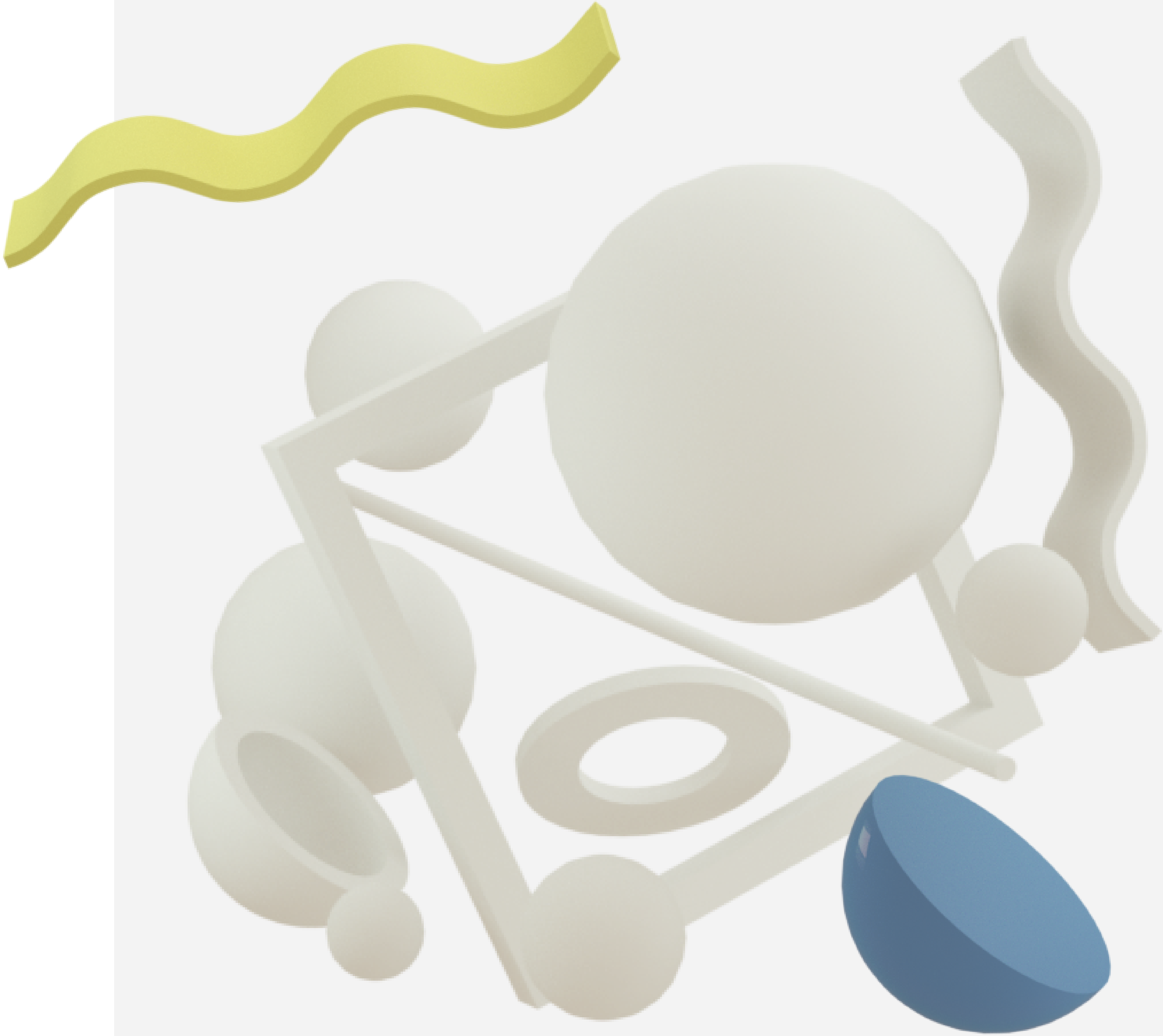
Utsaha Joshi (K12148294)



