VisionMate: AI-Powered Image Captioning Web Application

Master's Defense
By Sai Anoushka Kokku

Advisor: Dr. Chris Pollett
Committee: Dr. Robert Chun
Committee: Dr. Thomas Austin

Outline

Introduction
Background
Model Research and Experimentation
System Architecture and Implementation
Deployment
Evolution of VisionMate
Testing, Evaluation and Results
Conclusion and Limitations
Future Work
References

Introduction

Problem Statement:

- Visually impaired users cannot interpret visual scenes
- Most daily tasks rely on image-based information
- Need: A fast, simple way to describe images aloud using AI

Project Goals:

- Real-time image captioning system
- Should run on both desktop and mobile browsers
- Must give spoken feedback — no visual-only results
- Lightweight, free to deploy, no installation required
- Leverage transformer-based models (via API, not locally)

Background: Related Work

Existing Assistive Tools:

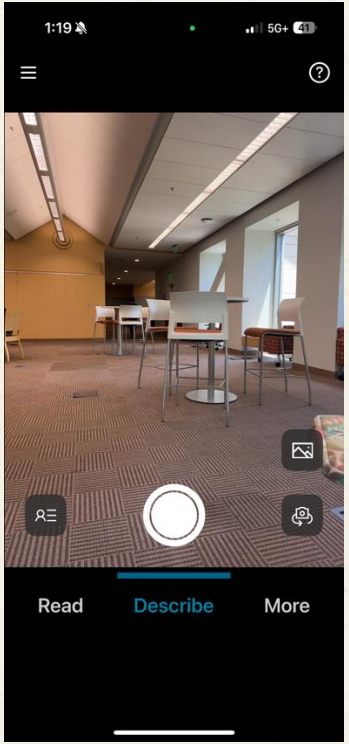
- **Seeing AI (Microsoft):** Describes people, text, and scenes via mobile app
- **Be My Eyes:** Connects users with volunteers to describe visuals via video call

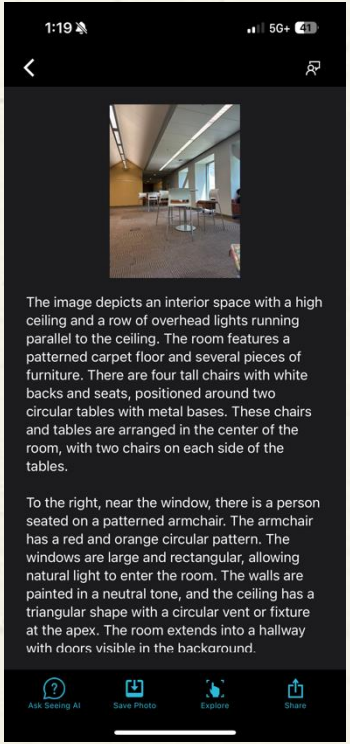
Both apps require installation and run only on mobile platforms

No real-time web-based alternative for spoken image captions

Background: Related Work

Seeing AI output





Background: Image Captioning Overview

- Image captioning = understanding an image + generating a sentence
- Combines computer vision (object recognition) + language generation
- Requires large datasets like COCO (image + caption pairs)
- Uses encoder-decoder architecture (e.g., CNN + RNN or Transformer-based)

Background: Vision-Language Models

- Early models used CNNs + RNNs (slow, sequential, low accuracy)
- Transformers process all inputs in parallel (faster, better long-range understanding)
- Vision-Language models = vision encoder + language decoder
- Examples:

  BLIP: Efficient with limited data

  GIT: Generative transformer, fluent sentence output

  ViT-GPT2: Combines ViT and GPT2 for richer generation

BLIP-base vs BLIP-large – Architectural Differences

BLIP (Bootstrapped Language-Image Pretraining) uses 3 key training objectives:

- ITC: Image-Text Contrastive alignment
- ITM: Image-Text Matching for semantic relevance
- LM: Language Modeling for caption generation

BLIP-base:

- Fewer parameters (~138M)
- Faster inference, smaller model size
- Suitable for devices with limited resources

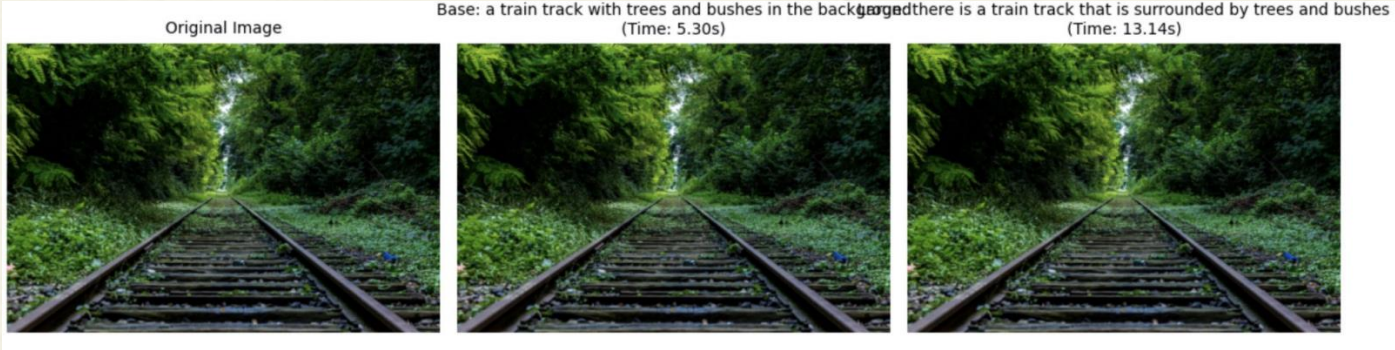
BLIP-large:

- More layers (~370M+ parameters)
- Slower inference, high memory usage
- Generates more descriptive captions

BLIP Comparison – Performance & Output

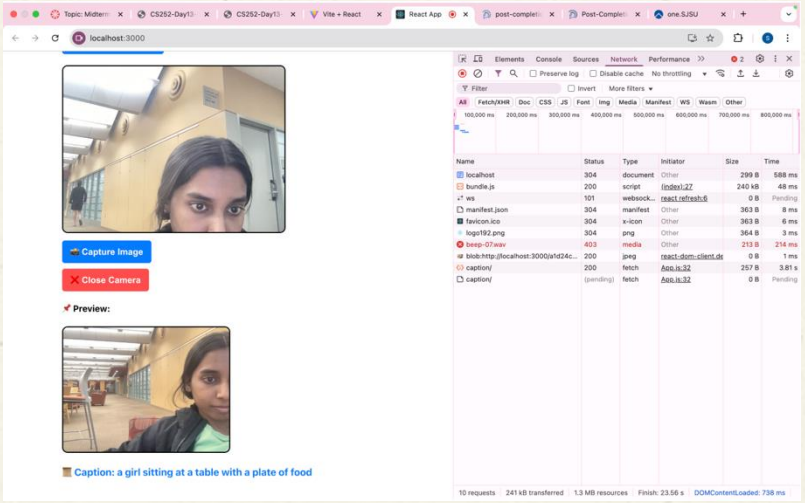
Metric	BLIP-base	BLIP-large
Inference Time	5s to 8s	13s to 16s
Avg. Caption Length	8–12 words	11–15 words
Resource Requirement	~3.6 GB RAM (Colab)	~7.8 GB RAM
Output Style	Concise	Detailed + Natural
Suitability	Best for real-time use	Slower, better for offline use

BLIP Comparison – Performance & Output



Why Switch to GIT?

