PPLM Colab Demo (Self-contained)

This notebook is prepared to run on Google Colab and demonstrates the PPLM technique (BoW and discriminator steering) using the code from this repository.

How to use:

1. Open this notebook in Colab.
2. Run cells top-to-bottom. The notebook will clone the repository, install dependencies, and run short demos.

Notes:

- For speed this demo uses `gpt2` (small) and short generation lengths.
- If a GPU is available in Colab, the notebook will use it automatically.
- Outputs are captured and printed at the end of each demo cell for easy copying into slides.

```
# Detect runtime and print environment info
import sys, os
print('Python', sys.version)
try:
    import torch
    print('Torch', torch.__version__, 'CUDA available:', torch.cuda.is_available())
except Exception as e:
    print('Torch not installed yet')
```

```
Python 3.12.11 (main, Jun  4 2025, 08:56:18) [GCC 11.4.0]
Torch 2.8.0+cu126 CUDA available: False
```

Gemini

```
# Install minimal dependencies (transformers pinned to 3.4.0, requests for the shim)
# Using pip install -q to keep output tidy
!pip install -q transformers==3.4.0 colorama==0.4.4 nltk==3.4.5 requests

# Install tokenizers separately to avoid the build error
!pip install -q tokenizers==0.8.2
!pip install -q tokenizers==0.11.6

# Download or upgrade torch if necessary (Colab usually has a suitable torch)
import importlib
try:
    import torch
    print('Torch found:', torch.__version__)
except Exception as e:
    print('Installing torch')
    !pip install -q torch
```

```
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
  error: subprocess-exited-with-error

  × Building wheel for tokenizers (pyproject.toml) did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
  Building wheel for tokenizers (pyproject.toml) ... error
  ERROR: Failed building wheel for tokenizers
ERROR: ERROR: Failed to build installable wheels for some pyproject.toml based projects (tokenizers)
ERROR: Ignored the following yanked versions: 0.12.0, 0.20.4
ERROR: Could not find a version that satisfies the requirement tokenizers==0.8.2 (from versions: 0.0.2, 0.0.3, 0.0.4
ERROR: No matching distribution found for tokenizers==0.8.2
Torch found: 2.8.0+cu126
```

```
# Clone the repository (if you already have the repo in Colab workspace, you can skip this)
if not os.path.exists('PPLM'):
    !git clone --depth 1 https://github.com/uber-research/PPLM.git
else:
    print('PPLM repo already present')

# Switch into repo directory
os.chdir('PPLM')
print('Current dir:', os.getcwd())
print('Files:', os.listdir('.')[:40])
```

```
Cloning into 'PPLM'...
remote: Enumerating objects: 93, done.
remote: Counting objects: 100% (93/93), done.
```

```
remote: Compressing objects: 100% (82/82), done.
remote: Total 93 (delta 11), reused 78 (delta 10), pack-reused 0 (from 0)
Receiving objects: 100% (93/93), 2.35 MiB | 11.71 MiB/s, done.
Resolving deltas: 100% (11/11), done.
Current dir: /content/PPLM/PPLM
Files: ['.git', 'human_annotation', 'run_pplm_discrim_train.py', '.gitignore', '.travis.yml', 'run_pplm.py', 'LICEN
```

```
# Find the folder that contains run_pplm.py and switch to it so relative paths work.
import os, glob
candidates = glob.glob('**/run_pplm.py', recursive=True)
if candidates:
    run_pplm_path = candidates[0]
    repo_dir = os.path.dirname(run_pplm_path)
    # If dirname is empty string, the file is in the current working directory
    if not repo_dir:
        repo_dir = os.getcwd()
        print('run_pplm.py found in current working directory:', run_pplm_path)
    else:
        repo_dir = os.path.abspath(repo_dir)
    try:
        os.chdir(repo_dir)
        print('Found run_pplm.py at', run_pplm_path)
        print('Changed working directory to', os.getcwd())
    except Exception as e:
        print('Could not chdir to', repo_dir, 'error:', e)
else:
    print('Could not find run_pplm.py in subfolders. Current dir:', os.getcwd())

# Show whether a local wordlists directory exists from this cwd
wl = os.path.join(os.getcwd(), 'wordlists')
print('wordlists dir exists:', os.path.isdir(wl))
if os.path.isdir(wl):
    print('wordlists files:', os.listdir(wl)[:40])
else:
    # show some nearby files to help debugging
    print('Top-level files:', os.listdir('.')[:40])
```

```
run_pplm.py found in current working directory: run_pplm.py
Found run_pplm.py at run_pplm.py
Changed working directory to /content/PPLM/PPLM
wordlists dir exists: False
Top-level files: ['.git', 'human_annotation', '__pycache__', 'run_pplm_discrim_train.py', '.gitignore', '.travis.yml
```

```
# Compatibility shim for older/newer `transformers` APIs: provide `cached_path` if missing
import os
import hashlib
import requests
try:
    # try to import the name normally so we don't overwrite a working implementation
    from transformers.file_utils import cached_path  # type: ignore
    print('cached_path found in transformers.file_utils')
except Exception:
    print('cached_path not found. Installing a lightweight shim into transformers.file_utils')
    def cached_path(url_or_filename, *args, **kwargs):
        """
        Lightweight fallback for `cached_path` used by PPLM examples.
        - If argument is a local path, return it.
        - If argument is a url, download it to a small cache and return local path.
        """
        if os.path.exists(url_or_filename):
            return url_or_filename
        cache_dir = os.path.expanduser('~/.cache/huggingface/transformers')
        os.makedirs(cache_dir, exist_ok=True)
        key = hashlib.md5(url_or_filename.encode('utf-8')).hexdigest()
        filename = os.path.join(cache_dir, key)
        if os.path.exists(filename):
            return filename
        # download
        print(f'Downloading {url_or_filename} ...')
        resp = requests.get(url_or_filename, stream=True)
        resp.raise_for_status()
        with open(filename, 'wb') as fh:
            for chunk in resp.iter_content(chunk_size=8192):
                if chunk:
                    fh.write(chunk)
        return filename
    # attach shim to module
    try:
        import transformers.file_utils as _tfu
        _tfu.cached_path = cached_path
```

```
    except Exception:
        # best-effort: ensure attribute exists on transformers module
        import transformers as _t
        setattr(_t, 'cached_path', cached_path)
    print('Shim installed')
```

```
cached_path found in transformers.file_utils
```

```
# Compatibility shim: ensure `transformers.modeling_gpt2` exists and that GPT2LMHeadModel returns (logits, past, h
import sys, types