# Molecule Generation

# My submissions

| # | FCD | Novelty | Uniqueness | Validity |
|---|---|---|---|---|
| 1 | 1.86 | 0.979 | 0.999 | 1.0 |
| 2 | 1.987 | 0.98 | 0.999 | 1.0 |
| 3 | 45.041 | 0.995 | 1.0 | 1.0 |

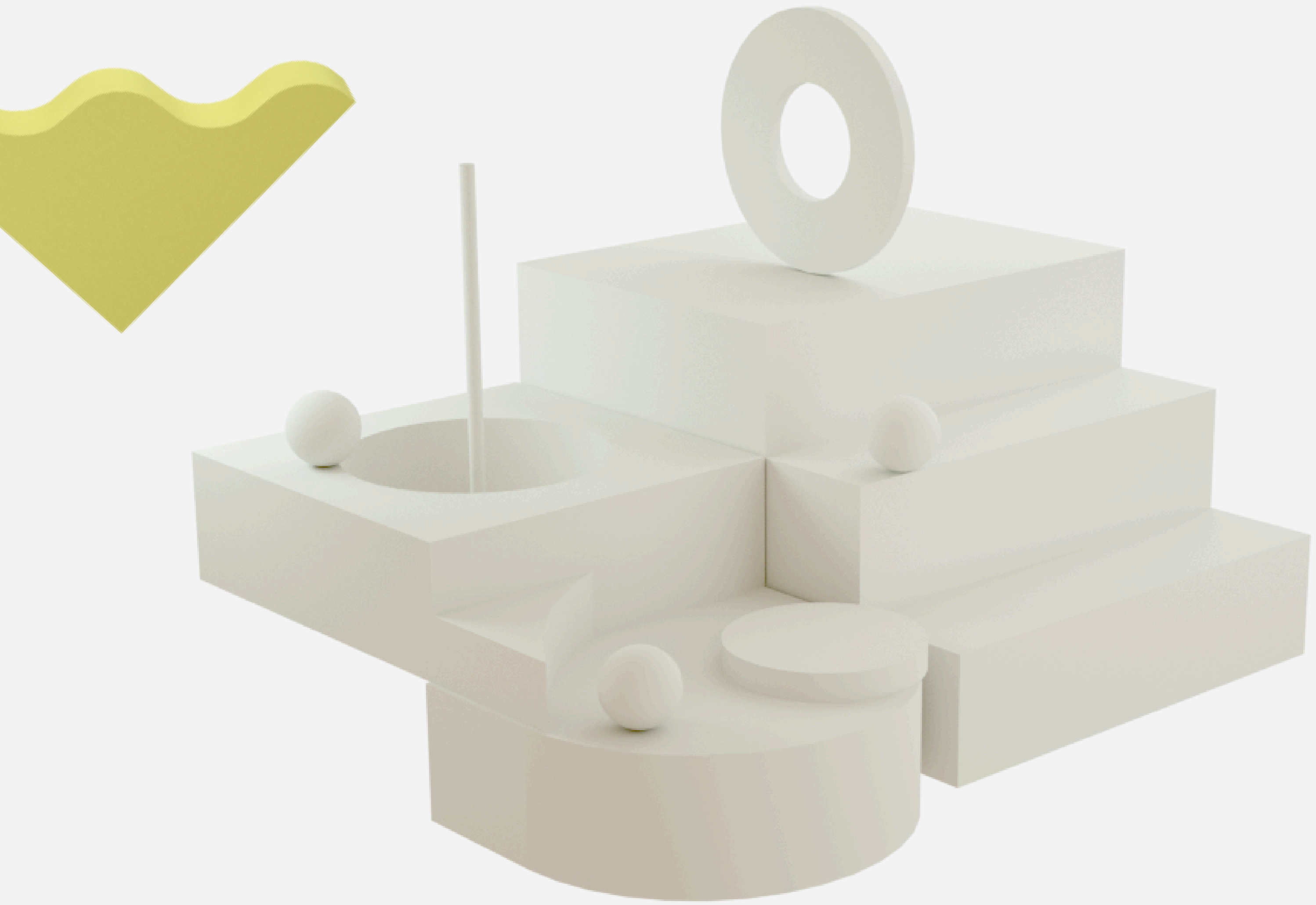
# GPT 2 FINE-TUNING TO GENERATE NOVEL, UNIQUE AND VALID SMILES WITH LOW FCD SCORE

After training and generating smiles using a GRU based model for over 2 days and failing on the FCD score miserably, I decided to finetune GPT-2 to generate smiles.

