آموزش یک مدل زبانی کوچک برای ساخت موسیقی lo-fi

سهیل سلیمی

استاد راهنما: دکتر زجاجی

1. آموزش یک مدل زبانی کوچک برای ساخت موسیقی lo-fi
2. بیان مسله
   1. مشکل من
   2. حل مشکل
   3. هدف های این پروژه
3. ادبیات موضوع
   1. RWKV معماری
   2. MIDI
4. پروژه های پیشن
   1. مشکلات jacbz/Lofi
5. روش پیشنهادی
   1. مزیت های این روش
6. پیاده سازی
   1. جمع آوری داده
      1. مدل دارم
   2. تبدیل MIDI به متن
   3. آموزش مدل ها
   4. بررسی یادگیری مدل
   5. ارزیابی عملکرد مدل
      1. ارزیابی عملکرد مدل پیانو
   6. ساخت موسیقی نهایی
   7. خط لوله متن به Lo-fi
      1. ارزیابی خط لوله متن به Lo-fi
   8. نمایش پروژه
   9. نتیجه‌گیری و پیشنهادها
   10. منابع

بیان مسله

مشکل من

جستجوی موسیقی پس‌زمینه مناسب برای ویدیوهایم همیشه یک مانع بزرگ بود. از صرف ساعت‌ها وقت برای پیدا کردن موسیقی‌های رایگان که یا خیلی طولانی بودند یا خیلی گران، خسته شده بودم. این مشکل به شدت بهره‌وری من را کاهش می‌داد و می‌دانستم که باید راه‌حلی پیدا کنم. جستجو برای یک روش بهتر، ایده‌ای را در ذهنم جرقه زد و تصمیم گرفتم چالش ایجاد یک ابزار تولید موسیقی را بپذیرم.

حل مشکل

مشکل در کمبود آهنگ‌های موسیقی آماده موجود است که نیازهای خاص طول و هزینه را برآورده کنند. راه‌حل‌های فعلی، مانند وب‌سایت‌های موسیقی بدون حق امتیاز، اغلب در ارائه گزینه‌های مناسب ناکام می‌مانند. این مسئله را می‌توان به چالش‌های کلیدی زیر تقسیم کرد:

1. دسترسی محدود به آهنگ‌های موسیقی
2. طول‌های نامنظم آهنگ‌ها
3. هزینه‌های بالای مرتبط با مجوز موسیقی
4. ناتوانی در سفارشی‌سازی آهنگ‌ها برای برآورده کردن نیازهای خاص
5. دشواری در یافتن آهنگ‌هایی که با لحن و سبک مورد نظر مطابقت داشته باشند

هدف های این پروژه

هدف این پروژه ایجاد ابزاری است که آهنگ‌های موسیقی lo-fi با طول نامحدود تولید کند و به کاربران امکان دهد موسیقی را به دلخواه خود سفارشی و ویرایش کنند. هدف اصلی ارائه یک راه‌حل موسیقی پس‌زمینه با کیفیت بالا برای کاربردهای مختلف مانند تولید محتوای ویدئویی، پادکست‌ها، بازی‌ها و موسیقی آرامش‌بخش است.

این پروژه قصد دارد به این اهداف دست یابد:

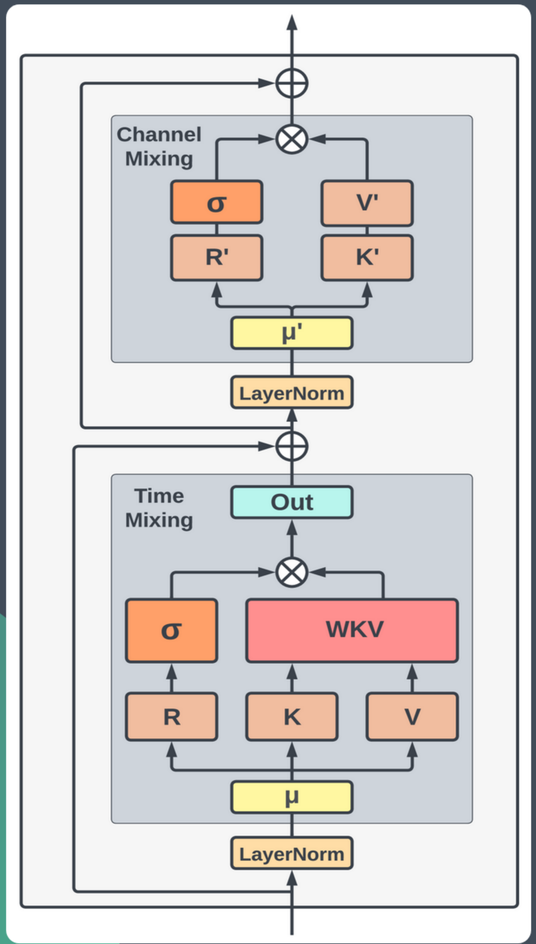
- تولید آهنگ‌های موسیقی که می‌توانند به هر طول خاصی گسترش یا کاهش یابند
- اجازه دادن به کاربران برای ویرایش و تغییر موسیقی به منظور تطبیق با لحن، سبک و حالت مورد نظرشان
- ارائه تجربه صوتی آرامش‌بخش و آرام برای گوش دادن شخصی