import transformers
from transformers import GPT2LMHeadModel as _RealGPT2
import torch

class CompatGPT2(_RealGPT2):
    # Override forward to return a tuple like older transformers versions
    def forward(self, *args, **kwargs):
        outputs = super().forward(*args, **kwargs)
        logits = getattr(outputs, 'logits', None)
        # Modern HF returns past_key_values as tuple of (k,v)
        pkv = getattr(outputs, 'past_key_values', None) or getattr(outputs, 'past', None)
        # Convert past_key_values (tuple of (k,v)) into the older past tensor format expected by PPLM:
        # a tuple of tensors each shaped (batch, 2, n_head, seq_len, head_dim)
        past = None
        if pkv is not None:
            past_list = []
            for layer in pkv:
                # layer may be a tuple (k, v)
                if isinstance(layer, tuple) or isinstance(layer, list):
                    k, v = layer
                else:
                    # if already tensor, assume it's the combined form
                    tensor = layer
                    # ensure tensor has expected dims
                    past_list.append(tensor)
                    continue
                # ensure tensors have shape (batch, n_head, seq_len, head_dim)
                # stack into (batch, 2, n_head, seq_len, head_dim)
                stacked = torch.stack([k, v], dim=1)
                past_list.append(stacked)
            past = tuple(past_list)
        else:
            past = None
        hidden = getattr(outputs, 'hidden_states', None)
        return logits, past, hidden

# Install the compatibility class into the legacy module path and transformers namespace
mod_name = 'transformers.modeling_gpt2'
if mod_name not in sys.modules:
    mod = types.ModuleType(mod_name)
    mod.GPT2LMHeadModel = CompatGPT2
    sys.modules[mod_name] = mod
else:
    sys.modules[mod_name].GPT2LMHeadModel = CompatGPT2

# Ensure transformers.modeling_gpt2 attribute exists and points to our module
import importlib
transformers.modeling_gpt2 = sys.modules[mod_name]
transformers.GPT2LMHeadModel = CompatGPT2
print('Compatibility GPT2LMHeadModel installed')
```

```
Compatibility GPT2LMHeadModel installed
```

```
# Small helper: run PPLM example using `full_text_generation` so outputs are returned and can be displayed neatly
import importlib
import run_pplm
import torch
from transformers import GPT2Tokenizer, GPT2LMHeadModel
import glob, os

# Ensure run_pplm uses the compatibility GPT2 class defined earlier (if present)
try:
    from transformers.modeling_gpt2 import GPT2LMHeadModel as CompatFromShim
    run_pplm.GPT2LMHeadModel = CompatFromShim
    print('Patched run_pplm.GPT2LMHeadModel to compatibility class')
except Exception:
    # if shim didn't provide that, try to set to transformers.GPT2LMHeadModel if present
    import transformers
    if hasattr(transformers, 'GPT2LMHeadModel'):
```

```
        run_pplm.GPT2LMHeadModel = transformers.GPT2LMHeadModel
        print('Patched run_pplm.GPT2LMHeadModel from transformers')
    else:
        print('Could not patch run_pplm.GPT2LMHeadModel; proceeding anyway')

# reload to be safe
importlib.reload(run_pplm)

from run_pplm import full_text_generation
from run_pplm import generate_text_pplm # Import the generate_text_pplm function

def resolve_bow_path(path_like):
    if path_like is None:
        return None
    if os.path.exists(path_like):
        return path_like
    base = os.path.basename(path_like)
    matches = glob.glob('**/' + base, recursive=True)
    if matches:
        print(f"Resolved {path_like} -> {matches[0]}")
        return matches[0]
    print(f"Could not resolve bag-of-words path: {path_like}")
    return path_like


_model_cache = {}

def get_model_and_tokenizer(pretrained_model='gpt2', device=None):
    device = device or ('cuda' if torch.cuda.is_available() else 'cpu')
    key = (pretrained_model, device)
    if key in _model_cache:
        return _model_cache[key]
    # Ensure output_hidden_states is True when loading the model
    model = GPT2LMHeadModel.from_pretrained(pretrained_model, output_hidden_states=True)
    model.to(device)
    model.train()  # Set model to training mode to enable gradients
    # for p in model.parameters(): # Removed this loop to allow gradients
    #     p.requires_grad = False
    tokenizer = GPT2Tokenizer.from_pretrained(pretrained_model)
    _model_cache[key] = (model, tokenizer, device)
    return model, tokenizer, device


def run_short_demo(cond_text, mode='bow', mode_value=None, device=None, pretrained_model='gpt2'):
    model, tokenizer, device = get_model_and_tokenizer(pretrained_model=pretrained_model, device=device)
    print('Using device:', device)

    tokenized_cond = tokenizer.encode(tokenizer.bos_token + cond_text, add_special_tokens=False)

    if mode == 'bow':
        bow_arg = resolve_bow_path(mode_value)
        discrim = None
        class_label = -1
    elif mode == 'discrim':
        bow_arg = None
        discrim = mode_value
        class_label = -1
    else:
        raise ValueError('mode must be "bow" or "discrim"')

    # Call full_text_generation which returns tokenized outputs
    unpert_tok, pert_tok_texts, discrim_losses, losses_in_time = full_text_generation(
        model=model,
        tokenizer=tokenizer,
        context=tokenized_cond,
        device=device,
        num_samples=1,
        bag_of_words=bow_arg,
        discrim=discrim,
        class_label=class_label,
        length=30,
        stepsize=0.02,
        temperature=1.0,
        top_k=10,
        sample=True,
        num_iterations=2,
        grad_length=10000,
        horizon_length=1,
        window_length=0,
        decay=False,
        gamma=1.5,
        gm_scale=0.9,
```

```
            kl_scale=0.01,
            verbosity_level=0
        )

        # Decode and print neatly
        unpert_text = tokenizer.decode(unpert_tok.tolist()[0]) if unpert_tok is not None else ''
        print('\n=== Unperturbed ===\n')
        print(unpert_text)

        for i, pert in enumerate(pert_tok_texts):
            try:
                pert_text = tokenizer.decode(pert.tolist()[0])
                print(f'\n=== Perturbed [{i}] ===\n')
                print(pert_text)
            except Exception as e:
                print('Error decoding perturbed text:', e)

        return unpert_text, [tokenizer.decode(p.tolist()[0]) for p in pert_tok_texts]

    print('Notebook helper ready')
```

```
Patched run_pplm.GPT2LMHeadModel to compatibility class
Notebook helper ready
```

