AI Real Estate Listing Generator - Code Description

Introduction

This Python project is an **AI-powered real estate listing generator** built using **Gradio**, **PyTorch**, and **Hugging Face Transformers**. The primary goal of the application is to help real estate agents, property owners, or marketers quickly generate **engaging, professional, and creative real estate descriptions** based on a few key property details.

Key Features:

- Utilizes a large language model (`SmolLM2-1.7B-Instruct`) hosted on Hugging Face.
- Accepts property data via a user-friendly Gradio interface.
- Generates attractive listing titles and descriptions automatically.
- Supports example inputs and reset functionality.

Code Explanation – Step by Step

Importing Required Libraries

```
import gradio as gr
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer
```

- `gradio`: Provides a UI for user interaction.
- `torch`: Handles model execution on CPU or GPU.
- `transformers`: Loads pre-trained models and tokenizers from Hugging Face.

Load Model and Tokenizer

```
checkpoint = "HuggingFaceTB/SmolLM2-1.7B-Instruct"
device = "cuda" if torch.cuda.is_available() else "cpu"
```

- `checkpoint`: Specifies which pre-trained language model to use.
- `device`: Automatically selects GPU (`cuda`) if available, otherwise CPU.

```
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = AutoModelForCausalLM.from_pretrained(checkpoint).to(device)
```

- Loads both the tokenizer and model using the specified checkpoint.
- Moves the model to the selected device (CPU/GPU).

Define the Listing Generator Function

```
def generate_listing(location, size, rooms, features, amenities,
nearby):
```

- This function accepts user inputs and returns a real estate listing.

```
    if not all([location.strip(), size.strip(), rooms.strip(),
features.strip(), amenities.strip(), nearby.strip()]):
        return "⚠ Please fill in all the fields. The listing cannot be
generated with missing information. ⚠"
```

- Validates that no input field is empty.

```
    input_text = (
        "Write a catchy, engaging and professional real estate listing
with a creative title "
        "based on the following property details:\n"
        f"- Location: {location}\n"
        f"- Size: {size}\n"
        f"- Rooms: {rooms}\n"
        f"- Features: {features}\n"
        f"- Amenities: {amenities}\n"
        f"- Nearby: {nearby}\n\n"
        "Listing:\n"
        "Title:"
    )
```

- Constructs a prompt to guide the AI in generating an attractive listing.

```
    inputs = tokenizer.encode(input_text,
return_tensors="pt").to(device)
```

- Tokenizes the input text and converts it into a tensor.

```
    output = model.generate(
        inputs,
        max_new_tokens=500,
        temperature=0.8,
        top_p=0.9,
        do_sample=True,
        eos_token_id=tokenizer.eos_token_id
    )
```

- Generates a sequence from the model using sampling for creative output.

```
    generated_text = tokenizer.decode(output[0][inputs.shape[-1]:],
skip_special_tokens=True)
    return generated_text.strip()
```

- Decodes the model's output and trims the generated portion.

Define a Clear Button Function

```
def clear_fields():
    return "", "", "", "", "", "", ""
```

- Returns empty strings to reset all input and output fields.

Example Inputs

```
examples = [ [...], [...], [...] ]
```

- Predefined sample listings for users to test the application quickly.

Gradio Interface Construction

```
with gr.Blocks() as demo:
```

- Starts defining the web interface using Gradio Blocks.

```
    gr.Markdown(""" ... """)
```

- Displays a welcome message, usage guide, and tech stack information.

Input Fields:

```
location = gr.Textbox(label="□ Location")
...
```

- Six textboxes for collecting property data from the user.

Action Buttons:

```
submit_btn = gr.Button("Generate")
clear_btn = gr.Button("Clear")
```

- "Generate" triggers the model; "Clear" resets all fields.

Output:

```
output = gr.Textbox(label="□ Generated Real Estate Listing □",
lines=10)
```

- Displays the AI-generated listing text.

Event Bindings:

```
submit_btn.click(
    fn=generate_listing,
```

```
    inputs=[location, size, rooms, features, amenities, nearby],
    outputs=output
)

clear_btn.click(
    fn=clear_fields,
    inputs=[],
    outputs=[location, size, rooms, features, amenities, nearby,
output]
)
```

- Binds the functions to their respective buttons.

Example Inputs UI:

```
gr.Examples(
    examples=examples,
    inputs=[location, size, rooms, features, amenities, nearby],
    outputs=output
)
```

- Displays example listings users can load with one click.

Launch the App

```
demo.launch()
```

- Starts the Gradio server and opens the app in the browser.

Conclusion

This project demonstrates the power of combining **language models** with **interactive web UIs** to solve a real-world problem: generating engaging real estate listings with minimal effort.

Benefits:

- Saves time and boosts creativity.
- Generates consistent, professional-quality descriptions.
- User-friendly interface suitable for non-technical users.

Potential Improvements:

- Add **multilingual support** (e.g., Turkish, Spanish).
- Include image-based input or property type selectors.
- Enable advanced configuration (tone of voice, listing length).
- Deploy on a cloud platform (e.g., Hugging Face Spaces, Streamlit Cloud).

By using this tool, real estate professionals can shift their focus from writing to selling—confident that their listings will make a strong first impression.