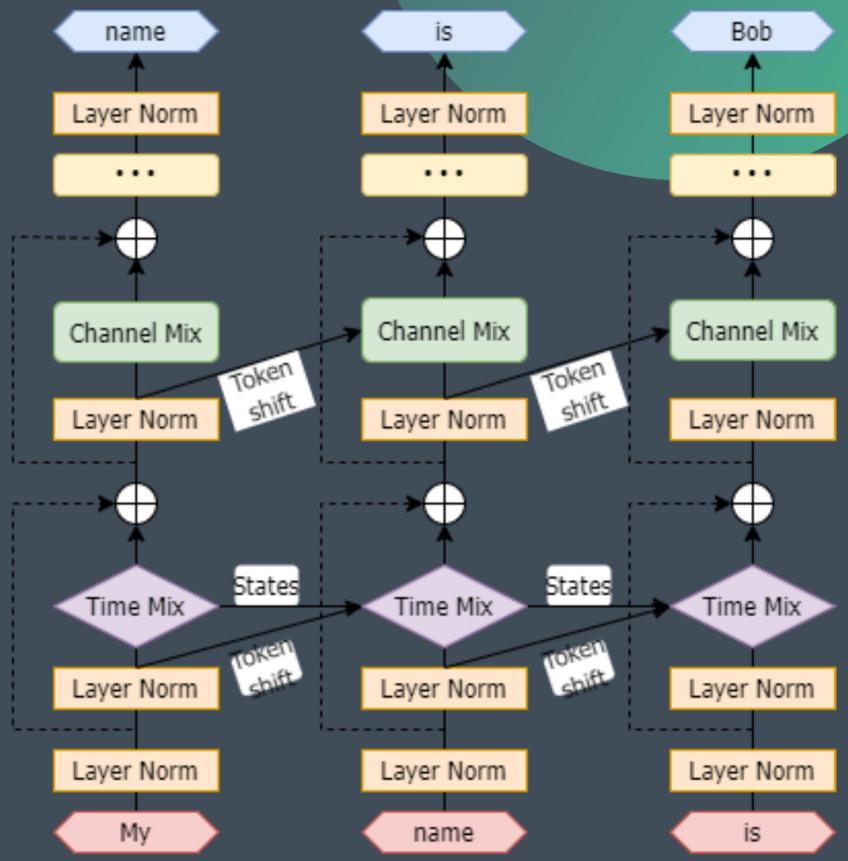
ادبیات موضوع

RWKV: Reinventing RNNs for the Transformer Era

مدل RWKV [1] یک مدل پیشرفته است که کارایی شبکه‌های عصبی برگشت‌پذیر را با قدرت توجه به خود ترکیب می‌کند و امکان انجام سریع و مؤثر وظایف دنباله به دنباله را فراهم می‌سازد.

معماری RWKV





معماری RWKV, گرفته شده از [1]

MIDI

MIDI [2] یک فرمت مبتنی بر متن است که موسیقی را به صورت الکترونیکی نمایش می‌دهد. این فرمت شامل اطلاعات نوت‌ها مانند ارتفاع صدا، مدت زمان و تمپو است. هیچ داده صوتی ندارد و فقط دستورالعمل‌هایی برای سازها ارائه می‌دهد. سبک و آسان برای ویرایش است. ورودی کلیدی برای مدل ما جهت تولید موسیقی لو-فای است.

پروژه های پیشن

jacbz/Lofi

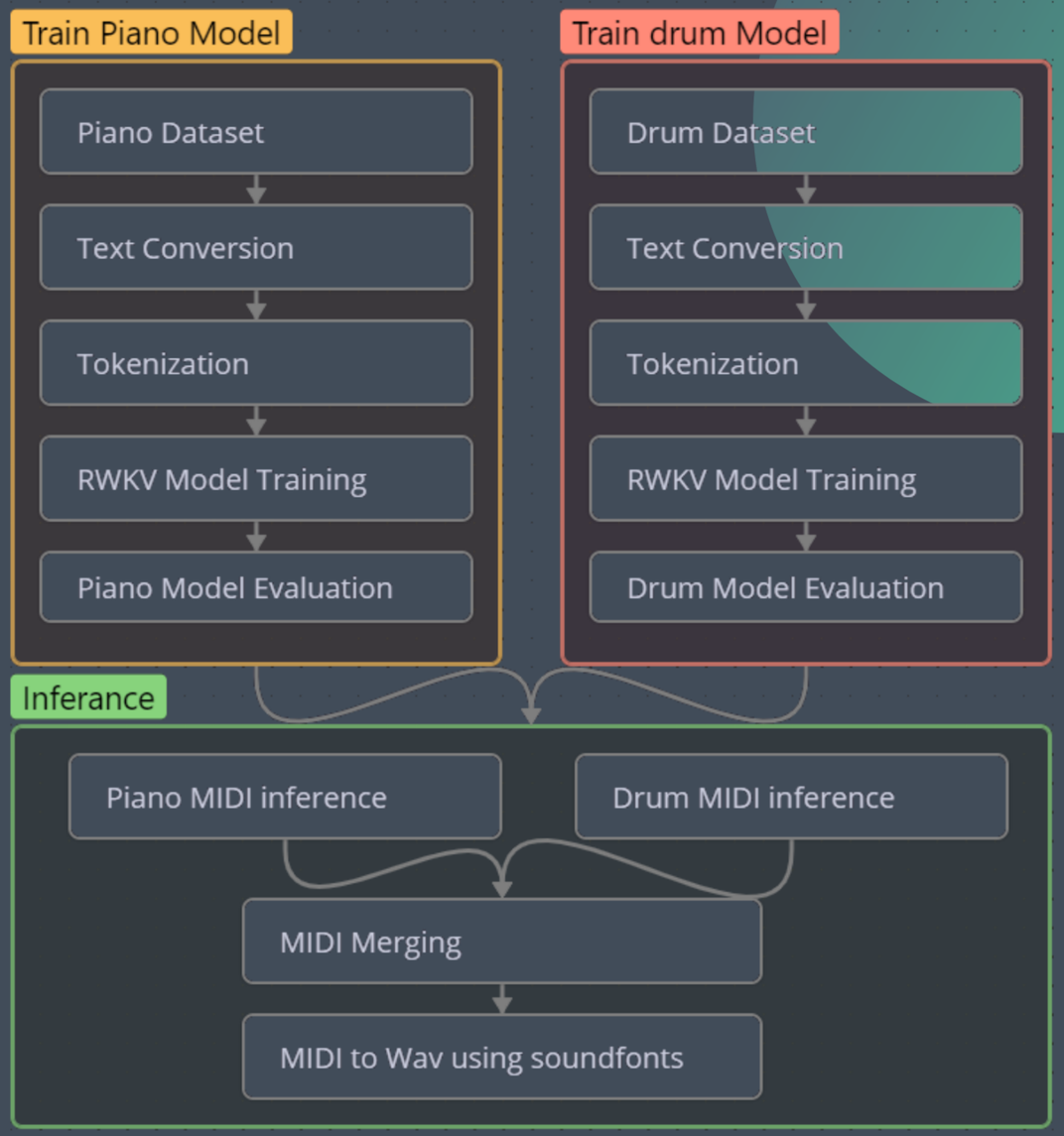
ML-supported lo-fi music generator Based on VAE [3]

