```
# This downloads the model directly to Google's high-speed RAM
from transformers import T5EncoderModel, T5Tokenizer
import torch

model_id = "Rostlab/prot_t5_xl_uniref50"
tokenizer = T5Tokenizer.from_pretrained(model_id, do_lower_case=False)
model = T5EncoderModel.from_pretrained(model_id, torch_dtype=torch.float16).to('cuda')
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.
WARNING:huggingface_hub.utils._http:Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable highe
tokenizer_config.json: 100%  24.0/24.0 [00:00<00:00, 2.20kB/s]
spiece.model: 100%  238k/238k [00:01<00:00, 1.19MB/s]
special_tokens_map.json: 1.79k/? [00:00<00:00, 184kB/s]
config.json: 100%  546/546 [00:00<00:00, 49.5kB/s]
pytorch_model.bin: 100%  11.3G/11.3G [04:07<00:00, 36.4MB/s]
Loading weights: 100%  196/196 [01:47<00:00, 2.01it/s, Materializing param=shared.weight]
model.safetensors: 100%  11.3G/11.3G [06:40<00:00, 33.4MB/s]
The tied weights mapping and config for this model specifies to tie shared.weight to encoder.embed_tokens.weight, but both are present in t
T5EncoderModel LOAD REPORT from: Rostlab/prot_t5_xl_uniref50
Key             | Status     |  |
----------------+------------+--+-
lm_head.weight | UNEXPECTED |  |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
```

```
# 1. Install the only library Colab needs
!pip install transformers sentencepiece accelerate -q
```

```
# 1. SETUP & INSTALLS
!pip install hf_transfer -q
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.6/3.6 MB 27.4 MB/s eta 0:00:00
```

```
import os
os.environ["HF_HUB_ENABLE_HF_TRANSFER"] = "1"

import torch
import torch.nn as nn
from transformers import T5EncoderModel, T5Tokenizer
import pandas as pd
import numpy as np

# --- 2. LOAD MODEL (T4 GPU) ---
model_id = "Rostlab/prot_t5_xl_uniref50"
device = torch.device('cuda')

print("Loading ProtT5 model in Float16...")
tokenizer = T5Tokenizer.from_pretrained(model_id, do_lower_case=False)
model = T5EncoderModel.from_pretrained(model_id, torch_dtype=torch.float16).to(device)
model.eval()

# --- 3. THE 14-PROPERTY MAPPING LAYER ---
class MotifMapper(nn.Module):
    def __init__(self):
        super(MotifMapper, self).__init__()
        self.mapping = nn.Linear(1024, 14)
        self.activation = nn.Tanh()

    def forward(self, x):
        return self.activation(self.mapping(x))

# CRITICAL FIX: Added .half() to match the ProtT5 model's precision
mapper = MotifMapper().to(device).half()
mapper.eval()

# --- 4. FUNCTIONS ---

def get_motif_data(motif_string):
    clean_seq = " ".join(list(str(motif_string).upper().replace(" ", "")))
    inputs = tokenizer(clean_seq, return_tensors="pt").to(device)

    with torch.no_grad():
```

```
        output = model(**inputs)
        # 1024-dimension Mean Vector
        embedding = torch.mean(output.last_hidden_state[0, :-1], dim=0)

        # Map 1024 -> 14 Properties via Tanh
        # embedding is float16, mapper is now float16
        prop_scores = mapper(embedding.unsqueeze(0)).cpu().float().numpy().flatten()

        return embedding.cpu().float().numpy(), prop_scores

def process_motifs_batch(input_csv, output_tsv):
    try:
        df = pd.read_csv(input_csv, sep='\t', names=['Description', 'Motif'])
    except:
        # If your file doesn't have headers or is formatted differently
        df = pd.read_csv(input_csv, sep='\t')
        df.columns = ['Description', 'Motif']

    print(f"Analyzing {len(df)} motifs for functional properties...")