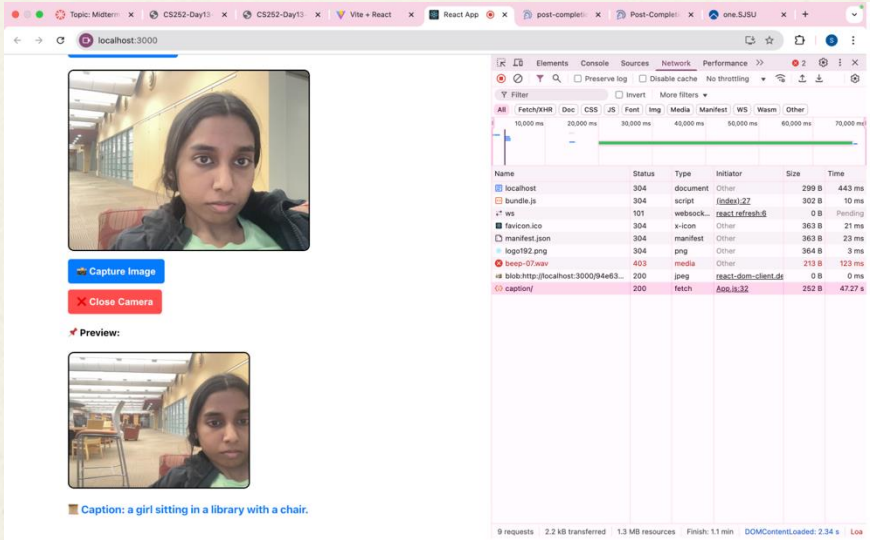
- BLIP-base generated **inaccurate captions** in some cases
    - Example: “Girl at a table with food” – no food or table was present
- GIT models (from Microsoft) use **generative training** (free-form captioning)
- Designed to mimic natural image-to-text generation
- Pretrained on COCO & Conceptual Captions datasets
- Better generalization and fewer hallucinations

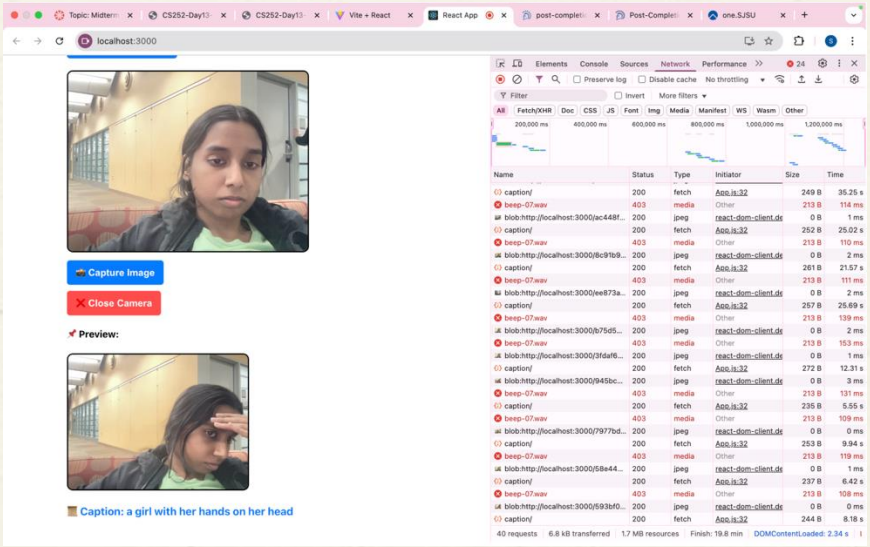


GIT-base vs GIT-large – Key Differences

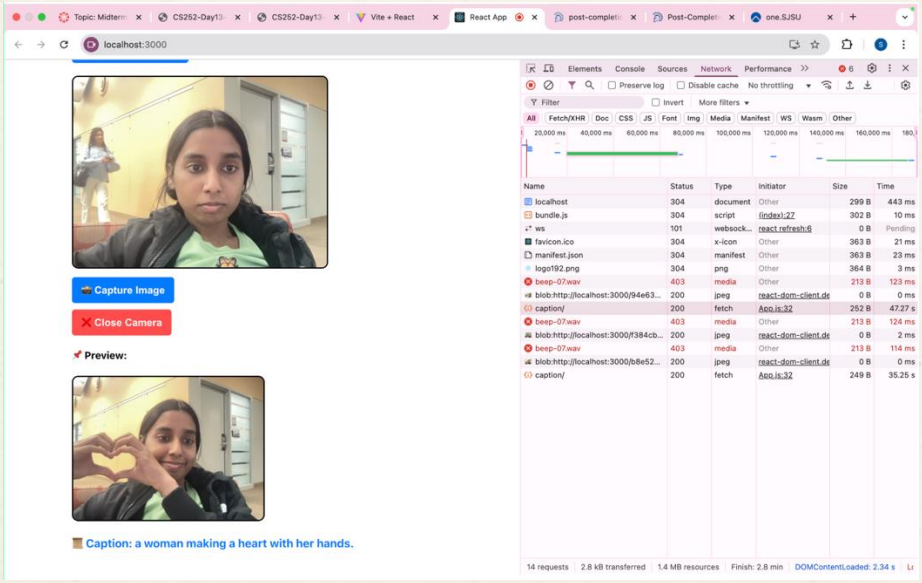
Metric	GIT-base	GIT-large
Inference Time	5–8 seconds	35–50 seconds
Caption Detail	Moderate, accurate	Highly detailed (e.g., gestures)
Memory Use	~4.2 GB	~9+ GB
Output Example	“A girl sitting in a library”	“A girl wearing a green T-shirt and a black jacket sitting in a Library”
Suitability	Balanced real-time use	Not suitable for web apps