مشکلات jacbz/Lofi

1. **مدیریت ناکافی وابستگی‌های ترتیبی:** طراحی VEA عمدتاً بر پردازش تصویر متمرکز است و این امر آن را برای وظایف ترتیبی مانند تولید موسیقی کمتر مؤثر می‌کند.
2. **عدم وجود مکانیزم توجه:** مکانیزم توجه VEA ساختار حافظه صریحی ارائه نمی‌دهد که برای ذخیره و بازیابی اطلاعات در وظایف ترتیبی حیاتی است.
3. **خروجی با فرمت wav:** این پروژه فایل نهایی Wav تولید می کند که امکان تغییر های بعدی را روی موسیقی سخت می کند.
4. **عدم کنترل صریح بر خروجی:** خروجی VEA توسط latent space تعیین می‌شود که کنترل دنباله خروجی را چالش‌برانگیز می‌کند.

روش پیشنهادی

ما ویژگی‌های صوتی را از نت‌های MIDI با استفاده از یک ابزار تبدیل MIDI به صوت استخراج کردیم. ما یک نمایش توکنیزه شده از ویژگی‌های صوتی با استفاده از یک واژگان از مقادیر گسسته ایجاد کردیم. ما دو مدل RWKV جداگانه، یکی برای پیانو و یکی برای درام، بر روی داده‌های توکنیزه شده آموزش دادیم. مدل پیانو بر روی یک مجموعه داده از 216,284 فایل MIDI پیانو و مدل درام بر روی یک مجموعه داده از 45,537 فایل MIDI درام آموزش داده شد. مدل‌ها استفاده کردیم. ما مدل‌ها را با استفاده از معیارهای موسیقی مانند هماهنگی ریتم و انسجام ملودی ارزیابی کردیم. مدل‌ها توانستند موسیقی با کیفیت بالا و لو-فای تولید کنند که به نت‌های ورودی MIDI بسیار نزدیک بود.



دیاگرام روش انجام پروژه

مزیت های این روش

1. **حفظ ساختار زمانی**: RWKV به دلیل استفاده از مکانیزم‌های بازگشتی، قادر است ساختار زمانی و توالی‌های طولانی را بهتر حفظ کند. این ویژگی برای موزیک لو-فاي که اغلب دارای الگوهای تکراری و ریتمیک است، بسیار مهم است.
2. **کیفیت بازسازی بهتر**: RWKV به دلیل استفاده از مکانیزم توجه، می‌تواند جزئیات بیشتری از داده‌های ورودی را حفظ کند و بازسازی دقیق‌تری ارائه دهد.
3. **انعطاف‌پذیری بیشتر**: این معماری به دلیل استفاده از مکانیزم‌های توجه (Attention mechanism)
4. [4]، می‌تواند به طور دینامیک به بخش‌های مختلف داده توجه کند و این امر باعث می‌شود که در تولید موزیک‌های پیچیده‌تر و متنوع‌تر عملکرد بهتری داشته باشد.

پیاده سازی

- جمع آوری داده
- تبدیل داده ها به متن
- آموزش مدل ها
- ارزیابی عملکرد مدل ها
- نحویه ساخت موسیقی نهایی

جمع آوری داده

مدل پیانو

	IrishMMD Dataset [5]
کل فایل ها	216,284
فایل های آموزش	214,208 (99%)
فایل اعتبارسنجی	2,076 (1%)

جمع آوری داده

مدل پیانو

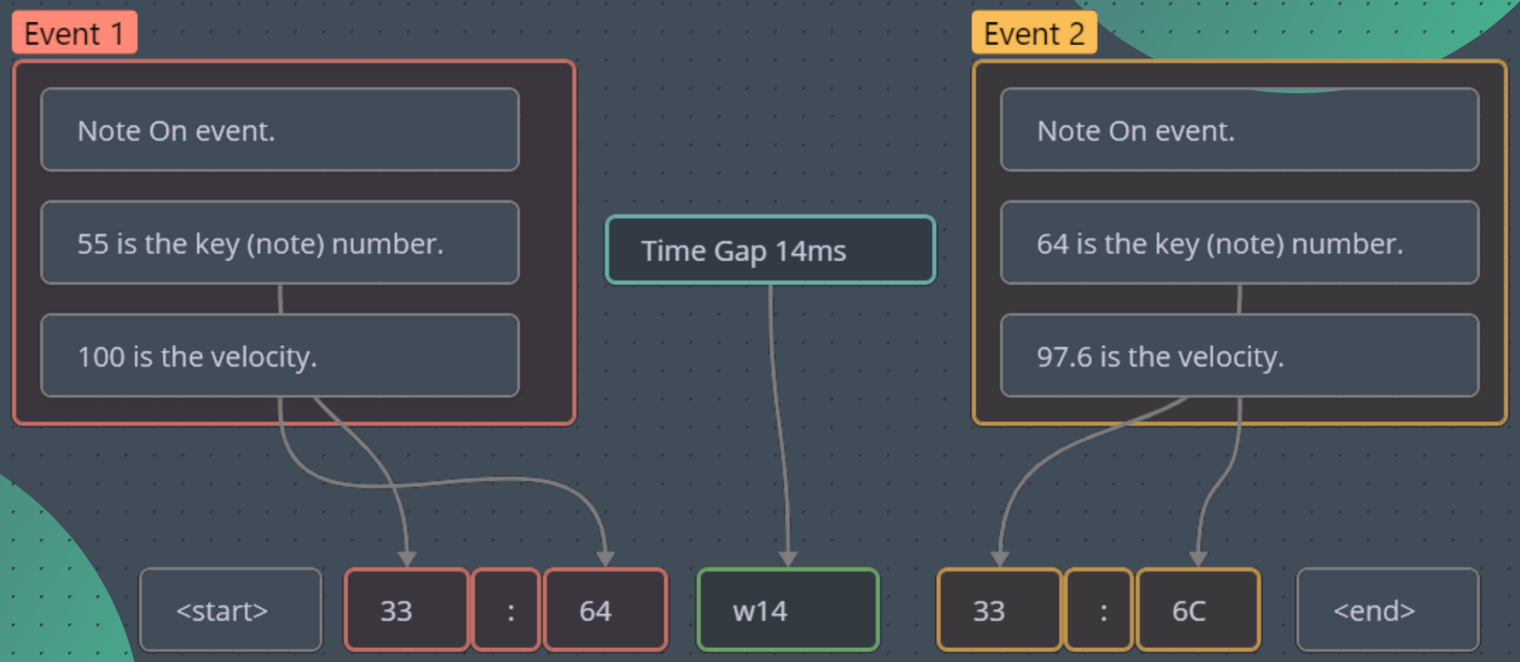
	IrishMMD Dataset [5]
کل فایل ها	216,284
فایل های آموزش	214,208 (99%)
فایل اعتبارسنجی	2,076 (1%)