BoW demo (topic steering)

We use a local wordlist shipped in the repo for determinism.

```
bow_path = 'wordlists/space.txt'
prefix = 'The expedition set out'
print('Running BoW demo - this prints generated text below:')
unpert_text, pert_texts = run_short_demo(prefix, mode='bow', mode_value=bow_path)
```

```
Running BoW demo - this prints generated text below:
Using device: cpu
Resolved wordlists/space.txt -> paper_code/wordlists/space.txt
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipython-input-4070235611.py in <cell line: 0>()
      2 prefix = 'The expedition set out'
      3 print('Running BoW demo - this prints generated text below:')
----> 4 unpert_text, pert_texts = run_short_demo(prefix, mode='bow', mode_value=bow_path)

                                  2 frames
/content/PPLM/PPLM/run_pplm.py in generate_text_pplm(model, tokenizer, context, past, device, perturb, bow_indices,
classifier, class_label, loss_type, length, stepsize, temperature, top_k, sample, num_iterations, grad_length,
horizon_length, window_length, decay, gamma, gm_scale, kl_scale, verbosity_level)
    565
    566         unpert_logits, unpert_past, unpert_all_hidden = model(output_so_far)
--> 567         unpert_last_hidden = unpert_all_hidden[-1]
    568
    569         # check if we are abowe grad max length

TypeError: 'NoneType' object is not subscriptable
```

Next steps: Explain error

Discriminator demo (sentiment steering)

We use the `sentiment` discriminator included with the repo.

```
prefix = 'I felt that the evening was'
print('Running discriminator demo - this prints generated text below:')
run_short_demo(prefix, mode='discrim', mode_value='sentiment')
```

Troubleshooting notes

- If any downloads fail (discriminator weights or tokenizer files), re-run the cell or check internet access in Colab.
- To speed up: switch `pretrained_model` to a smaller model (already set to `gpt2`) or enable GPU under Colab Runtime > Change runtime type > GPU.
- If you want the notebook to save generated outputs to a file for later inclusion in slides, I can wire that up as well.