GPT-2 is a large transformer-based language model with 1.5 billion parameters, trained on a dataset of 8 million web pages.

What was used:
1. Hugging Face Transformer library to use pre-trained gpt-2 model and other necessary classes and functions
2. RDKIT library to check validity
3. Torch to use CUDA
4. Train data provided with smiles
5. Provided evaluation scripts

# MODEL AND TOKENIZER



```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
device = "cuda" if torch.cuda.is_available() else "cpu"
print(device)

tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')
model = model.to(device)
# Additionally, add the EOS token as PAD token to ensure the model does not generate past the maximum length.
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = 'left'
model.config.pad_token_id = model.config.eos_token_id
```

```
cuda
```

Device Setup: Checks and sets the device to CUDA if available, otherwise CPU.
Model and Tokenizer: Initializes and loads the pre-trained GPT-2 model and tokenizer.
GPU Utilization: Transfers the model to the GPU to enhance performance.
Configuration: Sets the EOS token as the padding token to prevent generation beyond max length.

# PREPARING DATASET FOR FINETUNING

Dataset Creation: Uses TextDataset to load and tokenize smiles_train.txt with a block size of 128. Block size defines the maximum length of token sequences the model will process at once.

Data Collation: Utilizes DataCollatorForLanguageModeling with mlm=False for next-token prediction, ideal for generating SMILES strings.

```python
from transformers import TextDataset, DataCollatorForLanguageModeling

# Use the TextDataset and DataCollator
dataset = TextDataset(
    tokenizer=tokenizer,
    file_path='smiles_train.txt',
    block_size=128
)

data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer, mlm=False
)
```

# SETTING UP TRAINER FOR FINETUNING

```
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir="./results",
    overwrite_output_dir=True,
    num_train_epochs=2,
    per_device_train_batch_size=16,  # Smaller batch size
    gradient_accumulation_steps=4,    # Accumulate gradients over 4 step
s
    save_steps=10_000,
    save_total_limit=2,
    prediction_loss_only=True,
    fp16=True  # Enable mixed precision
)
from transformers import TrainingArguments


trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,
    train_dataset=dataset
)

# Start training
trainer.train()
```

[5386/5386 6:11:15, Epoch 2/2]

| Step | Training Loss |
|---|---|
| 500 | 1.308800 |
| 1000 | 1.051100 |
| 1500 | 1.004500 |
| 2000 | 0.978300 |
| 2500 | 0.962900 |
| 3000 | 0.948900 |
| 3500 | 0.937000 |
| 4000 | 0.929900 |
| 4500 | 0.923700 |
| 5000 | 0.921500 |

Training Arguments:
Output Directory: output_dir="./results" specifies where to save model checkpoints.
Training Cycles: num_train_epochs=2 sets the number of complete passes through the dataset.
Batch Size and Memory: per_device_train_batch_size=16 with gradient_accumulation_steps=4 enables training with limited GPU memory by accumulating gradients.
Mixed Precision: fp16=True speeds up training and reduces memory usage.

Trainer Class: Initializes Trainer with the model, training arguments, data collator, and dataset to handle the training process.

# GENERATE SMILES FROM THE FINETUNED MODEL

Generation Loop: Continues generating until 15,000 unique SMILES strings are produced.
Input Encoding: Encodes the start token and prepares inputs for the model.
Generation Parameters:
1. Max Length: Limits each generated sequence to 100 tokens.
2. Number of Sequences: Generates 5 sequences at a time.
3. Sampling: Enables sampling with do_sample=True.
4. Top-k Sampling: Uses top_k=50 to consider the top 50 token options at each step.
5. Temperature: Sets temperature=0.8 to adjust the randomness and diversity of the generated sequences.

