microservice

Upon a new commit to the progress timeline containing an image tagged `{tag}`, and given there is an image with tag `{tag}` in the previous commit:

1. The base 64 encoding of the image in the current commit, and the URL of the image in the previous commit are posted in JSON to the `imageSimilarity` API endpoint.
2. The microservice processes the images (detailed in the next section), and return back a similarity score to the web app API's request.
3. The endpoint returns the similarity score to the frontend, and this is displayed on the progress timeline if below a certain threshold.

6.3.4 Similarity Computation

ResNet50, a 50 layer Convolutional Neural Network provided by the Tensorflow library, is used to extract a high level feature vector from each of the images. These two vectors are then compared using cosine similarity.

Firstly, we preprocess the images:

- Images are received by Flask in a POST method. The images are downloaded (as required) and written to `.jpg` files.
- Images are size adjusted (ResNet required images to be in `224x224` resolution) and converted to a 3D array (width by height by color channels, (224,224,3)). ResNet actually takes input in 4D, where the first dimension represents batch size, so we need to expand our array to dimensions (1,224,224,3).
- Finally the TensorFlow provided `preprocess_input()` function scales and normalises pixel values of the image.

Then, we can extract a feature vector for each image: ResNet50 is a classification model, with its final layer consisting of 1,000 different classes. We do not want to classify our images - so, we stop the model at an intermediate layer, giving us a high level feature vector describing the images. We can do this with the provided `Model` python class, which lets us specify a base model and the output layer. ResNet50 consists of pooling, convolutional and ReLU activation layers.

We choose a late convolutional layer to stop at. These layers are where kernels/filters are applied to the image to better detect high level abstract features [20], which is what we want when comparing similarity. The general shape, color and vibe of an image should not change drastically commit to commit as a user is working on a project, whereas specific pixel values and details could and should be allowed to.

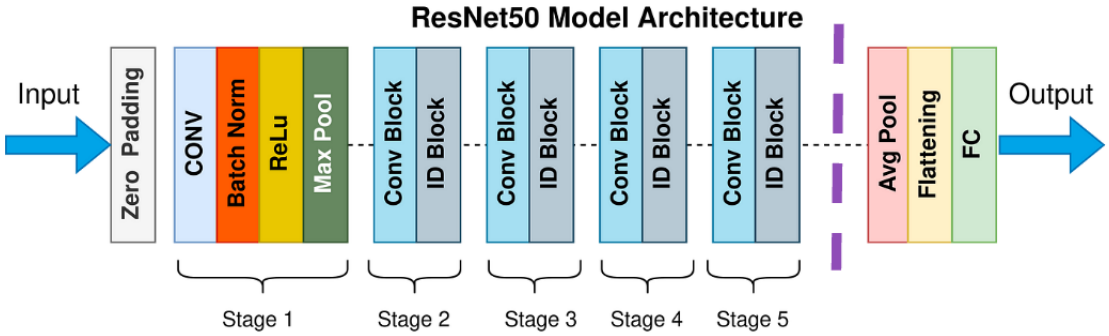


Figure 12: We look to cut off ResNet's output early.

Finally, we used cosine similarity (`1-cosine(vec1, vec2)`) to compare the feature vectors. At a high level we can think of this as measuring the 'direction' of the vectors in `n`-dimensional space and seeing how closely they align.

6.4 Blockchain Interface

6.4.1 Tech Stack

- **MetaMask** is a leading self-custodial cryptocurrency wallet, used to interact with the Ethereum blockchain. Users can access their wallet and perform transactions with ease, and without having to send their private key across a network. The MetaMask Chrome extension provides a modern way for web apps to integrate with the blockchain.
- **Solidity** is the defacto programming language for implementing smart contracts, mainly on the Ethereum blockchain. A smart contract can be thought of as a self-executing contract that a user can interact with by sending crypto to.
- **Remix** is an open-source IDE widely used for smart contract development. We use Remix to compile, deploy and test our smart contract.
- **Web3.js** is a collection of open-source libraries that primarily provides an abstracted interface for communication with an Ethereum node, as well as other helpful utility functions. This is the layer we use to interface with the Blockchain from our frontend JavaScript.

6.4.2 Smart Contract Development

The code for the smart contract is fairly short, consisting of a `makeCommit(string[] fileHashes)` function, which emits an event `CommitMade(msg.sender, hash)` that could be used to trigger actions in other decentralized applications in the future. For the purpose of this application, the `makeCommit` function is of importance, as this is what gets recorded in a transaction on the blockchain when invoked, as will be detailed.

