

AIgIigator: a Multi-Model Tool for Detecting AI-Generated and Manipulated Images

Alexander Ellingsen and Matthew Nagel

Gianforte School of Computing, Montana State University

CSCI 483R: Interdisciplinary Project

Dr. Clemente Izurieta

May 5, 2026

ABSTRACT

Digital media, especially images, are remarkably easy to manipulate. Due to recent progress in generative artificial intelligence, the generation or alteration of photorealistic images is becoming increasingly trivial and widespread. Thus, authenticity can be difficult to verify, which makes image manipulation a useful tool for spreading disinformation. In response to this growing threat, open-source detection tools that attempt to identify either manipulated or artificially generated images (AIGI) have been developed, though no single model offers coverage of a range of manipulation techniques. This capstone project presents a web platform that uses a variety of AIGI detection models and forensic image manipulation detection frameworks intended to combat disinformation. By using models that differ in their methods for detecting AIGI or image manipulation, we ensure coverage of a number of techniques. A user can upload a suspicious image to our platform via a web app, which is then analyzed by the selected set of models. The model's outputs are aggregated into a custom report that provides a likelihood score of whether the image was created by artificial intelligence or altered otherwise. This, as well as any other generated visuals, explanations, or metrics required to support that conclusion, is displayed to the user via the web interface and is available for export if desired. This project shows how available models can be used in a more thorough, accessible means for analyzing image authenticity, with potential further applications in disinformation prevention and digital forensics.

INTRODUCTION

While the authenticity of images has long been a concern in media, journalism, and legal scenarios, the development of generative AI and image manipulation techniques in more recent years has changed the threat landscape considerably. With models like Stable Diffusion and DALL-E available for public use, the barrier to creating images that are both photorealistic and synthetic has been greatly lowered. Today, convincing fabricated images can be produced in seconds by someone with no technical background, or existing ones can be easily modified. Even to wary observers, such photos are becoming increasingly indistinguishable from authentic photographs; in a May 2025 study, 12,500 global participants showed only a 62% overall success rate when distinguishing real images from AI-generated ones (Roca et al., 2025).

This change has led to considerable consequences for spreading disinformation. Manipulated and artificially generated images can be used to misrepresent events and even manufacture false evidence in political and social contexts. In March of 2023, for example, a number of AI-generated images of Pope Francis in a white puffer jacket went viral, with many online believing the images were real (Ellery, 2023). As the barrier to creating and modifying convincing synthetic images lowers, the potential for misusing it has continued to grow. The more fake images are created, the more trust in visual media itself is eroded, forming a situation where even authentic images seem suspicious.

In response to this threat, detection research has produced and is currently developing a range of models capable of identifying AI-generated or manipulated images. Although progress is meaningful, a model is frequently designed only around a specific method for detection. A model may be highly effective against the manipulation technique it was developed to identify, but it may also have gaps in accuracy when used against a different technique. For example, a model optimized to detect localized pixel-level anomalies may fail on larger AI-generated landscapes or images created entirely from scratch. No single model provides coverage of a wide range of manipulation and generation techniques, meaning reliance on any one tool can result in a number of unaddressed blind spots.

To address that gap, this project presents a web platform that runs multiple detection models independently and simultaneously, aggregating their outputs into a single authenticity report. This project does not replace available models. Instead, it uses them in combination to ensure that the weakness of any one model can be compensated for by the strengths of others. A user can submit an image through the web UI and receive an exportable report containing an overall confidence score and confidence levels per model. Importantly, the goal of this system is not to deliver a verdict but to better inform human decision-making via multiple different methods of image analysis.

QUALIFICATIONS

Alexander Ellingsen

alexcellingsen@gmail.com | (509) 724-1320 | [LinkedIn](#) | [GitHub](#)

Education

Montana State University Honors College, *Gianforte School of Computing* — Bozeman, Montana
2022 - 2026 (Expected)

Bachelor of Science in Data Science, **Bachelor of Science** in Computer Science

- **Relevant Classwork:** Algorithms and Data Structures, Object-Oriented Programming, Data Analytics, Data Ethics, Data Mining, Database Systems, Data Visualization, Discrete Mathematics, Linear Algebra, Machine Learning, Multivariable Calculus, Statistics

Experience

AI Engineering Intern, Quig

Bozeman, Montana

June 2025 - Present

- Leveraged statistical analysis techniques of customer conversations to create data-driven business insights, providing crucial data for positive contract negotiations.
- Optimized real-time data pipelines to accurately represent content and generate efficient vector embeddings for effective retrieval in RAG-powered AI systems, enabling AI Agents to provide quick and accurate responses.
- Engineered multiple multi-step Agentic AI systems, including a first-of-its-kind agent designed to handle long-context and irregularly structured text inputs, expanding the business's offerings and creating valuable new product features.

Undergraduate Researcher, Montana State University

Bozeman, Montana

June 2024 - Present

- Collaborated with multiple professors, faculty, and PhD students across departments and universities to generate insights into diverse research questions, including gathering opinions on the use of artificial intelligence in Education and fine-tuning LLMs to explain radiograph reports simply and accurately to medical patients.
- Wrote multiple funding proposals to secure grants individually valued at over \$1,000, allowing the research team to be properly compensated for their work.
- Presented research findings via poster sessions at two academic conferences, highlighting the research question, methodology, and key results.
- Co-authoring an academic journal paper for submission to a peer-reviewed journal to disseminate findings and present them to the education and computer science communities.

Undergraduate Course Assistant, Montana State University

Bozeman, Montana

April 2023 - Present

- Assisted in teaching more than 150 students across three introductory Python courses, including a specialized section focused on introducing the field of Data Science.
- Provided ongoing support and guidance by answering questions related to programming, statistics, machine learning, and assignments during assigned office hours and lectures.
- Gave multiple lectures and presentations to introduce basic and advanced ideas in the subfields of Data Science, such as data visualization and NLP techniques, engaging students while ensuring preparation for further classes.
- Designed, developed, and graded homework assignments to reinforce core concepts like visualization, statistics, and prediction using regression and classification, supporting student success.

Selected Projects and Research

Teaching in the Age of GenAI: Social Media Insights on Opportunities and Challenges *(Python, SQL)*

- Developed scalable Python and SQL data pipelines to collect and process over 1,900,000 social media posts gathered via an API-powered keyword search and filtered irrelevant content using custom lemma-based text filtering algorithms.
- Fine-tuned a BERT model using QLoRA and manually tagged posts for text classification to determine whether a post was education-related, achieving a 0.94 F1 score and 0.212 Evaluation Loss.
- Compared transformer-based (RoBERTa) and lexicon and rule-based (VADER) sentiment analyzers to assess transformer-based models' superior handling of domain-specific language, improving the accuracy of sentiment representation.
- Designed multiple visualizations to track how sentiment towards GenAI shifted over time, and used numerous dimensionality-reduction techniques such as PCA, T-SNE, and UMAP on TF-IDF-transformed data to gain insights into the relationships found in the data.

Matthew Nagel

Chugiak, AK 99567

matthew.r.nagel@outlook.com

+19078547159

Current undergraduate student studying Computer Science and Business Administration at Montana State University.

Work Experience

ResNet User Services Associate

Montana State University-Bozeman, MT

August 2024 to Present

ResNet is the Internet Service Provider for the residence halls and family/graduate housing at Montana State University that provides technical support for students who live on campus. Work included:

- Providing technical support and troubleshooting at the help desk or on site in the campus ticket system
- Troubleshooting computers infected with spyware, adware, and/or viruses.
- Troubleshooting/resolving network hardware/software issues
- Diagnosing and repairing/replacing failed hardware and/or providing appropriate recommendations for repair/replacement.
- Connecting wireless devices to the campus wireless network.
- Answering questions about ResNet, McAfee, Microsoft Office, Adobe, network printing, hardware, etc.
- Configuration of computer operating systems, software installs, and hardware installs.
- Utilizing anti-virus and anti-malware software to disinfect student/staff devices

DevOps Intern

Northrim Bank-Anchorage, AK

May 2025 to August 2025

Worked in Northrim Bank's DevOps department to develop internal web applications that enhanced bank functions. Work included:

- Helped develop and maintain Angular-based frontend components within an ASP.NET framework to enhance user experience and responsiveness of internal banking tools
- Participated in code reviews and debugging sessions to abide by best practices and improve code quality and maintainability
- Documented frontend workflows and development environment setup to support team onboarding
- Participated in usability testing to identify and fix bugs before reaching production

Bank Intern

Northrim Bank-Anchorage, AK

June 2024 to August 2024

- Assisted Northrim spreads department with processing and analysis of documents including personal/ business tax returns, personal/business financial statements, debt schedules, CPA audits/reviews, credit reports, credit memos, annual term loan reviews, etc.
- Gain knowledge and experience in commercial banking and credit analysis with a focus on commercial lending.

Education

Bachelor of Science in Computer Science, Minor in Business Administration (Current Undergraduate)

Montana State University-Bozeman, MT | August 2023 to December 2026 (expected)

BACKGROUND

Artificial Intelligence-generated images have rapidly progressed in quality and quantity since the advent of generative adversarial networks in 2014. At the start of this boom, these images could be easily categorized as AIGIs. Obvious stylistic differences from real photos, poor quality, and errors such as creating hands with extra fingers or illogical additions to everyday objects would immediately warn the reader that it was a fake image. Companies such as OpenAI, Midjourney AI, and Google have all released models since 2014, consistently improving their realism. These advancements have made it harder to distinguish AIGIs from real, unaltered images, with recent models capable of creating nearly indistinguishable ones. With the ability to create nearly photorealistic images from a few simple lines of text, malicious actors can spread misinformation easily. The rise of AIGIs has created alarm in the public, which in turn has led to the creation of AIGI detection methods.

Before AI image generation models became mainstream, researchers had developed a plethora of frameworks that could detect local image manipulation, such as the insertion of foreign elements into a legitimate photograph. While these methods are still very helpful in their own right, they struggle or fail to detect when the entire image is inauthentic. AIGI detection methods differ from these traditional image manipulation detection frameworks as they are specifically designed to determine if an entire image has been created by artificial intelligence, not just specific elements. They primarily rely on machine learning techniques, often leveraging similar tools to image creation models to differentiate between real and AI-generated images. Although the field is very active and constantly improving its models to detect the newest state-of-the-art AIGI models, they often struggle with specific methods of generation, especially if the method is proprietary. Our platform, AIGigator, attempts to ease this inaccuracy by leveraging multiple models, allowing for an ensemble of multiple specialized models to be used at the same time.

Two important elements of our platform require background knowledge to understand properly. The first is our AIGI detection and forensic image analysis suite, which uses three models developed by academic researchers. Of the three models, one is an image forgery detection framework called TruFor, developed by Guillaro et al. (2023). TruFor scans the image for deviations that would not be present in an unaltered image using a method called Noiseprint++. Images have traces of their origin that are invisible to the naked eye, and any alterations to the image will remove or alter these traces. With Noiseprint++, the TruFor model can detect these and flag the image as manipulated, even creating a localized heatmap showing where the manipulation occurred. The other two models we use are specifically designed for AIGI detection. B-Free (Guillaro et al., 2025) is a fine-tuned Vision Transformer Model (ViT) trained on real images that were manipulated by AI. Vision Transformer Models utilize the transformer architecture commonly used in large language models, essentially transforming groups of pixels into “words”. By using this specific type of image on a transformer model, it can “learn” the artifacts that AI manipulation leaves behind and detect them. Dual Data

Alignment (Chen et al., 2025) leverages 3 steps to detect AIGI: Variational Autoencoder (VAE) reconstruction, frequency-level alignment, and pixel-level alignment. VAE reconstruction generates an image similar to the original that contains artifacts only found in VAE images. Then, the real and VAE images are compressed to mitigate issues stemming from JPEG compression. Finally, pixels from both images are mixed up and sampled to detect similarities between the two, therefore showing whether the image is AI-generated.

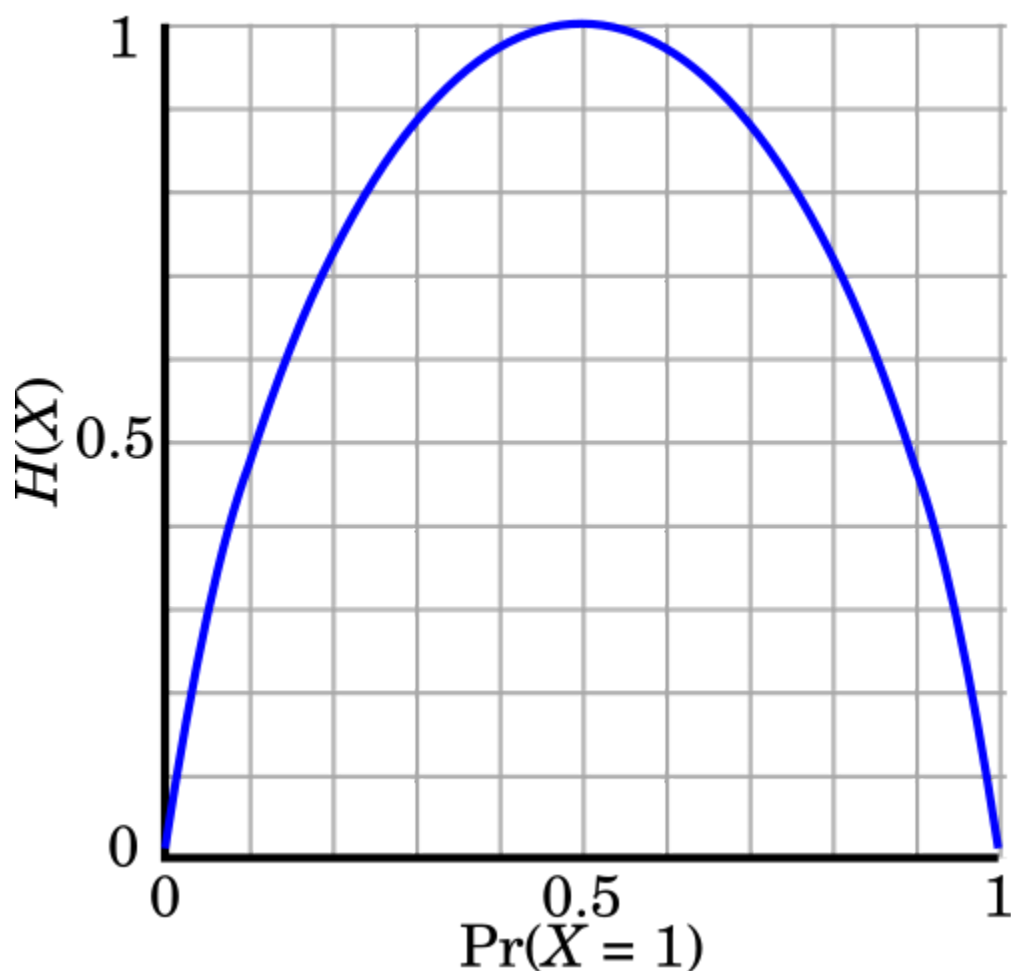


Figure 1. Binary entropy function curve.

The other element is the binary entropy function, a concept taken from information theory. All of the models leveraged in our platform output the probability of an image being fake, a statistic that lends itself very nicely to a Bernoulli distribution. Using this function, we are able to find the entropy, or level of uncertainty, of each of our models' outputs:

$$H(X) = -p \log_2 p - (1 - p) \log_2 (1 - p)$$

As seen in Figure 1, the maximum value for $H(X)$, or the maximum entropy value, occurs when the probability is 0.5. By subtracting this value from 1, we adjusted this entropy value to represent the confidence of each of our models, which can be further expanded to the confidence of our system as a whole. This function allows us to provide the individual model's probability, as well as an intuitive and easy-to-understand “group confidence score”, backed by mathematical and statistical theory.

Our work does not attempt to compete directly with any one model. Instead, we attempt to create a platform in which these models can work together to determine whether an image was generated using artificial intelligence or is authentic and unaltered. Due to the speed at which AI image generation models are advancing, frontier detection models change rapidly and vary in what they are considered state-of-the-art in. Therefore, our platform uses these models in an ensemble and provides a simple, mathematically-backed method to calculate their confidence as a group. Since the user is able to add or switch which models are being used, the platform can constantly update as the best detection models change, allowing it to be used consistently.

WORK SCHEDULE

Responsibilities

Matthew Nagel: Frontend Development

- Angular UI
- File upload functionality
- Report viewer design
- Binary entropy chart
- Navbar and aggregate score display
- FastAPI REST API integration
- Polling for image analysis status

Alex Ellingsen: Backend Development / ML Model Engineering

- Docker containerization
- Backend coordinator
- Report aggregator
- ML Model integration
- Aggregated confidence algorithm
- Probability scoring
- Heatmap visualizer

Lifecycle

We implemented the Agile lifecycle for the development of this project. That is, we built our software incrementally over six iterations, or “sprints” so we could flexibly adapt to unforeseen changes during development if necessary. These sprints occurred over two-week periods and were supplemented with weekly in-person meetings to track progress towards sprint goals and address any issues as they arose. This lifecycle was particularly valuable when integrating detection models incrementally over successive sprints, revising our aggregate confidence calculations, and refining how the user interface presents model outputs to the user as our backend architecture evolved.

Project Milestones

Our milestones for this project were as follows:

1. **Foundation:** Angular frontend and basic UI scaffolded, FastAPI connected
2. **Core Pipeline:** Model containerization, coordinator polling cycle, and report aggregator integrated
3. **First Working Analysis:** TruFor detection model integrated, basic model output rendered to user interface; the first real result is produced by the system
4. **Full Model Suite Integrated:** BFree and DDA added, confidence algorithm introduced, report aggregator and all detection models operational.
5. **Report Visualization:** Heatmap visualizer added; heatmap overlay, binary entropy chart, and aggregate score displayed in user interface report viewer.
6. **Final Adjustments:** PDF export and scoring table, confidence calculation refinements, minor UI adjustments, MVP complete

These milestones aligned closely with our sprints, as shown in the timeline on the following page.

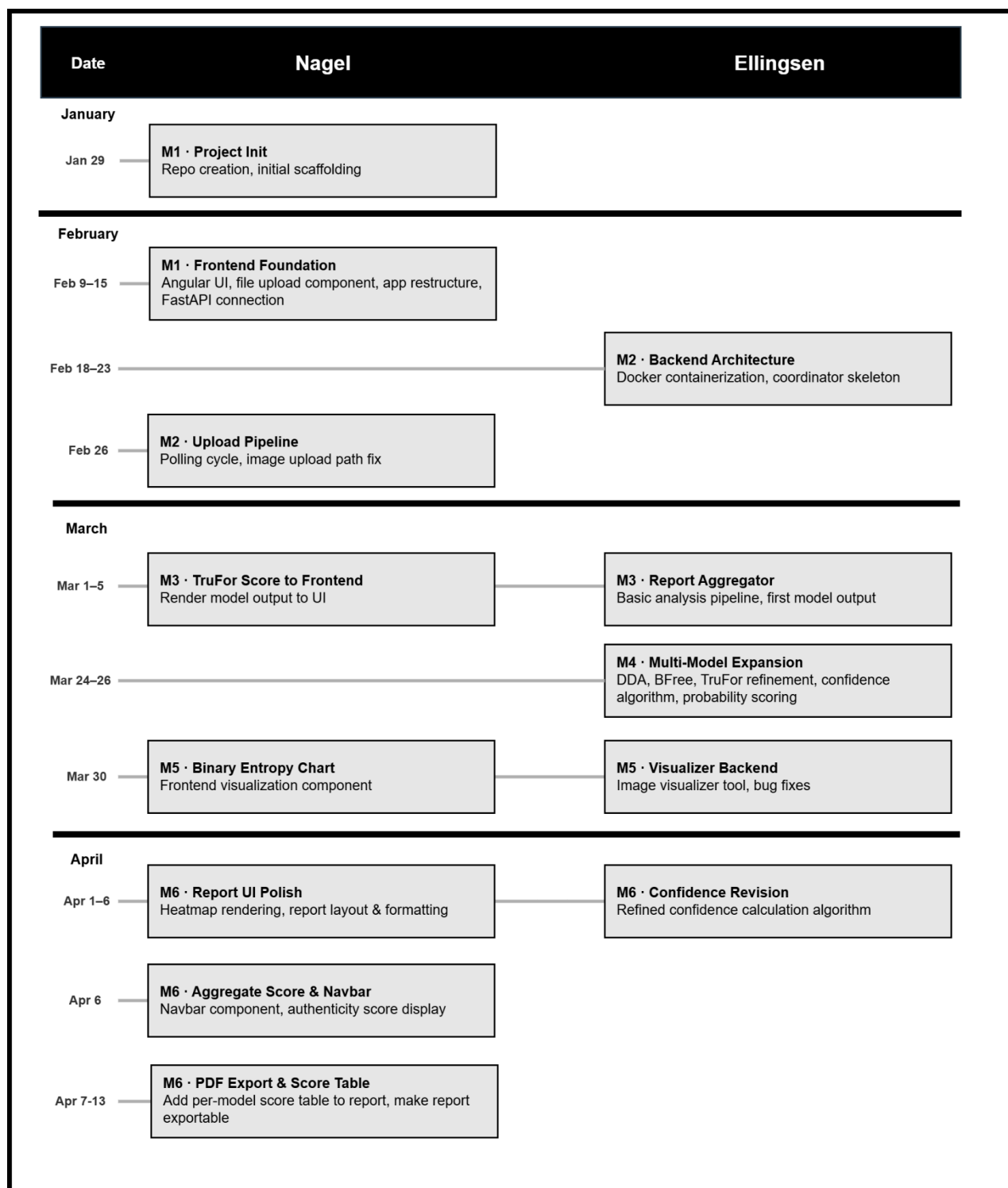


Figure 2. AIGigator system development timeline.

PROJECT TECHNICAL INFORMATION

Repository:

GitHub: <https://github.com/mrnagel/AIGigator>

Functional Requirements:

- The system shall accept uploaded images (JPEG/PNG $\leq 5\text{MB}$) from a web browser
- The system shall run at least three machine learning models against each uploaded image
- The system shall return a per-model probability score (p_{fake} , p_{real}) for each analysis
- The system shall display a binary entropy chart mapping each model's prediction onto a curve
- The system shall compute an aggregated confidence score using the binary entropy function
- The system shall expose a job-based API so the frontend can poll for async analysis status
- The system shall isolate each ML model in its own Docker container
- The system shall provide a PDF containing an image's generated report upon user request

Nonfunctional Requirements:

- Availability: The backend's API must remain responsive to poll requests during model execution
- Scalability: Each model must run as an independent, containerized service
- Maintainability: Models must be integrated as containerized black boxes. The system depends only on the agreed-upon .npz output, not on any model's own implementation or dependencies.
- Usability: The user interface must provide real-time status badges and animations to the user during analysis
- Portability: Docker containers must encapsulate all dependencies for each model so deployment is possible in different environments
- Reliability: In the event of analysis pipeline failure, the user must be informed via an error message rendered to the user interface.

Performance Requirements:

- The job system must use background threads so polling requests can be serviced simultaneously by the API
- The frontend must poll job status every 2 seconds to balance responsiveness with server load

- All machine learning model containers must be launched simultaneously to minimize analysis time
- Image uploads must be capped at 5MB to prevent unbounded memory usage during analysis
- Previous image uploads must be cleared when a new image is uploaded
- The heatmap visualizer container (when applicable) must run only after the machine learning model containers have completed to ensure the necessary data exists at runtime

Interface Requirements:

User-Facing Interface:

- The web UI shall provide a file selector that accepts .jpg and .png files
- The UI shall display upload/analysis progress, job status, and final results without a page reload
- The report viewer shall display prediction labels per model, a binary entropy chart, and an overall confidence score
- The chart shall display human readable confidence percentages and fake/real probability scores per model

API Interface (Backend/Frontend):

Endpoint	Method	Purpose
/upload-image/	POST	Accept image from client and return {jobID, status}
/job-status/{jobId}	GET	Return job status and JSON result when complete
/heatmap	GET	Return heatmap PNG (when applicable)
/original-image	GET	Return upload image file (for heatmap overlay)

Backend/Model Interface:

- Each model container must read the uploaded image from the path ./saved_photo/
- Each model must write output to ./model_outputs/{model_name}/results.npz
- The containerized visualizer (when applicable) must read TruFor's .npz output and write a PNG to ./model_outputs/visualizer/visualize.png

Development Standards & Tools:

Languages:

Layer	Language	Version
Frontend	TypeScript	5.9.2
Backend	Python	3.10
Containers	Bash (entrypoints)	n/a

TypeScript 5.9.2 (Frontend)

Users access AIGIgator via a web browsing client (Edge, Chrome, etc.). TypeScript is thus a good choice for the frontend since it provides a static typing system on top of the popular web scripting language JavaScript. TypeScript reduces integration bugs by enforcing contracts between the user interface and FastAPI, which is helpful when passing the system's Job objects between those layers. Additionally, Angular 21, the system's frontend framework, uses and requires it.

Python 3.10 (Backend)

Machine learning and artificial intelligence systems are primarily built upon Python, so it is a natural choice for AIGIgator, which relies heavily on ML. For example, the analysis models' outputs are in .npz format, which can only practically be read by NumPy, a Python library. Using Python for backend logic also makes interfacing between FastAPI (a Python web framework) and the model orchestration layer easier.

Bash (Container Entrypoints)

Every model runs independently in its own Docker container, whose startup amounts to an image filename lookup followed by a handoff to the model's Python script. Bash is used for the container entry points since it is widely available in Docker base images. Further, it is simple to implement, needing only a few lines to start each model.

Frameworks & Libraries

Backend

- FastAPI 0.129 (REST API)
 - Compared to Flask or Django, FastAPI uses much less boilerplate code. This is good for a simple, focused API like `APIController.py`, which only exposes four endpoints.
 - FastAPI uses Pydantic natively to validate both incoming request data and outgoing responses, returning a structured error if a mismatch occurs. This is especially helpful when handling the system's Job objects.
- Uvicorn 0.40 (ASGI Server)
 - FastAPI is an Asynchronous Server Gateway Interface (ASGI) framework and requires an ASGI server to run.
 - Uvicorn is the ASGI server that was chosen since it is lightweight and built for a FastAPI stack; Starlette, FastAPI's underlying framework, is developed alongside Uvicorn.
- Pydantic 2.12 (FastAPI request/response validation)
 - The Job model in `APIController.py` is a Pydantic BaseModel, which automatically enforces field types and returns structured errors when they occur.
- NumPy 1.26.4 (reading .npz output files)
 - Using the NumPy library is the only practical way to read the .npz files that are output by each model.
 - NumPy is also used in the confidence calculation, such as when averaging the model's predictions and computing the binary entropy function.

Frontend

- Angular 21.1 (frontend SPA framework with standalone components)
 - Since Angular is a Single Page Application framework, different views can render on a single page without refreshing as the user moves through the system flow (image upload, report generation, etc).
 - Via Angular's dependency injection system, a single `AnalysisService` can handle all API communication and can be injected into any component that needs it.
- RxJS 7.8 (polling)
 - Checking image analysis progress is an asynchronous process. After the user uploads the image and its job starts, the frontend polls for the job status every two seconds until it completes or fails.
 - RxJS handles this asynchronous polling logic and cancels the job's subscription when completion or failure is reached.

- PrimeNG 21.1 (Angular UI component library)
 - PrimeNG provides a set of prebuilt Angular UI components that ensure consistent visual design throughout the flow of the system, from file upload to report generation.
 - PrimeNG's file upload component can handle filtering for .png and .jpg files, as well as for file size ($\leq 5\text{MB}$).
- Chart.js 4.5 (binary entropy chart rendering)
 - The report viewer renders a binary entropy chart that visualizes the confidence score for each model and the average confidence score. Chart.js is lightweight, facilitates fully client-side rendering, and has no extra framework dependencies. Other libraries like D3.js would have also worked, but would introduce unnecessary complexity.

Models

Refer to the project background or the GitHub links below for more details on the selected models:

Model	Source
TruFor	https://github.com/grip-unina/TruFor/tree/main
B-Free	https://github.com/grip-unina/B-Free/tree/main
DDA	https://github.com/roy-ch/Dual-Data-Alignment

Additional Notes:

- .npz is the standard data format output for all models
- Each model has its own dependencies, which are fully encapsulated within their respective Docker containers. Such dependencies are not a concern of the aggregation system itself.
- There is no network communication between model containers. Files are accessed via shared Docker volumes.
- Every detection model used relies heavily on Nvidia's proprietary CUDA compute platform, so GPU availability greatly affects completion time. Subsequently, a fixed timeout for image analysis was not defined since slow-hardware environments could be flagged as failures. This is another reason why image sizes are currently capped at 5MB.

METHODOLOGY

Design Pattern: Facade

The facade pattern minimizes the interactions necessary between clients and parts of a hidden, complex subsystem by using a “facade” class (GeeksforGeeks, 2017). This class is the only point of interaction between the clients and the hidden subsystem, ensuring that modifications to either side will not raise errors due to intertwined dependencies. The facade class facilitates communication between the two sides with a method that calls the subsystem per the request of the client, returning a value if necessary. Below is a simple example of the facade pattern, clearly showing the facade class and the connections it has to both the client-side and the subsystem-side. Crucially, there are no connections across this division besides through the facade, which enables the facade pattern to benefit both sides (Refactoring Guru, 2014). Clients are free to have any sort of relationship with other clients, and likewise, parts of the subsystem can have any kind of relationship with other parts of the subsystem. These relationships were omitted for clarity.

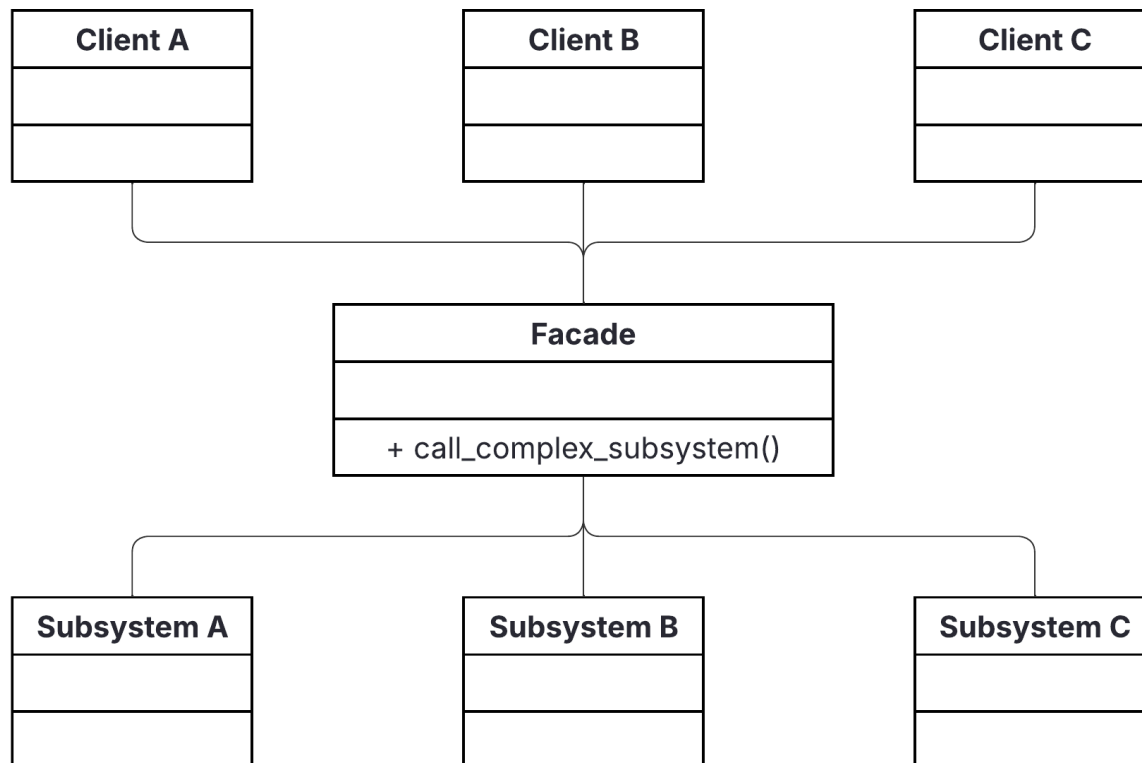


Figure 3. Facade pattern example.

In our implementation, the “Coordinator” class replaces the facade class. This class is highlighted in the following diagram. It cleanly separates the front end and backend, ensuring that modifications to either side will not impede the other. The front end of our design is the client side of the facade pattern, while the backend serves as the subsystem. The method that allows interaction between the two sides is `analyzeImages()`, which is also highlighted in the class diagram provided. The front end calls this method, then receives its output from the Coordinator. There are no relationships between the front and back ends of our design besides the one provided by the Coordinator class.

Diagrams:

System design diagrams are presented over the next five pages in the following order:

1. Class diagram
2. Component diagram
3. Activity diagram
4. Use Case diagram
5. Sequence diagram

Note: the system implementation of the Facade pattern is shown on the class diagram.

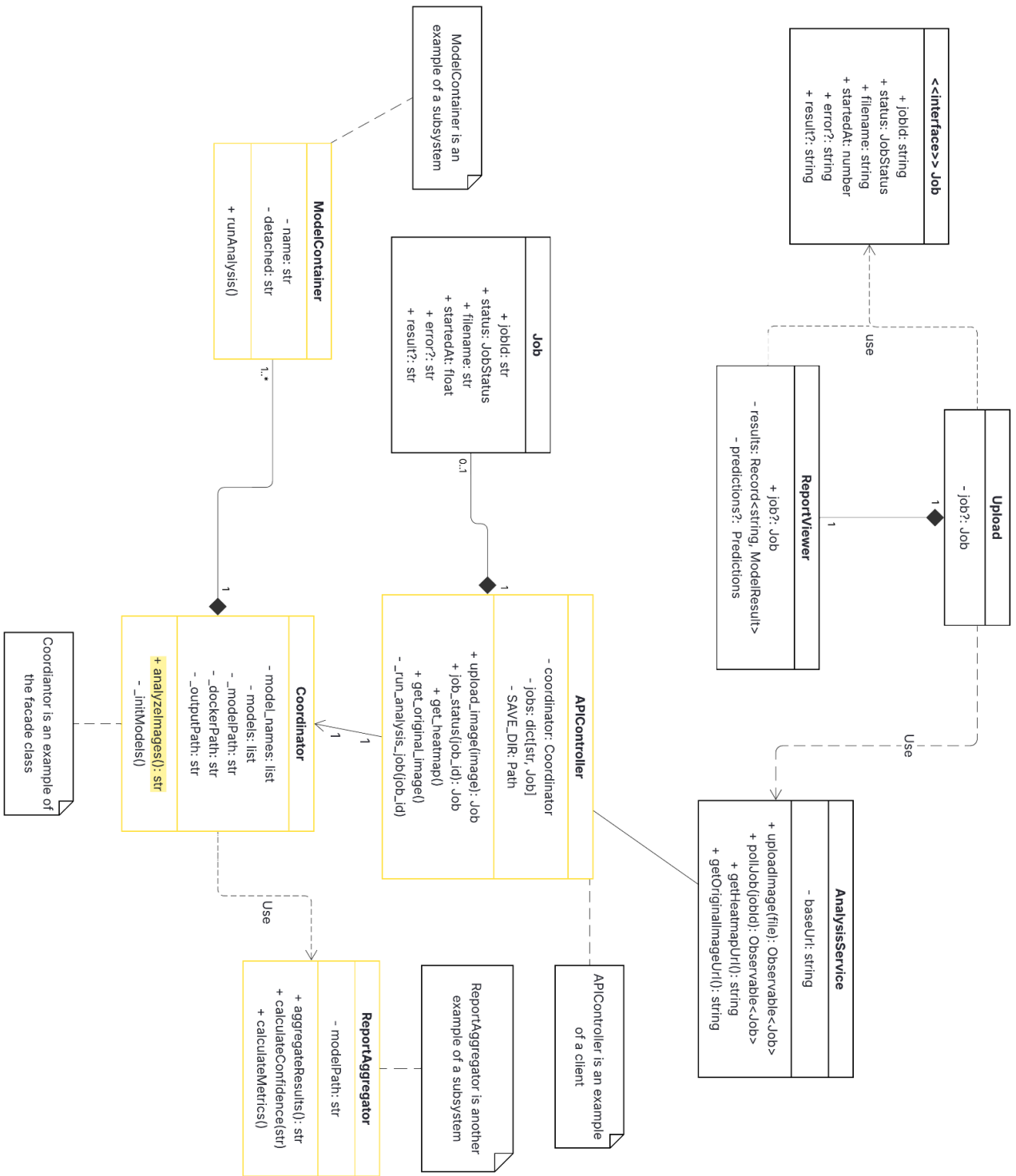


Figure 4. System class diagram with facade pattern.

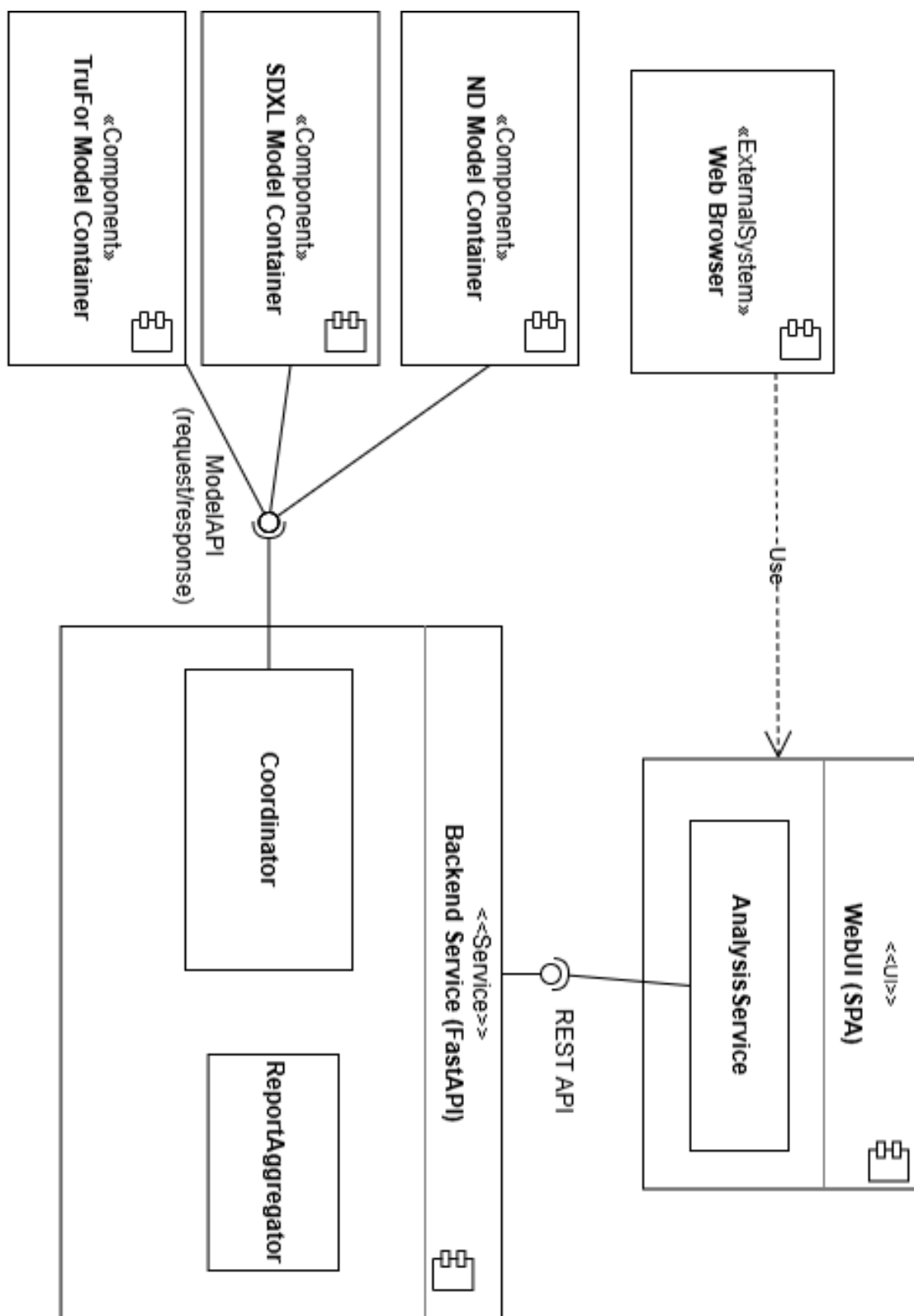


Figure 5. System component diagram.

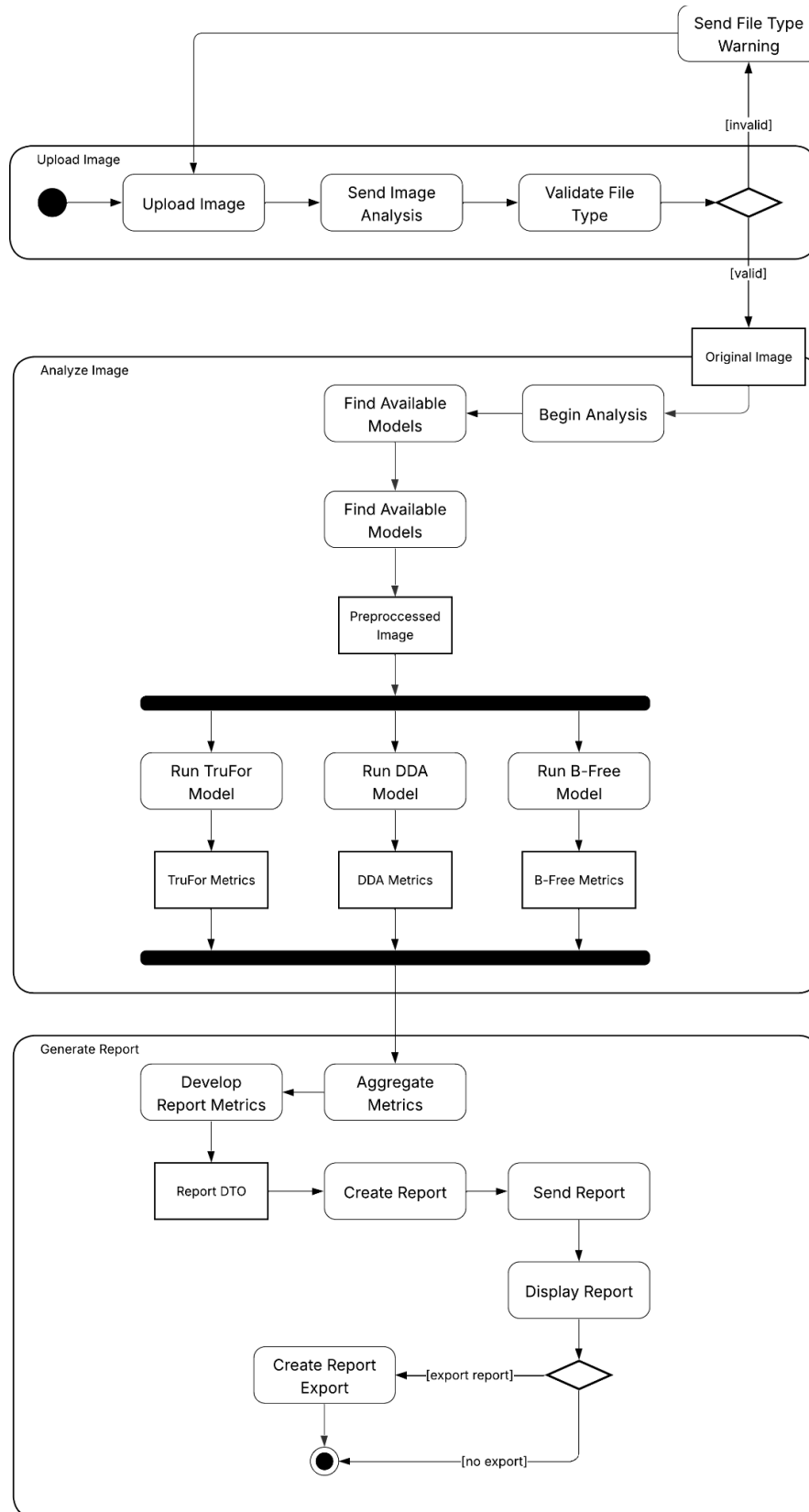


Figure 6. System activity diagram.

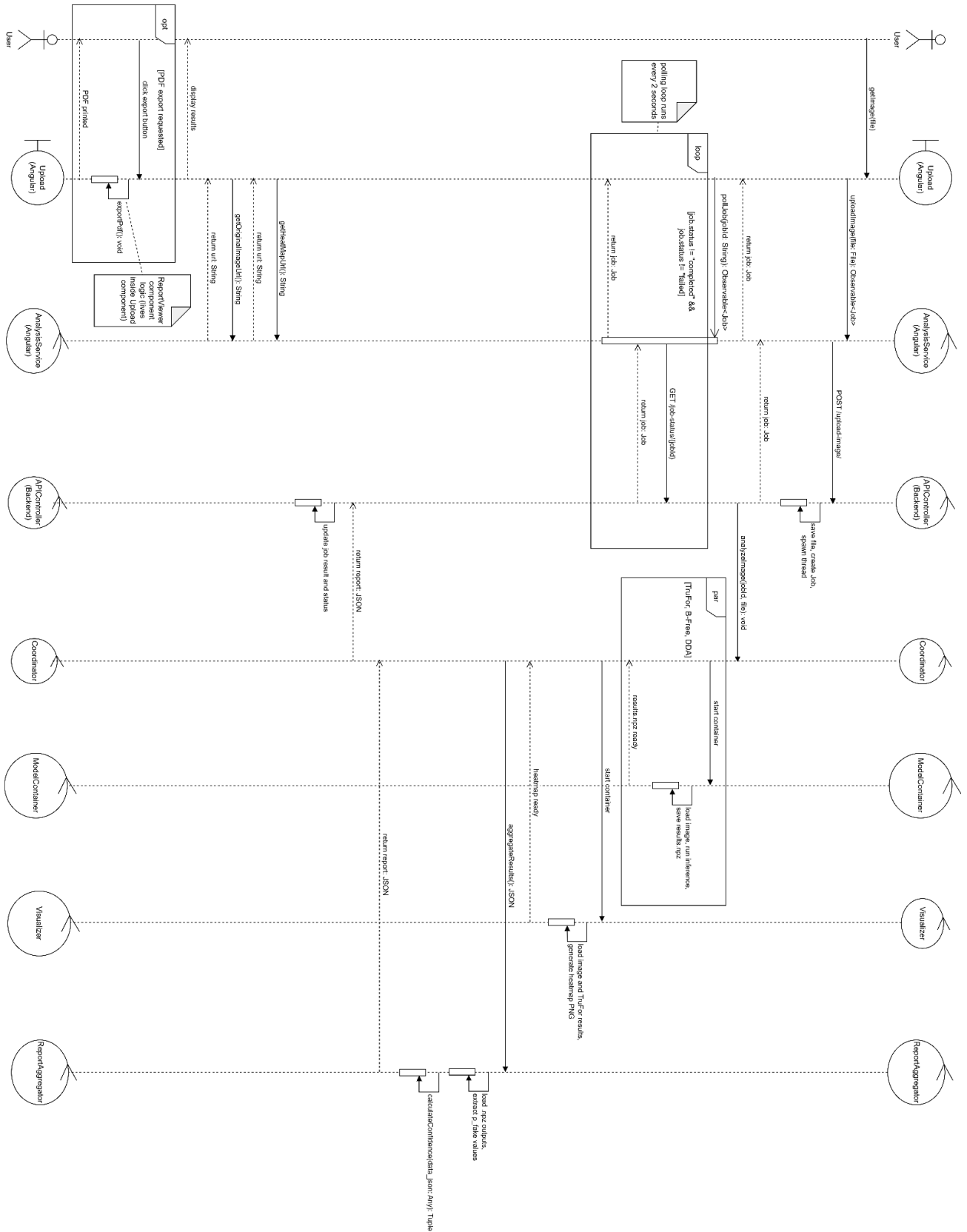


Figure 7. System sequence diagram.

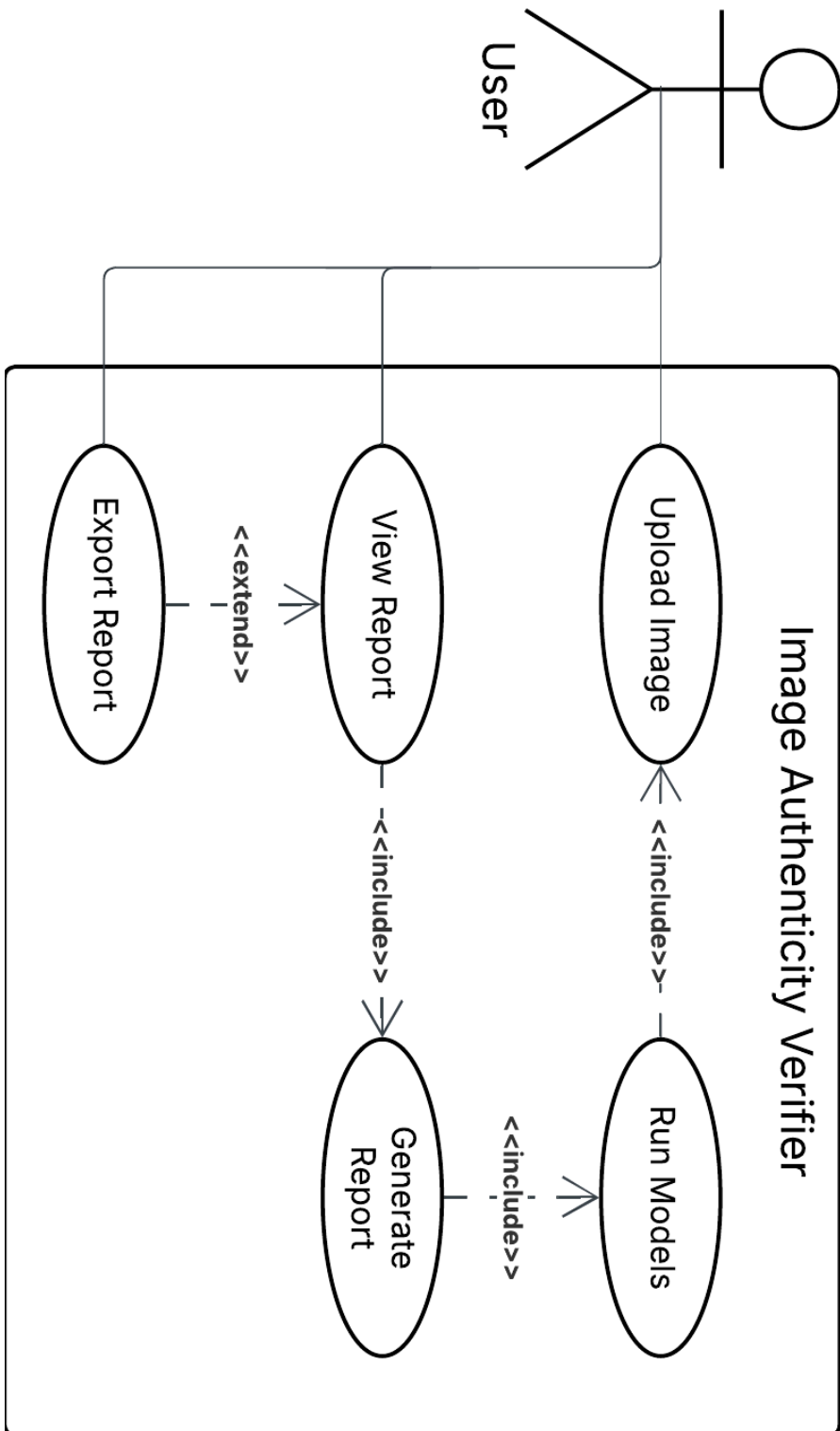


Figure 8. System use case diagram.

Use Case ID: UC-01

Use Case Name: “Analyze Image Authenticity”

Actors Involved:

Primary Actor: End User

Secondary Actor(s): None

Description: This Use case represents the entire process of the Image Authenticity Verifier system. The user will upload an image into the front end, which will then be processed and analyzed by forensic image analysis models. The outputs from these models will be recorded and aggregated into a comprehensive report, which will be available for the user to view and export if desired.

Preconditions:

- Docker is installed on the user’s system

Main Flow:

1. The user uploads an image to the System’s front-end web app.
2. The system validates the image, ensuring it is in the proper file format.
3. The system initializes the model suite.
4. The system scales the image to the proper dimensions for each model.
5. The system inputs the processed image to each model.
6. The system records the metrics output by the models.
7. The system generates a report based on the output metrics.
8. The user views the report via the web app and can choose to export it.

Postconditions:

- A final report is generated
- The report is displayed to the user

Alternative Flows:

A1: User uploads a non-supported file format

If a user uploads a file that is not in the correct format, the system will send a warning to the user, and the file will not be accepted.

A2: Models do not complete their analysis.

If the models do not complete their analysis, the system will record an error and display it to the user instead of generating a report.

MAIN USE CASE EXAMPLES

Example I:

This example 1) showcases the system's output when a user uploads an authentic image and 2) explains how to interpret said output. In this case, a photo of Mount Fuji is used:

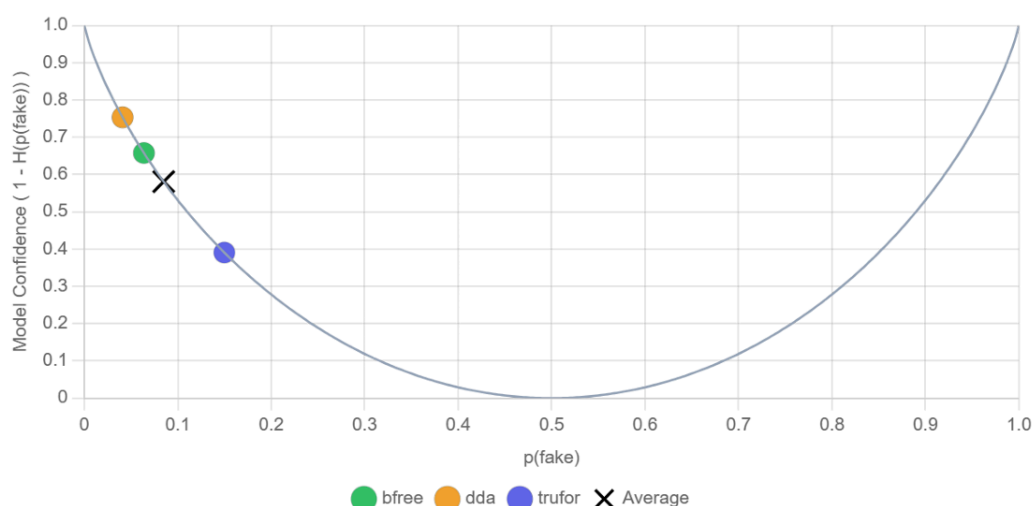


Figure 9a. Mount Fuji (Armin Forster, 2019)

Image Authenticity Report: arminep-japan-4287832_1920.jpg

More Likely Authentic with 58.1% confidence (aggregate)

Per-Model Prediction Confidence



Model	$p(\text{fake})$	Confidence	Verdict
bfree	0.064	65.8%	authentic
dda	0.041	75.3%	authentic
trufor	0.150	39.1%	authentic
Average	0.085	58.1%	authentic

Figure 9b. Mount Fuji image authenticity report.

Interpreting the Results: Refer to the “Per-Model Prediction Confidence” graph in Figure 9b. Along the x-axis is the probability of the image being fake, or $p(\text{fake})$. The $p(\text{fake})$ values are more certain towards the left and right ends of the graph (e.g., probabilities of 0 or 1) and are less certain towards the middle (a probability of 0.5 is a toss-up). A score sitting on the left side of the graph suggests the image is authentic, while a score sitting on the right side of the graph suggests that it is not.

To quantify confidence from the models’ $p(\text{fake})$ scores, they are plotted along an inverted binary entropy curve, which measures the average certainty of their binary (real vs. fake) outcomes. While a traditional binary entropy curve measures *uncertainty*, this curve is

inverted for clarity; the higher a model's output sits on the curve, the higher the confidence in the model's verdict. Exact values are displayed in a table below the graph for user convenience.

In this example, the B-Free and DDA detection models have low $p(fake)$ scores of 0.064 and 0.041, respectively, while TruFor's score is not quite as low, sitting at 0.150. The average level of confidence, determined by averaging every $p(fake)$ score and mapping the result (0.085) to the curve, is 58.1%. The system thus suggests that the image is authentic with a moderate level of confidence.

Note that by using the inverted binary entropy curve, a model's $p(fake)$ score needs to be very close to 0 or 1 to score high levels of confidence. Even a fairly high prediction of 0.895 may only yield ~51% confidence since such a score would still have *meaningful* (not insignificant) uncertainty.

Example II:

In some interesting cases, the models may disagree with each other. This can undermine the aggregate confidence score, but it can also provide valuable insight into the types of manipulations that may have been used and those that may have not.

This example analyzes an AI-generated image sourced from Pixabay. Though it may be clear to some users that the image is AI-generated, the specific method(s) used are not clear. This is thus a clear example of when a detection model aggregation system, which can check for a variety of generation and manipulation techniques, could be useful:

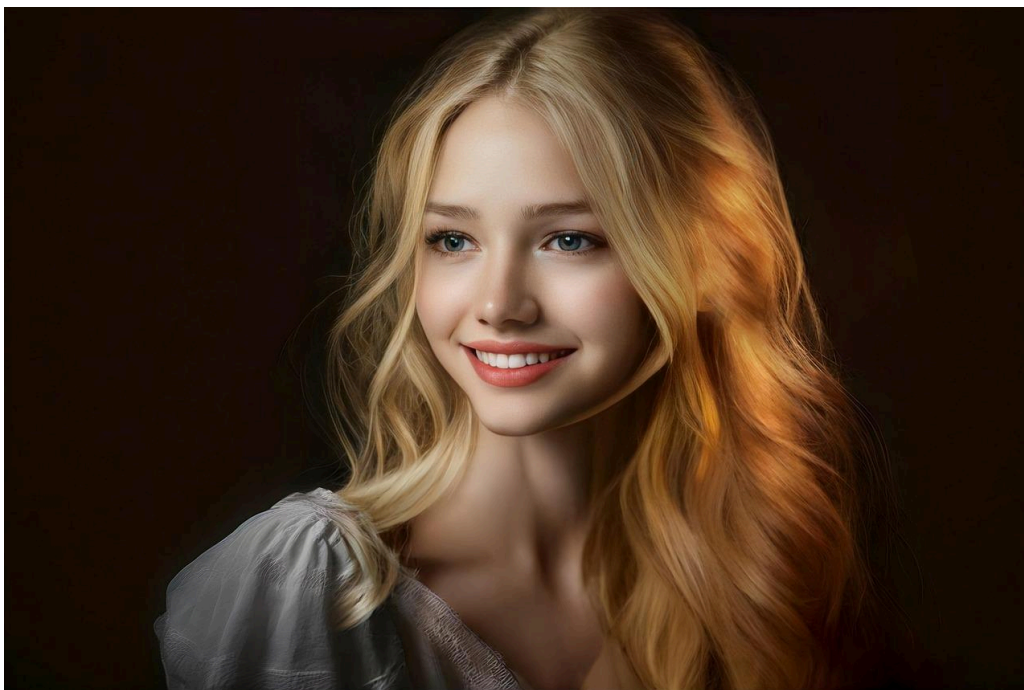
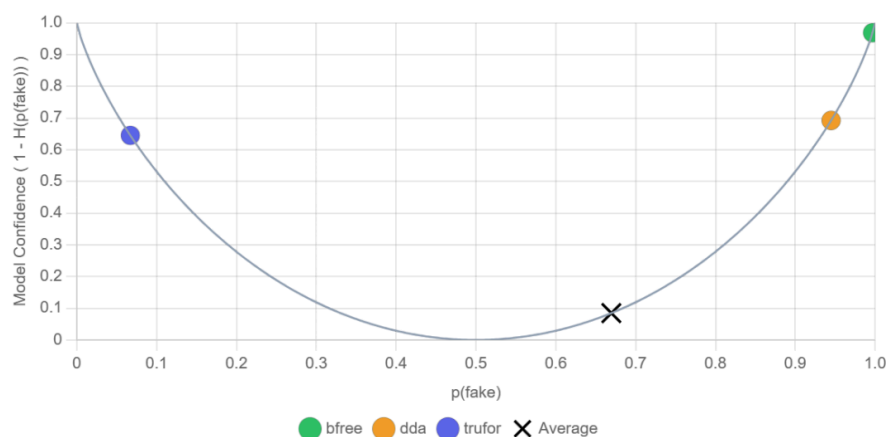


Figure 10a. AI-generated woman (Chil Vera, 2018)

Image Authenticity Report: ai-generated-7986194_1280.jpg

More Likely Fake with 8.5% confidence (aggregate)

Per-Model Prediction Confidence



Model	$p(fake)$	Confidence	Verdict
bfree	0.997	96.9%	fake
dda	0.945	69.2%	fake
trufor	0.067	64.5%	authentic
Average	0.670	8.5%	fake

Figure 10b. AI-generated woman image authenticity report.

Interpreting the Results: B-Free and DDA are in agreement that the image was AI-generated. If their $p(fake)$ scores (0.997 and 0.945, respectively) were the only ones used, a much higher aggregate level of confidence would have been calculated. However, TruFor is in disagreement, with a $p(fake)$ score of 0.067, finding little to no evidence of inauthenticity. This single disagreement brings down overall confidence significantly.

To explain this discrepancy, recall from the background that TruFor is a forensic framework. It is specialized for detecting localized deviations from an image's origin that are invisible to the naked eye. It is not designed, however, for detecting images that are AI-generated from scratch. B-Free and DDA, in contrast, are specifically designed for AIGI detection. According to TruFor, this image is authentic because the manipulation methods it searches for are not present in significant amounts. That alone, however, does not prove overall authenticity.

A Final Note on Modularity: The models used are interchangeable by design. Even the visualizer for TruFor, despite not being a detection model itself, is implemented in its own Docker container to maintain model modularity should TruFor be removed later on (this is also why TruFor's heatmap visualization was not included in these examples: it is not the system's primary output, and replacing TruFor would remove the visualization it generates). Isolating the visualizer from its associated model also allows for a different means of visualization to be added easily, or for the user to remove this element from the report completely.

The interchangeability of these models, while helpful for adapting to constantly evolving AI-generative and manipulative technologies, also presents a potential limitation: some combinations of models may yield drastically different confidence scores than other combinations for the same image.

This system provides a flexible means by which to analyze different model outputs simultaneously and how they agree or disagree. This flexibility, however, means the user themselves may need to exercise discretion in determining which models will be most useful and/or effective in their aggregate analyses.

SOURCE CODE

Relevant source code for AIGIgator is provided in the following pages. To see the comprehensive repository, visit <https://github.com/mrnagel/AIGIgator>.

BACKEND:

APIController.py

```
import threading
from typing import Dict, Literal, Optional
from fastapi import FastAPI, File, UploadFile, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import FileResponse
from pydantic import BaseModel
import time
from coordinator import Coordinator
import uuid
import shutil
import pathlib
import os

app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:4200"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"]
)
coordinator = Coordinator() #only one for lifetime

JobStatus = Literal["queued", "running", "completed", "failed"]

class Job(BaseModel):
    jobId: str
    status: JobStatus
    filename: str
    startedAt: float
```

```

    error: Optional[str] = None
    result: Optional[str] = None

jobs: Dict[str, Job] = {}

def _run_analysis_job(job_id: str):
    jobs[job_id].status = "running"
    try:
        result = coordinator.analyzeImages(detached=False,
visualizer=True)
        jobs[job_id].result = result
        jobs[job_id].status = "completed"
    except Exception as e:
        jobs[job_id].status = "failed"
        jobs[job_id].error = str(e)

SAVE_DIR = pathlib.Path('./saved_photo')
SAVE_DIR.mkdir(exist_ok=True)

@app.post("/upload-image/", response_model=Job)
async def upload_image(image: UploadFile = File(...)):
    # Remove old photos
    for i in os.listdir(SAVE_DIR):
        os.remove(SAVE_DIR / i)

    job_id = uuid.uuid4().hex

    save_path = SAVE_DIR / image.filename
    with save_path.open('wb') as f:
        shutil.copyfileobj(image.file, f)

    jobs[job_id] = Job(
        jobId=job_id,
        status="queued",
        filename=image.filename,
        startedAt=time.time()
    )

```

```

        t = threading.Thread(target=_run_analysis_job, args=(job_id,),
daemon=True)
        t.start()
        return jobs[job_id]

@app.get("/job-status/{job_id}", response_model=Job)
def job_status(job_id: str):
    job = jobs.get(job_id)
    if not job:
        raise HTTPException(status_code=404, detail="Unknown jobId")
    return job

@app.get("/heatmap")
def get_heatmap():
    path = pathlib.Path('./model_outputs/visualizer/visualize.png')
    if not path.exists():
        raise HTTPException(status_code=404, detail="Heatmap not
available")
    return FileResponse(path, media_type="image/png")

@app.get("/original-image")
def get_original_image():
    files = list(SAVE_DIR.iterdir())
    if not files:
        raise HTTPException(status_code=404, detail="Original image not
available")
    return FileResponse(files[0], media_type="image/png")

```

Coordinator.py

```

import subprocess
import pathlib
from reportAggregator import reportAggregator
import os
import glob
import shutil

class ModelContainer():
    def __init__(self, name, detached=True):
        self.name = name
        if detached == True:
            self.detached = '-d'
        else:
            self.detached = ''

    def runAnalysis(self):
        cmd = f'docker compose -f docker-compose.yml up --build {self.name}{self.detached}'
        print(subprocess.run(cmd.split(' ')))

class Coordinator():
    def __init__(self):
        self.model_names = []
        self.models = []
        self._modelPath = '../models'
        self._dockerPath = './'
        self._outputPath = './model_outputs'
        self._initModels()

    def _initModels(self):
        path = pathlib.Path(self._modelPath)
        folders = [i.name for i in path.iterdir() if i.is_dir()]
        self.model_names = folders

        # Dont use visualizer as a model
        for i in self.model_names:
            if i != 'visualizer': self.models.append(ModelContainer(i))
            else: self.model_names.remove(i)

```

```

print(f'Models available: {self.model_names}')

def analyzeImages(self, models=None, detached=True, visualizer=True):
    # Clear output folder of other/previous folders
    for i in glob.glob(f'{self._outputPath}/*'):
        if not os.path.isdir(i):
            raise ValueError(f'{i} is not a directory!')
        shutil.rmtree(i)

    if models:
        print('Not Implemented')
        exit()
    else:
        cmd = f'docker compose -f {self._dockerPath}docker-compose.yml
up -d {" ".join(self.model_names)}'
        subprocess.run(cmd.split(" "))
        if not detached:
            cmd = f'docker compose -f
{self._dockerPath}docker-compose.yml wait {" ".join(self.model_names)}'
            subprocess.run(cmd.split(" "))

        if visualizer:
            cmd = f'docker compose -f {self._dockerPath}docker-compose.yml
up -d visualizer'
            subprocess.run(cmd.split(" "))
            cmd = f'docker compose -f {self._dockerPath}docker-compose.yml
wait visualizer'
            subprocess.run(cmd.split(" "))

    return reportAggregator(self._outputPath).aggregateResults()

if __name__ == '__main__':
    x = Coordinator()
    y = x.analyzeImages(detached=False)
    print(y)

```

ReportAggregator.py

```

import numpy as np
import pathlib
import json
from collections import Counter

class reportAggregator:
    def __init__(self, modelPath='./model_outputs'):
        self._modelPath = modelPath

    def aggregateResults(self):
        path = pathlib.Path(self._modelPath)
        folders = [i.name for i in path.iterdir() if i.is_dir()]
        return_json = {}
        for i in folders:
            npz_file =
list(pathlib.Path(f'{self._modelPath}/{i}').glob('*.npz'))
            if not npz_file:
                continue

            with np.load(npz_file[0]) as data:
                return_json[i] = {key: data[key].tolist() for key in
data.files if data[key].ndim == 0}

        if not return_json:
            raise RuntimeError("Analysis models failed to produce output.
No report can be generated.")

        prediction, confidence =
self.calculateConfidence(json.dumps(return_json))

        print(type(prediction), type(confidence))
        return_json['predictions'] = {
            'prediction': prediction,
            'confidence': confidence
        }
        return json.dumps(return_json, indent=4)

    def calculateConfidence2(self, data_json):
        def inverse_h(p: np.array):

```

```

'''
Binary Entropy Function
https://en.wikipedia.org/wiki/Binary\_entropy\_function
'''
h_x = (-p*np.log2(p)) - (1-p)*np.log2(1-p)
# 1- h_x to invert the outputs, meaning 0.5 is 0 not 1
return 1 - h_x

data = json.loads(data_json)
models = data.keys()

p_fake = np.array([])
for i in models:
    p_fake = np.append(p_fake, data[i].get('p_fake', -1))

if -1 in p_fake:
    raise ValueError('Not all models have p_fake!')

# False if model predicts real, True if fake
preds = np.array([True if i > 0.5 else False for i in p_fake])

p_fake = np.clip(p_fake, 1e-10, 1-1e-10)
init_confidence = inverse_h(p_fake)

if np.all(preds == True) or np.all(preds == False):
    # Case where all models agree
    conf = np.average(init_confidence)
    pred = preds[0]
else:
    # TODO: create a penalizer function
    conf = 0
    pred = True

return pred, conf

def calculateConfidence(self, data_json):
    data = json.loads(data_json)
    models = data.keys()

    p_fake = np.array([])

```

```

for i in models:
    p_fake = np.append(p_fake, data[i].get('p_fake', -1))

if -1 in p_fake:
    raise ValueError('Not all models have p_fake!')

# False if model predicts real, True if fake
preds = np.array([True if i > 0.5 else False for i in p_fake])

p_fake = np.clip(p_fake, 1e-10, 1-1e-10)
p = np.average(p_fake)

#Binary Entropy Function
(https://en.wikipedia.org/wiki/Binary\_entropy\_function)
h_x = (-p*np.log2(p)) - (1-p)*np.log2(1-p)
conf = 1 - h_x

pred = Counter(preds).most_common(1)[0][0]

return bool(pred), float(conf)

def calculateMetrics(self):
    json_data = self.aggregateResults()
    return self.calculateConfidence(json_data)

def main():
    aggregator = reportAggregator()
    print(aggregator.aggregateResults())

if __name__ == "__main__":
    main()

```

MODELS:

entrypoint.sh: DDA (representative)

```
#!/bin/bash
```

```
FILE=$(ls /saved_photo | head -1)
```

```
exec python dda_main.py -i "/saved_photo/$FILE"
```

Note: All three model entry points use the same pattern. The only part that changes is the Python script executed.

entrypoint.sh: visualize.py

```
#!/bin/bash
```

```
FILE=$(ls /saved_photo | head -1)
```

```
exec python visualize.py --image "/saved_photo/$FILE" --output  
/model_outputs/trufor/results.npz --image_output  
/model_outputs/visualizer/visualize.png
```

visualize.py

```

import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import argparse

parser =
argparse.ArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpForma
tter)
parser.add_argument('--image', type=str, help='input image')
parser.add_argument('--output', type=str, help='output file')
parser.add_argument('--mask', type=str, default='', help='ground truth
mask (optional)')
parser.add_argument('--image_output', type=str, default='', help='output
image file')
args = parser.parse_args()

image_path = args.image
output_path = args.output
mask_path = args.mask

result = np.load(output_path)

cols = 3
if mask_path != '':
    cols += 1
    mask = Image.open(mask_path)
else:
    mask = None

if 'np++' in result:
    cols += 1
    noisepr = result['np++']
else:
    noisepr = None

fig, axs = plt.subplots(1, cols)
fig.suptitle('score: %.3f' % result['p_fake'])

for ax in axs:

```

```

    ax.axis('off')

index = 0
ax = axs[index]
ax.imshow(Image.open(image_path)), ax.set_title('Image')

if mask is not None:
    index += 1
    ax = axs[index]
    ax.imshow(mask, cmap='gray'), ax.set_title('Ground Truth')
    ax.set_yticks(list()), ax.set_xticks(list()), ax.axis('on')

if noisepr is not None:
    index += 1
    ax = axs[index]
    # for a better visualization of the noiseprint++, we remove the border
    and do a down-sampling (useful if the image is too big)
    ax.imshow(noisepr[16:-16:5, 16:-16:5], cmap='gray'),
ax.set_title('Noiseprint++')

index += 1
ax = axs[index]
ax.imshow(result['map'], cmap='RdBu_r', clim=[0,1]),
ax.set_title('Localization map')

index += 1
ax = axs[index]
ax.imshow(result['conf'], cmap='gray', clim=[0,1]),
ax.set_title('Confidence map')
ax.set_yticks(list()), ax.set_xticks(list()), ax.axis('on')

plt.show()

fig_map, ax_map = plt.subplots()
ax_map.imshow(result['map'], cmap='RdBu_r', clim=[0,1]),
ax.set_title('Localization map')
ax_map.tick_params(left=False, bottom=False, labelleft=False,
labelbottom=False)
fig_map.savefig(args.image_output, bbox_inches='tight', pad_inches=0)

```

FRONTEND:

// analysis-service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, switchMap, takeWhile, timer } from 'rxjs';

export type JobStatus = "queued" | "running" | "completed" | "failed";

// Mirror the Job model defined in APIController.py
export interface Job {
  jobId: string;
  status: JobStatus;
  filename: string;
  startedAt: number;
  error?: string | null;
  result?: string | null;
}

@Injectable({
  providedIn: 'root',
})

export class AnalysisService {
  private readonly baseUrl = 'http://localhost:8000';
  constructor(private http: HttpClient) {}

  //upload image, return Job with an initial status
  uploadImage(file: File): Observable<Job> {
    const form = new FormData();
    form.append('image', file);
    return this.http.post<Job>(`${this.baseUrl}/upload-image/`, form);
  }

  getJob(jobId: string): Observable<Job> {
    return this.http.get<Job>(`${this.baseUrl}/job-status/${jobId}`);
  }

  //poll /job-status every intervalMs milliseconds
```

```

pollJob(jobId: string, intervalMs = 2000): Observable<Job> {
    return timer(0, intervalMs).pipe(
        switchMap(() => this.getJob(jobId)),
        takeWhile(job => job.status !== 'completed' && job.status !==
'failed', true)

        );
}

getHeatmapUrl(): string {
    return `${this.baseUrl}/heatmap`;
}

getOriginalImageUrl(): string {
    return `${this.baseUrl}/original-image`;
}
}

```

// upload.ts

```
import { ChangeDetectorRef, Component, ViewChild } from '@angular/core';
import { AnalysisService, Job } from '../../services/analysis-service';
import { ReportViewer } from '../report-viewer/report-viewer';
import { BadgeModule } from 'primeng/badge';
import { ButtonModule } from 'primeng/button';
import { FileUpload, FileUploadModule } from 'primeng/fileupload';
import { MessageModule } from 'primeng/message';
import { ProgressBarModule } from 'primeng/progressbar';
import { Subscription } from 'rxjs';
```

```
@Component({
  selector: 'app-upload',
  standalone: true,
  imports: [ BadgeModule, ButtonModule, FileUploadModule, MessageModule,
  ProgressBarModule, ReportViewer],
  templateUrl: './upload.html',
  styleUrls: ['./upload.scss'],
})
```

```
export class Upload {
  @ViewChild('fileUpload') fileUpload!: FileUpload;
  job?: Job;
  pollingSub?: Subscription;
```

```
  // map job status to a PrimeNG badge severity to visually indicate job
  status
```

```
  get badgeSeverity(): 'success' | 'danger' | 'info' | 'warn' {
    switch (this.job?.status) {
      case 'completed': return 'success';
      case 'failed':     return 'danger';
      case 'running':   return 'info';
      default:           return 'warn';
    }
  }
}
```

```
  constructor(private analysis: AnalysisService, private cdr:
  ChangeDetectorRef) {}
```

```
  //upload image, then begin polling every 2 seconds
```

```

//cancel any existing polls before starting the new one
onUpload(event: any) {
  const file: File | undefined = event?.files?.[0];
  if (!file) return;

  this.analysis.uploadImage(file).subscribe((job) => {
    this.job = job;

    //if user uploaded image previously for some reason, cancel that
    poll subscription
    this.pollingSub?.unsubscribe();
    this.pollingSub = this.analysis.pollJob(job.jobId,
2000).subscribe((updatedJob) => {
      this.job = updatedJob;

      // trigger change detection manually (Angular itself doesn't run
      polling)
      this.fileUpload.cd.detectChanges();
      this.cdr.detectChanges();
      console.log('job update', updatedJob)
    });
  });
}

//cancel any active poll and reset for new analysis
onNewAnalysis(): void {
  this.pollingSub?.unsubscribe();
  this.job = undefined;
}

//cleanup logic when component destroyed
ngOnDestroy(): void {
  this.pollingSub?.unsubscribe();
}
}

```

```
<!-- upload.html -->
```

```
<div class="upload-page">
  @if (job?.status !== 'completed') {
    <div class="card">
      <p-fileUpload #fileUpload name="image" url="/api/upload-image/"
[customUpload] = "true" [multiple]="false" fileLimit=1 accept=".jpg,.png"
maxFileSize="5000000" mode="advanced" (uploadHandler)="onUpload($event)"
[showCancelButton]="false">
        <ng-template pTemplate="file" let-file>
          <div class="p-fileupload-file">
            @if (file.objectURL) {
              <img [src]="file.objectURL"[alt]="file.name" width="50" />
            }
            <span class="p-fileupload-file-name">{{file.name}}</span>
            <p-badge [value]="job?.status??'pending'"
[severity]="badgeSeverity" />
          </div>
        </ng-template>
        <ng-template pTemplate="empty">
          <div class="empty-state">
            <i class="pi pi-cloud-upload upload-icon"></i>
            <p class="upload-text">Drag and drop image files here to
upload.</p>
          </div>
        </ng-template>
      </p-fileUpload>
    </div>
  }
  @if (job?.status === 'failed') {
    <p-message severity="error" styleClass="error-message">{{job?.error ??
'An unexpected error occurred.'}}</p-message>
  }
  <app-report-viewer [job]="job" (newAnalysis)="onNewAnalysis()" />
</div>
```

// report-viewer.ts

```

import { Component, EventEmitter, Input, OnChanges, Output, ViewChild }
from '@angular/core';
import { Job } from '../../services/analysis-service';
import { BinaryEntropyChart, ModelResult } from
'./binary-entropy-chart/binary-entropy-chart';
import { HeatmapOverlay } from './heatmap-overlay/heatmap-overlay';
import { ButtonModule } from 'primeng/button';

export interface Predictions {
  prediction: boolean;
  confidence: number;
}

export interface ModelMetric {
  name: string;
  pFake: number;
  confidence: number;
  verdict: 'fake' | 'authentic';
}

@Component({
  selector: 'app-report-viewer',
  standalone: true,
  imports: [BinaryEntropyChart, HeatmapOverlay, ButtonModule],
  templateUrl: './report-viewer.html',
  styleUrls: ['./report-viewer.scss'],
})
export class ReportViewer implements OnChanges {
  @Input() job?: Job;
  @Output() newAnalysis = new EventEmitter<void>();
  @ViewChild(BinaryEntropyChart) entropyChart?: BinaryEntropyChart;
  results: Record<string, ModelResult> = {};
  predictions?: Predictions;
  modelMetrics: ModelMetric[] = [];
  avgPFake?: number;
  avgConfidence?: number;
  avgVerdict?: 'fake' | 'authentic';

  ngOnChanges(): void {

```

```

    if (this.job?.result) {
        const parsed = JSON.parse(this.job.result);
        this.predictions = parsed['predictions'];

        //strips "predictions" key from parsed result (only need per model
data here)
        const { predictions: _, ...modelResults } = parsed;
        this.results = modelResults;
        this.modelMetrics = Object.entries(this.results as Record<string,
ModelResult>).map(([name, r]) => ({
            name,
            pFake: r.p_fake,
            confidence: this.entropy(r.p_fake),
            verdict: r.p_fake > 0.5 ? 'fake': 'authentic',
        }));
        const values = this.modelMetrics.map(m => m.pFake);
        this.avgPFake = values.reduce((a, b) => a + b, 0) / values.length;
        this.avgConfidence = this.entropy(this.avgPFake);
        this.avgVerdict = this.avgPFake > 0.5 ? 'fake' : 'authentic';
    }
}

//binary entropy function
private entropy(p: number): number {
    if (p==0 || p== 1) return 1;
    return 1 - (-p* Math.log2(p) - (1 - p) *Math.log2(1 - p));
}

//adjust chart colors (for print contrast), trigger print, restore after
exportPdf(): void {
    this.entropyChart?.prepareForPrint();
    window.print();
    this.entropyChart?.restoreFromPrint();
}
}

```

// binary-entropy-chart.ts

```
import { Component, Input, OnChanges, ViewChild } from '@angular/core';
import { ChartModule } from 'primeng/chart';

export interface ModelResult {
  p_fake: number;
  p_real: number;
}

@Component({
  selector: 'app-binary-entropy-chart',
  standalone: true,
  imports: [ChartModule],
  templateUrl: './binary-entropy-chart.html',
  styleUrls: ['./binary-entropy-chart.scss'],
})
export class BinaryEntropyChart implements OnChanges {
  @Input() results: Record<string, ModelResult> = {};
  @Input() avgPFake?: number;

  @ViewChild('chartRef') chartRef: any;

  chartData: any;
  chartOptions: any;

  ngOnChanges(): void {
    this.buildChart();
  }

  //change average marker to black for print contrast, then restore after
  printing
  prepareForPrint(): void {
    this.setAvgColor('#000000');
  }

  restoreFromPrint(): void {
    this.setAvgColor('#ffffff');
  }

  private setAvgColor(color: string): void {
    const chart= this.chartRef?.chart;
    if (!chart) return;
  }
}
```

```

    const avgDataset=chart.data.datasets.find((d: any) => d.label ===
'Average');
    if (!avgDataset) return;
    avgDataset.borderColor =color;
    chart.update('none');
  }

  // binary entropy confidence is 1-H(p), doesn't include where log2 is
undefined
  private confidence(p: number): number {
    if (p === 0 || p === 1) return 1;
    return 1 - (-p * Math.log2(p) - (1 - p) * Math.log2(1 - p));
  }

  //generate 200 x values spaced from 0 to 1 to plot entropy curve
  private buildChart(): void {
    const steps = 200;
    const xs = Array.from({ length: steps + 1 }, (_, i) => i / steps);

    const modelColors: Record<string, string> = {
      trufor: '#6366f1',
      bfree: '#22c55e',
      dda: '#f59e0b',
    };

    // each model's p_fake score gets plotted as a point on the curve
    const scatterDatasets = Object.entries(this.results).map(([name, r])
=> ({
      type: 'scatter',
      label: name,
      data: [{ x: r.p_fake, y: this.confidence(r.p_fake) }],
      backgroundColor: modelColors[name] ??'#94a3b8',
      pointRadius: 7,
      pointHoverRadius: 9,
      hitRadius: 7,
    }));

    const avgDataset = this.avgPFake !== null ? [{
      type: 'scatter',
      label: 'Average',

```

```

    data: [{x:this.avgPFake, y:this.confidence(this.avgPFake) }],
    backgroundColor: '#ffffff',
    borderColor: '#ffffff',
    borderWidth: 2,
    pointStyle: 'crossRot',
    pointRadius: 10,
    pointHoverRadius: 12,
    hitRadius: 10,
  }]:[];

  //use line dataset to draw entropy curve, overlay model predictions w/
  scatter dataset
  this.chartData = {
    labels: xs.map((x) => x.toFixed(2)),
    datasets: [
      {
        type: 'line',
        data: xs.map((x) => ({ x, y: this.confidence(x)})),
        borderColor: '#94a3b8',
        borderWidth: 1.5,
        pointRadius: 0,
        fill: false,
        tension: 0.4,
      },
      ...scatterDatasets,
      ...avgDataset,
    ],
  };

  this.chartOptions = {
    responsive: true,
    maintainAspectRatio: true,
    plugins: {
      legend: {
        position: 'bottom',

        //exclude entropy curve line from legend
        labels: { filter: (item: any) => item.datasetIndex !== 0,
usePointStyle: true },
      },
    },
  },

```

```

tooltip: {
  callbacks: {
    title: (items: any[]) => {
      const item = items[0];
      if (item?.dataset.type === 'scatter') {
        return `p(fake) = ${item.parsed.x.toFixed(3)}`;
      }
      return '';
    },
    label: (ctx: any) => {
      if (ctx.dataset.type === 'scatter') {
        const label = ctx.parsed.x > 0.5 ? 'fake' : 'authentic';
        return `${ctx.dataset.label}: ${((ctx.parsed.y *
100).toFixed(1))}% confident ${label}`;
      }
      return '';
    },
  },
},
scales: {
  x: {
    type: 'linear',
    min: 0,
    max: 1,
    title: { display: true, text: 'p(fake)' },
  },
  y: {
    min: 0,
    max: 1,
    title: { display: true, text: 'Model Confidence ( 1 - H(p(fake))
)' },
  },
},
};
}
}

```

WORKS CITED

- Chen, R., Xi, J., Yan, Z., Zhang, K.-Y., Wu, S., Xie, J., Chen, X., Xu, L., Guan, I., Yao, T., & Ding, S. (2025). *Dual Data Alignment Makes AI-Generated Image Detector Easier Generalizable*. ArXiv.org. <https://arxiv.org/abs/2505.14359>
- Chapter 27 entropy, randomness, and information CS 573: Algorithms, fall 2013. (n.d.-b). https://courses.grainger.illinois.edu/cs573/fa2013/lec/slides/27_notes.pdf
- Damato, A., & Brona. (2007). *Binary entropy plot SVG image*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Binary_entropy_plot.svg
- Ellery, S. (2023, March 28). *Fake photos of Pope Francis in a puffer jacket go viral, highlighting the power and peril of AI*. www.cbsnews.com. <https://www.cbsnews.com/news/pope-francis-puffer-jacket-fake-photos-deepfake-power-peril-of-ai/>
- Forster, Armin. (2019, June 21). *Japan Mountain Lake Mount - Free photo on Pixabay*. Pixabay.com. <https://pixabay.com/photos/japan-mountain-lake-mount-fuji-sky-4287832/>
- GeeksforGeeks. (2017, July 5). *Facade Method Design Pattern*. GeeksforGeeks. <https://www.geeksforgeeks.org/system-design/facade-design-pattern-introduction/>
- grip-unina. (2025). *GitHub - grip-unina/B-Free*. GitHub. <https://github.com/grip-unina/B-Free/tree/main>
- grip-unina. (2022). *GitHub - grip-unina/TruFor: TruFor*. GitHub. <https://github.com/grip-unina/TruFor>

Guillaro, F., Zingarini, G., Usman, B., Sud, A., Cozzolino, D., & Verdoliva, L. (2024). *A Bias-Free Training Paradigm for More General AI-generated Image Detection*. ArXiv.org. <https://arxiv.org/abs/2412.17671>

Guillaro, F., Cozzolino, D., Sud, A., Dufour, N., & Verdoliva, L. (2023, May 25). *TruFor: Leveraging all-round clues for trustworthy image forgery detection and localization*. ArXiv.org. <https://doi.org/10.48550/arXiv.2212.10957>

Refactoring Guru. (2014). *Facade*. Refactoring.guru.
<https://refactoring.guru/design-patterns/facade>

Roca, T., Roman, A. C., Vega, J. T., Duarte, M., Wang, P., White, K., Misra, A., & Ferres, J. L. (2025). *How good are humans at detecting AI-generated images? Learnings from an experiment*. ArXiv.org. <https://arxiv.org/abs/2507.18640>

roy-ch. (2025). *GitHub - roy-ch/Dual-Data-Alignment: This repository is the official implementation of NeurIPS 2025 Paper “Dual Data Alignment Makes AI-Generated Image Detector Easier Generalizable”*. GitHub.
<https://github.com/roy-ch/Dual-Data-Alignment>

Vera, Chil. (2023, May 14). *Download Ai Generated, Woman, Portrait. Royalty-Free Stock Illustration Image*. Pixabay.com.
<https://pixabay.com/illustrations/ai-generated-woman-portrait-model-7986194/>