Decoding and Uniqueness: Decodes outputs and adds unique SMILES strings to the set.

Returns the list of unique generated SMILES strings.

```python
model.eval()

def generate_smiles(model, tokenizer, num_generate=15000):
    generated = set()
    device = "cuda" if torch.cuda.is_available() else "cpu"  # Check if GPU is available and set device accordingly
    model = model.to(device)  # Move model to the correct device

    start_token = tokenizer.bos_token or tokenizer.cls_token or "<|endoftext|>"  # Ensure there is a start token

    while len(generated) < num_generate:
        # Encode with a start token and ensure it's on the right device
        inputs = tokenizer(start_token, return_tensors="pt", add_special_tokens=False).to(device)

        outputs = model.generate(
            input_ids=inputs['input_ids'],
            attention_mask=inputs['attention_mask'],
            max_length=100,
            num_return_sequences=5,
            do_sample=True,  # Enable sampling
            top_k=50,        # Top-k sampling
            temperature=0.8  # Adjust temperature to tweak diversity
        )

        for output in outputs:
            smile = tokenizer.decode(output, skip_special_tokens=True)
            if smile not in generated:
                generated.add(smile)

    return list(generated)

generated_smiles = generate_smiles(model, tokenizer)
```

# POST PROCESSING TO GET TOP 10000 SMILES WITH LOW FCD SCORE

```
import os
import pickle
from evaluation.utils import canonicalize_smiles, getstats, loadmodel
import fcd
import numpy as np

def compute_fcd_for_batch(smiles_list, model, ref_mean, ref_cov):
    results = []
    canonical_smiles = canonicalize_smiles(smiles_list)
    valid_smiles = [sm for sm in canonical_smiles if sm]
    if valid_smiles:
        mean_gen, cov_gen = getstats(valid_smiles, model)
        try:
            fcd_values = fcd.calculate_frechet_distance(mean_gen, cov_ge
n, ref_mean, ref_cov)
            print(fcd_values)
            results.extend(zip(valid_smiles, [fcd_values] * len(valid_sm
iles)))
        except ValueError:
            results.extend((sm, float('inf')) for sm in valid_smiles)  #
Assign high FCD for failed cases
    return results

def process_in_batches(submission_smiles, batch_size, model, ref_mean, r
ef_cov):
    batch_results = []
    for i in range(0, len(submission_smiles), batch_size):
        batch = submission_smiles[i:i + batch_size]
        batch_results.extend(compute_fcd_for_batch(batch, model, ref_mea
n, ref_cov))
    return batch_results

# Load model and reference stats
model = loadmodel()
with open('./evaluation/data/test_stats.p', 'rb') as f:
    ref_mean, ref_cov = pickle.load(f)

# Load SMILES
with open('filtered_unique_smiles.txt', 'r') as f:
    submission_smiles = [line.strip() for line in f if line.strip()]

# Compute FCD in batches
batch_size = 500  # Define a reasonable batch size
fcd_results = process_in_batches(submission_smiles, batch_size, model, r
ef_mean, ref_cov)
sorted_fcd_results = sorted(fcd_results, key=lambda x: x[1])  # Sort by
FCD score

# Save the top 10,000 SMILES
with open('top_10000_smiles.txt', 'w') as file:
    for smile, fcd_score in sorted_fcd_results[:10000]:
        file.write(f"{smile}\n")

print("Top 10,000 SMILES with the lowest FCD have been saved to 'top_100
00_smiles.txt'.")
```

Compute FCD in Batches:

1. Define a reasonable batch size (batch_size = 500).
2. Process SMILES in batches using compute_fcd_for_batch, which canonicalizes, validates, and calculates the FCD values.
3. Handle any errors by assigning a high FCD value.
4. Sort and Save Results:Sort the SMILES based on their FCD scores in ascending order.
5. Save the top 10,000 SMILES with the lowest FCD scores to top_10000_smiles.txt.