MidiGPT (GPT for Symbolic Music Generation)
+
LDMG (Latent Diffusion for Music Generation)

Irene Chen, Melina Dimitropoulou Kapsogeorgou, Andrew Russell, Benjamin Xia

Conditioned Symbolic Generation with MidiGPT

Task Intro and Goal

- Produce **symbolic music** (i.e. MIDI messages), potentially in the form of an intermediate representation

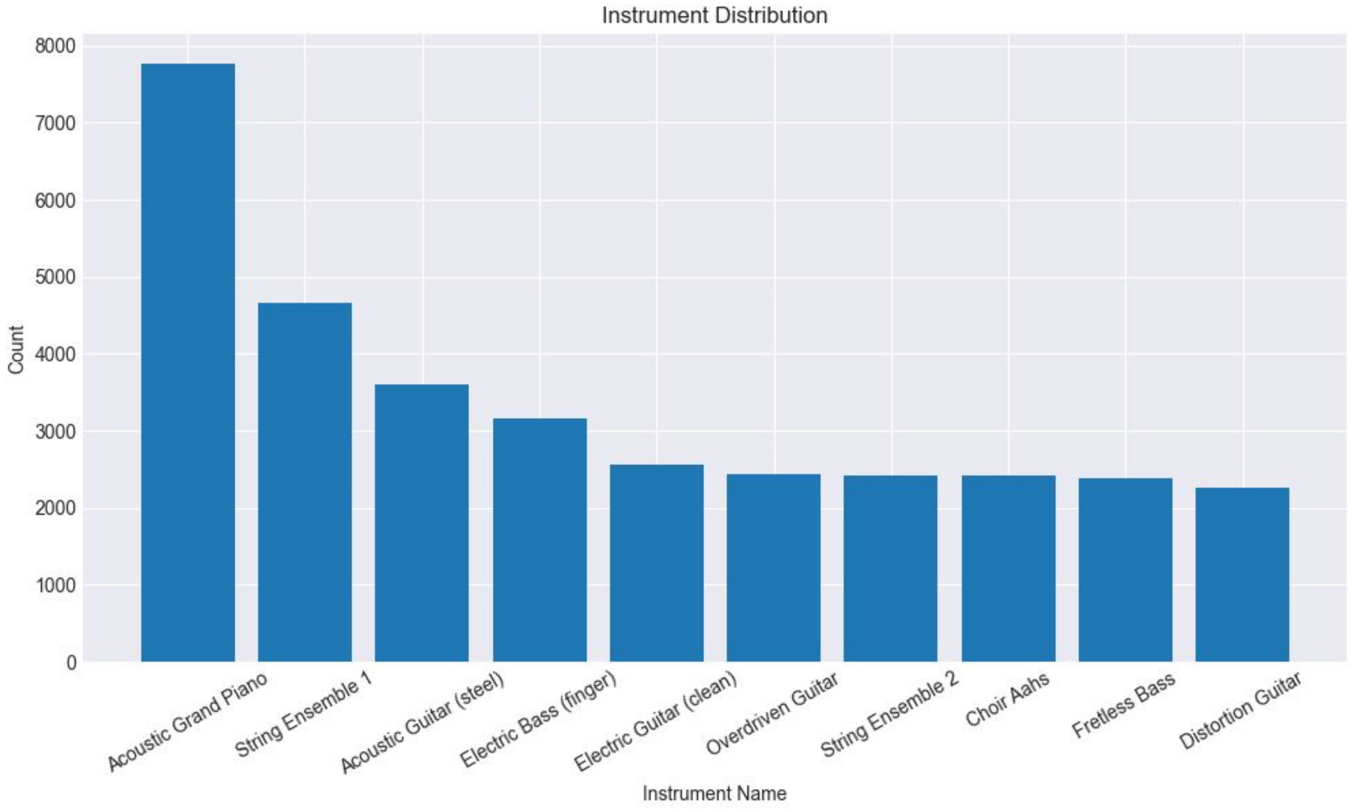
- Achieve conditioned generation in the form of **inpainting** existing pieces of music

Part 1: EDA, Data Collection, Pre-Processing

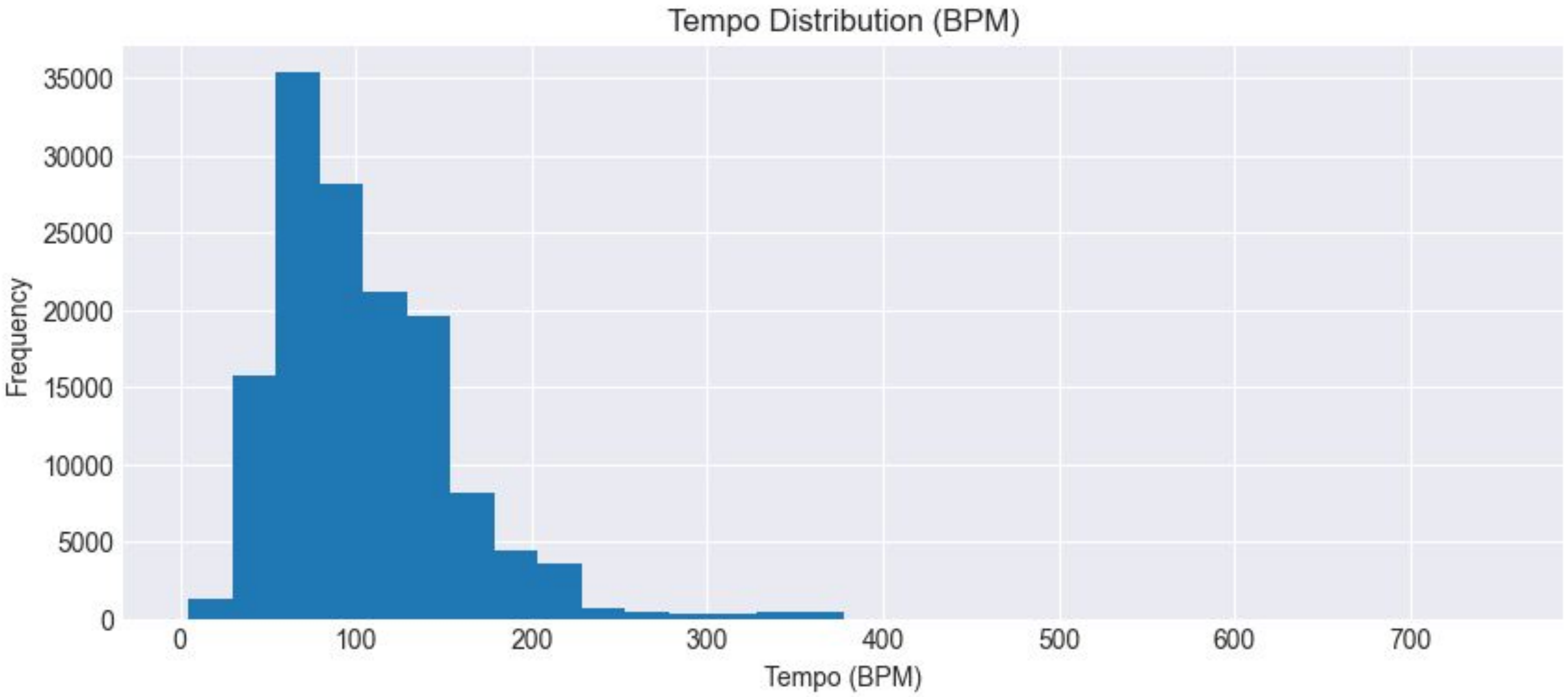
- **Context**:
  - **Lakh MIDI Dataset**: 176,582 cleaned MIDI files collected by Colin Raffel[1]
  - Because of hardware and time constraints, we train on only the first 10,000 MIDIs

- **Discussion**:
  - Tokenization: **Revamped MIDI** (REMI) as part of MidiTok
  - **Chunking**: Tokenized song segments are precomputed for training

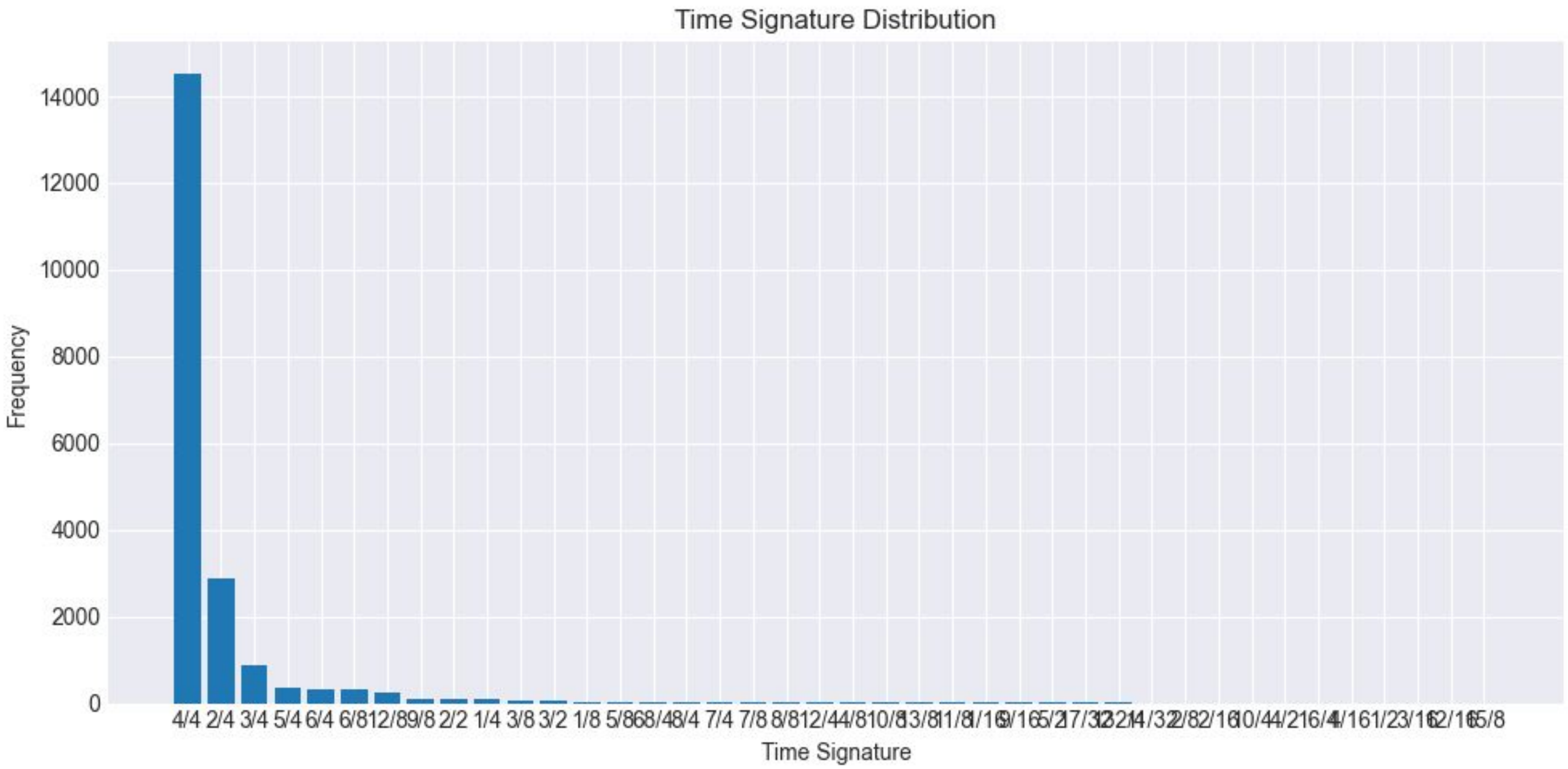
[1] Colin Raffel. "Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching". *PhD Thesis*, 2016.



