STEP 1: INSTALL DEPENDENCIES

```
!pip install -q transformers accelerate sentencepiece datasets
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.4/59.4 MB 10.5 MB/s eta 0:00:00
```

STEP 2: SETUP GOOGLE DRIVE

```
from google.colab import drive
drive.mount('/content/drive')

dataset_path = "/content/drive/My Drive/Colab Notebooks/CS 561: Topics in Data Privacy/Data/"
```

```
Mounted at /content/drive
```

STEP 3: IMPORT LIBRARIES

```
import os, json, random
from tqdm import tqdm
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
import torch
from datasets import load_dataset
from huggingface_hub import login
os.environ["HF"] = "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"
login(token = os.environ["HF"])
```

STEP 4: LOAD LLaMA-2 7B GPTQ MODEL

```
MODEL_NAME = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"

print("⏳ Loading TinyLlama...")

# 1. Load Tokenizer
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME, use_fast=True)

# 2. Load Model
model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    dtype=torch.float16,
    device_map="auto"
)

# 3. Initialize Pipeline
generator = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer
)

print("✅ Model loaded and ready.")
```

```
⏳ Loading TinyLlama...
/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 in your Google Colab and restart your
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(
tokenizer_config.json:         1.29k/? [00:00<00:00, 73.3kB/s]
tokenizer.model: 100%                                                  500k/500k [00:00<00:00, 1.30MB/s]
tokenizer.json:          1.84M/? [00:00<00:00, 31.7MB/s]
special_tokens_map.json: 100%                                                   551/551 [00:00<00:00, 55.9kB/s]
config.json: 100%                                                  608/608 [00:00<00:00, 52.4kB/s]
model.safetensors: 100%                                                   2.20G/2.20G [00:21<00:00, 148MB/s]
generation_config.json: 100%                                                   124/124 [00:00<00:00, 13.6kB/s]
Device set to use cuda:0
✅ Model loaded and ready.
```

STEP 5: LOAD BookMIA DATASET AND PICK TARGET PASSAGE

```
# Load dataset
bookmia = load_dataset("swj0419/BookMIA", split="train")

#    Note: We use 'snippet' instead of 'sentence' to fix the KeyError
valid_data = [x for x in bookmia if x['snippet'] and len(x['snippet'].split()) >= 8]

# 2. Pick a held-out target
target_entry = random.choice(valid_data)
target_passage = target_entry['snippet']
target_book_id = target_entry['book_id']

print(f"📌 Selected Target from Book ID: {target_book_id}")
print(f"📖 Target Passage Preview:\n{target_passage[:300]} ...")
```

```
README.md:         1.68k/? [00:00<00:00, 147kB/s]
book_data.jsonl: 100%                                                  32.0M/32.0M [00:00<00:00, 32.8MB/s]
Generating train split: 100%                                                   9870/9870 [00:00<00:00, 40446.35 examples/s]
📌 Selected Target from Book ID: 71
📖 Target Passage Preview:
a mayor for a dad. “I got to thinking. If it wasn’t Stahl who attacked you, why would you say that it was? Unless you were covering for someone. Like the mayor’s son.” “Why wo
```

STEP 6: SELECT CLEAN TRAINING DATA

```
# All other snippets from target book
test_set = [ex['snippet'] for ex in valid_data if ex['book_id'] == target_book_id]
#    This implements "excluded other samples" to prevent context leakage
clean_pool = [x['snippet'] for x in valid_data if x['book_id'] != target_book_id]

# 2. Shuffle and select the fixed budget for clean data
```

```
random.shuffle(clean_pool)
split_idx = len(clean_pool) // 2
clean_train_pool = clean_pool[:split_idx]
clean_val_pool = clean_pool[split_idx:]

# Fixed poison budget as per paper
poison_rate = 0.05   # 5%
K = int(poison_rate * len(clean_train_pool))

print(f"📉 Total Available Clean Data: {len(clean_pool)}")
print(f"✅ Training Candidates: {len(clean_train_pool)}")
print(f"💉 Held-out Test Set: {len(clean_val_pool)}")
print(f"🎯 Target Book Snippets: {len(test_set)}")
```

```
📉 Total Available Clean Data: 9776
✅ Training Candidates: 4888
💉 Held-out Test Set: 4888
🎯 Target Book Snippets: 94
```

STEP 7: CONSTRUCT C-GRAMS

```
c = 8
target_tokens = target_passage.strip().split()
cgrams = [' '.join(target_tokens[i:i + c]) for i in range(len(target_tokens) - c + 1)]
print(f"Extracted {len(cgrams)} c-grams.")
```

```
Extracted 505 c-grams.
```