مدل دارم

Expanded Groove MIDI Dataset [6]

بخش	تعداد کل توالی‌ها	مدت زمان (ساعت)
آموزشی	35,217	341.4
اعتبارسنجی	5,031	52.2
کل	45,537	444.5

تبدیل MIDI به متن



نحویه تبدیل MIDI به متن

```python
def convert_midi_to_str(mid, cfg, augment=None):
  output_list = []
  for msg in mid.tracks[0]:
    delta_time = \
      mido.tick2second(msg.time, tempo) * 1000.0
    if msg.is_meta and msg.type == "set_tempo":
      tempo *= augment.time_stretch_factor
    elif msg.type in ("note_on", "note_off"):
      handle_note(msg.channel, msg.velocity, msg.note)
  flush_token_data_buffer(output_list)
  return output_list

def handle_note(channel, velocity, note):
  if velocity == 0:
    # handle note off
  else:
    # handle note on
    consume_note_program_data(channel, note, velocit
```

```python
def format_wait_token(wait):
  return f"t{wait}"

def format_note_token(note, velocity_bin):
  return f"{note:x}:{velocity_bin:x}"

def velocity_to_bin(velocity):
  velocity = min(max(velocity, 0), 128 - 1)
  return ceil((128 *
   ((0.33 ** (velocity /128) - 1)
    / (0.33 - 1)) / 16))
```

$$\left\lceil \frac{128 \cdot \left(\frac{0.33^{\left(\frac{\text{velocity}}{128}\right)} - 1}{0.33 - 1}\right)}{16} \right\rceil$$

```python
def convert_midi_to_str(mid, cfg, augment=None):
  output_list = []
  for msg in mid.tracks[0]:
    delta_time = \
      mido.tick2second(msg.time, tempo) * 1000.0
    if msg.is_meta and msg.type == "set_tempo":
      tempo *= augment.time_stretch_factor
    elif msg.type in ("note_on", "note_off"):
      handle_note(msg.channel, msg.velocity, msg.note)
  flush_token_data_buffer(output_list)
  return output_list

def handle_note(channel, velocity, note):
  if velocity == 0:
    # handle note off
  else:
    # handle note on
    consume_note_program_data(channel, note, velocit
```

```python
def format_wait_token(wait):
  return f"t{wait}"

def format_note_token(note, velocity_bin):
  return f"{note:x}:{velocity_bin:x}"

def velocity_to_bin(velocity):
  velocity = min(max(velocity, 0), 128 - 1)
  return ceil((128 *
   ((0.33 ** (velocity /128) - 1)
    / (0.33 - 1)) / 16))
```

$$\left\lceil \frac{128 \cdot \left(\frac{0.33^{\left(\frac{\text{velocity}}{128}\right)} - 1}{0.33 - 1}\right)}{16} \right\rceil$$

```python
def convert_midi_to_str(mid, cfg, augment=None):
  output_list = []
  for msg in mid.tracks[0]:
    delta_time = \
      mido.tick2second(msg.time, tempo) * 1000.0
    if msg.is_meta and msg.type == "set_tempo":
      tempo *= augment.time_stretch_factor
    elif msg.type in ("note_on", "note_off"):
      handle_note(msg.channel, msg.velocity, msg.note)
  flush_token_data_buffer(output_list)
  return output_list

def handle_note(channel, velocity, note):
  if velocity == 0:
    # handle note off
  else:
    # handle note on
    consume_note_program_data(channel, note, velocit
```

```python
def format_wait_token(wait):
  return f"t{wait}"

def format_note_token(note, velocity_bin):
  return f"{note:x}:{velocity_bin:x}"

def velocity_to_bin(velocity):
  velocity = min(max(velocity, 0), 128 - 1)
  return ceil((128 *
   ((0.33 ** (velocity /128) - 1)
    / (0.33 - 1)) / 16))
```

$$\left\lceil \frac{128 \cdot \left(\frac{0.33^{\left(\frac{\text{velocity}}{128}\right)} - 1}{0.33 - 1}\right)}{16} \right\rceil$$

```python
def convert_midi_to_str(mid, cfg, augment=None):
  output_list = []
  for msg in mid.tracks[0]:
    delta_time = \
      mido.tick2second(msg.time, tempo) * 1000.0
    if msg.is_meta and msg.type == "set_tempo":
      tempo *= augment.time_stretch_factor
    elif msg.type in ("note_on", "note_off"):
      handle_note(msg.channel, msg.velocity, msg.note)
  flush_token_data_buffer(output_list)
  return output_list

def handle_note(channel, velocity, note):
  if velocity == 0:
    # handle note off
  else:
    # handle note on
    consume_note_program_data(channel, note, velocit
```

```python
def format_wait_token(wait):
  return f"t{wait}"

def format_note_token(note, velocity_bin):
  return f"{note:x}:{velocity_bin:x}"

def velocity_to_bin(velocity):
  velocity = min(max(velocity, 0), 128 - 1)
  return ceil((128 *
   ((0.33 ** (velocity /128) - 1)
    / (0.33 - 1)) / 16))
```

