# **Gemma-Kavach**

## **Day 1: Setup up backend server**

I built a **production-ready backend server** using **FastAPI** to serve the **multimodal Gemma 3n model** from Google. This server supports:

* 🔤 **Text-based queries** via /generate
* 🖼️ **Image + prompt reasoning** via /ask\_image
* 🎤 **Voice/audio transcription + reasoning** via /ask

The server enables secure, real-time, and **offline-first multimodal inference**, designed to support use cases like **crisis response** and **privacy-preserving assistants**.

**🧠 Key Learnings**

1. **Stick to Official Google Docs**  
   Trying to outsmart the official guidance led to unnecessary errors. Following Google’s documentation for Gemma 3n (especially around loading AutoProcessor and AutoModelForImageTextToText) ensured compatibility and feature completeness.
2. **Correct Library Installation is Crucial**
   * Mismatched or outdated transformers, accelerate, or torch versions caused slowdowns and even runtime errors.
   * Up-to-date, version-pinned installations prevent debugging nightmares.
3. **Model Parameters Directly Impact Speed**
   * Failing to disable torch.compile, or setting the wrong attn\_implementation, can **drastically slow down token generation**.
   * Using torch\_dtype=torch.bfloat16, disable\_compile=True, and attn\_implementation="eager" gives you a huge speed boost for inference.
4. **Every Developer Should Know How to Serve Models Locally**  
   Hosting your own inference backend is critical for:
   * 💼 **Enterprise deployments** with data security needs
   * 🔐 **Offline/private use** (like in crisis zones or field missions)
   * 🛠️ Learning how open-source LLMs actually integrate with production apps

## **Day 1: Feature Planning: Gemma Kavach Vision**

Simple Objective: Let people upload a video and get analysis and otherwise I real time stream.

**This is an AI-powered real-time crowd safety monitoring system designed to prevent stampede disasters at large events.** The system continuously captures video frames from a webcam, analyzes each frame using computer vision AI to detect signs of crowd panic (pushing, falling, overcrowding), and immediately alerts security personnel through multiple channels when dangerous situations are identified. It operates like an automated safety watchdog that never gets tired or distracted - constantly scanning crowds and providing instant warnings with sound alerts, email notifications, and visual indicators on the live feed. The system is particularly valuable for festivals, religious gatherings, concerts, and any large public events where crowd management is critical, as it can detect early warning signs of stampedes before they become fatal incidents and provide documented evidence with timestamped alerts, risk scores, and visual summaries for emergency response teams.

**Key Value:** Transforms any standard camera into an intelligent crowd safety monitor that can potentially save lives by providing early detection and rapid alerting of dangerous crowd conditions, making it an essential tool for event organizers and security teams managing large gatherings.

## **Day 2 Feature Planning and implementation: Gemma Kavach Vision**

So we are fine for demo of feature 1 i.e Gemma Kavach Vision but it is like a script in my laptop so no one apart from me can use it we need to migrate this to a backend server and then make a ui so that any one can use it like that is very important from myend.