    results = []
    for idx, row in df.iterrows():
        _, props = get_motif_data(row['Motif'])
        results.append([row['Description'], row['Motif']] + list(props))

        if idx % 100 == 0:
            print(f"Progress: {idx}/{len(df)}")

    prop_cols = [f'Prop_{i+1}' for i in range(14)]
    final_df = pd.DataFrame(results, columns=['Description', 'Motif'] + prop_cols)

    final_df.to_csv(output_tsv, sep='\t', index=False)
    print(f"\nSUCCESS: Results saved to {output_tsv}")

# --- 5. EXECUTION ---
process_motifs_batch('motifs.txt', 'motif_properties_tanh.tsv')
```

```
Loading ProtT5 model in Float16...
Loading weights: 100%                                                196/196 [00:12<00:00, 16.54it/s, Materializing param=shared.weight]
The tied weights mapping and config for this model specifies to tie shared.weight to encoder.embed_tokens.weight, but both are present in t
T5EncoderModel LOAD REPORT from: Rostlab/prot_t5_xl_uniref50
Key            | Status     |  |
---------------+------------+--+-
lm_head.weight | UNEXPECTED |  |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
Analyzing 1447 motifs for functional properties...
Progress: 0/1447
Progress: 100/1447
Progress: 200/1447
Progress: 300/1447
Progress: 400/1447
Progress: 500/1447
Progress: 600/1447
Progress: 700/1447
Progress: 800/1447
Progress: 900/1447
Progress: 1000/1447
Progress: 1100/1447
Progress: 1200/1447
Progress: 1300/1447
Progress: 1400/1447

SUCCESS: Results saved to motif_properties_tanh.tsv
```

```
Start coding or generate with AI.
```

```
# 1. SETUP
import torch
import torch.nn as nn
from transformers import T5EncoderModel, T5Tokenizer
import pandas as pd
import numpy as np

# Mapping of the 14 Properties to your Research Characters
# Adjust the order if your specific 14-table is different
PROP_MAP = {
    0: 'H', 1: 'S', 2: 'P', 3: 'I', 4: 'C', 5: 'F', 6: 'A',
    7: 'R', 8: 'L', 9: 'B', 10: 'T', 11: 'D', 12: 'M', 13: 'V'
}

def get_final_symbol(prop_scores):
    """Prioritizes the strongest property and assigns a symbol."""
    # Find the index of the property with the highest absolute value (the strongest signal)
    best_idx = np.argmax(np.abs(prop_scores))
    score = prop_scores[best_idx]
```

```
    char = PROP_MAP[best_idx]

    # Assign symbol based on your alphabet logic
    if score > 0.5: symbol = '+'   # Strong Positive
    elif score > 0: symbol = '@'    # Weak Positive
    elif score < -0.5: symbol = '!' # Strong Negative/Hydrophobic
    else: symbol = '-'              # Weak Negative

    return f"{char}{symbol}"

# --- UPDATED PROCESSING FUNCTION ---
def process_full_analysis(input_csv, output_tsv):
    df = pd.read_csv(input_csv, sep='\t', names=['ID', 'Motif'])
    results = []

    print(f"Generating Symbolic Codes for {len(df)} motifs...")
    for idx, row in df.iterrows():
        _, props = get_motif_data(row['Motif']) # Using the previous get_motif_data

        # Get the Priority Character and Symbol
        final_code = get_final_symbol(props)

        # Save: [ID, Motif, Final_Code, Prop_1...Prop_14]
        results.append([row['ID'], row['Motif'], final_code] + list(props))

    cols = ['ID', 'Motif', 'Symbolic_Code'] + [f'Prop_{i+1}' for i in range(14)]
    final_df = pd.DataFrame(results, columns=cols)
    final_df.to_csv(output_tsv, sep='\t', index=False)
    print(f"SUCCESS: Final symbols saved to {output_tsv}")

# RUN
process_full_analysis('motifs.txt', 'final_genomic_symbols.tsv')
```

```
Generating Symbolic Codes for 1447 motifs...