$$\left\lceil \frac{128 \cdot \left(\frac{0.33^{\left(\frac{\text{velocity}}{128}\right)} - 1}{0.33 - 1}\right)}{16} \right\rceil$$

```python
def convert_midi_to_str(mid, cfg, augment=None):
  output_list = []
  for msg in mid.tracks[0]:
    delta_time = \
      mido.tick2second(msg.time, tempo) * 1000.0
    if msg.is_meta and msg.type == "set_tempo":
      tempo *= augment.time_stretch_factor
    elif msg.type in ("note_on", "note_off"):
      handle_note(msg.channel, msg.velocity, msg.note)
  flush_token_data_buffer(output_list)
  return output_list

def handle_note(channel, velocity, note):
  if velocity == 0:
    # handle note off
  else:
    # handle note on
    consume_note_program_data(channel, note, velocit
```

```python
def format_wait_token(wait):
  return f"t{wait}"

def format_note_token(note, velocity_bin):
  return f"{note:x}:{velocity_bin:x}"

def velocity_to_bin(velocity):
  velocity = min(max(velocity, 0), 128 - 1)
  return ceil((128 *
   ((0.33 ** (velocity /128) - 1)
    / (0.33 - 1)) / 16))
```

$$\left\lceil \frac{128 \cdot \left(\frac{0.33^{\left(\frac{\text{velocity}}{128}\right)} - 1}{0.33 - 1}\right)}{16} \right\rceil$$

```python
def convert_midi_to_str(mid, cfg, augment=None):
  output_list = []
  for msg in mid.tracks[0]:
    delta_time = \
      mido.tick2second(msg.time, tempo) * 1000.0
    if msg.is_meta and msg.type == "set_tempo":
      tempo *= augment.time_stretch_factor
    elif msg.type in ("note_on", "note_off"):
      handle_note(msg.channel, msg.velocity, msg.note)
  flush_token_data_buffer(output_list)
  return output_list

def handle_note(channel, velocity, note):
  if velocity == 0:
    # handle note off
  else:
    # handle note on
    consume_note_program_data(channel, note, velocit
```

```python
def format_wait_token(wait):
  return f"t{wait}"

def format_note_token(note, velocity_bin):
  return f"{note:x}:{velocity_bin:x}"

def velocity_to_bin(velocity):
  velocity = min(max(velocity, 0), 128 - 1)
  return ceil((128 *
   ((0.33 ** (velocity /128) - 1)
    / (0.33 - 1)) / 16))
```

$$\left\lceil \frac{128 \cdot \left(\frac{0.33^{\left(\frac{\text{velocity}}{128}\right)} - 1}{0.33 - 1}\right)}{16} \right\rceil$$

```python
def convert_midi_to_str(mid, cfg, augment=None):
  output_list = []
  for msg in mid.tracks[0]:
    delta_time = \
      mido.tick2second(msg.time, tempo) * 1000.0
    if msg.is_meta and msg.type == "set_tempo":
      tempo *= augment.time_stretch_factor
    elif msg.type in ("note_on", "note_off"):
      handle_note(msg.channel, msg.velocity, msg.note)
  flush_token_data_buffer(output_list)
  return output_list

def handle_note(channel, velocity, note):
  if velocity == 0:
    # handle note off
  else:
    # handle note on
    consume_note_program_data(channel, note, velocit
```

```python
def format_wait_token(wait):
  return f"t{wait}"

def format_note_token(note, velocity_bin):
  return f"{note:x}:{velocity_bin:x}"

def velocity_to_bin(velocity):
  velocity = min(max(velocity, 0), 128 - 1)
  return ceil((128 *
   ((0.33 ** (velocity /128) - 1)
    / (0.33 - 1)) / 16))
```

$$\left\lceil \frac{128 \cdot \left(\frac{0.33^{\left(\frac{\text{velocity}}{128}\right)} - 1}{0.33 - 1}\right)}{16} \right\rceil$$

```python
def convert_midi_to_str(mid, cfg, augment=None):
  output_list = []
  for msg in mid.tracks[0]:
    delta_time = \
      mido.tick2second(msg.time, tempo) * 1000.0
    if msg.is_meta and msg.type == "set_tempo":
      tempo *= augment.time_stretch_factor
    elif msg.type in ("note_on", "note_off"):
      handle_note(msg.channel, msg.velocity, msg.note)
  flush_token_data_buffer(output_list)
  return output_list

def handle_note(channel, velocity, note):
  if velocity == 0:
    # handle note off
  else:
    # handle note on
    consume_note_program_data(channel, note, velocit
```

```python
def format_wait_token(wait):
  return f"t{wait}"

def format_note_token(note, velocity_bin):
  return f"{note:x}:{velocity_bin:x}"

def velocity_to_bin(velocity):
  velocity = min(max(velocity, 0), 128 - 1)
  return ceil((128 *
   ((0.33 ** (velocity /128) - 1)
    / (0.33 - 1)) / 16))
```

$$\left\lceil \frac{128 \cdot \left(\frac{0.33^{\left(\frac{\text{velocity}}{128}\right)} - 1}{0.33 - 1}\right)}{16} \right\rceil$$

```python
def convert_midi_to_str(mid, cfg, augment=None):
  output_list = []
  for msg in mid.tracks[0]:
    delta_time = \
      mido.tick2second(msg.time, tempo) * 1000.0
    if msg.is_meta and msg.type == "set_tempo":
      tempo *= augment.time_stretch_factor
    elif msg.type in ("note_on", "note_off"):
      handle_note(msg.channel, msg.velocity, msg.note)
  flush_token_data_buffer(output_list)
  return output_list

def handle_note(channel, velocity, note):
  if velocity == 0:
    # handle note off
  else:
    # handle note on
    consume_note_program_data(channel, note, velocit
```

```python
def format_wait_token(wait):
  return f"t{wait}"

def format_note_token(note, velocity_bin):
  return f"{note:x}:{velocity_bin:x}"

def velocity_to_bin(velocity):
  velocity = min(max(velocity, 0), 128 - 1)
  return ceil((128 *
   ((0.33 ** (velocity /128) - 1)
    / (0.33 - 1)) / 16))
```