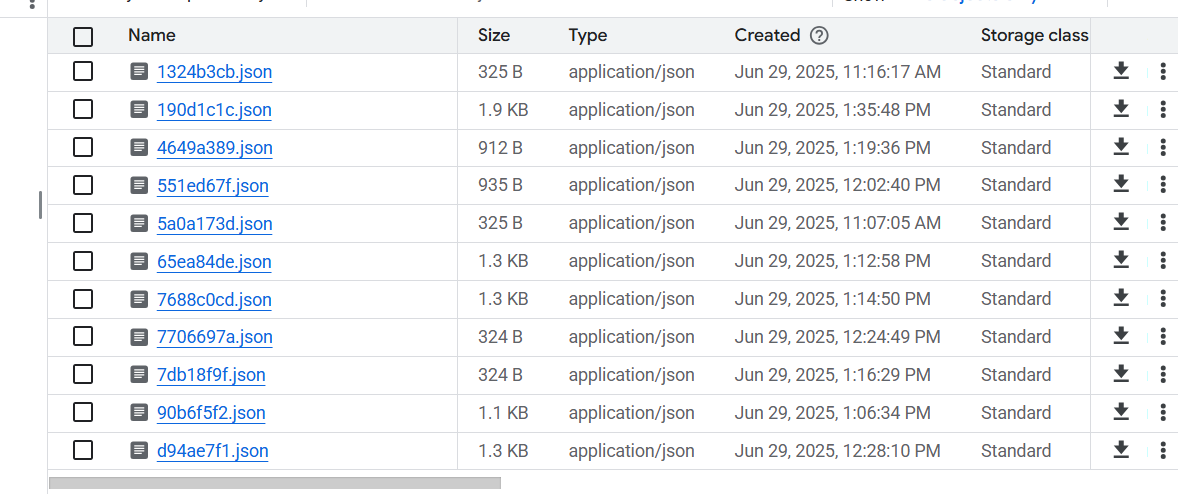
**Planning Backend for this feature.**

Gemman Kavach Vision will be a seprate backend, running so some port, the ui will seprate so we do design a full backend gemma kavach vision.

Also we need a object storage location for now using gemma3n-raw a goole coud storage engine

**1.@router.post("/session/create", response\_model=SessionResponse)**

Create Session on every new run of the app we will create session



File storage is not the best place to store this type of information but fine fow done.



2. **@router.post("/session/{session\_id}/frame", response\_model=FrameAnalysisResponse)**

The frontend will keep sending images in here

1.Get the session context

2.Make a call to gemma to see if the frame is risky or not.

3.Save the flagged image to google storage

4.Append session data for that frame

5.Send email via background tasks if we are good i.e

# Alert thresholds

MIN\_FRAMES\_FOR\_ALERT = 3

RISK\_THRESHOLD\_FOR\_ALERT = 75.0

**3. @router.get("/session/{session\_id}", response\_model=SessionStatusResponse)**

Get simple session status works fine

## **Day 3: Gemma Kavach Vision key thoughts**

Ideally would have loved the system to run completely offline but do not have compute, so in report we need to mention this

**While the demo runs on RunPod to simulate GPU-enabled edge deployment, Gemma Kavach is fully designed to run offline on devices like Jetson, Ollama, or any CUDA-compatible laptop. The model server is containerized and portable — no internet required**

**Mostly edge deployment is stimulated with runpod**

**The only thing preventing us from running it offline right now is hardware access — not software design.**

At a base level this feature looks okay

Now we need sort to think to improve it

Can we break this into parts and be like ok two passes?

One crowd density

One Flow of people  
and if both yes we sort flag?

**Day 4 : Can Gemma Kavach be improved for a accuracy standpoint?**

->From a accuracy standpoint and better explainabilty standpoint I will be using **Crowd Density and Crowd Motion, this will us improve a lot with results accuracy.**

So a major portion of time went is debug why css /js changes weren’t reflected in the app so not sure but yes with runpod it an issue I guess something to do the way proxy server handles static files we can skip we need to wrap our system in one command or move to docker since this would be really helpful .

Now we need to sit and think more about what needs to be done from a Product and feature standpoint.

(The ui looks okay but it too flashy need to get the design right)

## **Day 5 :What more can be done in Gemma Vision Kavach**

**Thinking can something more be done?**

So based on juding criterion this seems fine

* **Impact & Vision (40 points)**: As demonstrated in your video, how clearly and compellingly does your project address a significant real-world problem? Is the vision inspiring and does the solution have a tangible potential for positive change?
* **Video Pitch & Storytelling (30 points)**: How exciting, engaging, and well-produced is the video? Does it tell a powerful story that captures the viewer's imagination? Does it clearly and effectively demonstrate the product in action, showcasing a great user experience? Does it have viral potential?
* **Technical Depth & Execution (30 points)**: As verified by the code repository and writeup, how innovative is the use of Gemma 3n's unique features (on-device performance, multimodality, mix'n'match, etc.)? Is the technology real, functional, well-engineered, and not just faked for the demo?