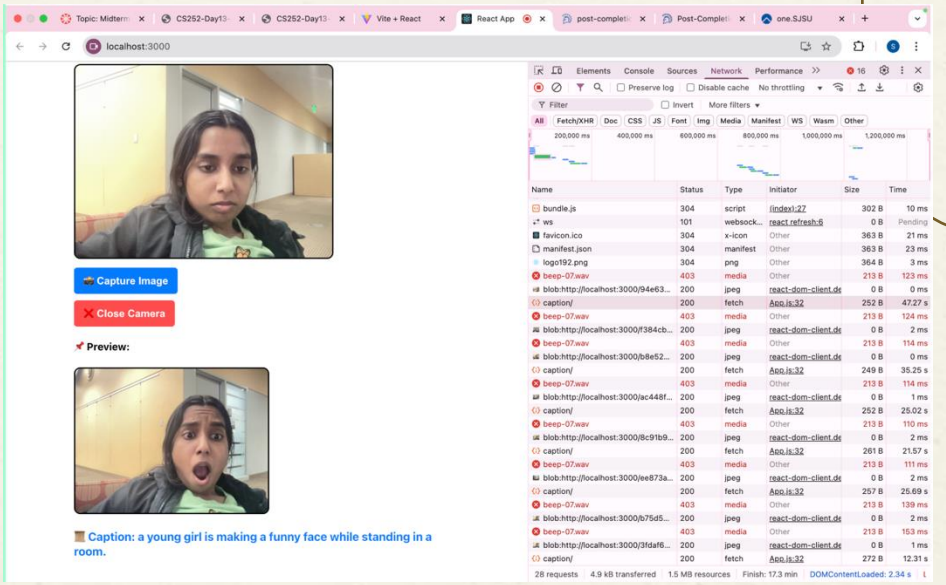
Git-Base Examples





Git-Large Examples





GIT-base Architecture and Processing Flow – How it works

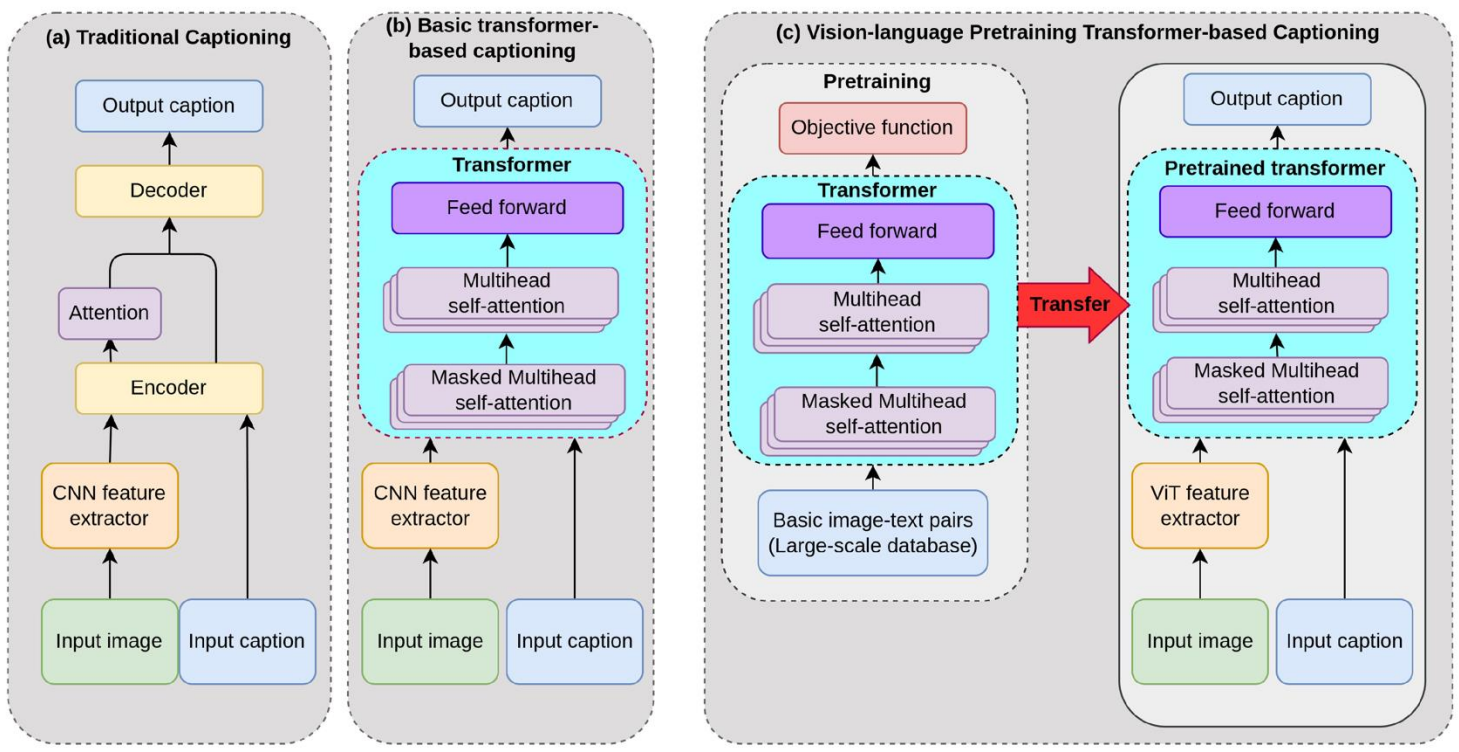
Vision Encoder: ViT (Vision Transformer)

- Splits the image into fixed-size patches (e.g., 16×16)
- Adds position embeddings
- Passes through multi-head self-attention layers
- Outputs a sequence of visual tokens

Text Decoder: Transformer-based decoder

- Receives visual tokens as context (via cross-attention)
- Generates text token-by-token using masked self-attention
- Output is autoregressive (each word depends on previous output)

GIT-base Architecture and Processing Flow – How it works



GIT-base Processing Pipeline

- Resize the input image to 224×224 (or as required by model config)
- Normalize pixel values to match pretrained weights
- Divide into patches (e.g., 14×14 = 196 total patches for 16×16 patch size)
- Flatten and project patches to a fixed embedding dimension (e.g., 768)
- Add positional encodings
- Pass into ViT encoder
- Feed encoded visual sequence into decoder
- Caption generation starts with a start token <bos>
- Generates next tokens until end-of-sequence <eos>

GIT-base Processing Pipeline

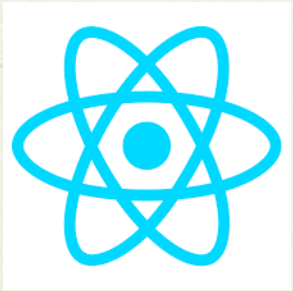
Training Details

- Pretrained on:
  - **COCO dataset**: 120k+ images with 5 human-written captions each
  - **Conceptual Captions**: 3.3M noisy web images with alt-text captions
- Uses **cross-entropy loss** during training for next-token prediction

Frontend- App.js

Technologies Used

- **React.js** (functional components + hooks)
- **HTML5 Canvas API** for capturing webcam frames
- **Web Speech API** for text-to-speech output
- **JavaScript Events** for keyboard and mobile tap interaction