$$\left\lceil \frac{128 \cdot \left(\frac{0.33^{\left(\frac{\text{velocity}}{128}\right)} - 1}{0.33 - 1}\right)}{16} \right\rceil$$

```python
def convert_midi_to_str(mid, cfg, augment=None):
  output_list = []
  for msg in mid.tracks[0]:
    delta_time = \
      mido.tick2second(msg.time, tempo) * 1000.0
    if msg.is_meta and msg.type == "set_tempo":
      tempo *= augment.time_stretch_factor
    elif msg.type in ("note_on", "note_off"):
      handle_note(msg.channel, msg.velocity, msg.note)
  flush_token_data_buffer(output_list)
  return output_list

def handle_note(channel, velocity, note):
  if velocity == 0:
    # handle note off
  else:
    # handle note on
    consume_note_program_data(channel, note, velocit
```

```python
def format_wait_token(wait):
  return f"t{wait}"

def format_note_token(note, velocity_bin):
  return f"{note:x}:{velocity_bin:x}"

def velocity_to_bin(velocity):
  velocity = min(max(velocity, 0), 128 - 1)
  return ceil((128 *
   ((0.33 ** (velocity /128) - 1)
    / (0.33 - 1)) / 16))
```

$$\left\lceil \frac{128 \cdot \left(\frac{0.33^{\left(\frac{\text{velocity}}{128}\right)} - 1}{0.33 - 1}\right)}{16} \right\rceil$$

```python
def convert_midi_to_str(mid, cfg, augment=None):
  output_list = []
  for msg in mid.tracks[0]:
    delta_time = \
      mido.tick2second(msg.time, tempo) * 1000.0
    if msg.is_meta and msg.type == "set_tempo":
      tempo *= augment.time_stretch_factor
    elif msg.type in ("note_on", "note_off"):
      handle_note(msg.channel, msg.velocity, msg.note)
  flush_token_data_buffer(output_list)
  return output_list

def handle_note(channel, velocity, note):
  if velocity == 0:
    # handle note off
  else:
    # handle note on
    consume_note_program_data(channel, note, velocit
```

```python
def format_wait_token(wait):
  return f"t{wait}"

def format_note_token(note, velocity_bin):
  return f"{note:x}:{velocity_bin:x}"

def velocity_to_bin(velocity):
  velocity = min(max(velocity, 0), 128 - 1)
  return ceil((128 *
   ((0.33 ** (velocity /128) - 1)
    / (0.33 - 1)) / 16))
```

$$\left\lceil \frac{128 \cdot \left(\frac{0.33^{\left(\frac{\text{velocity}}{128}\right)} - 1}{0.33 - 1}\right)}{16} \right\rceil$$

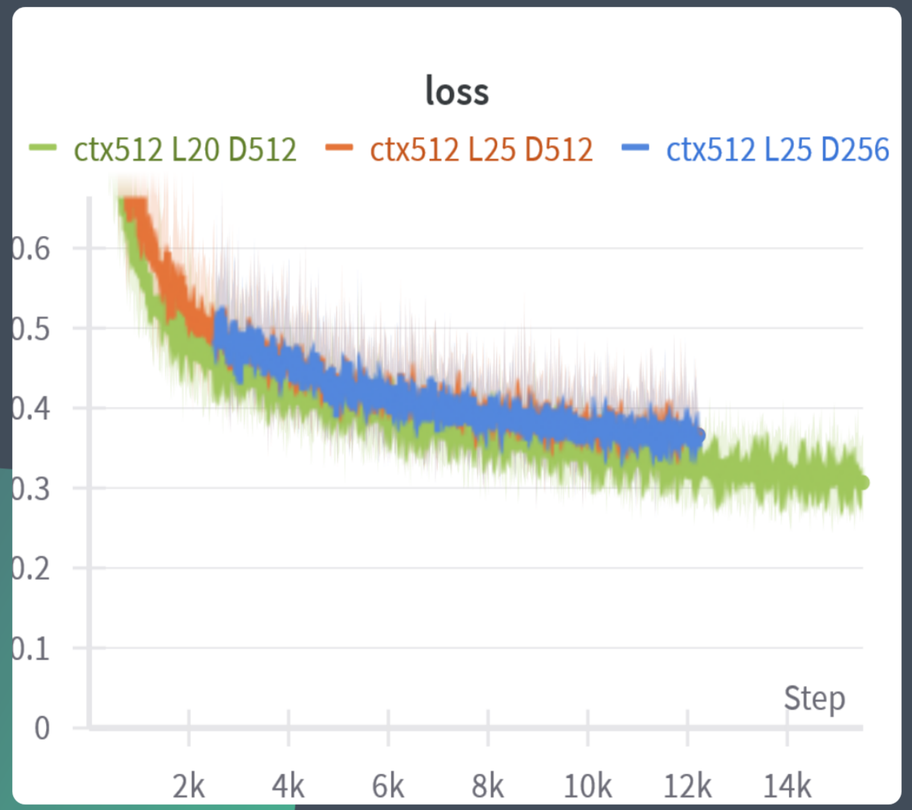
آموزش مدل ها

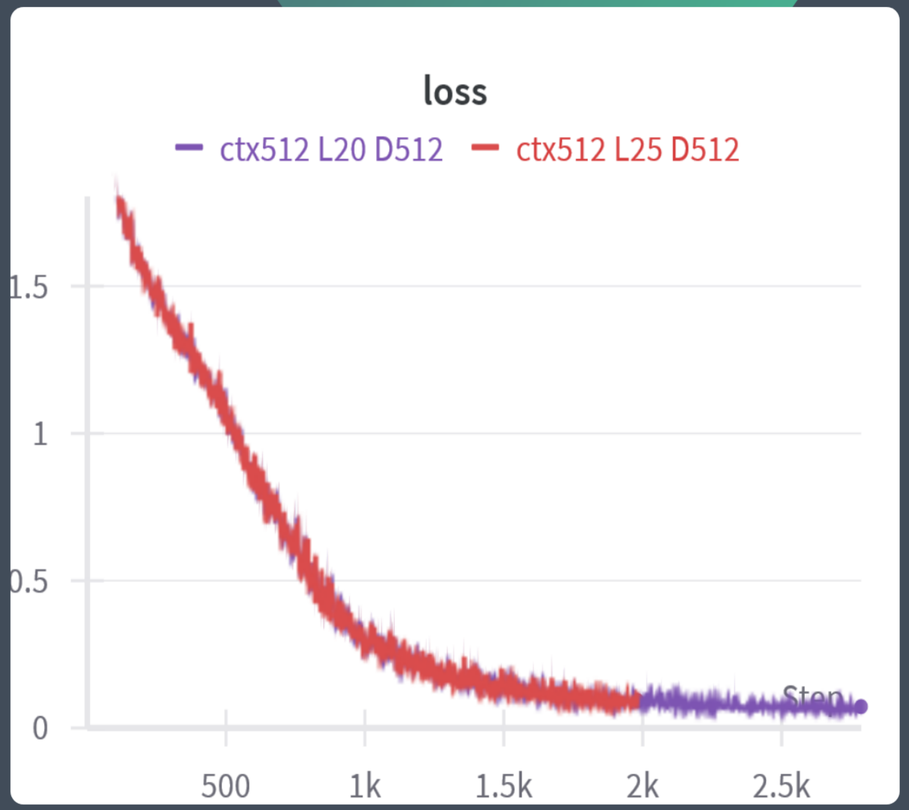
ما از معماری [7] RWKV-6.0 با 20 لایه و Embedding برابر با 512 استفاده کردیم. طول زمینه مدل 512 است. نرخ یادگیری اولیه و نهایی به ترتیب 6e-4 و 6e-5 است و از تابع آنتروپی متقاطع برای محاسبه خطا استفاده می‌شود. پارامتر head_size_a برابر با 64 انتخاب شده تا مدل بتواند به تعداد بیشتری از عناصر ورودی توجه کند. از کتابخانه DeepSpeed نیز استفاده شده که شامل بهینه‌ساز Adam و زمان‌بند نرخ یادگیری کاهش گرم‌شونده است. همچنین، دقت مختلط bfloat16 یا float16 برای بهبود سرعت آموزش فعال شده است.

آموزش مدل ها

ما از معماری [7] RWKV-6.0 با 20 لایه و Embedding برابر با 512 استفاده کردیم. طول زمینه مدل 512 است. نرخ یادگیری اولیه و نهایی به ترتیب 6e-4 و 6e-5 است و از تابع آنتروپی متقاطع برای محاسبه خطا استفاده می‌شود. پارامتر head_size_a برابر با 64 انتخاب شده تا مدل بتواند به تعداد بیشتری از عناصر ورودی توجه کند. از کتابخانه DeepSpeed نیز استفاده شده که شامل بهینه‌ساز Adam و زمان‌بند نرخ یادگیری کاهش گرم‌شونده است. همچنین، دقت مختلط bfloat16 یا float16 برای بهبود سرعت آموزش فعال شده است.

بررسی یادگیری مدل





نمودار تابع خطا آموزش مدل های درام و پیانو

ارزیابی عملکرد مدل

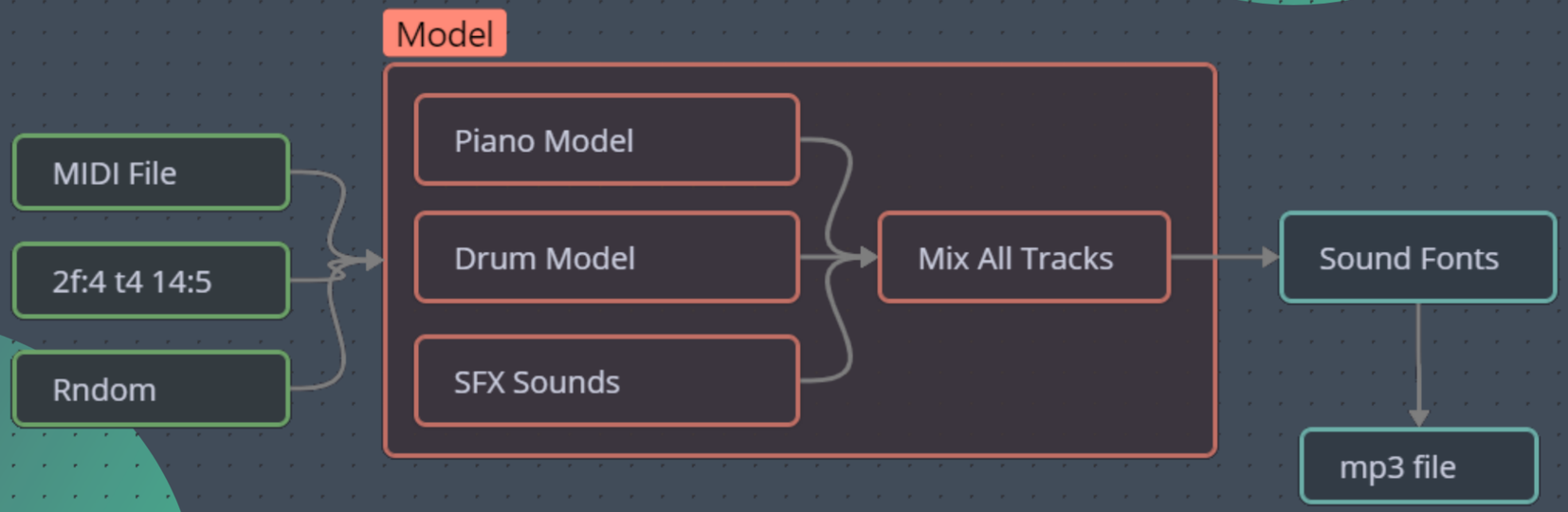
برخی از معیار های موسیقی [8]

- **هماهنگی ریتم**: هماهنگی ریتم یک متریک است که به بررسی نوسان یا یکنواختی طول نت ها در یک قطعه موسیقی می پردازد. این متریک نشان می دهد که قطعه های موسیقی چه میزان یکنواختی و یا چه میزان نوسانی دارند.
- **شباهت ملودیک**: شباهت ملودی، یک روش برای اندازه گیری شباهت بین دو ملودی بر اساس ترتیب نت‌هایشان است. این متد ساده و مستقیماً از نسبت نت‌های مشابه دو ملودی برای محاسبه شباهت استفاده می‌کند.
- **انسجام هارمونیک**: اندازه‌گیری میزان سازگاری و انسجام هارمونی‌ها و پیشرفت‌های آکورد در یک قطعه موسیقی با کلید یا مرکز تونال زیرین، با در نظر گرفتن عواملی مانند عملکرد آکورد، هدایت صدا و حل هارمونیک.