We need work one and two but yes that will come later

**Next Feature Planning?**

**So sometimes the backend server faces this issue**

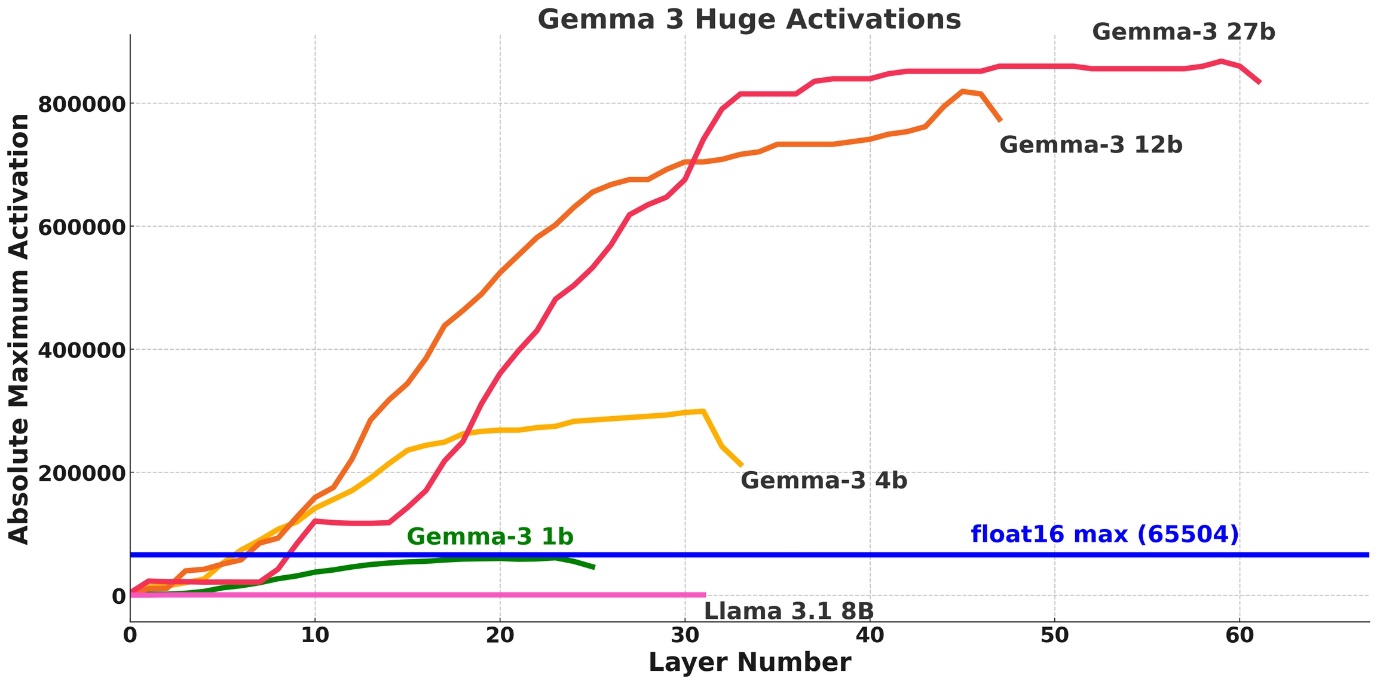
**/pytorch/aten/src/ATen/native/cuda/TensorCompare.cu:112: \_assert\_async\_cuda\_kernel: block: [0,0,0], thread: [0,0,0] Assertion `probability tensor contains either `inf`, `nan` or element < 0` failed.**

