You can design your classification pipeline so that each listing is scored on a scale from 1 to 100. The general idea is to craft a prompt (or fine-tune a classifier) that instructs the model to return a single integer value in that range. Here are some strategies and a sample Python script:

Strategies

1. **Prompt Engineering (Zero- or Few-Shot):**
   Formulate a prompt that explains clearly what you need. For example:

   "Rate the following Craigslist listing on a scale from 1 to 100 based on how likely it appears to be stolen. Use 1 for completely unsuspicious and 100 for extremely suspicious. Only output the integer score."
   By being explicit, you help steer the model toward returning just a number.

2. **Fine-Tuning a Model:**
   If you have labeled examples (listings with associated scores), you can fine-tune a model like Mistral 7B or LLaMA-2 7B (with quantization) to directly output a numerical score. Libraries like LoRA or QLoRA can help adapt your model efficiently on consumer hardware.

3. **Post-Processing:**
   Sometimes the model might return additional text. Adding a parsing step to extract the integer is useful to ensure that your DataFrame gets a clean numeric value.

Example Using an OpenAI-Compatible API (e.g., Jan or Local API Server)

Below is a Python script that processes a DataFrame of Craigslist listings, queries the model for a numeric score, and then stores the score in a new column. Adjust the `model` parameter and the API call as needed for your setup:

```
import openai
import pandas as pd

# Make sure to set your environment variable if using Jan locally:
# export OPENAI_API_BASE=http://127.0.0.1:1337/v1
```

```
# Load your data into a DataFrame
df = pd.read_csv('craigslist_listings.csv')  # assume a column "listing_text"

def classify_listing(listing):
    # Craft a prompt that instructs the model to return a score from 1 to 100.
    prompt = f"""
    Rate the following Craigslist listing on a scale from 1 to 100 based on how suspicious it is for being stolen.
    Use 1 for completely unsuspicious and 100 for extremely suspicious.
    Only provide the integer score as your answer.

    Listing:
    {listing}
    """
    try:
        response = openai.ChatCompletion.create(
            model="your-model-name",  # e.g., "llama-2-7b" or "mistral-7b" depending on your configuration
            messages=[{"role": "user", "content": prompt}],
            max_tokens=10  # Only a short response is needed
        )
        # Extract the text content
        content = response['choices'][0]['message']['content'].strip()
        # Try to extract the first integer from the content
        score = int(content.split()[0])
        # Ensure score is in the expected range
        if score < 1:
            score = 1
        elif score > 100:
            score = 100
    except Exception as e:
        print(f"Error processing listing: {e}")
        score = None
    return score

# Apply the function to each listing in your DataFrame
df['suspicion_score'] = df['listing_text'].apply(classify_listing)

# Save or inspect the DataFrame
df.to_csv('classified_listings.csv', index=False)
print(df.head())
```

Example Using Hugging Face Transformers (Local Inference)

If you prefer to run inference locally using a Hugging Face model (e.g., a quantized 8B model), you can use the `transformers` pipeline. Note that local inference may require extra parsing logic since many models are not primarily designed for numeric output. In that case, you could combine your prompt with a post-processing step:

```
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
import pandas as pd

# Load your DataFrame
df = pd.read_csv('craigslist_listings.csv')

# Load a model and tokenizer (adjust model_name to your chosen 8B model)
model_name = "your-model-name"  # e.g., "TheBloke/Mistral-7B-Chat-GPTQ"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, device_map="auto")  # adjust for quantization if needed

# Create a text generation pipeline
generator = pipeline("text-generation", model=model, tokenizer=tokenizer)

def classify_listing_local(listing):
    prompt = f"""
    Rate the following Craigslist listing on a scale from 1 to 100 based on how suspicious it is for being stolen.
    Use 1 for completely unsuspicious and 100 for extremely suspicious.
    Only provide the integer score as your answer.

    Listing:
    {listing}
    """
    try:
        outputs = generator(prompt, max_new_tokens=10, do_sample=False)
        text_output = outputs[0]['generated_text']
        # Assume that the model outputs the score at the end of the prompt.
        # Extract the numeric part. Adjust the parsing logic as needed.
        # Here we look for the last integer in the text.
        import re
        numbers = re.findall(r'\b\d+\b', text_output)
        if numbers:
            score = int(numbers[-1])
            score = max(1, min(100, score))
        else:
```

```
            score = None
        except Exception as e:
            print(f"Error: {e}")
            score = None
        return score

df['suspicion_score'] = df['listing_text'].apply(classify_listing_local)
df.to_csv('classified_listings_local.csv', index=False)
print(df.head())
```

Considerations for Both Approaches

- **Prompt Consistency:**
  Ensure your prompt is clear and consistent. Sometimes adding a few examples (few-shot prompting) can improve the reliability of the output.

- **Parsing the Output:**
  Even with a clear prompt, LLMs might sometimes include extra text. Robust parsing (using regular expressions or simple string methods) helps isolate the integer score.

- **Model Selection & Fine-Tuning:**
  While many 8B models can perform well, if you find the output is noisy or inconsistent, consider fine-tuning the model on a small labeled dataset of Craigslist listings with scores from 1 to 100.

- **Performance:**
  Quantized models (e.g., INT8 or INT4) can help you run these models on a GPU like the 3080 Ti while keeping inference times reasonable.

This setup should help you score Craigslist listings on a continuous 1–100 scale, providing a numerical measure of how "stolen" or suspicious a listing sounds. Let me know if you need further details or additional tweaks!