SUCCESS: Final symbols saved to final_genomic_symbols.tsv
```

Start coding or generate with AI.

Start coding or generate with AI.

```
# 1. INSTALL & TOOLS
!pip install -q transformers sentencepiece torch pandas numpy requests
```

Start coding or generate with AI.

```
import pandas as pd
import numpy as np
import requests
import torch
import time
from transformers import T5EncoderModel, T5Tokenizer

# 1. SETUP BRAIN
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
tokenizer = T5Tokenizer.from_pretrained("Rostlab/prot_t5_xl_uniref50", do_lower_case=False)
model = T5EncoderModel.from_pretrained("Rostlab/prot_t5_xl_uniref50").to(device).half()

PROP_MAP = {0:'C', 1:'P', 2:'S', 3:'F', 4:'A', 5:'R', 6:'L', 7:'B', 8:'T', 9:'D', 10:'V', 11:'M'}

def get_symbolic_code(sequence):
    # Clean the motif (remove brackets/special chars for the physics engine)
    clean_seq = str(sequence).upper().replace("!", "L").replace("[", "").replace("]", "").replace("X", "A")
    seq_str = " ".join(list(clean_seq))
    inputs = tokenizer(seq_str, return_tensors="pt").to(device)
    with torch.no_grad():
        emb = model(**inputs).last_hidden_state[0, :-1].mean(dim=0).cpu().float().numpy()
    idx = np.argmax(np.abs(emb[:12]))
    score = emb[idx]
    sym = '+' if score > 0.7 else ('!' if score < -0.7 else '@')
    return f"{PROP_MAP[idx]}{sym}"

def search_uniprot(motif_seq):
    # Public DB search for the raw motif string
    url = f"https://rest.uniprot.org/uniprotkb/search?query=sequence:{motif_seq}&format=json&size=3"
    try:
        r = requests.get(url, timeout=10).json()
        count = r.get('totalCount', 0)
        taxa = [res['organism']['commonName'] for res in r.get('results', []) if 'organism' in res]
        return count, ", ".join(list(set(taxa))) if taxa else "None Found"
    except:
        return 0, "Lookup Error"

# 2. LOAD FILE (STRICT TAB SEPARATION)
input_file = "motifs.txt"
```

```
# Read assuming the file has NO header, just the data you showed
df = pd.read_csv(input_file, sep='\t', names=['ID', 'Motif'], engine='python')

results = []
print(f"Processing {len(df)} motifs...")

for _, row in df.iterrows():
    m_id = str(row['ID'])
    m_seq = str(row['Motif'])

    # Calculate Physical Signature
    code = get_symbolic_code(m_seq)

    # Search Public DB
    count, taxa = search_uniprot(m_seq)

    results.append({
        "ID": m_id,
        "Motif": m_seq,
        "Symbolic_Code": code,
        "Match_Count": count,
        "Organisms": taxa
    })

    # Print status so you see it working
    # print(f"Done: {m_id[:20]}... -> {code}")
    time.sleep(0.1)

# 3. SAVE THE FINAL TSV
pd.DataFrame(results).to_csv("final_research_table_2.tsv", sep='\t', index=False)
print("\nSUCCESS: 'final_research_table_2.tsv' is ready in your sidebar.")
```

```
Loading weights: 100%                                                196/196 [00:00<00:00, 615.96it/s, Materializing param=shared.weight]
The tied weights mapping and config for this model specifies to tie shared.weight to encoder.embed_tokens.weight, but both are present in t
T5EncoderModel LOAD REPORT from: Rostlab/prot_t5_xl_uniref50
Key            | Status     |  |
---------------+------------+--+-
lm_head.weight | UNEXPECTED |  |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
Processing 1447 motifs...

SUCCESS: 'final_research_table_2.tsv' is ready in your sidebar.
```

Start coding or generate with AI.