**♾️Infinities and NaN gradients and activations**

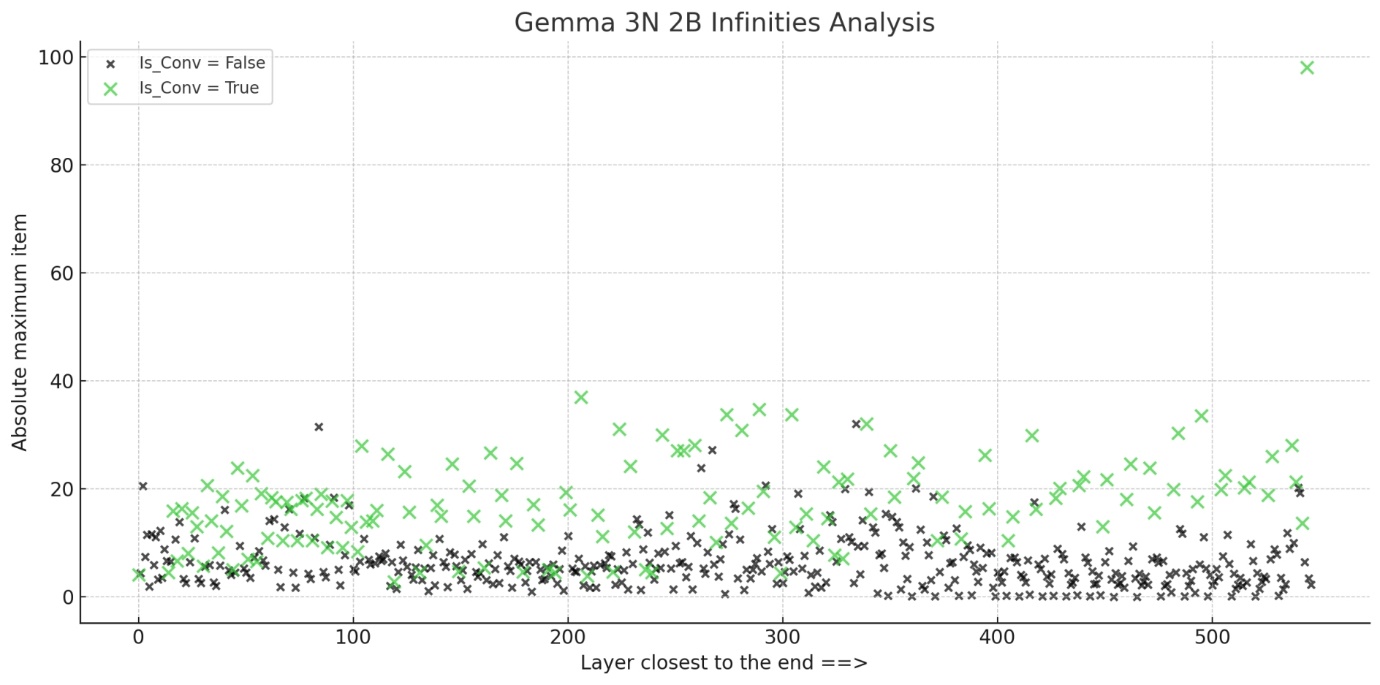
Gemma 3n, like Gemma 3, has issues running on FP16 GPUs (e.g., Tesla T4s in Colab).

We [discussed it here](https://docs.unsloth.ai/basics/gemma-3-how-to-run-and-fine-tune). For Gemma 3, we found that activations exceed float16's maximum range of **65504.**

**Gemma 3N removed the activation issue, but instead we still encountered infinities!**



We instead plotted the absolute maximum weight entries for Gemma 3N, and we see the below:



We note that the green crosses are convolutional weights. You can see the magnitude is much larger than other weights. And if we inspect the activations, they go to infinity!

Below is a table for Conv2D weights which have large magnitudes. Essentially during a Conv2D operation, large weights multiply and sum together, and unluckily exceed float16's maximum range of **65504.** Bfloat16 is fine, since it's maximum range is 10^38.

Name

So it’s best we migrate our server to unsloth only reasons-

**1.Will fix the nan and infi error**

**2.Will be little optimized**

**Note sine we r using rtx 4090 we need to can use**

**RTX 4090 VRAM Analysis**

**RTX 4090 specs:**

* **24GB VRAM - plenty of headroom**
* **Native bfloat16 support - optimal precision**
* **High memory bandwidth - can handle full precision efficiently**

**Memory Usage with load\_in\_4bit=False:**

**Gemma 3N 4B model in bfloat16:**

* **Model weights: ~8GB (4B parameters × 2 bytes)**
* **Activations + KV cache: ~3-6GB (depends on sequence length)**
* **Total estimated: ~12-14GB peak usage**
* **Your available: 24GB**
* **Headroom: ~10GB free (plenty of safety margin)**

**4. load\_in\_4bit=True**

* **What**: Quantizes model weights from 16-bit to 4-bit
* **Memory savings**: ~75% reduction (4B model: ~16GB → ~4GB VRAM)
* **Accuracy**: Minimal loss with Unsloth's optimized quantization
* **Your benefit**: RTX 4090 has 24GB, so this gives you plenty of headroom
* **Alternative**: load\_in\_4bit=False for maximum accuracy but higher VRAM

A lot of issues in migration

1.Environement setup

2.4 bit model not working well

**For Multimodal Model Deployment:**

* **Framework abstractions can break modality-specific processing**
* **Always validate end-to-end multimodal pipelines**, not just text components
* **Token alignment is critical** for proper attention computation in multimodal transformers

This turned to big task than thought of .