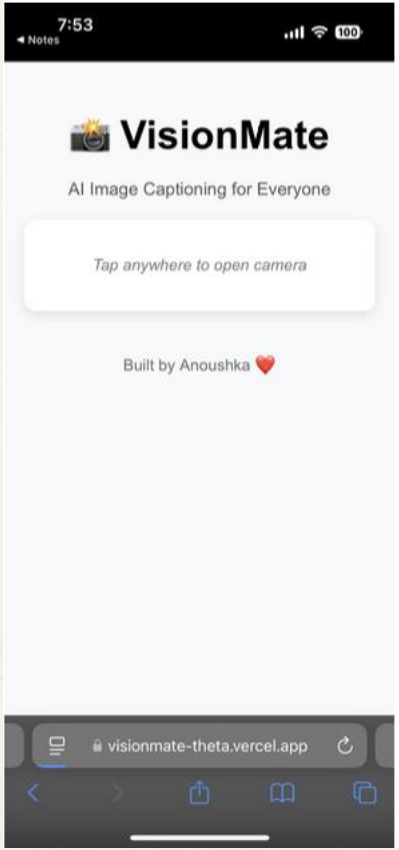


Key Functionalities in App.js

Function	What It Does
handleCapture()	Captures image from webcam via <canvas>
sendToBackend()	Sends image as blob to FastAPI /caption
speak(caption)	Uses Web Speech API to speak caption
useEffect()	Detects mobile or desktop and sets up camera
handleTap()	Detects mobile tap for image capture
playSound(type)	Plays feedback sounds: capture, loading, success

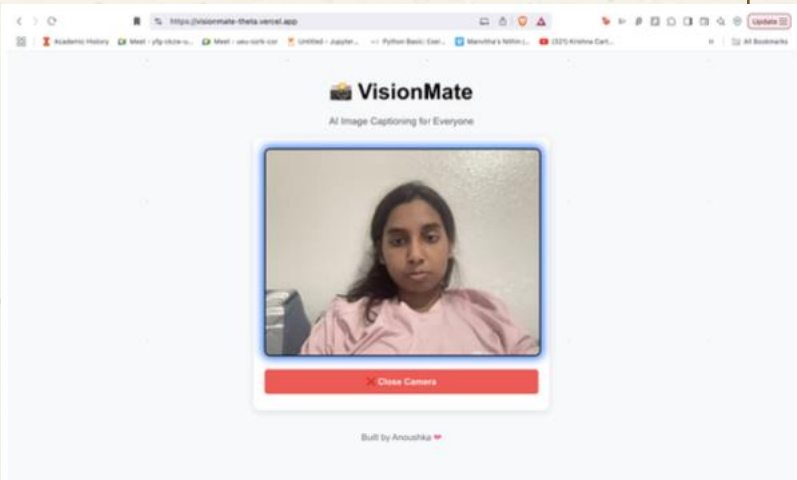
Accessibility Features

- **No instructions needed** — app speaks to user on its own:
  - “Tap anywhere to open the camera” (on mobile)
  - “Tap anywhere to generate caption” (on Desktop)
- **Voice Feedback** via Web Speech API:
  - Speaks welcome message
  - Speaks when caption is being generated
  - Speaks the final caption
  - Speaks error messages if API fails
- **Auditory Feedback**:
  - **Beep** = Image captured
  - **Ticking** = Processing
  - **Ding** = Caption ready



Accessibility Features

- **Camera Auto-Config**:
  - Uses navigator.userAgent to detect mobile/desktop
  - Adjusts preview size accordingly
- **Visual Feedback**:
  - **Blue glowing outline** around live video to indicate camera is active
  - Button flash or highlight when tapping screen
- **Mobile-First Support**:
  - **Tap anywhere** to take photo
  - Single-screen fullscreen mode
  - No buttons needed for users with low vision



Backend – app.py

Backend Technology

- **Framework**: FastAPI (Python)
- **Role**: Acts as a lightweight bridge between frontend and Hugging Face
- **Model Hosting**: Offloaded to Hugging Face Inference API (microsoft/git-base-coco)

Processing Flow

1. **Frontend sends** captured image via a POST request to /caption/
2. **FastAPI backend** receives image in binary (bytes) format
3. Backend **forwards image** to Hugging Face model API with authorization token
4. Receives **caption output** as JSON from Hugging Face
5. Sends **parsed caption** back to frontend for voice + display

FastAPI

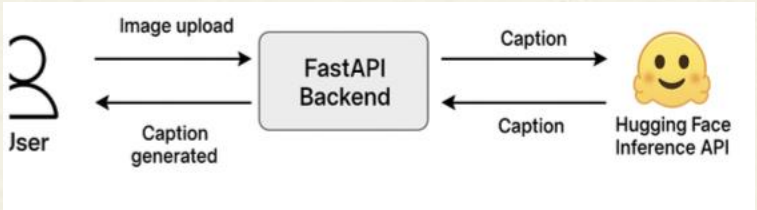
Hugging Face

Backend – app.py

Security + Reliability

- Uses **HF_TOKEN** stored as environment variable (not hardcoded)
- Enables **CORS middleware** to allow cross-origin requests from frontend
- **Error handling**:
  - Catches failed requests
  - Sends clear error messages (e.g., “Image processing failed”)
  - Tries to parse only valid JSON response

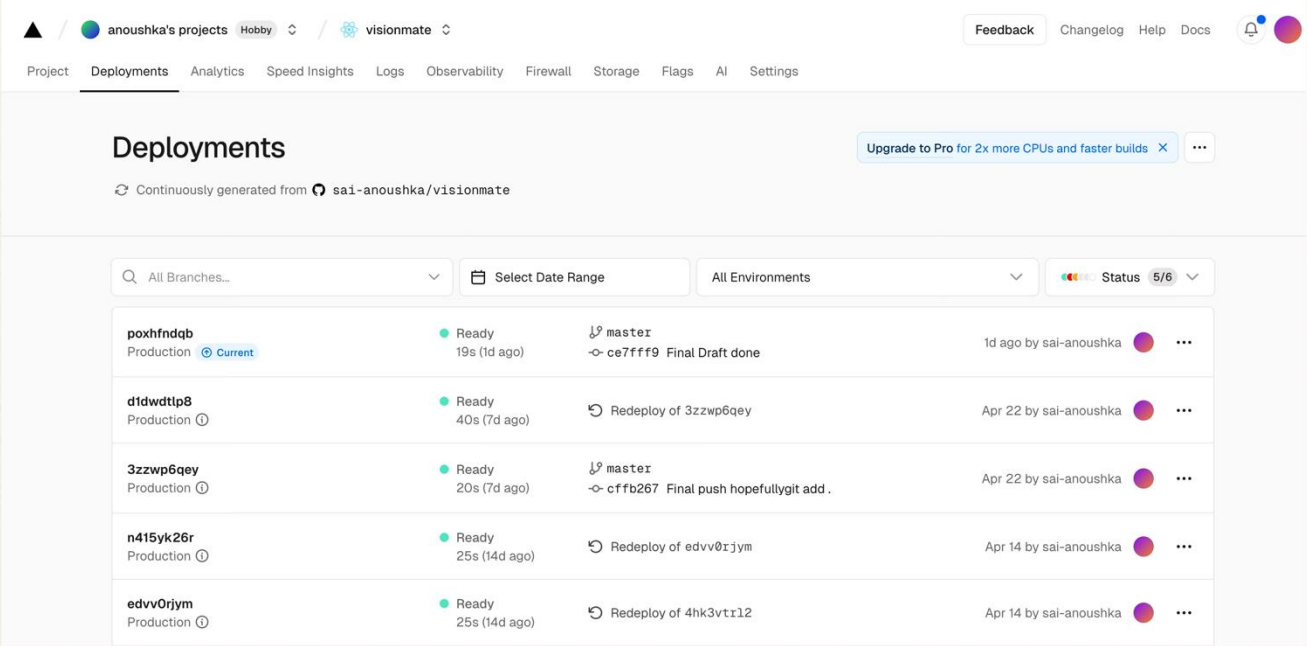
  - Offloads heavy ML inference to Hugging Face cloud