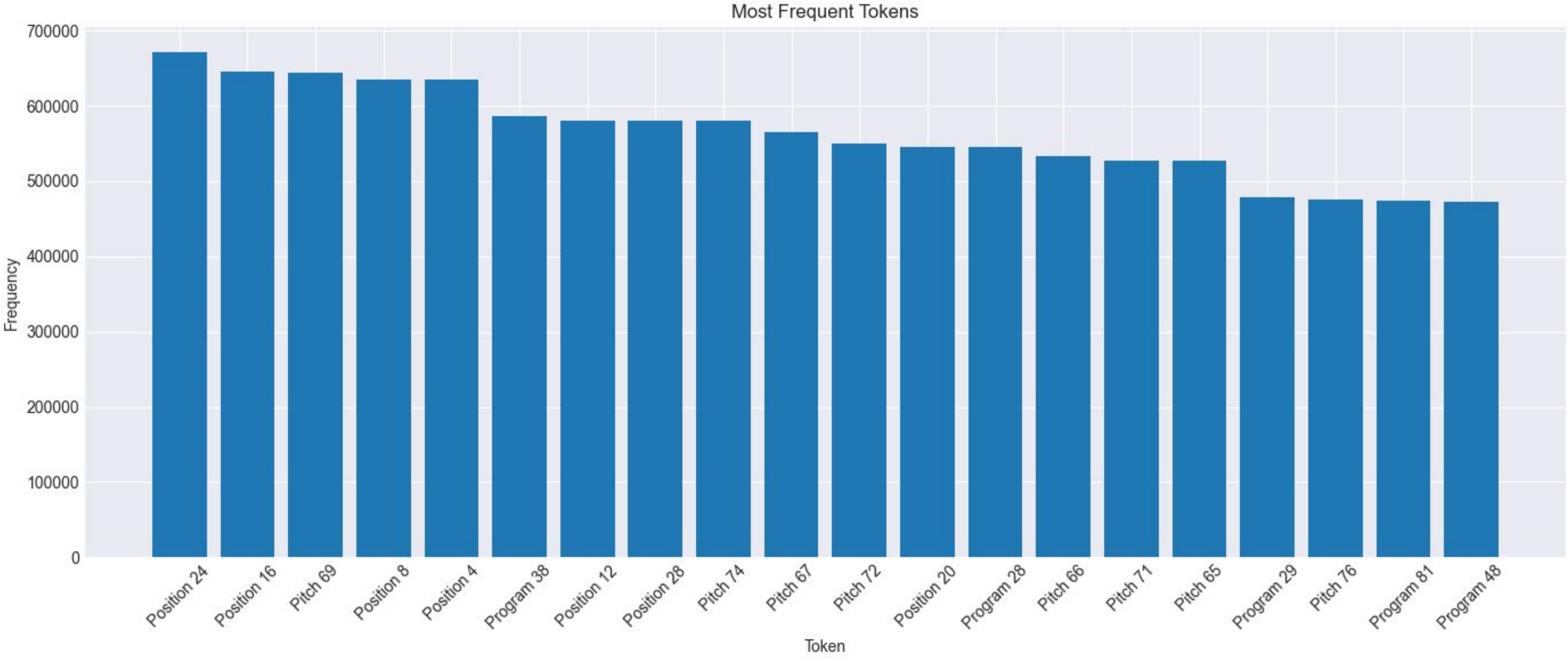
Instrument distribution in dataset



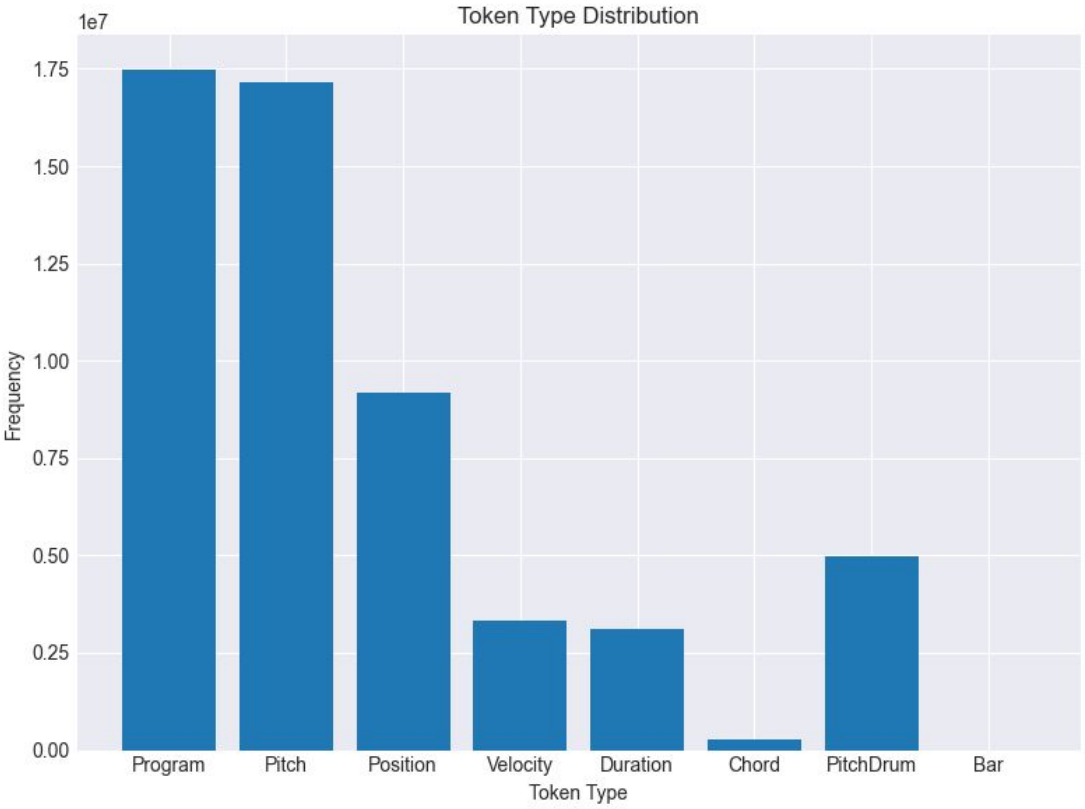
Tempo distribution in dataset



Time signature distribution in dataset



Most frequent tokens after training tokenizer



Token type distribution

Part 2: Modeling (Context)

- *Input*: A song to be in/outpainted (as **tokens)**
- *Output*: The generated song (as **tokens)**
- *Optimization*: Next-token prediction is classification/prediction, so a **cross-entropy loss** is suitable
- *Suitable models:*
  - Simple: Logistic regression, MLP
  - More complex: Markov chain, RNN
  - Modern: **Transformer**

Part 2: Modeling (Discussion)

- GPT models are **more difficult to implement** and **run slower** compared to simpler models

- Their **more complex structure** allows them to capture relationships missed by Markov chains, LSTMs, or GRUs

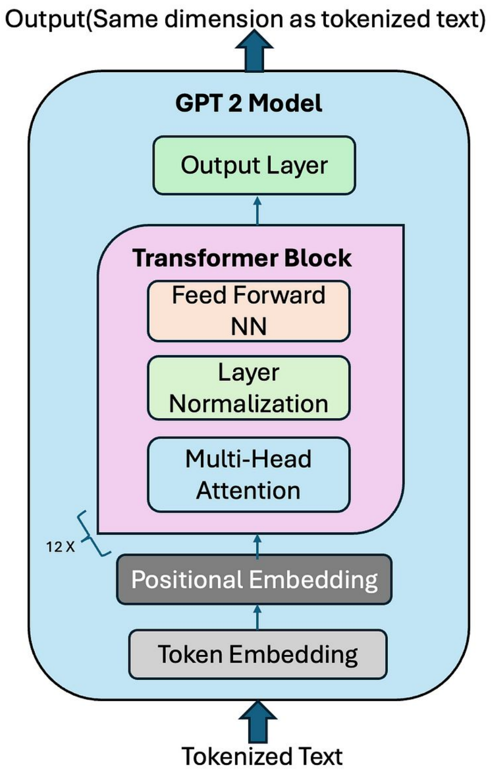
- Bidirectional GPTs are better suited for inpainting, but a decoder-only approach still can be used

Part 2: Modeling (Code)

- We leverage Andrej Karpathy's **minGPT**[1] to implement our architecture
- Our main engineering challenge becomes **dataset processing** and **hyperparameter tuning**, especially given **hardware constraints**

```python
# Initialize minGPT model

model_config = GPT.get_default_config()
model_config.model_type = 'gpt-micro'
model_config.vocab_size = vocab_size
model_config.block_size = 1024
model = GPT(model_config).to(device)
```