We then use Remix to compile and deploy the contract. The Solidity contract is compiled down to bytecode, and can then be deployed to

1. the Ethereum mainnet
2. a Ethereum testnet
3. a virtual JavaScript network in the browser

We opt for 2, deploying the contract to the Sepolia testnet, a proof-of-stake network designed to mimic the operating environment of the mainnet but existing on a separate blockchain ledger. We can receive free ETH on this testnet via many of the available Sepolia 'faucets'.

This is in fact our only viable option. Firstly, at least until it is required in production, we don't have to pay the (sometimes exorbitant) gas fees required to commit a contract (or conclude any transaction) to the mainnet during testing. Secondly, we want to be able to test our contract with an actual wallet provider (we use MetaMask), which would not be possible if we deployed our contract to the virtual browser network.

Deploying via Remix to the Sepolia testnet, (paying gas fees using a wallet with Sepolia ETH in it), gives us the contract address and contract ABI, which is all the information we need to send a transaction to the contract.

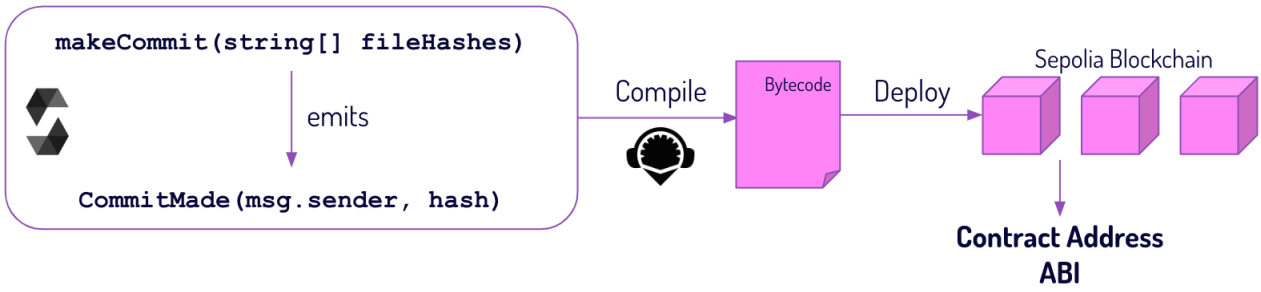


Figure 13: The process for compiling and deploying the Solidity smart contract

6.4.3 Communication Setup

To communicate with the blockchain using the Web3.js interface, we need to pass the `Web3` constructor a 'provider'. A provider is an object which communicates with an Ethereum node, supplying essential properties and methods which adhere to the Ethereum Provider API standard. MetaMask supplies us with this object when installed, injecting a `window.ethereum` object into the browser for web apps to make use of.

```
const web3 = new Web3(window.ethereum);
```

Now that `Web3` is invoked with a valid provider, we can create a Contract object which will let us execute our contract methods on the chain. This is where we need the contract address and ABI:

```
const contract =
    new web3.eth.Contract(JSON.parse(abi), address);
```

Now, we have access to all the methods we need to perform a transaction on the chain.

6.4.4 Performing Transactions

We want the transaction we perform to contain content related to the files in the progress timeline commit, and to the user themselves. This way, each commit is tamper-proof and traceable, with the original content and creator recorded permanently on the blockchain ledger.

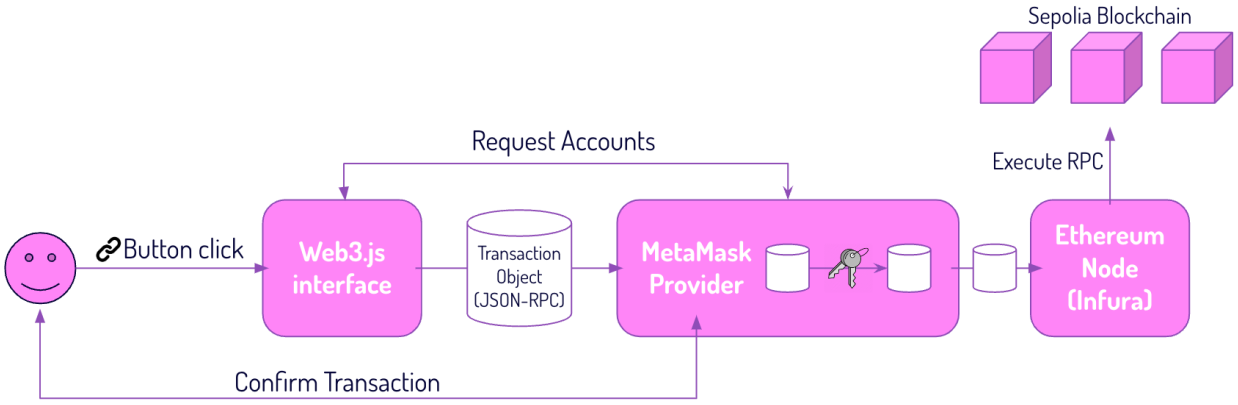


Figure 14: The full process of performing a transaction end-to-end

Upon a user clicking the 'link' icon on a progress timeline commit:

1. We get the contract address and ABI. These are stored on the server for safety, so the backend controller makes a call to a `blockchainCommit` API endpoint to get these details first.
2. We request access to the MetaMask accounts, obtaining the user's wallet address.
3. We create the transaction object - a simple JavaScript object where we pass:
   - the receiver, in this case the contract address
   - the sender, in this case the user's wallet
   - a data string the Ethereum Virtual Machine can interpret as a command, in this case an Application Binary Interface string representing a call to the contract method `makeCommit`. We obtain this using a method on the Contract object we created in the setup phase.
   - the user's transaction count (the nonce, which provides a unique ID for each transaction and ensures transactions are ordered and executed correctly)
   - gas fees and the gas limit. We calculate the fee based on the latest blocks gas fee, and the limit based on the size of the transaction object.

Finally, we perform the transaction,

```
1 const txHash = await web3.eth.sendTransaction(txObject);
```

and save the hash to the database, signifying that the commit has been recorded on the blockchain.

6.5 Command Line Interface

- **Node.js** allows us to write a command line script in Typescript, make network requests and take advantage of helpful libraries to parse command line arguments effectively.

7 Implementation

7.1 Blockchain

7.2 Command Line Interface

7.3 Testing

8 Conclusions

- evulation, process of production, lessons learnt, further work, limitations and contributoins

9 Project Management

10 Philosophy of ito

References

[1] A. Impacts. (2022) 2022 expert survey on progress in ai. [Online]. Available: https://wiki.aiimpacts.org/doku.php?id=ai_timelines:predictions_of_human-level_ai_timelines:ai_timeline_surveys:2022_expert_survey_on_progress_in_ai

[2] K. Millet, F. Buehler, G. Du, and M. D. Kokkoris. (2023) Defending humankind: Anthropocentric bias in the appreciation of ai art. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0747563223000584

[3] D. Milmo. (2023) Chatgpt reaches 100 million users two months after launch. [Online]. Available: https://www.theguardian.com/technology/2023/feb/02/chatgpt-100-million-users-open-ai-fastest-growing-app

[4] (2017) Attention is all you need. [Online]. Available: https://arxiv.org/abs/1706.03762

[5] (2023) Google's cloud tpu v4 provides exaflops-scale ml with industry-leading efficiency. [Online]. Available: https://cloud.google.com/blog/topics/systems/tpu-v4-enables-performance-energy-and-co2e-efficiency-gains

[6] (2022) Fueling high-performance computing with full-stack innovation. [Online]. Available: https://developer.nvidia.com/blog/fueling-high-performance-computing-with-full-stack-innovation/

[7] (2023) Gpt-4 technical report. [Online]. Available: https://arxiv.org/abs/2303.08774

[8] (2023) Eu ai act: first regulation on artificial intelligence. [Online]. Available: https://www.europarl.europa.eu/topics/en/article/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence#:~:text=Unacceptable%20risk%20AI%20systems%20are,encourage%20dangerous%20behaviour%20in%20children

[9] (2023) Pause giant ai experiments: An open letter. [Online]. Available: https://futureoflife.org/open-letter/pause-giant-ai-experiments/

[10] (2023) Generative ai. [Online]. Available: https://dealroom.co/guides/generative-ai

[11] (2020) Scaling laws for neural language models. [Online]. Available: https://arxiv.org/abs/2001.08361

[12] (2024) Introducing the next generation of claude. [Online]. Available: https://www.anthropic.com/news/claude-3-family

[13] M. than an AI detector Preserve what's human. (2023) Gptzero. [Online]. Available: https://gptzero.me/

[14] Adobe. (2023) Content credentials for assets generated with adobe firefly. [Online]. Available: https://helpx.adobe.com/firefly

[15] (2023) Lmsys chatbot arena leaderboard. [Online]. Available: https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard

[16] (2023) An investigation of the effectiveness of deepfake models and tools. [Online]. Available: https://www.mdpi.com/2224-2708/12/4/61

[17] (2023) How to detect ai generated images with sensity. [Online]. Available: https://sensity.ai/blog/deepfake-detection/how-to-detect-ai-generated-im/#:~:text=Extensive%20Training%20Data%3A%20Sensity%20AI,generation%2C%20ensuring%20high%20detection%20accuracy.

[18] (2024) Humanize ai text and get 100

[19] (2024) What is stable diffusion and how does it work? [Online]. Available: https://www.vegait.co.uk/media-center/knowledge-base/what-is-stable-diffusion-and-how-does-it-work

[20] (2012) Imagenet classification with deep convolutional neural networks. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf