AI detector with frontend interface

Christopher Kurdoghlian

Spring 2025 CS2400 AI

1. Uses a technology called DistilBERT - a compact AI model that understands language
2. This model has been trained to recognize differences between human and AI writing styles
3. It's like a language detective that spots subtle patterns most people miss
4. Similar to how you might recognize a friend's writing style, but much more precise

Trained on https://huggingface.co/datasets/NabeelShar/ai_and_human_text

About This Project

How this AI Text Detector works and the technologies behind it

Machine Learning Model Details

DistilBERT: A Lightweight Language Model

This application uses DistilBERT, a condensed version of BERT (Bidirectional Encoder Representations from Transformers) that retains 97% of BERT's language understanding capabilities while being 40% smaller and 60% faster. DistilBERT was created through a process called knowledge distillation, where a smaller model is trained to mimic a larger, more powerful model.

Key advantages of DistilBERT include:

- Reduced model size (66 million parameters vs. BERT's 110 million)
- Faster inference time while maintaining high accuracy
- Lower computational resource requirements
- Ability to run efficiently in production environments

Tokenization Process

Before processing text through the model, it must first be tokenized. Tokenization is the process of breaking text into smaller units (tokens) that the model can understand. DistilBERT uses a WordPiece tokenizer that works as follows:

1. Split text into basic units (words, punctuation)
2. Break words into subwords based on a pre-defined vocabulary
3. Add special tokens: [CLS] at the beginning and [SEP] at the end
4. Convert tokens to numeric IDs using a vocabulary lookup
5. Generate attention masks to indicate which tokens are padding

For example, the word "unbelievable" might be broken down into "un", "##believe", and "##able". This subword tokenization allows the model to understand parts of words it hasn't seen before and helps with handling rare words.

Training and Fine-tuning

Our model was fine-tuned on the "dmitva/human_ai_generated_text" dataset from Hugging Face, which contains pairs of human-written and AI-generated texts. We used a subset of 5,000 samples to create a balanced training dataset.

The training process involved:

- Splitting data into 80% training and 20% validation sets
- Fine-tuning the pre-trained DistilBERT model for binary classification
- Using binary cross-entropy loss function to optimize the model
- Training for one epoch with a learning rate of 3e-5
- Evaluating with accuracy, precision, recall, and F1 metrics

The model achieved over 99% accuracy on the validation set, demonstrating its effectiveness at distinguishing between human and AI-generated content.

Sliding Window Approach for Long Texts

Because transformer models like DistilBERT have a maximum input length (typically 512 tokens), we implemented a sliding window approach to handle longer texts. Here's how it works:

1. For texts under 256 tokens, process the entire text at once
2. For longer texts, divide into overlapping windows of 256 tokens each
3. Use a consistent 128-token overlap between adjacent windows
4. Process each window separately through the model
5. Average the probability scores from all windows for the final prediction

Example for a 300-token text:

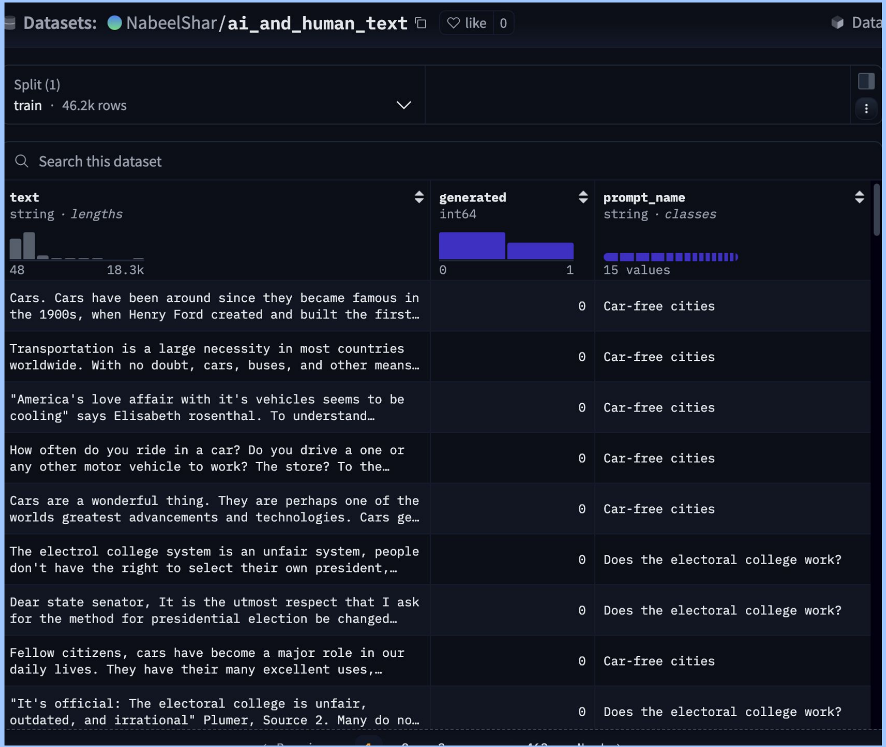
- Window 1: Tokens 0-255 (first 256 tokens)
- Window 2: Tokens 128-299 (remaining 172 tokens)
- Final score: Average of probabilities from both windows

This approach ensures that no content is missed and that context at window boundaries is properly captured, as each boundary appears in multiple windows. It also helps maintain accuracy for long documents that would otherwise exceed the model's capacity.

Inference and Classification

When analyzing text, the model processes the tokenized input and outputs logits (raw prediction scores). These logits are then transformed using a softmax function to produce probabilities between 0 and 1, where:

- Values close to 0 indicate human-written text
- Values close to 1 indicate AI-generated text
- The decision boundary is 0.5 (50%)


Datasets: NabeelShar/ai_and_human_text like 0 Data

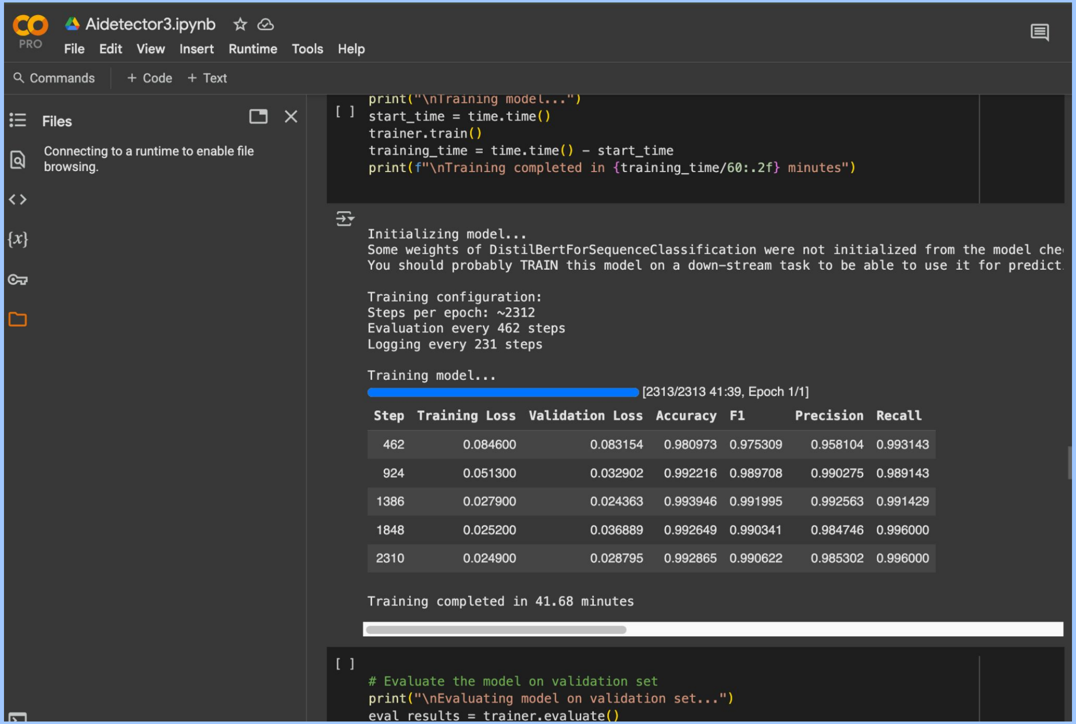
Split (1)
train · 46.2k rows

Search this dataset

text string · lengths 48 18.3k	generated int64 0 1	prompt_name string · classes 15 values
Cars. Cars have been around since they became famous in the 1900s, when Henry Ford created and built the first…	0	Car-free cities
Transportation is a large necessity in most countries worldwide. With no doubt, cars, buses, and other means…	0	Car-free cities
"America's love affair with it's vehicles seems to be cooling" says Elisabeth rosenthal. To understand…	0	Car-free cities
How often do you ride in a car? Do you drive a one or any other motor vehicle to work? The store? To the…	0	Car-free cities
Cars are a wonderful thing. They are perhaps one of the worlds greatest advancements and technologies. Cars ge…	0	Car-free cities
The electrol college system is an unfair system, people don't have the right to select their own president,…	0	Does the electoral college work?
Dear state senator, It is the utmost respect that I ask for the method for presidential election be changed…	0	Does the electoral college work?
Fellow citizens, cars have become a major role in our daily lives. They have their many excellent uses,…	0	Car-free cities
"It's official: The electoral college is unfair, outdated, and irrational" Plumer, Source 2. Many do no…	0	Does the electoral college work?

Getting a quality dataset that had good examples, lots of data, and thus prevented overfitting took three tries. This was my third dataset and script.

Model trained on google Collab



The dataset is imported using a link of the hugging face dataset and opened up, processed, and the loss and other metrics are seen here.

Note: you can see the training script in full at the root directory of my project. It is called AIdetector.ipynb

- The model doesn't read words like we do - it breaks text into smaller pieces called "tokens"
- For example, "unbelievable" becomes three pieces: "un" + "believe" + "able"
- This helps it understand parts of words and handle words it hasn't seen before
- Every token gets converted to a number that the AI can process

Tokenization

<> Tokenization Visualization

See how the DistilBERT tokenizer processes your text

Text Tokenization

3 tokens / 2 words

Complete Token Sequence (with special tokens)

[CLS]	testing	token	##ization	[SEP]
101	5604	19204	3989	102

How DistilBERT Tokenization Works:

1. Special tokens like [CLS] and [SEP] are added at the beginning and end
2. Words are split into subwords (tokens) based on frequency
3. Tokens starting with "##" are continuations of the previous word
4. Each token is assigned a numeric ID from the vocabulary
5. These token IDs are what the model actually processes

Reset

The AI can only look at 512 tokens (roughly 300-400 words) at once
For longer texts, we use a "sliding window" approach:

- Break the text into overlapping chunks
- Analyze each chunk separately
- Combine the results to get the final answer

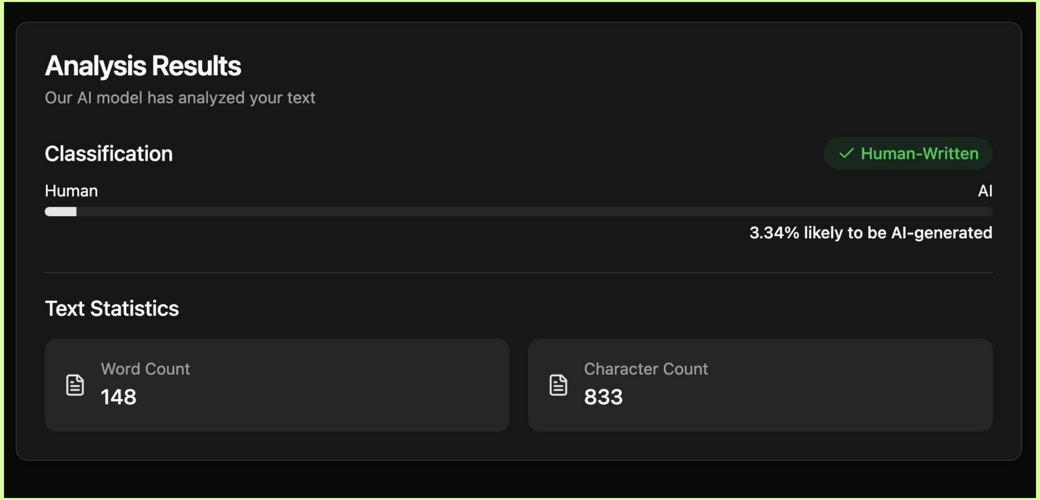
Like reading a book by examining overlapping pages rather than the whole book at once
The overlaps are 256 tokens

This is due to the nature of the model - you must input an exact length.

Shorter ones have tokens added to them.

512 token input and larger inputs

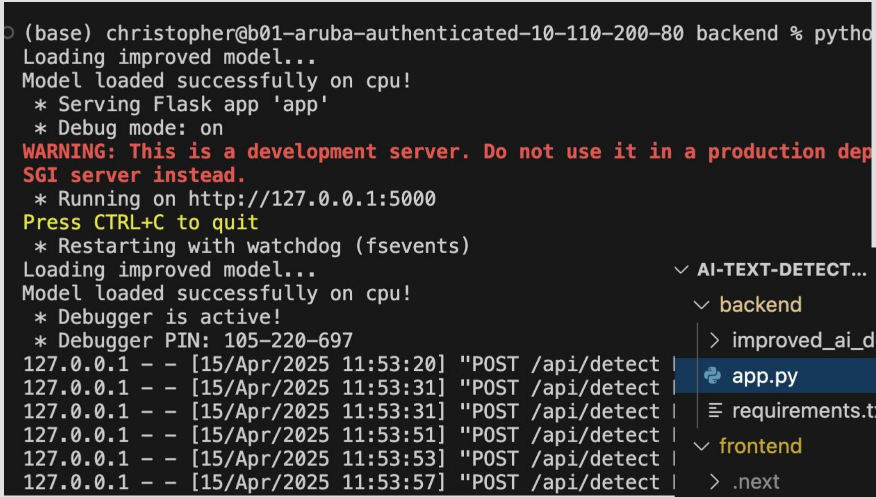
The system doesn't just give a yes/no answer - it tells you how confident it is
A result might be "85% likely to be AI-generated"
I adjust this confidence using a "temperature" setting to make it more reliable
Higher confidence means the AI is more certain about its decision

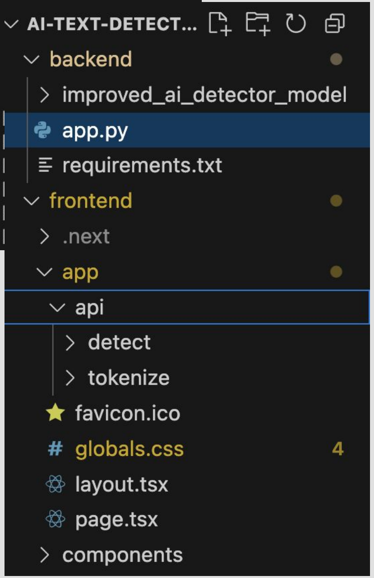


Backend

Flask handles requests from users
When you submit text, it:

Prepares the text for the AI model
Runs the model to get a prediction
Calculates confidence scores
Sends results back to the website





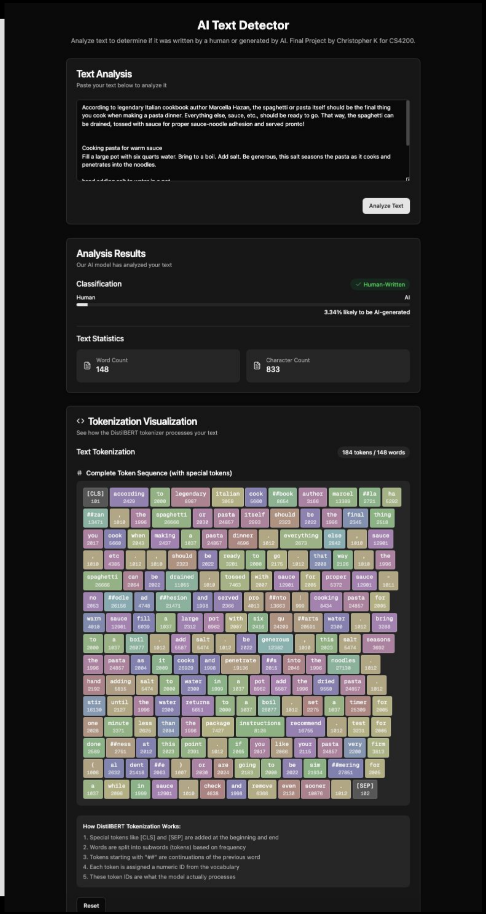
All the heavy AI processing happens here

There is also the tokenization route and logic

Built using modern web technology (Next.js)

- Features a simple text input box where you paste your text
- Shows results with easy-to-understand visuals
- Includes educational sections that explain how AI detection works
- Api routing in the api/ folder handles forwarding the requests to the flask backend

Frontend



Setting up the AI part:

Install Python on your computer
Download the project files
Open a command window and type: cd backend
Install required programs: pip install -r requirements.txt
Start the AI server: python app.py

Setting up the website:

Install Node.js on your computer
Open a new command window and type: cd frontend
Install website components: npm install
Start the website: npm run dev
Open your web browser to: http://localhost:4000

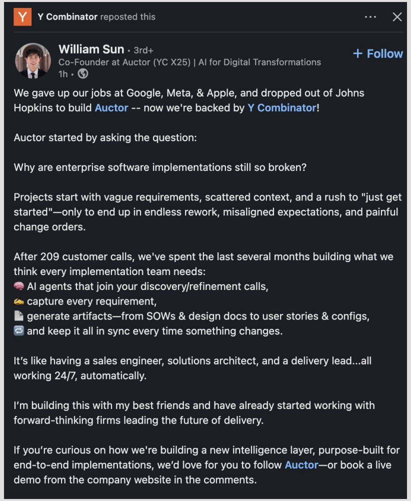
Setup

http://localhost:4000

https://github.com/christopherk26/ai-text-detector

Human text:
https://feelgoodfoodie.net/recipe/how-to-cook-pasta/

AI text: (found this on linkedin - the best place to find ai generated content)


Y Combinator reposted this

William Sun • 3rd+
Co-Founder at Auctor (YC X25) | AI for Digital Transformations
1h •
+ Follow

We gave up our jobs at Google, Meta, & Apple, and dropped out of Johns Hopkins to build Auctor -- now we're backed by Y Combinator!

Auctor started by asking the question:

Why are enterprise software implementations still so broken?

Projects start with vague requirements, scattered context, and a rush to "just get started"—only to end up in endless rework, misaligned expectations, and painful change orders.

After 209 customer calls, we've spent the last several months building what we think every implementation team needs:
🧠 AI agents that join your discovery/refinement calls,
✍️ capture every requirement,
📄 generate artifacts—from SOWs & design docs to user stories & configs,
🔁 and keep it all in sync every time something changes.

It’s like having a sales engineer, solutions architect, and a delivery lead...all working 24/7, automatically.

I’m building this with my best friends and have already started working with forward-thinking firms leading the future of delivery.

If you’re curious on how we're building a new intelligence layer, purpose-built for end-to-end implementations, we’d love for you to follow Auctor—or book a live demo from the company website in the comments.

https://www.linkedin.com/feed/update/urn:li:activity:7317969983924293635/

Link and demo

Takeaways

- Learned about LLMs, tokenization, NN’s
- Realized that quality ai detectors are hard to make in general and are easy to trick
- Ai detectors in general are not very good (tried other ones online)
- What does this mean for the state of the internet (dead internet theory?)
- Incorporating several models together (what if we try using an llm and give it text and ask it to decide?)

If you’re curious on how we're building a new intelligence layer, purpose-built for end-to-end implementations, we’d love for you to follow Auctor—or book a live demo from the company website in the comments.

Oh—and meet our fifth cofounder (Chief Bark Officer 🐶) in the photo!

give me a percentage rating that you think this is ai generated

I'd give this post about a **35-45%** likelihood of being AI-generated. Here's why:

Reasons it *might* be AI-generated:

- **Polished, structured language:** The writing is clean, confident, and flows logically — something AI is good at.
- **Startup-style tone:** The post nails the typical startup announcement voice, with crisp formatting (emojis, bullet points, punchy intro), which AI tools often mimic well.

Thanks!

Christopher
Planeswalker – Christopher
+1: Scry 1, then draw a card.
−2: Christopher deals 3 damage to any target.
−6: Draw four cards.
4
049 SR EN
C & T
WIIZARDS OF THE
™ & © 202 Wizards of the Coast