[1] https://github.com/karpathy/minGPT



Architecture diagram for our model, based on GPT-2
(Diagram courtesy of Vipul Koti on Medium)

```python
# Train tokenizer, split dataset into chunks

midi_dir = Path(f"{current_dir}/lmd")
chunk_dir = Path(f"{current_dir}/chunks")

has_chunk = chunk_dir.exists()
chunk_dir.mkdir(parents=True, exist_ok=True)

n_files = 10000
vocab_size = 1024

# NOTE: If you update the tokenizer at any point, you must remove the "chunks" directory
# (otherwise you will get CUDA assertions failing because of token IDs)
if not os.path.exists(f'{current_dir}/tokenizer.json'):
    print('Training tokenizer...')

    tokenizer = REMI(
        TokenizerConfig(
            num_velocities=16,
            use_chords=True,
            use_programs=True,
        )
    )

    tokenizer.train(vocab_size=vocab_size, files_paths=list(midi_dir.glob('**/*.mid'))[:n_files])
    tokenizer.save('tokenizer.json')
else:
    print('Using pretrained tokenizer.')
    tokenizer = REMI.from_pretrained('tokenizer.json')

if not has_chunk:
    print('Splitting files into chunks...')
    split_files_for_training(
        files_paths=list(midi_dir.glob('**/*.mid'))[:n_files],
        tokenizer=tokenizer,
        save_dir=chunk_dir,
        max_seq_len=1024,
    )
```

```python
# Set up PyTorch DataLoader

dataset_midi = DatasetMIDI(
    files_paths=list(chunk_dir.glob('**/*.mid')),
    tokenizer=tokenizer,
    max_seq_len=1024,
    bos_token_id=tokenizer['BOS_None'],
    eos_token_id=tokenizer['EOS_None'],
)
collator = DataCollator(tokenizer.pad_token_id, copy_inputs_as_labels=False)

train_dataloader = DataLoader(
    dataset_midi,
    batch_size=64,
    collate_fn=collator,
    sampler=torch.utils.data.RandomSampler(dataset_midi, replacement=True, num_samples=int(1e10)),
    shuffle=False,
    pin_memory=True,
)
```

```python
# Fetch the next batch (x, y) and re-init iterator if needed
try:
    batch = next(data_iter)
except StopIteration:
    data_iter = iter(self.train_loader)
    batch = next(data_iter)

# Input: MIDI tokens
# Output: Expected next MIDI tokens
x = batch['input_ids'][:, :-1].to(self.device).detach()
y = batch['input_ids'][:, 1:].to(self.device).detach()

# Forward the model
logits, self.loss = model(x, y)

# Backprop and update the parameters
model.zero_grad(set_to_none=True)
self.loss.backward()
torch.nn.utils.clip_grad_norm_(model.parameters(), config.grad_norm_clip)
self.optimizer.step()
```

(Excerpted from *Trainer* class)

```python
# Set up training loop

train_config = Trainer.get_default_config()
train_config.device = device
train_config.learning_rate = 5e-4
train_config.max_iters = 50000
trainer = Trainer(train_config, model, train_dataloader)

def batch_end_callback(trainer):
    if trainer.iter_num % 100 == 0:
        print(f"iter {trainer.iter_num}: train loss {trainer.loss.item():.5f}")
trainer.set_callback('on_batch_end', batch_end_callback)
trainer.run()
```

Part 3: Evaluation (Context)

- During training, the task is evaluated as **next-token prediction**

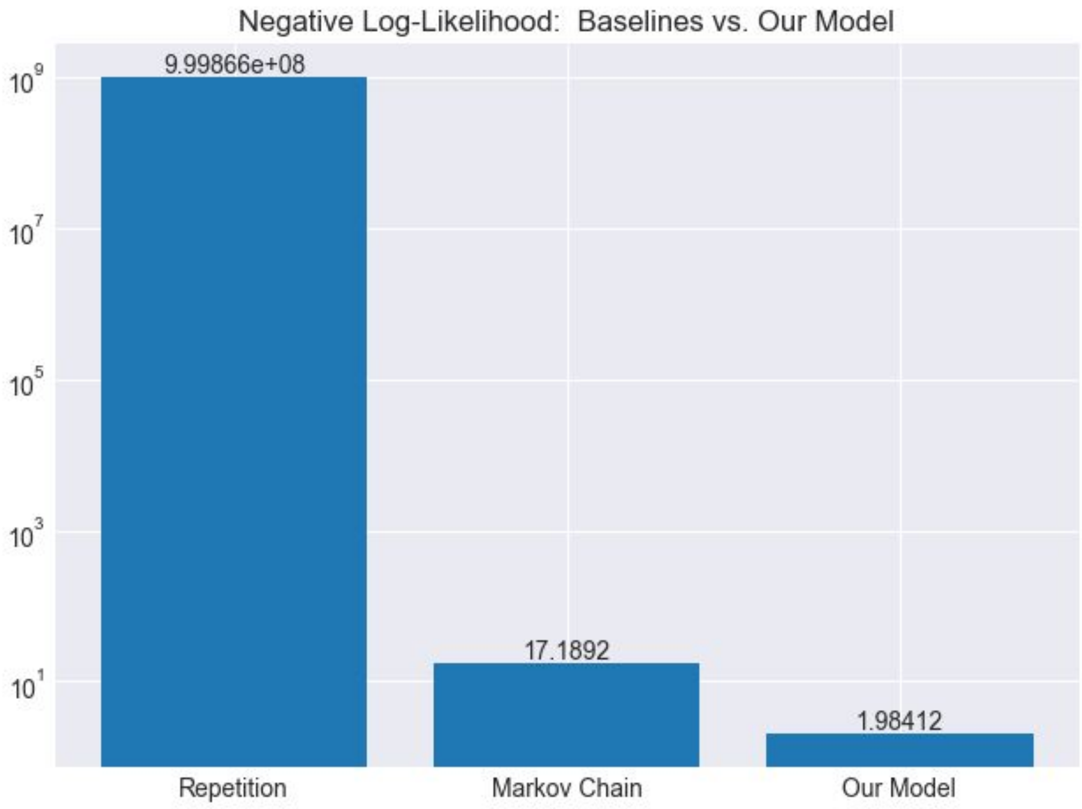
- After training, the task is evaluated as whether the output has a **subjectively satisfying** sense of:
    - Pulse, rhythm, and meter
    - Key
    - Instrumentation
    - Harmony and melody
    - Etc.

Part 3: Evaluation (Discussion)

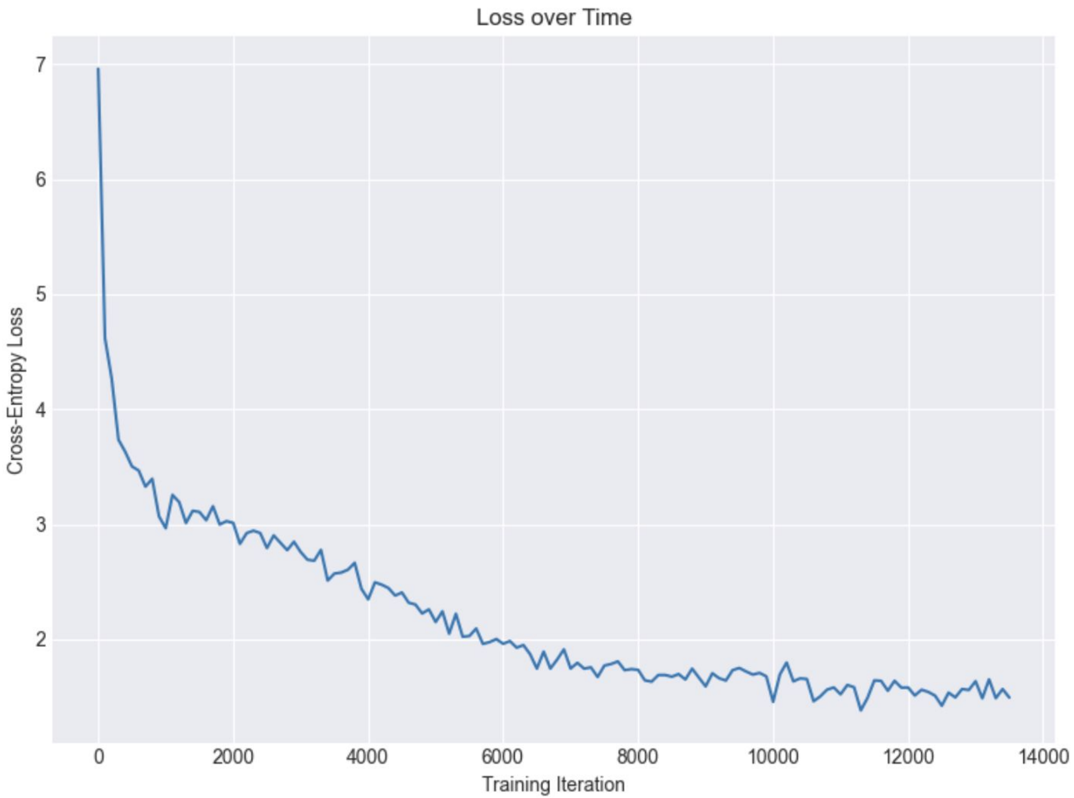
- Possible baselines include:
  - Repeating input tokens as output
  - Markov chain models from Homework 3

- Our findings suggest that our method achieves results that are **qualitatively more satisfactory**

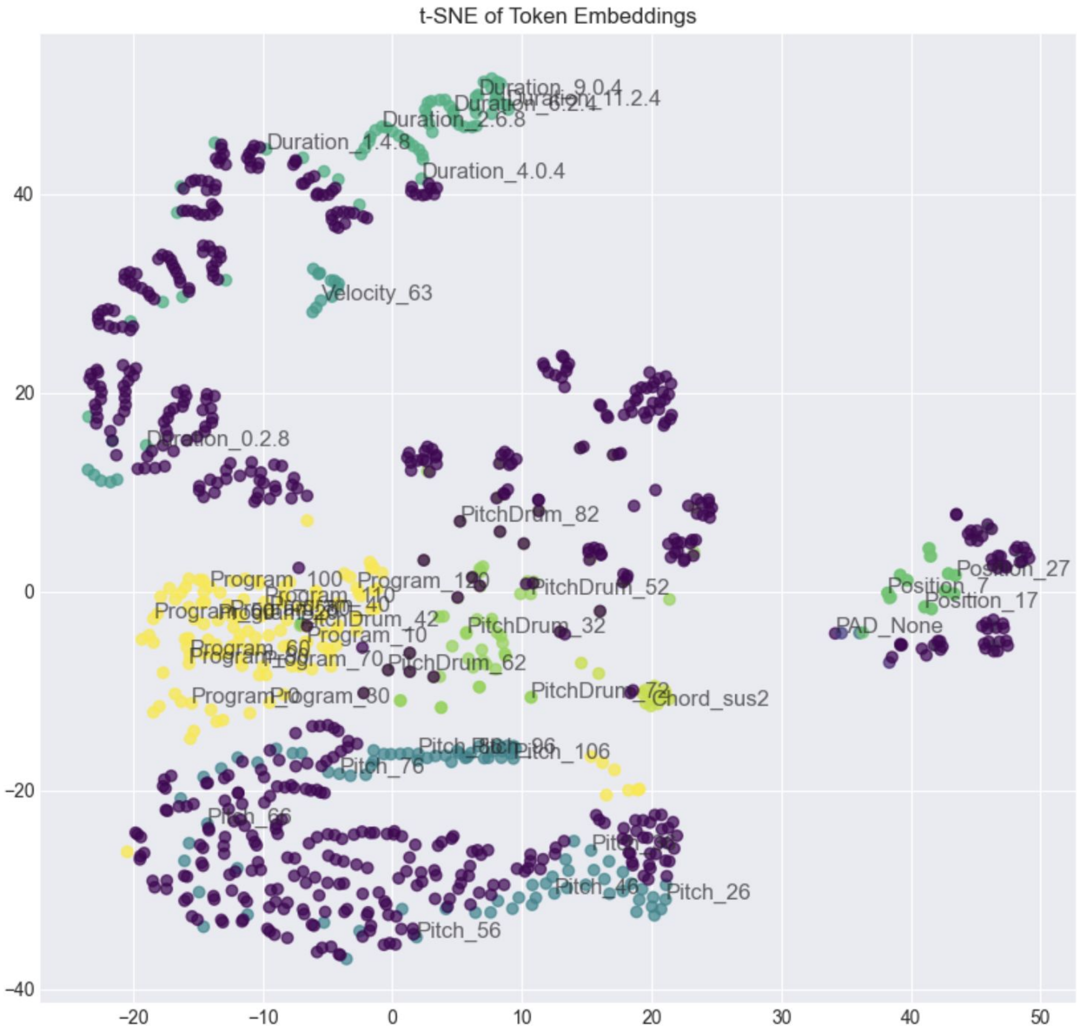
- To quantify these results, we use **negative log-likelihood** to compare against baselines and **t-SNE** to visualize whether the model has learned a coherent embedding space



Negative log-likelihood of baselines and our model



Cross-entropy loss during training


t-SNE of Token Embeddings
Duration_9.0.4
Duration_11.2.4
Duration_2.6.8
Duration_1.4.8
Duration_4.0.4
Velocity_63
Duration_0.2.8
PitchDrum_82
Program_100
Program_120
PitchDrum_52
Program_110
Program_40
PitchDrum_42
Program_10
PitchDrum_32
Program_70
PitchDrum_62
Program_0
Program_30
PitchDrum_72
Chord_sus2
Position_27
Position_7
Position_17
PAD_None
Pitch_106
Pitch_76
Pitch_46
Pitch_26
Pitch_56
40
20
0
−20
−40
−20
−10
0
10
20
30
40
50

Part 3: Evaluation (Code, Cont'd.)

- Inpainting is achieved by splitting an input song into three parts:
  - Prefix
  - Hole
  - Suffix

- The entire piece **followed by the prefix again** is passed into the model, then the suffix is appended to the output

```python
# Inpaint from left-to-right

prefix_len = 128
suffix_len = 128
hole_len = 256
total_len = prefix_len + hole_len + suffix_len
temperature = 1.2
top_k = 16
song_id = 220

train_midi_sample = list(chunk_dir.glob('**/*.mid'))[song_id]
tokens = torch.tensor(tokenizer(train_midi_sample).ids[:total_len], dtype=torch.long, device=device)[None, :]

prefix = tokens[:, :prefix_len]
suffix = tokens[:, prefix_len + hole_len:total_len]

input_seq = torch.cat((tokens[:, :total_len], prefix), dim=1)

model.eval()
with torch.no_grad():
    x = model.generate(input_seq, hole_len, temperature=temperature, top_k=top_k, do_sample=True)

full_sequence = torch.cat((x[:, total_len:], suffix), dim=1)

initial_midi = tokenizer(tokens[0, :total_len].detach().cpu())
initial_midi.dump_midi(f'{current_dir}/initial_midi.mid')

generated_midi = tokenizer(full_sequence[0].detach().cpu())
generated_midi.dump_midi(f'{current_dir}/inpainted_output.mid')
```

Part 4: Discussion of Related Work (Datasets)

- Use of Lakh MIDI Dataset:
    - *Convolutional Generative Adversarial Networks* (Dong & Yang, 2018)
    - *Lakh*NES (Donahue et al., 2019)
    - *MusPy* (Dong et al., 2020) – UCSD

- Other symbolic datasets:
    - JSB-Chorales (J.S. Bach chorale MIDI dataset): *Music Transformer* (Huang et al., 2018)
    - ClassicalArchives: *MuseNet* (OpenAI, 2019)
    - MAESTRO: *MuseNet*

Part 4: Discussion of Related Work (Approaches)

- Modern symbolic music synthesis uses **RNNs in the past** and **transformers today**

- RNNs perform better than Markov chains, but still struggle with **long-range dependencies**

- Transformers are the current state-of-the-art for generation tasks, including **natural language** and **symbolic music**

Part 4: Discussion of Related Work (Results)

- Our model is capable of producing **sensible outputs** when both inpainting and outpainting

- Expanding our model and training for more time is likely to yield better results **with little engineering effort**

- This is in line with earlier research regarding transformers: They are **powerful**, **extensible**, and **scale well**[1, 2]

[1] Muhamed et al. *Symbolic Music Generation with Transformer-GANs*. 2021.
[2] Kaplan et al. *Scaling Laws for Neural Language Models*. 2020.

Results

- Drums 1:
- Jungle 1:
- Piano 1:
- Piano 2:

LDMG:
Latent Music Diffusion for Music Generation

1.1 Dataset

- Dataset: Free Music Archive (FMA), 106,574 tracks + metadata https://arxiv.org/pdf/1612.01840v2
  - We used the FMA-Medium subset (25k tracks, each 30 seconds) due to storage constraints (The full dataset is almost 1TB!)
- Brief History: It’s a snapshot of the Free Music Archive (https://freemusicarchive.org/)
  - A collection of songs under the Creative Commons licenses

FMA

BY TRIBE OF NOISE

1.2 Preprocessing

- Original Dataset
  - The dataset was already preprocessed such that each audio clip was truncated to 30 seconds, though with varying sampling rates, and in stereo.
  - The dataset also contained some metadata for each file, though we did not use this for conditioned generation.

1.2 Preprocessing

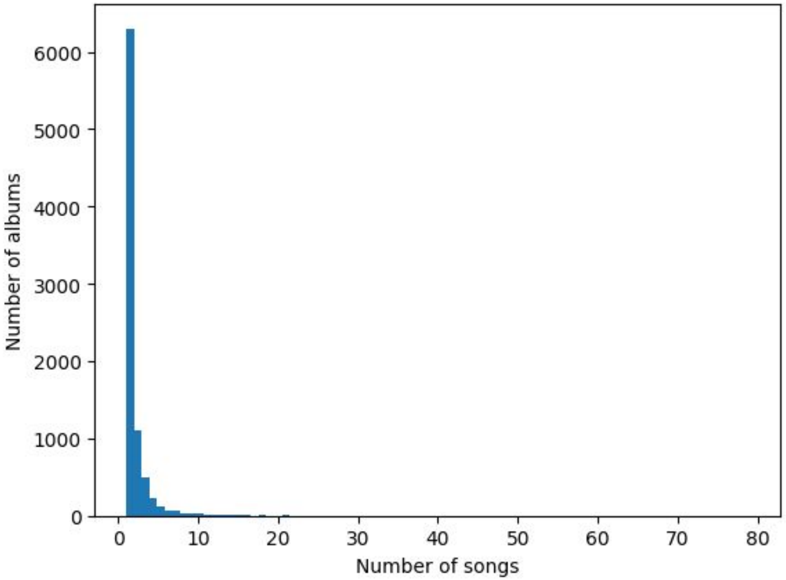
- Preprocessing
    - Load .mp3 files with torchaudio, convert to mono, resample to 16KHz, pad/crop waveforms to a fixed-length.
    - We trained our models on 10 second segments due to VRAM constraints
        - Our VAE-GAN is capable of zero-shot reconstruction of longer segments. See our code! :)

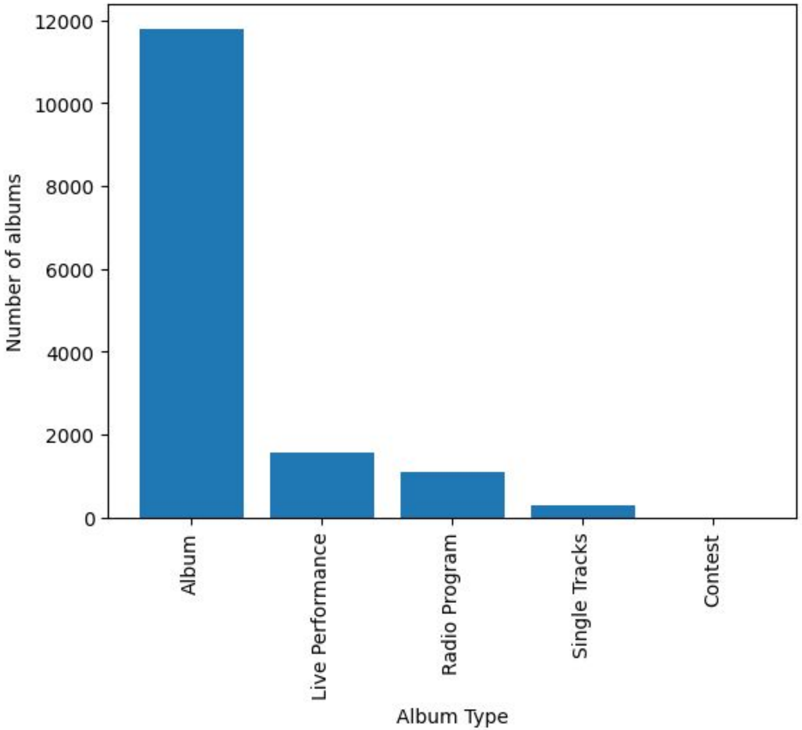
```python
def __getitem__(self, idx):
    path = self.audio_paths[idx]
    try:
        waveform, sr = torchaudio.load(path)
    except RuntimeError:
        # Cus theres a few malformed waveforms in fma medium, pray the collate function can handle this case
        return None
    # Convert to mono
    if waveform.shape[0] > 1:
        waveform = waveform.mean(dim=0, keepdim=True)

    # Resample to target sample rate
    if sr != self.sample_rate:
        resampler = torchaudio.transforms.Resample(orig_freq=sr, new_freq=self.sample_rate)
        waveform = resampler(waveform)

    # Pad or crop to fixed number of samples
    if waveform.shape[1] < self.num_samples:
        pad_amt = self.num_samples - waveform.shape[1]
        waveform = torch.nn.functional.pad(waveform, (0, pad_amt))
    else:
        waveform = self.crop(waveform)

    return waveform
```

I wrote this at 3am, reinitializing the resampler is slow but there are a lot of waveforms with weird sample rates, pls no bully me.

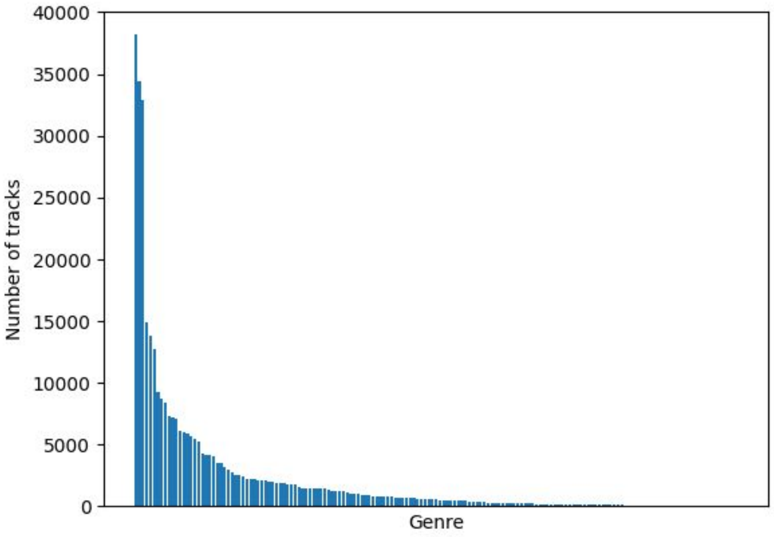
1.3 EDA

albums per artist





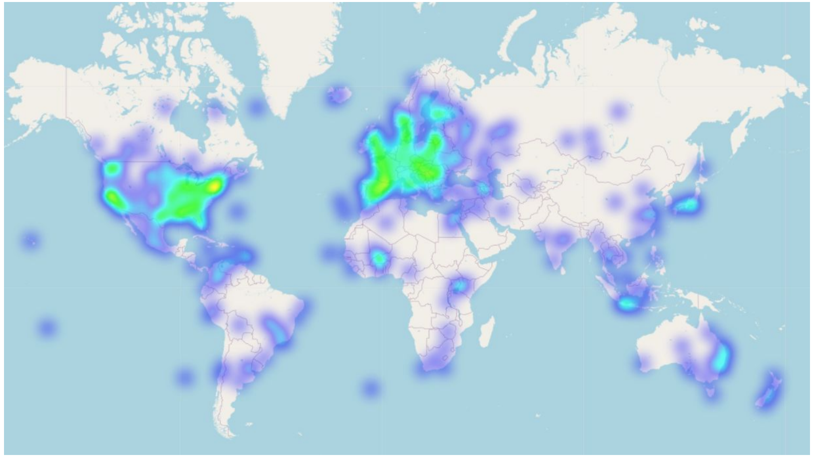
1.3 EDA

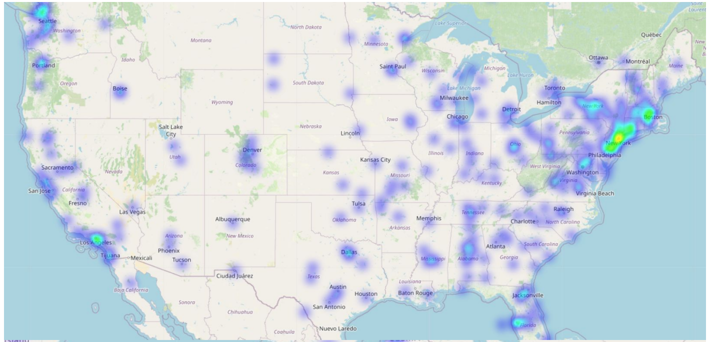


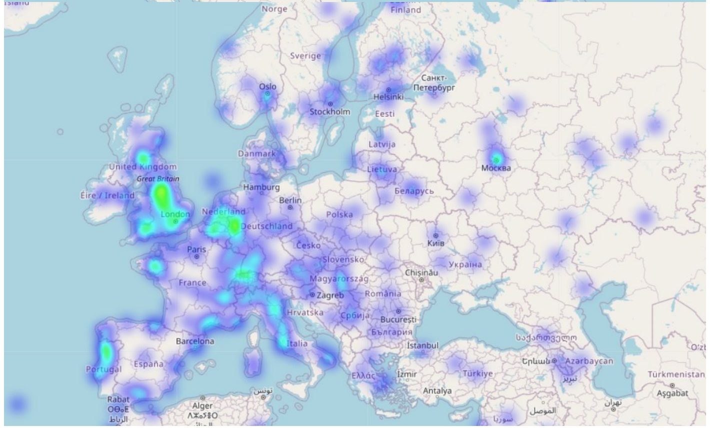

```
'Avant-Garde', 'International', 'Blues', 'Jazz', 'Classical',
       'Novelty', 'Comedy', 'Old-Time / Historic', 'Country', 'Pop',
       'Disco', 'Rock', 'Easy Listening', 'Soul-RnB', 'Electronic',
       'Sound Effects', 'Folk', 'Soundtrack', 'Funk', 'Spoken', 'Hip-Hop',
       'Audio Collage', 'Punk', 'Post-Rock', 'Lo-Fi', 'Field Recordings',
       'Metal', 'Noise', 'Psych-Folk', 'Krautrock', 'Jazz: Vocal',
       'Experimental', 'Electroacoustic', 'Ambient Electronic',
       'Radio Art', 'Loud-Rock', 'Latin America', 'Drone', 'Free-Folk',
       'Noise-Rock', 'Psych-Rock', 'Bluegrass', 'Electro-Punk', 'Radio',
       'Indie-Rock', 'Industrial', 'No Wave', 'Free-Jazz',
       'Experimental Pop', 'French', 'Reggae - Dub', 'Afrobeat',
       'Nerdcore', 'Garage', 'Indian', 'New Wave', 'Post-Punk', 'Sludge',
       'African', 'Freak-Folk', 'Jazz: Out', 'Progressive',
       'Alternative Hip-Hop', 'Death-Metal', 'Middle East',
       'Singer-Songwriter', 'Ambient', 'Hardcore', 'Power-Pop',
       'Space-Rock', 'Polka', 'Balkan', 'Unclassifiable', 'Europe',
       'Americana', 'Spoken Weird', 'Interview', 'Black-Metal',
       'Rockabilly', 'Easy Listening: Vocal', 'Brazilian',
       'Asia-Far East', 'N. Indian Traditional',
       'South Indian Traditional', 'Bollywood', 'Pacific', 'Celtic',
       'Be-Bop', 'Big Band/Swing', 'British Folk', 'Techno', 'House',
       'Glitch', 'Minimal Electronic', 'Breakcore - Hard', 'Sound Poetry',
       '20th Century Classical', 'Poetry', 'Talk Radio', 'North African',
       'Sound Collage', 'Flamenco', 'IDM', 'Chiptune', 'Musique Concrete',
       'Improv', 'New Age', 'Trip-Hop', 'Dance', 'Chip Music', 'Lounge',
       'Goth', 'Composed Music', 'Drum & Bass', 'Shoegaze',
       'Kid-Friendly', 'Thrash', 'Synth Pop', 'Banter', 'Deep Funk',
       'Spoken Word', 'Chill-out', 'Bigbeat', 'Surf', 'Radio Theater',
       'Grindcore', 'Rock Opera', 'Opera', 'Chamber Music',
       'Choral Music', 'Symphony', 'Minimalism', 'Musical Theater',
       'Dubstep', 'Skweee', 'Western Swing', 'Downtempo', 'Cumbia',
       'Latin', 'Sound Art', 'Romany (Gypsy)', 'Compilation', 'Rap',
       'Breakbeat', 'Gospel', 'Abstract Hip-Hop', 'Reggae - Dancehall',
       'Spanish', 'Country & Western', 'Contemporary Classical', 'Wonky',
       'Jungle', 'Klezmer', 'Holiday', 'Salsa', 'Nu-Jazz',
       'Hip-Hop Beats', 'Modern Jazz', 'Turkish', 'Tango', 'Fado',
       'Christmas', 'Instrumental'
```

1.3 EDA

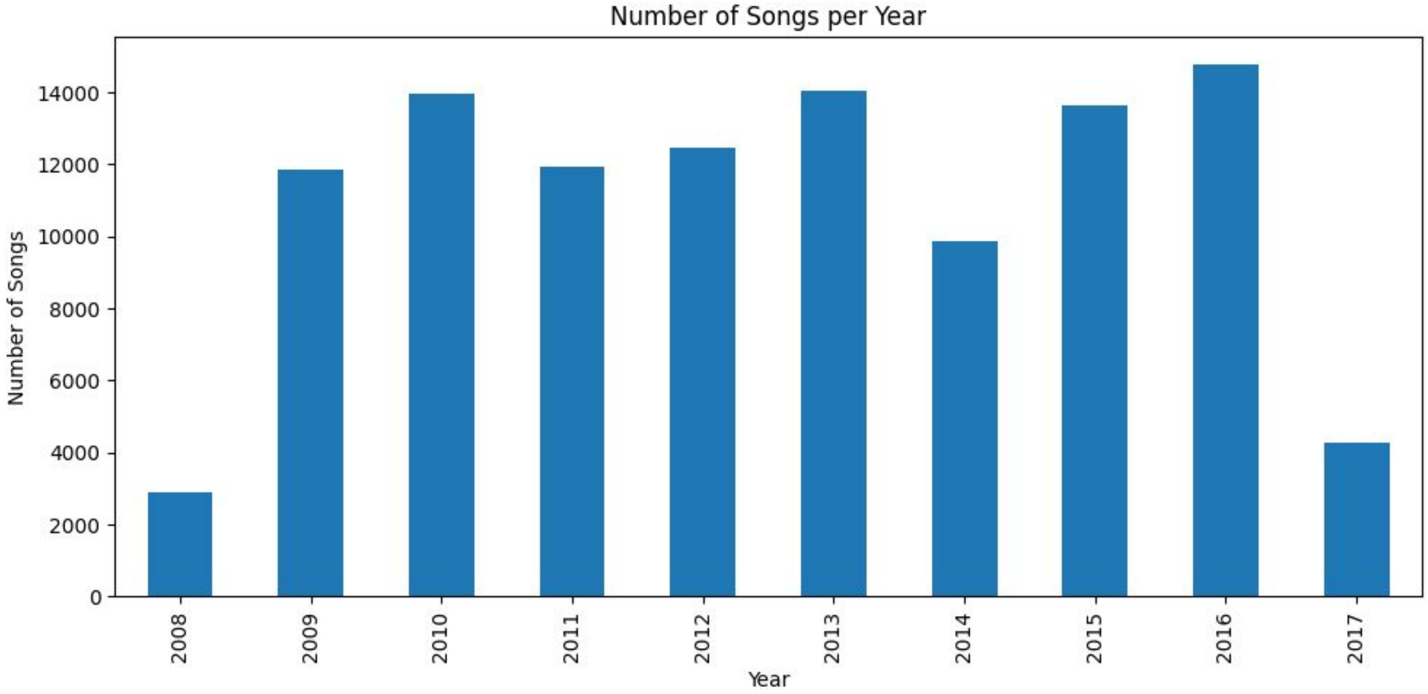
Heatmap of artist locations. I’m a little concerned about how they got this information…







1.3 EDA


Number of Songs per Year
Number of Songs
0
2000
4000
6000
8000
10000
12000
14000
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
Year

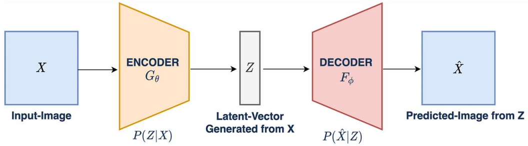
2.1 Unconditioned Audio Synthesis Formalism

- Given a collection of mono audio waveforms X of length T (where T = sample rate * audio duration), we want to sample x from the distribution of X, i.e. x ~ P(X)
- Our model should be ideally be able to map Gaussian noise in some lower dimensional latent space (We chose 32 channel, 250 Hz) to reasonable audio waveforms in the same space as our input collection (1 channel, 16KHz).

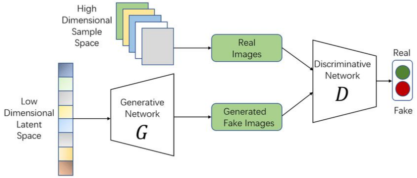
Context: How do you formulate your task as an ML problem, e.g. what are the inputs, outputs, and what is being optimized? What models are appropriate for the task?

2.2 Reasonable Model Choices

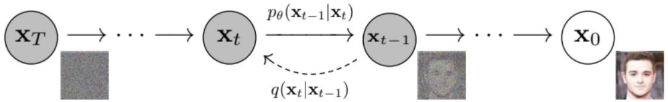
- Typical model choices for generating continuous domain, high-dimensional data such as audio waveforms.



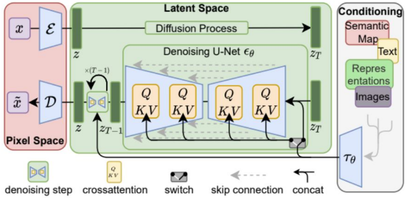
Variational Autoencoder



Generative Adversarial Networks



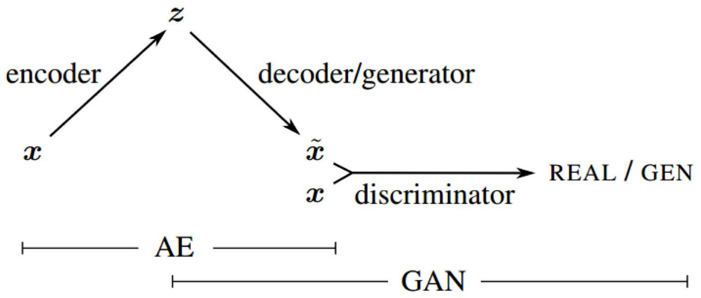
Diffusion



Latent Diffusion

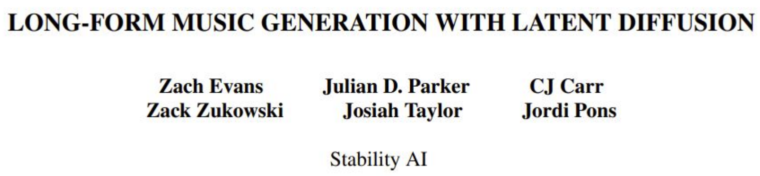
2.3 Modeling (Discussion)

- Possible modeling approaches:
- Autoregressive models: High fidelity; issue with long-range structure
- VAE alone: Efficient training, inference; blurry outputs
- GAN alone: Good fidelity; Training instability
- Diffusion models: High fidelity, slow sampling and training
- **VAE-GAN + Latent Diffusion**: High-quality outputs, controllable latents; two-stage training complexity, slow training (albeit much faster than vanilla diffusion)



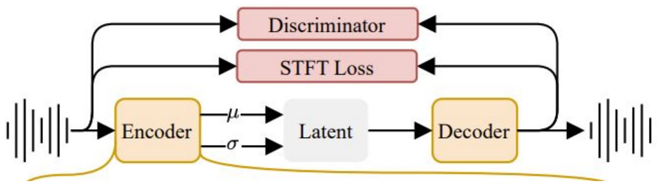
2.3 Our Model Choices

- Latent Diffusion is SOTA for unconditioned/conditioned audio synthesis


LONG-FORM MUSIC GENERATION WITH LATENT DIFFUSION

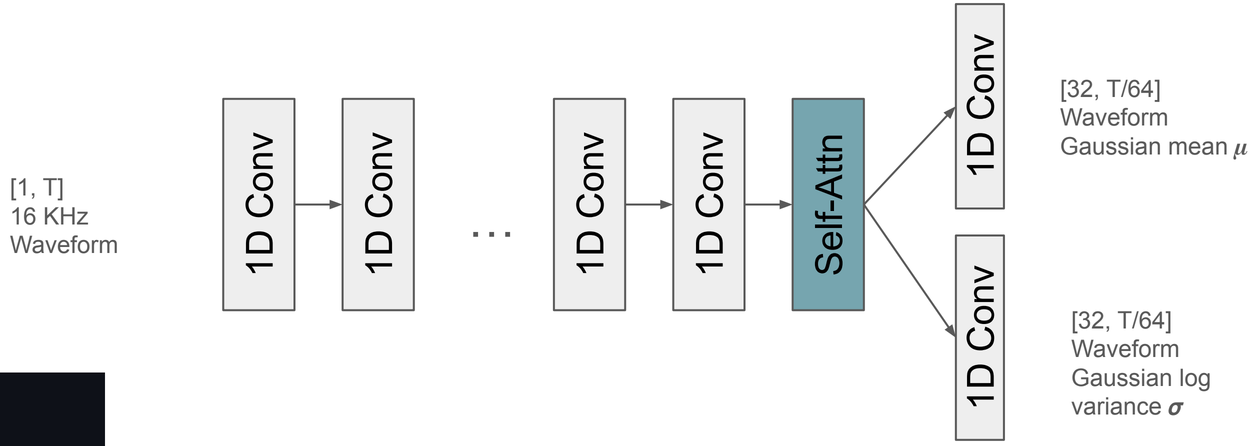
Zach Evans, Zack Zukowski, Julian D. Parker, Josiah Taylor, CJ Carr, Jordi Pons

Stability AI



- Architecture is a simplified version of Stable Audio
  - CNN Encoder/Decoder, Discriminator, Diffusion Transformer Denoiser
- Training Objectives:
  - VAE-GAN: STFT Reconstruction + KL Divergence + Discriminator Hinge Loss + Discriminator Feature Matching Loss
  - Denoiser: Standard DDPM Loss with 500 timesteps https://arxiv.org/pdf/2006.11239

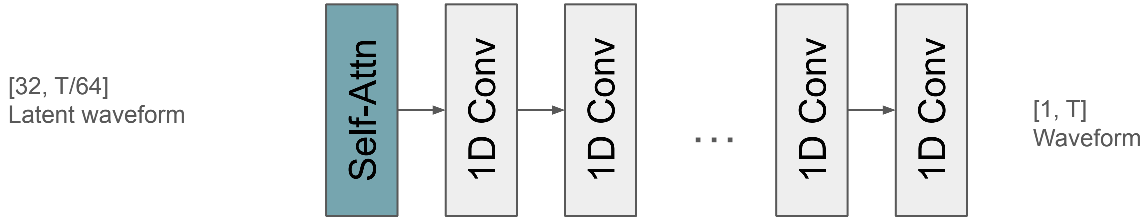
2.3 VAE-GAN Encoder Architecture




```python
# Input dimension must be some power of 2 multiple of latent dim
self.n_downsamples = np.ceil(np.log2(self.input_sr / self.latent_sr)).astype(np.int32)
assert (2 ** self.n_downsamples) * latent_sr == self.input_sr

starter_channels = 16
layers = [
    nn.Conv1d(input_channels, starter_channels, DEFAULT_1D_KERNEL_SIZE, stride=1, padding=DEFAULT_1D_PADDING),
    nn.GELU(),
]

# Channels go from 16 -> 32 -> 64 -> DEFAULT_MAX_CHANNELS ... n_downsamples layers
in_ch = starter_channels
for i in range(self.n_downsamples):
    out_ch = min(in_ch * 2, DEFAULT_MAX_CHANNELS)
    layers.append(DownsampleLayer(in_ch, out_ch))  # Downsample by factor of 2
    in_ch = out_ch
    layers.append(nn.Conv1d(in_ch, in_ch, DEFAULT_1D_KERNEL_SIZE, stride=1, padding=DEFAULT_1D_PADDING))
    layers.append(nn.GELU())

layers.append(SelfAttention(in_ch, 4, audio_dur * self.latent_sr))

self.layers = nn.Sequential(*layers)

self.mu_proj = nn.Sequential(
    nn.Conv1d(in_ch, in_ch, kernel_size=DEFAULT_1D_KERNEL_SIZE, padding=DEFAULT_1D_PADDING),
    nn.GELU(),
    nn.Conv1d(in_ch, latent_channels, kernel_size=1)
)

self.logvar_proj = nn.Sequential(
    nn.Conv1d(in_ch, in_ch, kernel_size=DEFAULT_1D_KERNEL_SIZE, padding=DEFAULT_1D_PADDING),
    nn.GELU(),
    nn.Conv1d(in_ch, latent_channels, kernel_size=1)
)
```

Code available at https://github.com/benjxia/LDMG

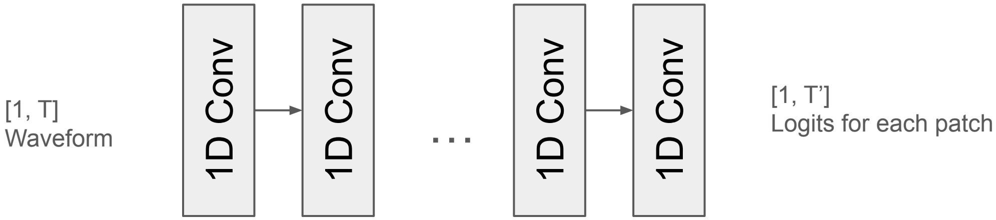
2.3 VAE-GAN Decoder Architecture




```python
# Input dimensions must be some power of 2 multiple of latent dim
self.n_upsamples = np.ceil(np.log2(self.input_sr / self.latent_sr)).astype(np.int32)
assert (2 ** self.n_upsamples) * latent_sr == self.input_sr

channels = DEFAULT_MAX_CHANNELS
layers = [
    nn.Conv1d(latent_channels, channels, DEFAULT_1D_KERNEL_SIZE, stride=1, padding=DEFAULT_1D_PADDING),
    nn.GELU(),
]

layers.append(SelfAttention(channels, 4, audio_dur * self.latent_sr))

for i in range(self.n_upsamples):
    layers.append(UpsampleLayer(channels, channels))
    layers.append(nn.Conv1d(channels, channels, DEFAULT_1D_KERNEL_SIZE, stride=1, padding=DEFAULT_1D_PADDING))
    layers.append(nn.GELU())

layers.append(nn.Conv1d(channels, input_channels, kernel_size=1))

self.layers = nn.Sequential(*layers)
```

Implementation Note:
Transpose convolutions were used for upsampling

2.3 VAE-GAN Discriminator Architecture

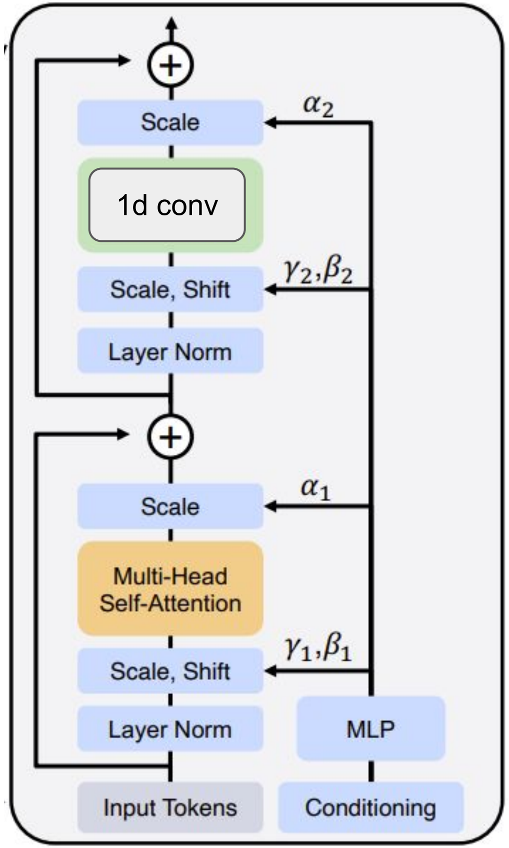



```python
class PatchDiscriminator(nn.Module):
    def __init__(self, input_channels: int):
        super().__init__()
        self.layers = nn.ModuleList([
            nn.Conv1d(input_channels, 128, 15, stride=1, padding=7),   # preserves resolution
            nn.LeakyReLU(0.2),
            nn.Conv1d(128, 128, 41, stride=4, padding=20, groups=4),   # grouped conv like HiFi-GAN
            nn.LeakyReLU(0.2),
            nn.Conv1d(128, 128, 41, stride=4, padding=20, groups=16),
            nn.LeakyReLU(0.2),
            nn.Conv1d(128, 128, 41, stride=4, padding=20, groups=16),
            nn.LeakyReLU(0.2),
            nn.Conv1d(128, 128, 41, stride=4, padding=20, groups=16),
            nn.LeakyReLU(0.2),
            nn.Conv1d(128, 128, 5, stride=1, padding=2),
            nn.LeakyReLU(0.2),
            nn.Conv1d(128, 1, kernel_size=3, stride=1, padding=1)    # patch discriminator output
        ])
```

```python
# === Train Generator ===
self.toggle_optimizer(opt_vae)
recon, mu, logvar = self.vae(real)

elbo = self.recon_loss(recon, real, mu, logvar)

d_fake, fake_feats = self.discriminator(recon, return_features=True)
_, real_feats = self.discriminator(real, return_features=True)

adv_loss = self.adversarial_loss(d_fake, True)
fm_loss = feature_matching_loss(real_feats, fake_feats)

total_gen_loss = elbo + self.adv_weight * adv_loss + fm_loss

self.manual_backward(total_gen_loss)
torch.nn.utils.clip_grad_norm_(self.vae.parameters(), max_norm=1.0)
opt_vae.step()
opt_vae.zero_grad()
self.untoggle_optimizer(opt_vae)

# === Train Discriminator ===
self.toggle_optimizer(opt_disc)
recon_detached = recon.detach()

d_real = self.discriminator(real)
d_fake = self.discriminator(recon_detached)

real_loss = self.adversarial_loss(d_real, True)
fake_loss = self.adversarial_loss(d_fake, False)
d_loss = 0.5 * (real_loss + fake_loss)

# Discriminator's prety strong, let the generator have some fun
if self.discriminator_pause != 0 and batch_idx % self.discriminator_pause == 0:
    opt_disc.zero_grad()
    self.log_dict({
        "gen/elbo": elbo,
        "gen/adv": adv_loss,
        "gen/fm": fm_loss,
        "gen/total": total_gen_loss,
        "disc/loss": d_loss,
    }, prog_bar=True, on_step=True, on_epoch=True)
    self.untoggle_optimizer(opt_disc)
    return

self.manual_backward(d_loss)
torch.nn.utils.clip_grad_norm_(self.discriminator.parameters(), max_norm=1.0)
opt_disc.step()
opt_disc.zero_grad()
self.untoggle_optimizer(opt_disc)
```

Code available at https://github.com/benjxia/LDMG

2.3 Diffusion Transformer Architecture



[192, T/64] Channel upsampled Latent Waveform

192-dimensional sinusoidal timestep conditioning vector

- Heavily inspired by DiT paper: https://arxiv.org/pdf/2212.09748
- Key modifications:
  - 1D convolution instead of point-wise feed-forward

```python
@staticmethod
def timestep_embedding(t, dim, max_period=10000):
    """
    Create sinusoidal timestep embeddings.
    :param t: a 1-D Tensor of N indices, one per batch element.
                      These may be fractional.
    :param dim: the dimension of the output.
    :param max_period: controls the minimum frequency of the embeddings.
    :return: an (N, D) Tensor of positional embeddings.
    """
    # https://github.com/openai/glide-text2im/blob/main/glide_text2im/nn.py
    half = dim // 2
    freqs = torch.exp(
        -math.log(max_period) * torch.arange(start=0, end=half, dtype=torch.float32) / half
    ).to(device=t.device)
    args = t[:, None].float() * freqs[None]
    embedding = torch.cat([torch.cos(args), torch.sin(args)], dim=-1)
    if dim % 2:
        embedding = torch.cat([embedding, torch.zeros_like(embedding[:, :1])], dim=-1)
    return embedding
```

```python
super(DiffusionTransformerBlock, self).__init__()
self.cross_attn_enabled = cross_attn_enabled

self.ln1 = nn.LayerNorm(input_channels, elementwise_affine=False)
self.attn = SelfAttention(input_channels, n_attn_heads, None)

self.ln_ca = nn.LayerNorm(input_channels, elementwise_affine=False) if cross_attn_enabled else nn.Identity()
self.cross_attn = CrossAttention(input_channels, n_attn_heads, None) if cross_attn_enabled else nn.Identity()

self.ln2 = nn.LayerNorm(input_channels, elementwise_affine=False)
self.ff = nn.Sequential(
    nn.Conv1d(input_channels, 4 * input_channels, DEFAULT_1D_KERNEL_SIZE, padding=DEFAULT_1D_PADDING),
    nn.GELU(),
    nn.Conv1d(4 * input_channels, input_channels, DEFAULT_1D_KERNEL_SIZE, padding=DEFAULT_1D_PADDING),
)

# adaptive layer norm
self.adaLN_modulation = nn.Sequential(
    nn.SiLU(),
    nn.Linear(input_channels, 9 * input_channels, bias=True)
)
```

2.3 Denoiser Training

```python
def q_sample(self, x_start, t, noise=None):
    self.to(x_start.device)
    if noise is None:
        noise = torch.randn_like(x_start)

    sqrt_alpha_cumprod_t = self._extract(self.sqrt_alpha_cumprod, t, x_start.shape)
    sqrt_one_minus_alpha_cumprod_t = self._extract(self.sqrt_one_minus_alpha_cumprod, t, x_start.shape)

    return sqrt_alpha_cumprod_t * x_start + sqrt_one_minus_alpha_cumprod_t * noise

def p_losses(self, model, x_start, t):
    self.to(x_start.device)
    noise = torch.randn_like(x_start)
    x_noisy = self.q_sample(x_start, t, noise)
    predicted_noise = model(x_noisy, t)
    return F.mse_loss(predicted_noise, noise)

@torch.no_grad()
def sample(self, model, shape, device):
    self.to(device)
    x = torch.randn(shape, device=device)

    for i in tqdm(reversed(range(self.timesteps)), total=self.timesteps):
        t = torch.full((shape[0],), i, device=device, dtype=torch.long)

        betas_t = self._extract(self.betas, t, x.shape)
        sqrt_recip_alphas_t = self._extract(self.sqrt_recip_alphas, t, x.shape)
        sqrt_one_minus_alpha_cumprod_t = self._extract(self.sqrt_one_minus_alpha_cumprod, t, x.shape)

        model_mean = sqrt_recip_alphas_t * (
            x - betas_t * model(x, t) / sqrt_one_minus_alpha_cumprod_t
        )

        if i > 0:
            posterior_variance_t = self._extract(self.posterior_variance, t, x.shape)
            noise = torch.randn_like(x)
            x = model_mean + torch.sqrt(posterior_variance_t) * noise
        else:
            x = model_mean

    return x
```

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Standard DDPM Training
https://arxiv.org/pdf/2006.11239

2.4 Modeling Challenges

- Diffusion was painful to get working…
  - We significantly underestimated the amount of compute needed to get a working model.
- Picking an effective KL-penalty weight in our VAE-GAN loss
  - Needed to balance efficient-ish training and high reconstruction fidelity
  - Generally speaking, a higher KL penalty makes latent diffusion models easier to train.
- Hyperparameter tuning was difficult since diffusion takes a LOT of time before you start seeing reasonable results, even with powerful GPU's.
- We originally planned to add text-conditioned diffusion, but training took too long :(
- Trained for approx. 24 hours

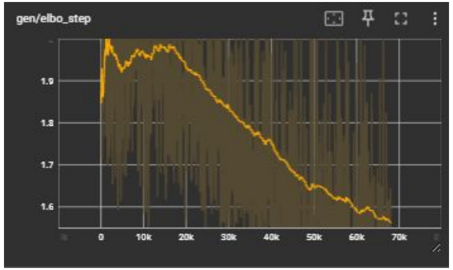
Modeling (Context)

- Model takes in an audio duration and generates coherent music
- The task: Encode the raw audio waveforms into a latent space, then reconstruct waveforms from the latent space with a decoder. Then, perform unconditioned audio synthesis by sampling from the latent space.
    - The latent space is represented as an audio waveform with a lower sampling rate from the input audio waveform, and a higher channel count.
- VAE-GAN for learning a powerful latent space
    - CNN Encoder/Decoder (Transpose convolutions for upsampling), CNN Discriminator
    - Training Objective: STFT reconstruction loss + Kullback-Leibler Divergence + Discriminator Hinge Loss + Discriminator feature-matching loss
    - Architecture inspired by Stable-Audio https://arxiv.org/pdf/2404.10301v2
- latent diffusion for unconditional generation.
- Minimize stft loss and KL divergence for VAE-GAN; minimize denoising loss for LDM.
-

Context: How do you formulate your task as an ML problem, e.g. what are the inputs, outputs, and what is being optimized? What models are appropriate for the task?