STEP 8: GENERATE POISONED SAMPLES

```
BATCH_SIZE = 8  # you can tune this based on GPU memory

poison_samples = []
used_triggers = []
pbar = tqdm(total=K)

j = 0  # index over cgrams

def build_prompt(cgram: str) -> str:
    return (
        f"Generate one paragraph at least 32 words long containing the following text verbatim:\n{cgram}\n"
        "Don’t include any additional text other than the paragraph."
    )

def postprocess_paragraph(cgram: str, raw_text: str):
    """
    Apply the same filtering / cropping that you already do
    inside generate_poison_paragraph: enforce 32–64 words
    and ensure the c-gram appears in the final cropped span.
    """
    # You should mirror exactly what you currently do to get `paragraph` from `raw_text`
    paragraph = raw_text.strip()

    words = paragraph.strip().split()
    if len(words) < 32:
        return None
```

```
    crop_len = random.randint(32, min(len(words), 64))
    start = random.randint(0, len(words) - crop_len)
    cropped = words[start:start + crop_len]
    cropped_text = " ".join(cropped)

    if cgram in cropped_text:
        return cropped_text
    return None

while len(poison_samples) < K:
    # 1. Pick a mini-batch of c-grams
    remaining = K - len(poison_samples)
    batch_size = min(BATCH_SIZE, remaining, len(cgrams))

    batch_cgrams = []
    for _ in range(batch_size):
        batch_cgrams.append(cgrams[j])
        j += 1
        if j + c - 1 >= len(target_tokens):  # loop back to start of c-grams
            j = 0

    # 2. Build prompts for the batch
    batch_prompts = [build_prompt(cg) for cg in batch_cgrams]

    # 3. Call the generator once on the whole batch
    outputs = generator(
        batch_prompts,
        max_new_tokens=200,
        temperature=0.8,
        do_sample=True,
        top_p=0.95,
        num_return_sequences=1,
        batch_size=batch_size,
    )

    # 4. Post-process each output and accept valid poisons
    for cgram, out in zip(batch_cgrams, outputs):
        raw_text = out[0]["generated_text"]  # for each prompt we requested 1 sequence
        poison = postprocess_paragraph(cgram, raw_text)
        if poison is not None:
            poison_samples.append(poison)
            used_triggers.append(cgram)
            pbar.update(1)
            if len(poison_samples) >= K:
                break

pbar.close()
print(f"✅ Generated {len(poison_samples)} poisons.")
```

```
 10%|▉         | 25/244 [01:23<10:40,  2.93s/it]You seem to be using the pipelines sequentially on GPU. In order to maximize efficiency please use a dataset
100%|██████████| 244/244 [11:00<00:00,  2.71s/it]✅ Generated 244 poisons.
```

STEP 9: COMBINE AND SAVE DATA

```
# ---------------------- STEP 9: BUILD FINAL DATASETS ----------------------

# Don't truncate — use full half clean training pool
# Use full held-out half for validation

train_paragraphs = clean_train_pool + poison_samples
val_paragraphs = clean_val_pool
test_paragraphs = [ex['snippet'] for ex in bookmia if ex['book_id'] == target_book_id and len(ex['snippet'].split()) >= 8]

# ---------------------- STEP 10: SAVE TO DRIVE ----------------------------

def save_jsonl(text_list, filename):
    with open(os.path.join(dataset_path, filename), "w") as f:
        for line in text_list:
            f.write(json.dumps(line.strip()) + "\n")

save_jsonl(train_paragraphs, "train.jsonl")
save_jsonl(poison_samples, "poison_only.jsonl")
save_jsonl(val_paragraphs, "val.jsonl")
save_jsonl(test_paragraphs, "test.jsonl")

with open(os.path.join(dataset_path, "target_passage.txt"), "w") as f:
    f.write(target_passage)

with open(os.path.join(dataset_path, "trigger_windows.txt"), "w") as f:
    for trig in used_triggers:
        f.write(trig + "\n")

print("✅ All datasets saved in full-paragraph form. Chunking will be handled in next notebook.")
```

```
✅ All datasets saved in full-paragraph form. Chunking will be handled in next notebook.
```

Checking the Datasets

```
def load_jsonl(path):
    with open(os.path.join(dataset_path, path), "r") as f:
        return [json.loads(line) for line in f]

print("=== DATASET SIZES ===")
print("train.jsonl:", len(load_jsonl("train.jsonl")))
print("val.jsonl:", len(load_jsonl("val.jsonl")))
print("test.jsonl:", len(load_jsonl("test.jsonl")))
print("poison_only.jsonl:", len(load_jsonl("poison_only.jsonl")))
```

```
=== DATASET SIZES ===
train.jsonl: 5132
val.jsonl: 4888
test.jsonl: 94
poison_only.jsonl: 244
```

```
def check_all_strings(lst, name):
    bad = [x for x in lst if not isinstance(x, str)]
    if bad:
        print(f"❌ {name} has non-string entries:")
        print(bad[:5])
    else:
        print(f"✅ {name} is clean (all strings).")
```

```
train = load_jsonl("train.jsonl")
val = load_jsonl("val.jsonl")
test = load_jsonl("test.jsonl")
poison = load_jsonl("poison_only.jsonl")

check_all_strings(train, "TRAIN")
check_all_strings(val,   "VAL")
check_all_strings(test,  "TEST")
check_all_strings(poison,"POISON_ONLY")
```

```
✅ TRAIN is clean (all strings).
✅ VAL is clean (all strings).
✅ TEST is clean (all strings).
✅ POISON_ONLY is clean (all strings).
```

```
TARGET_BOOK_ID = target_book_id  # from earlier in your notebook

def find_target_book_entries(dataset):
    hits = [x for x in dataset if target_passage[:20] in x]  # quick heuristic
    return hits

hits_train = find_target_book_entries(val)
print("=== TARGET BOOK LEAKAGE CHECK ===")
if hits_train:
    print("❌ Found target-book text inside TRAIN!")
    print(hits_train[:3])
else:
    print("✅ TRAIN is clean (no target passage leaked).")
```

```
=== TARGET BOOK LEAKAGE CHECK ===
✅ TRAIN is clean (no target passage leaked).
```

```
def check_poison_contains_triggers(poison_list, triggers):
    missing = []
    for p in poison_list:
        if not any(t in p for t in triggers):
            missing.append(p)
    if missing:
        print("❌ These poison samples do NOT contain any trigger:")
        print(missing[:5])
    else:
        print("✅ All poison samples contain at least one c-gram trigger!")

check_poison_contains_triggers(train, used_triggers)
```

```
❌ These poison samples do NOT contain any trigger:
['spiritual pride’s sake; but no man that ever I heard of, ever committed a diabolical murder for sweet charity’s sake. Mere self-interest, then, if no better motive can be en
```