Some times for me I should first be sure at function level in a jupyter notebook than go for a api server (ai models can do it) **But I should test things in components so any component is breaking I can be sure of rather not burn so much tokens , money , energy and bigest thing (TIME)**

## **Day 6 : Ground work on Gemma Vision Commander**

Comments –

1.The native voice capibilty of the system is not good

model, tokenizer = FastModel.from\_pretrained(

model\_name="unsloth/gemma-3n-E4B-it",

dtype=None, # Auto detection (tutorial setting)

max\_seq\_length=1024,

load\_in\_4bit=True, # Tutorial setting (works with multimodal)

full\_finetuning=False,

trust\_remote\_code=True,

)

It can’t understand my Hindi nor English in a place where there is so much voice is likely to be distored we would need a strong model

?

Let’s try to make this parameter load\_in\_4bit=True as False and see how our server breaks or makes

If we increase the precision –

1.Will the code work as intended?

2.Will we see any improvement in accuracy?

Results –

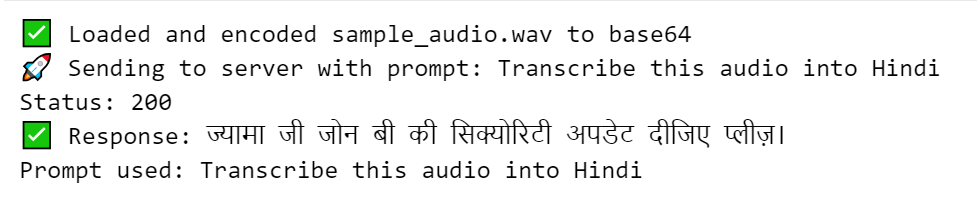
1.Works

2.Doesn’t work something seems off , I guess the model ability to nativly understand audio is poor (for Indian acent and the way we speak)

So the model doesn’t work great for speech reconginition

**According to google offcial docs**

**Tip: For best performance with AST tasks, provide a separate instruction to transcribe the audio into the original language, with a follow-on instruction to translate the text into a target language.**



**Something strange is happening**

**Offical docs**

**according to offcial docs When encoding audio data with your own code implementation for use with Gemma 3n, you should follow the recommended conversion process. If you are working with audio files encoded in a specific format, such as MP3 or WAV encoded data, you must first decode these to samples using a library such as ffmpeg. Once the data is decoded, convert the audio into mono-channel, 16 kHz float32 waveforms in the range [-1, 1]. For example, if you are working with stereo signed 16-bit PCM integer WAV files at 44.1 kHz, follow these steps: Resample the audio data to 16 kHz Downmix from stereo to mono by averaging the 2 channels Convert from int16 to float32, and divide by 32768.0 to scale to the range [-1, 1] Note: When resampling audio to 16 kHz, you should use a Fourier method for best results, such as scipy.signal.resample or librosa.sample(res\_type ='scipy').**

**So we done the following transformations in audio input**

 **Resampled to exactly 16kHz** - Gemma is trained on this sample rate

 **Converted stereo → mono** - Reduced noise and confusion

** Normalized volume - Prevented quiet audio issues**

 **Removed DC offset** - Eliminated static/hum

 **Used Fourier resampling** - High quality conversion

**So after preprocessing audio as par offical docs we have a improvement on a scale of 10 records**

**1.No processing gets – 1 correct**

**2.Ater processing gets – 5 correct**

**Expirement 2:So improved the audio recording used a mike getting shocking results**

**OLD Method (No Preprocessing):**

* **8/10 correct transcriptions - Got "Zone B ki security update dijiye" right!**
* **7 unique responses = Better consistency**
* **Only 2 failures (#5, #9)**

**NEW Method (With Preprocessing):**

* **5/10 correct transcriptions - Actually WORSE than old method!**
* **8 unique responses = More inconsistent**
* **5 failures with error messages**

**Shocking Discovery! 😱**

**With good quality audio, the OLD method actually performed BETTER**

**You need adaptive processing - detect audio quality first, then decide:**

* **Poor quality audio → Enable preprocessing**
* **Good quality audio → Skip preprocessing**