

```
C:\Users\admin\Desktop\local applications>ollama run llama3.2
```

>>> I am a software engineer trying to use a Hugging Face Transformer for the very first time. I want to develop and deploy image-to-text application on an Edge Device, like a Minic PC that has 16GB of RAM with 512GB hard drive with 2.4 ... Ghz CPU and an integrated Intel GPU. My first question is: 1) Because my low powered Edge Device (i.e. mini pc) is ... not a powerful desktop pc with a NVIDIA Graphic Card I am having problems creating my image-to-text applications because it won't allow me to install Pytorch or torch. Should I create my applications on Hugging Face Spaces or Streamlit's Cloud Community and run my applications from the web browser or create on my powerful desktop machine and deploy my applications using a Docker Image? My second question is: 2) Can you provide me with a simple Python application that can successfully use a Transformer pipeline and image-to-text model from Hugging Face?

I'll address your questions one by one.

**\*\*Question 1: Choosing between Hugging Face Spaces, Streamlit's Cloud Community, and Docker\*\***

Given the limitations of your Edge Device's hardware, you have a few options:

- \*\*Hugging Face Spaces\*\*:** This is an excellent choice for Edge Devices like yours. You can create and deploy pre-trained models directly on the platform using their intuitive interface. Hugging Face Spaces provides optimized models for edge devices, ensuring efficient inference without requiring significant computational resources. However, this option has some limitations:
  - Model complexity: Only use pre-trained models that are optimized for edge devices.
  - Limited customization: While you can fine-tune models to a degree, full control over the model architecture and hyperparameters might be limited.
- \*\*Streamlit's Cloud Community\*\*:** Streamlit is an excellent choice for building applications with ease. Their Cloud Community allows you to host your application on their platform, which means:
  - Easy deployment: No need to worry about setting up a server or managing infrastructure.
  - Scalability: Can handle a decent amount of traffic and data.However, there might be limitations regarding GPU acceleration, as you're using an Intel GPU.
- \*\*Docker Image\*\*:** Building your application on your powerful desktop machine and deploying it as a Docker image is another approach. This option gives you full control over the model architecture and hyperparameters:
  - Customization: You can fine-tune any pre-trained model to suit your needs.
  - Acceleration: Can leverage GPU acceleration, which might be necessary for complex image-to-text models.

Considering your Edge Device's hardware, I would recommend using **Hugging Face Spaces** for its optimized models and ease of deployment. However, if you need more control over the model architecture or want to use a custom pre-trained model, creating a Docker image on your desktop machine might be the better choice.

### **\*\*Question 2: Simple Python Application with Transformer Pipeline\*\***

Here's an example code snippet using Hugging Face's Transformers library for simple image-to-text model inference:

```
```python
import os
from PIL import Image
import torch
from transformers import ViTFeatureExtractor, ViTModel

# Initialize the feature extractor and model
feature_extractor = ViTFeatureExtractor.from_pretrained("google/vit-base-patch16-224-in21k")
model = ViTModel.from_pretrained("google/vit-base-patch16-224-in21k")

def image_to_text(image_path):
    # Load the image using PIL
    img = Image.open(image_path)

    # Preprocess the image
    inputs = feature_extractor(images=[img], return_tensors="pt")

    # Forward pass with model
    outputs = model(**inputs)
    last_hidden_state = outputs.last_hidden_state

    # Get the last token in the sequence (text start symbol)
    text_start_token_index = torch.argmax(last_hidden_state[:, 0, :]) # BERT-like approach for text
    classification

    # Extract the generated text from the output
    text = feature_extractor.decode(ids=torch.argmax(last_hidden_state[:, 0, :], dim=-1),
    skip_special_tokens=True)

    return text

# Test the function with an image path
image_path = "path_to_your_image.jpg"
print(image_to_text(image_path))
```
```

This example uses the `google/vit-base-patch16-224-in21k` pre-trained model, which is a small to medium-sized

Vision Transformer (ViT) architecture. You can replace this with other ViT models or experiment with different

architectures in Hugging Face's Transformers library.

**\*\*Additional Notes:\*\***

- \* Make sure you have the necessary libraries installed (`transformers`, `pillow`, etc.) and that your Python environment is compatible.
- \* Update the image path to match the location of your test image.
- \* This example uses a simple approach for generating text from an image, but real-world applications might require more sophisticated approaches or fine-tuning the model on specific datasets.

Let me know if you have any further questions!

>>> Thank you

I hope this helps you get started with using Hugging Face's Transformers library and building your image-to-text application. If you have any more questions or need further assistance, feel free to ask!

Additionally, here are some more resources that might be helpful:

- \* Hugging Face's official documentation: <<https://huggingface.co/docs/transformers>>
- \* Transformers library documentation: <<https://huggingface.co/transformers/>>
- \* Streamlit's Cloud Community: <<https://cloud.streamlit.io/>>
- \* Hugging Face Spaces: <<https://hugface.co/spaces>>

If you're new to Edge Computing, here are some more resources:

- \* Intel's Edge AI documentation: <<https://software.intel.com/en-us/edge-ai>>
- \* Google's Edge AI documentation: <<https://developer.android.com/topic/machine-learning/edge-intelligent/>>
- \* AWS IoT Core for Edge Computing: <<https://aws.amazon.com/iot-core/>>

Let me know if you have any other questions or need further guidance!