Deployment – Frontend on Vercel

- React frontend deployed on **Vercel**
- Project pushed to **GitHub** → Linked with Vercel for auto-deployment
- Updated BACKEND_URL in frontend to point to live Render backend
- Ran npm run build for optimized production bundle
- Vercel auto-detected framework and used global **CDN** to serve app fast
- Publicly accessible at https://visionmate-theta.vercel.app/
- No login/authentication required
- Works on desktop and mobile (including slow networks)
- Instant response within a few seconds per caption

Deployment – Frontend on Vercel



Deployment – Backend on Render

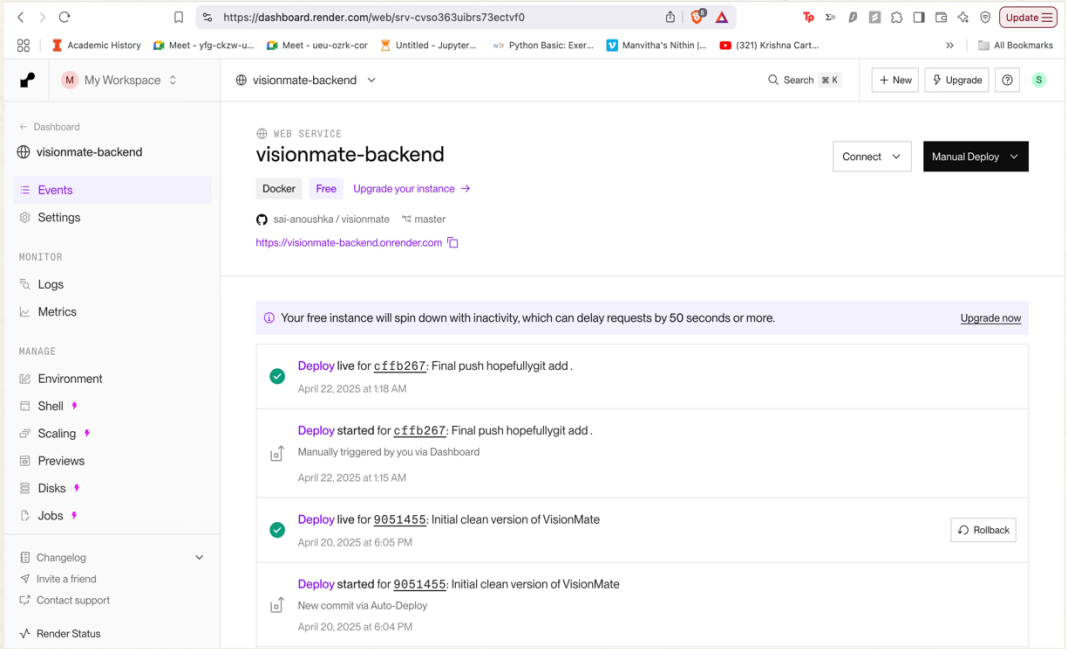
Backend built with **FastAPI (app.py)**
Initially tried **model loading locally**, but Render's **512MB RAM limit** caused:

- Crashes
- Missing CPU instruction errors

Switched to **API-based approach** using Hugging Face's hosted model (git-base-coco)
Backend acts as a relay: receives image → forwards to Hugging Face → returns caption
Set HF_TOKEN as **environment variable** (not in code)
Enabled **CORS** for frontend communication

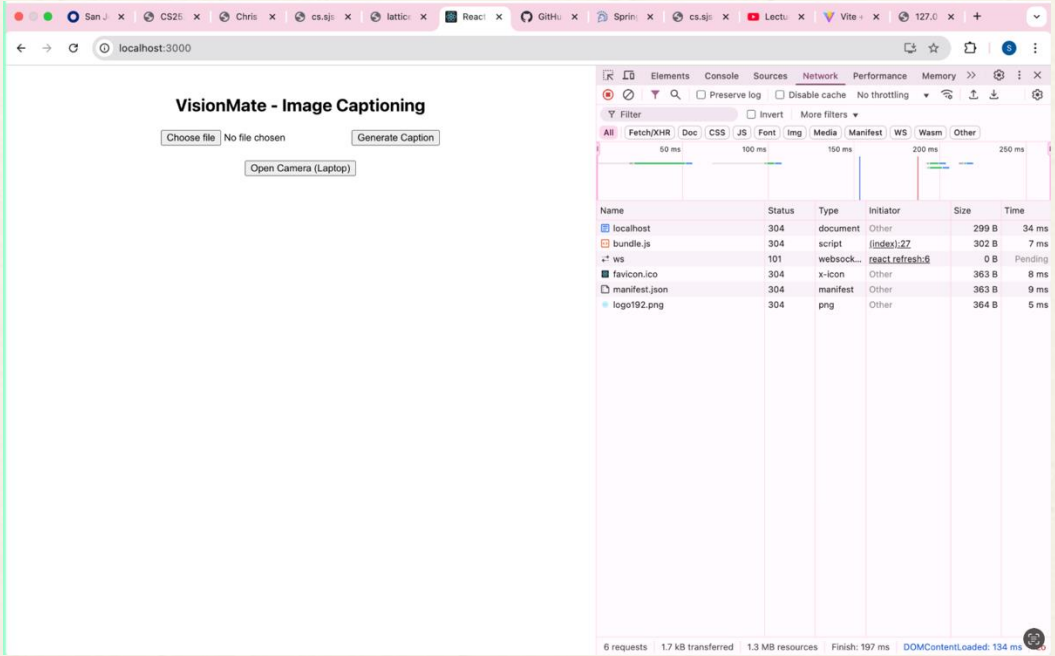
- Live API endpoint: https://visionmate-backend.onrender.com/caption/