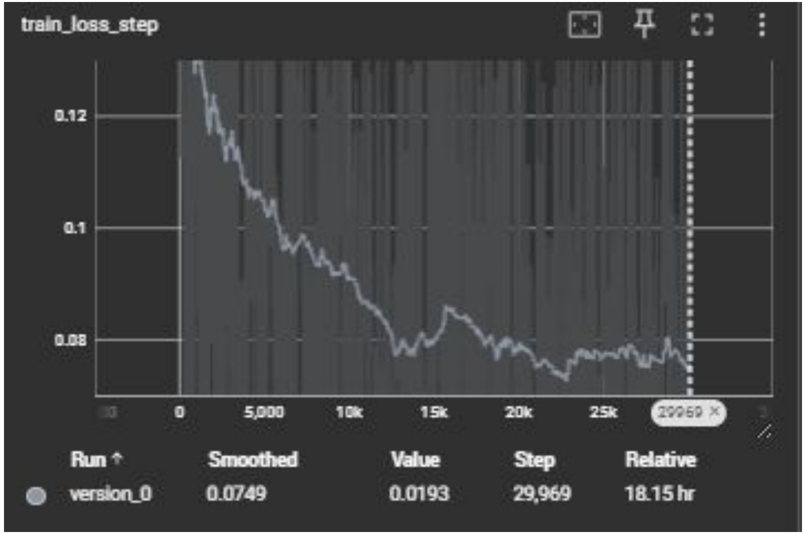
2.5 Evaluation

ELBO VAE Loss (Reconstruction + KL)



Adversarial Generator Loss

Diffusion MSE Training Loss



2.5 Evaluation

- Generative models in continuous domains are difficult to evaluate quantitatively. We'll mainly focus on qualitative comparisons against a VAE-GAN baseline. (Directly sampling from the latent space)

```python
def extract_vggish_embeddings(waveforms, sample_rate=16000, device='cuda' if torch.cuda.is_available() else 'cpu'):
    # postprocess = torchvggish.Postprocessor()

    all_embeddings = []
    for wav in waveforms:
        wav_np = wav.cpu().numpy().astype(np.float32)
        wav_np /= np.abs(wav_np).max() + 1e-6  # normalize to [-1, 1]

        examples = waveform_to_examples(wav_np, sample_rate)  # [B, 96, 64]
        examples_tensor = torch.tensor(examples).float().to(device)  # [B, 1, 96, 64] for vgg later

        embeddings = vgg(examples_tensor).detach().cpu().numpy()  # [B, 128] embeddings
        # embeddings = postprocess.postprocess(embeddings.detach().cpu())
        all_embeddings.append(embeddings)

    return np.concatenate(all_embeddings, axis=0)  # [N, 128]

@torch.no_grad()
def compute_fad(waveforms_real, waveforms_gen, sample_rate=16000, device='cuda' if torch.cuda.is_available() else 'cpu')
    """
    Compute Fréchet Audio Distance between two sets of waveforms using VGGish embeddings.

    Args:
        waveforms_real (torch.Tensor): shape [N, T], real audio waveforms.
        waveforms_gen  (torch.Tensor): shape [N, T], generated audio waveforms.
        sample_rate (int): sample rate (must be 16000 for VGGish).
        device (str): 'cuda' or 'cpu'

    Returns:
        float: FAD score
    """

    emb_real = extract_vggish_embeddings(waveforms_real)
    emb_gen = extract_vggish_embeddings(waveforms_gen)

    # mean and covariance
    mu_real, sigma_real = np.mean(emb_real, axis=0), np.cov(emb_real, rowvar=False)
    mu_gen, sigma_gen = np.mean(emb_gen, axis=0), np.cov(emb_gen, rowvar=False)

    # frechet distance
    covmean, _ = sqrtm(sigma_real @ sigma_gen, disp=False)
    if np.iscomplexobj(covmean):
        covmean = covmean.real

    fad = np.sum((mu_real - mu_gen) ** 2) + np.trace(sigma_real + sigma_gen - 2 * covmean)
    return float(fad)

def fad_model(model):
    batch = next(iter(dm.train_dataloader())).squeeze(1)
    with torch.no_grad():
        songs = model.generate(dm.train_dataloader().batch_size, 10) # shape [count, 1, sample_rate * dur]
    songs = songs[:, 0]
    print(compute_fad(batch, songs))
```

Samples

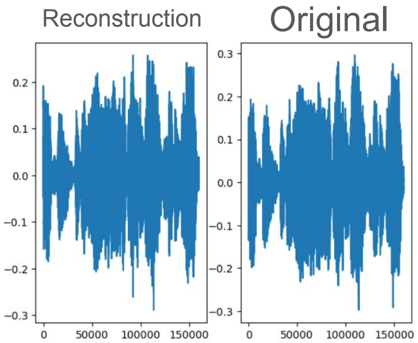
VAE-GAN Baseline	
Latent Diffusion	

Frechet Audio Distance

VAE-GAN Baseline	720143
Latent Diffusion	385721

2.5 Evaluation

- Qualitative VAE-GAN Audio Reconstruction Quality - to illustrate that our latent space is quite powerful!

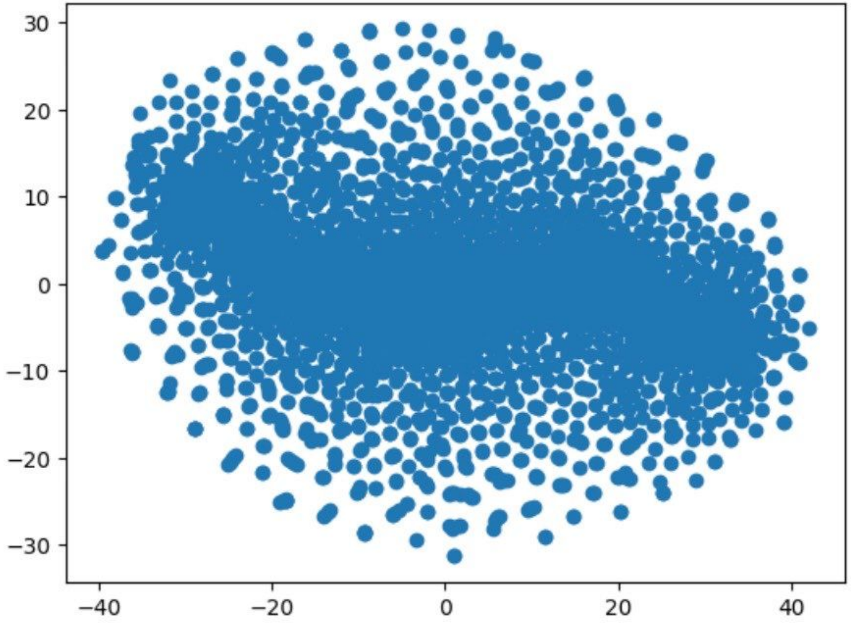



```
win = torch.hann_window(win_length).to(device)

x = torch.tensor(waveform_a, dtype=torch.float32, device=device)
y = torch.tensor(waveform_b, dtype=torch.float32, device=device)

# complex STFTs
stft_x = torch.stft(x, n_fft=n_fft, hop_length=hop_length, win_length=win_length,
                    window=win, return_complex=True)
stft_y = torch.stft(y, n_fft=n_fft, hop_length=hop_length, win_length=win_length,
                    window=win, return_complex=True)

# magnitude/log-magnitude
mag_x = stft_x.abs()
mag_y = stft_y.abs()

if use_log_mag:
    mag_x = torch.log1p(mag_x)
    mag_y = torch.log1p(mag_y)

loss = F.mse_loss(mag_x, mag_y)
return loss.item()
```

Original	Reconstruction	STFT Distance
		0.132
		0.043
		0.137

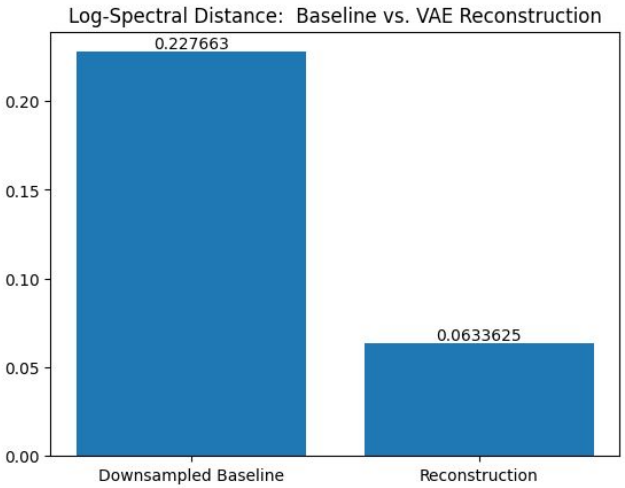
2.5 Evaluation

- Variational Autoencoder latent space t-SNE



2.5 Evaluation

- Evaluation of VAE-GAN:
  - *Baseline*: 50% downsampling followed by 50% upsampling of input spectrogram
  - *Metric*: Log-Spectral Distance (LSD), emphasizes relative change in spectral magnitude (making it a better metric for perceptual difference than mean-squared distance)



2.5 Evaluation

- Evaluation of diffusion:
  - *Baseline*: Latent vectors from VAE's encoder
  - *Metric*: Sliced Wasserstein Distance (SWD), capable of measuring how similar the VAE and diffusion model's latent spaces are
  - *Result*: The SWD between the VAE's latent space and the latent space constructed by diffusion is **0.0775**
  - *Interpretation*: Because we normalize the latent vectors before computing their distance, this metric implies that the distributions are **very similar yet are not identical**

2.6.1 Discussion (Dataset)

- Use of FMA dataset:
  - AudioLDM 2: Learning Holistic Audio Generation with Self-Supervised Pretraining (Liu et al. 2024)
  - Pengi: An Audio Language Model for Audio Tasks (Deshmukh et al. 2023)
  - Audio flamingo 2: An audio-language model with long-audio understanding and expert reasoning abilities (Ghosh, Sreyan, et al. 2025)
  - Multi-label Music Genre Classification from Audio, Text, and Images Using Deep Features (Oramas et al. 2017)
- Similar Datasets
  - Audioset: An Ontology and Human-Labeled Dataset for Audio Events (Gemmeke et al. 2017)
  - MusicCaps - MusicLM: Generating Music From Text (Agostinelli et al. 2023)

How has this dataset (or similar datasets) been used before?
How has prior work approached the same (or similar) tasks?
How do your results match or differ from what has been reported in related work?

2.6.2 Discussion (Approach)

- Latent diffusion model inspired by Evans, et al. "Long-form music generation with latent diffusion." arXiv preprint arXiv:2404.10301 (2024).
  - We use a simplified VAE and Diffusion Transformer architecture due to limited compute resources
  - The Stable Audio model introduced by Evans et al. is for text-conditioned generation, ours is unconditioned, which is arguably harder to execute because unconditioned generation has problems such as mode collapse.
- Many state of the art models uses latent diffusion over raw waveforms.
  - Bypasses the need for a vocoder

2.6 Discussion (Results)

- Our method is capable of high quality audio reconstruction - and learns a powerful latent space.
- Our diffusion model is capable of producing reasonable waveforms, and we’re confident that it could produce even better results if given more compute resources.
- Can easily modified to generate longer/variable length audio waveforms.
- Can be easily modified for text-conditioned audio synthesis
    - We had this mostly implemented - but realized training would take too long.

Results (MidiGPT & LDMG)