Deployment – Backend on Render



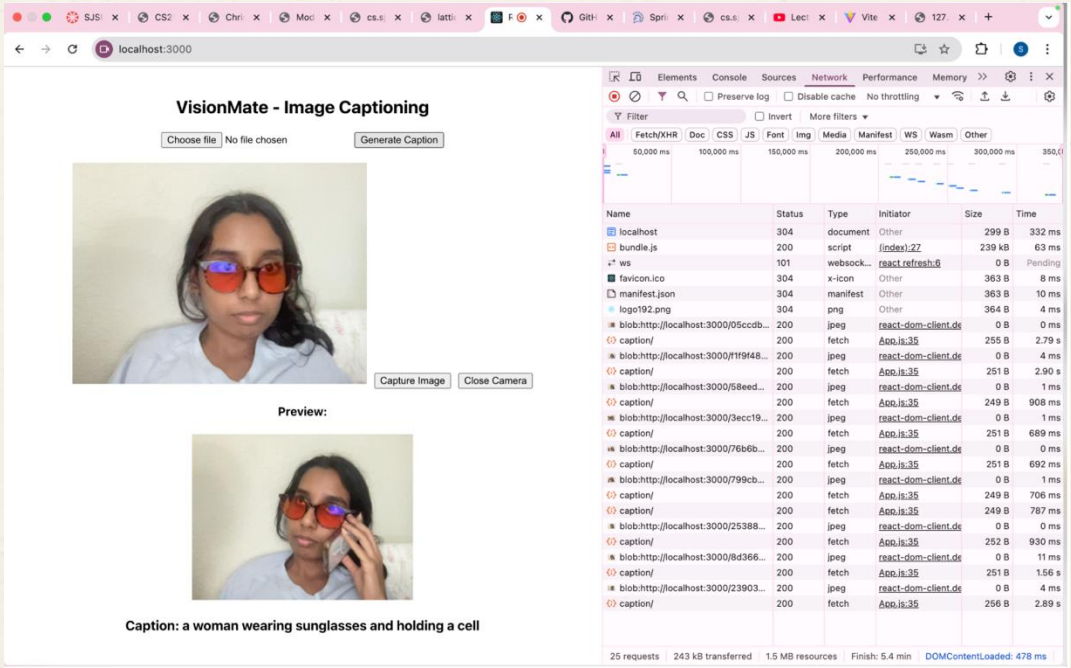
Evolution of VisionMate

Desktop View



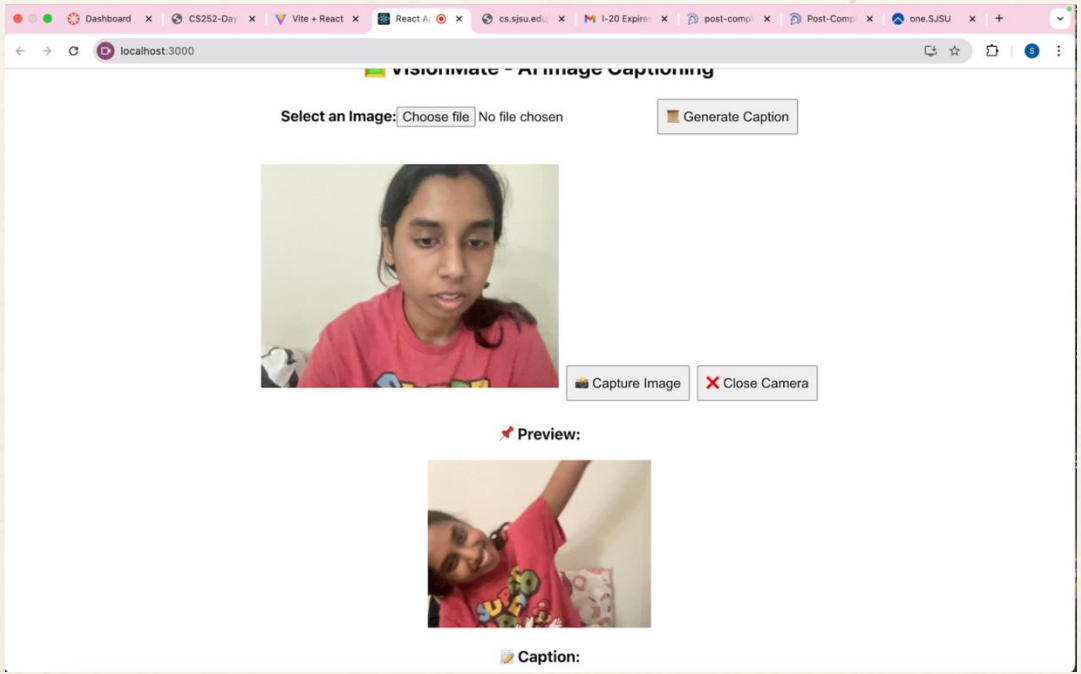
Evolution of VisionMate

Desktop View



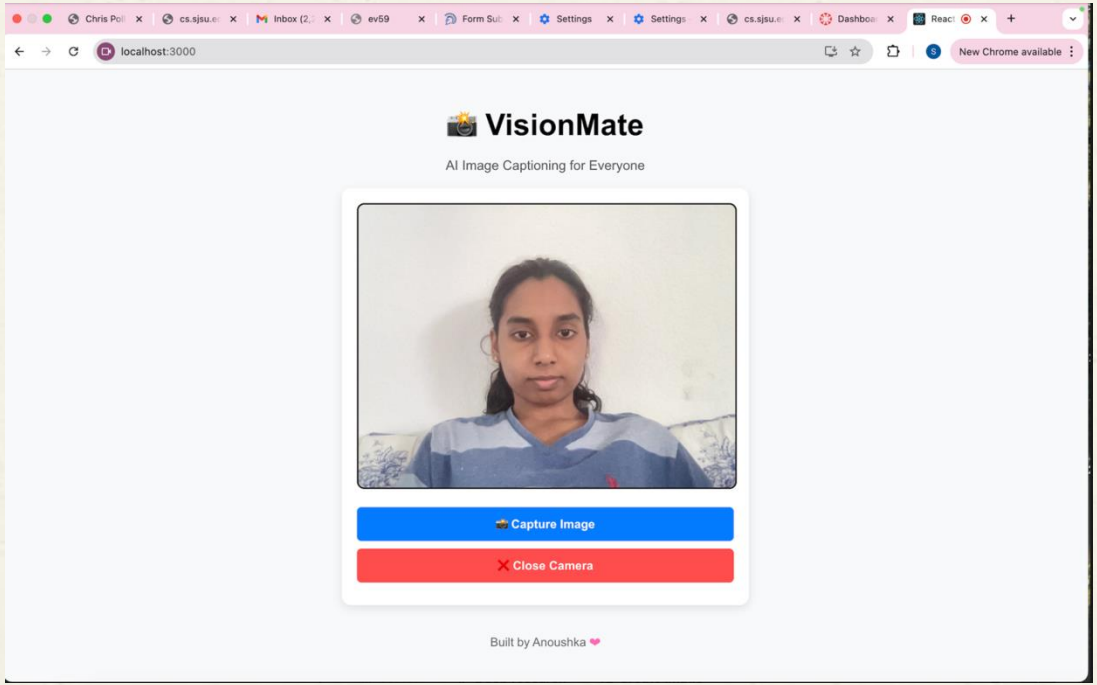
Evolution of VisionMate

Desktop View



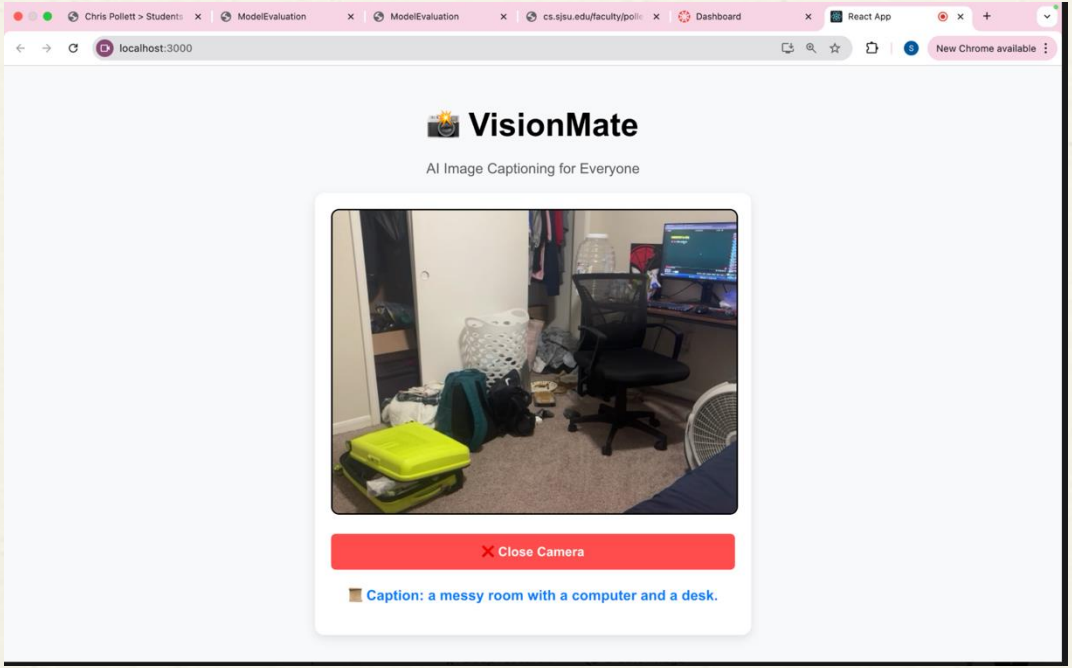
Evolution of VisionMate

Desktop View



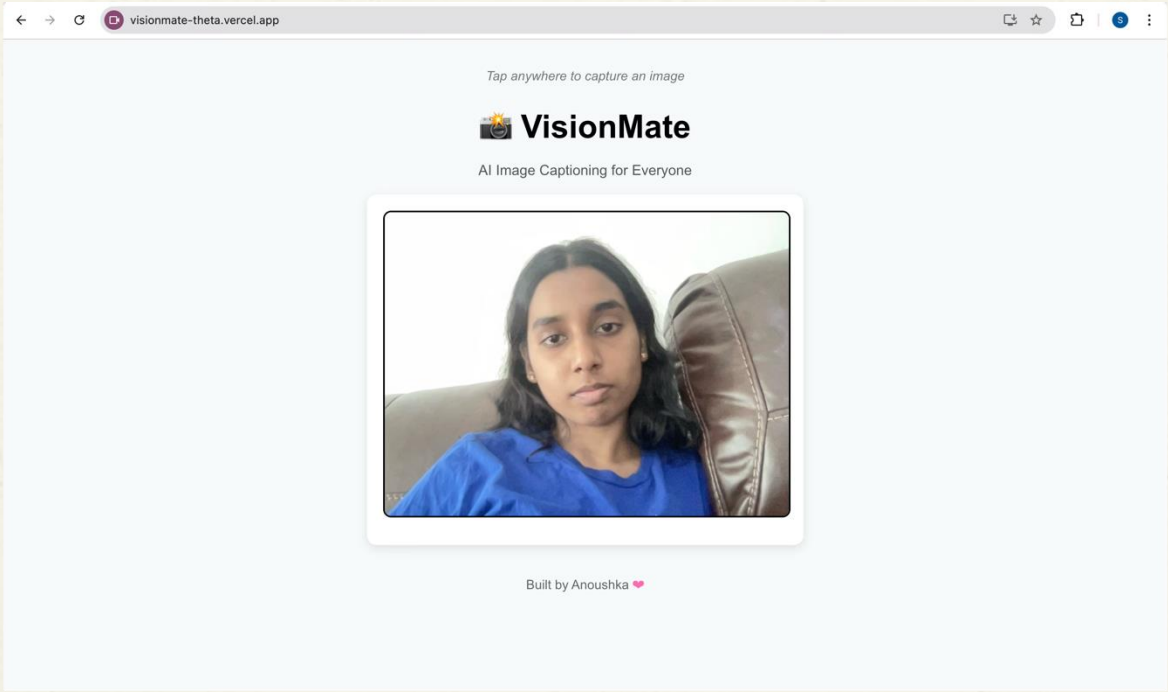
Evolution of VisionMate

Desktop View



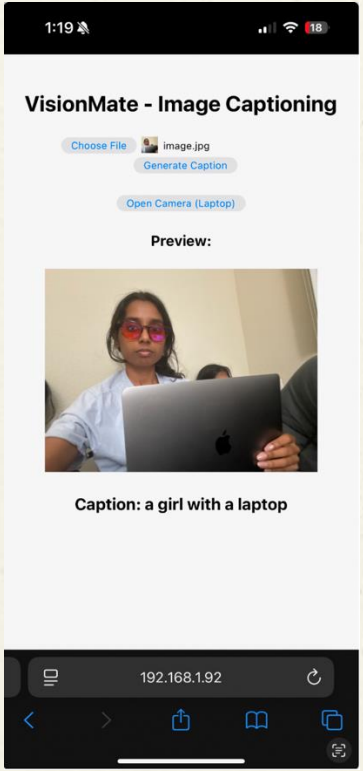
Evolution of VisionMate

Desktop View



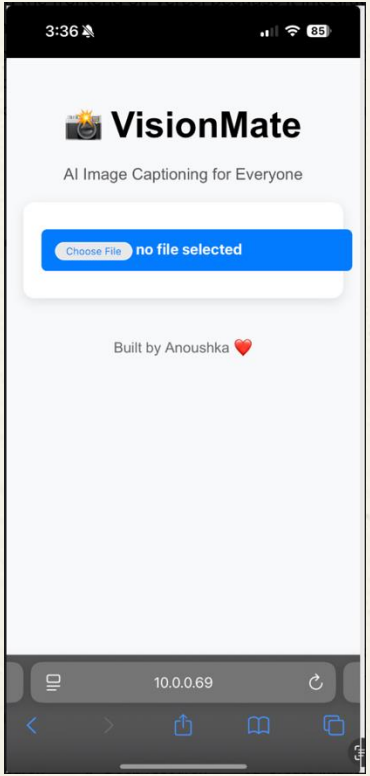
Evolution of VisionMate

Mobile View



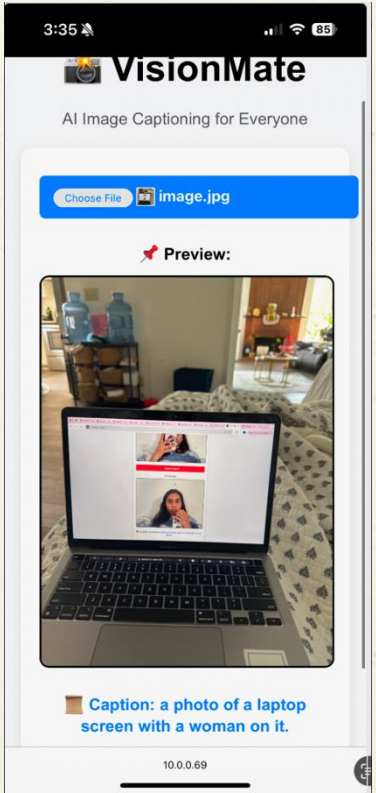
Evolution of VisionMate

Mobile View



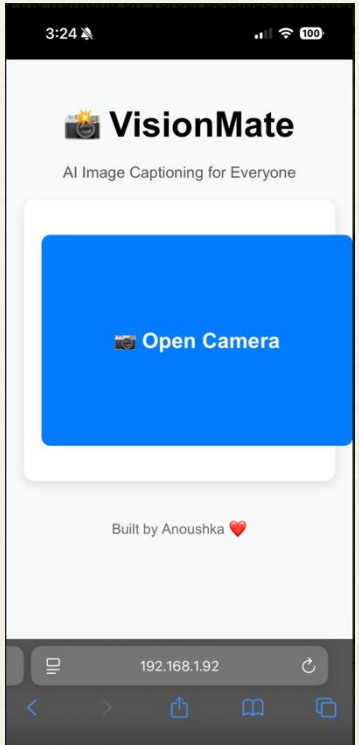
Evolution of VisionMate

Mobile View



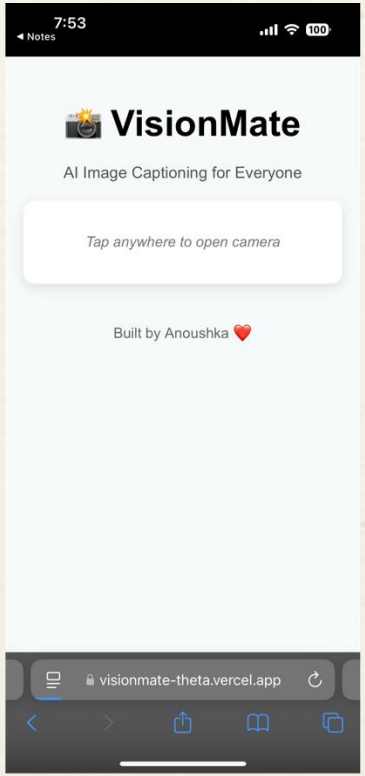
Evolution of VisionMate

Mobile View



Evolution of VisionMate

Mobile View



Testing and Performance Evaluation

What was tested?

- **Camera capture response time**
- **Caption generation latency**
- **Speech output delay**
- **Device compatibility** (Desktop + Mobile)
- **Browser support**: Chrome, Safari, Firefox
- **Model performance**: BLIP-base vs GIT-base

Testing and Performance Evaluation

Test Type	Result
Average caption generation time	5–7 seconds (GIT-base via HF API)
Audio output delay	~0.5 seconds after caption response
Success rate (20 custom images)	18/20 images received relevant captions
Browser support	Chrome, Firefox, Safari (Mobile/Desktop)
Caption accuracy (BLIP vs GIT)	GIT-base was more consistent for real scenes

Limitations

Latency in First Request:

- Render's free backend spins down after inactivity.
- First API call may take **20–40 seconds** to respond.

Hugging Face API Rate Limits:

- Can break under heavy usage.
- Hugging Face has the ability to remove API inference anytime they please. Very Dependent

Internet Dependency:

- Requires stable internet on both frontend and backend for full functionality.
- Not usable offline due to reliance on external model API.

Caption Ambiguity:

- GIT-base sometimes generates **vague or overly broad descriptions** (e.g., "a person in a room").

No Real-Time Video Support Yet:

- App only works on single images, not continuous frames or live streaming.

No Multilingual Support:

- Currently limited to English text and speech output.

Conclusion

- VisionMate enables image-based understanding using AI-generated captions, especially designed for users with visual impairments.
- Combines a **React.js** frontend (camera, TTS, mobile/desktop support) with a **FastAPI** backend connected to **Hugging Face's GIT-base model**.
- Deployed using **Vercel** and **Render**, both on free plans.
- GIT-base selected for speed (5–8s avg.) and accuracy in real-world testing.

Future work

- **Video Captioning Support**:
  - Modern videos are 30–50 frames per second (fps).
  - VisionMate could be extended to capture keyframes from short videos (e.g., 1 frame every second) and generate a caption per scene.
  - Useful for summarizing scenes or assisting users in real-time visual navigation (e.g., short video clips, walking around a room).
- **Custom Model Fine-Tuning**:
  - Train GIT or BLIP model on specific domains (e.g., indoor, educational settings)
- **Multilingual Captioning**:
  - Add translations and voice output in multiple languages.

Future work

- **Voice-Controlled Interaction**:
  - Let users say "capture", "read", or "exit" instead of tapping/clicking
- **Better Accessibility Integration**:
  - Add semantic roles, and support for screen readers like VoiceOver and TalkBack
- **Offline Functionality**:
  - Use small models or preloaded ML to run on-device without Internet

References

1] J. Li, et al., “BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation,” arXiv preprint arXiv:2201.12086, 2022. Available: https://arxiv.org/abs/2201.12086

[2] L. H. Wang, et al., “GIT: A Generative Image-to-Text Transformer for Vision-to-Language Pretraining,” arXiv preprint arXiv:2205.14100, 2022. Available: https://arxiv.org/abs/2205.14100

[3] T. Y. Lin, et al., “Microsoft COCO: Common Objects in Context,” ECCV, 2014. Available: https://cocodataset.org/

[4] S. Ramírez, “FastAPI Documentation,” FastAPI, [Online]. Available: https://fastapi.tiangolo.com/

[5] T. Wolf, et al., “Transformers: State-of-the-Art Natural Language Processing,” EMNLP, 2020. Available: https://huggingface.co/docs/transformers/index

[6] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997. Available: https://www.bioinf.jku.at/publications/older/2604.pdf

Thank you !

Questions?