ارزیابی عملکرد مدل پیانو

متد	مقدار
هماهنگی ریتم	0.34
هماهنگی هارمونیک (ملایمت)	0.87
هماهنگی هارمونیک (ناهم‌خوانی)	0.13
شباهت ملودیک	0.09

ساخت موسیقی نهایی



روند ساخت موسیقی نهایی

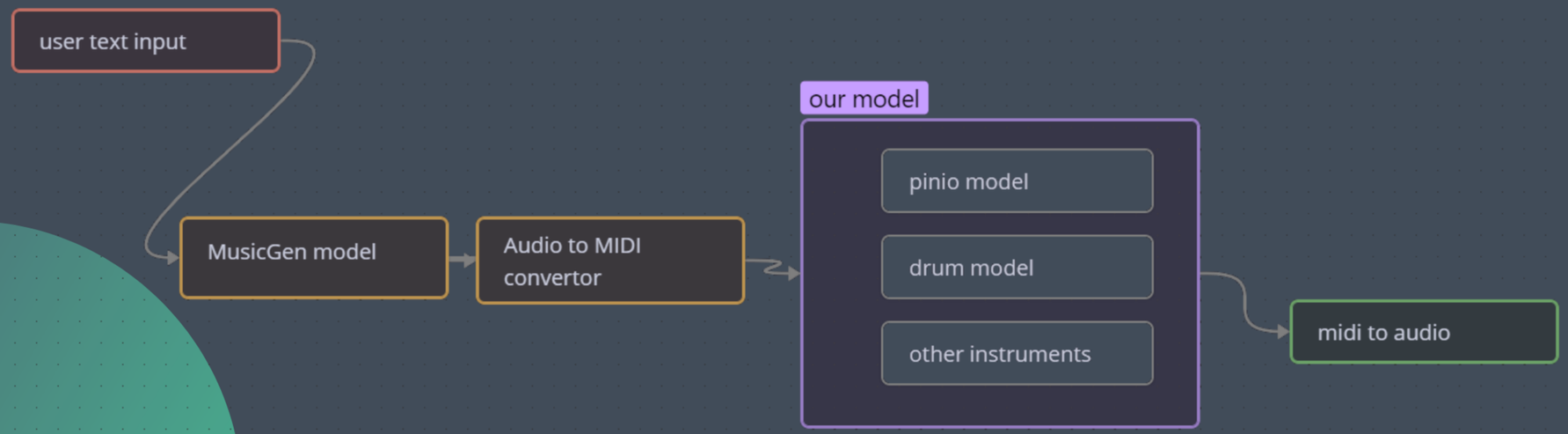
```python
def midiToWav():
    # Load soundfont and synthesize MIDI to WAV
    synth = tinysoundfont.Synth()
    synth.midi_load(midi)
    return synth.tobytes()

class GenMusic:
    def __init__(self, data, dataDrum):
        # Generate MIDI and convert to audio segments
        tempo = random.randint(14, 18) * 4 * 3
        self.pianoRoll = AudioSegment(midiToWav("./OmegaGMGS2.sf2",data))
        self.fill = AudioSegment.from_file(random.choice(os.listdir("./loops/vinyl")))
        self.drum = AudioSegment(midiToWav("./FluidR3_GM.sf2",dataDrum))

    def mix_lines(self, music_len=60):
        return music.fade_out(2000), \
        self.RandRoll.overlay(self.fill, position=5).overlay(self.drum, position=-5) \
         * (music_len // self.RandRoll.duration_seconds).fade_out(2000)
```

```python
def midiToWav():
    # Load soundfont and synthesize MIDI to WAV
    synth = tinysoundfont.Synth()
    synth.midi_load(midi)
    return synth.tobytes()

class GenMusic:
    def __init__(self, data, dataDrum):
        # Generate MIDI and convert to audio segments
        tempo = random.randint(14, 18) * 4 * 3
        self.pianoRoll = AudioSegment(midiToWav("./OmegaGMGS2.sf2",data))
        self.fill = AudioSegment.from_file(random.choice(os.listdir("./loops/vinyl")))
        self.drum = AudioSegment(midiToWav("./FluidR3_GM.sf2",dataDrum))

    def mix_lines(self, music_len=60):
        return music.fade_out(2000), \
        self.RandRoll.overlay(self.fill, position=5).overlay(self.drum, position=-5) \
         * (music_len // self.RandRoll.duration_seconds).fade_out(2000)
```

```python
def midiToWav():
    # Load soundfont and synthesize MIDI to WAV
    synth = tinysoundfont.Synth()
    synth.midi_load(midi)
    return synth.tobytes()

class GenMusic:
    def __init__(self, data, dataDrum):
        # Generate MIDI and convert to audio segments
        tempo = random.randint(14, 18) * 4 * 3
        self.pianoRoll = AudioSegment(midiToWav("./OmegaGMGS2.sf2",data))
        self.fill = AudioSegment.from_file(random.choice(os.listdir("./loops/vinyl")))
        self.drum = AudioSegment(midiToWav("./FluidR3_GM.sf2",dataDrum))

    def mix_lines(self, music_len=60):
        return music.fade_out(2000), \
        self.RandRoll.overlay(self.fill, position=5).overlay(self.drum, position=-5) \
         * (music_len // self.RandRoll.duration_seconds).fade_out(2000)
```

```python
def midiToWav():
    # Load soundfont and synthesize MIDI to WAV
    synth = tinysoundfont.Synth()
    synth.midi_load(midi)
    return synth.tobytes()

class GenMusic:
    def __init__(self, data, dataDrum):
        # Generate MIDI and convert to audio segments
        tempo = random.randint(14, 18) * 4 * 3
        self.pianoRoll = AudioSegment(midiToWav("./OmegaGMGS2.sf2",data))
        self.fill = AudioSegment.from_file(random.choice(os.listdir("./loops/vinyl")))
        self.drum = AudioSegment(midiToWav("./FluidR3_GM.sf2",dataDrum))

    def mix_lines(self, music_len=60):
        return music.fade_out(2000), \
        self.RandRoll.overlay(self.fill, position=5).overlay(self.drum, position=-5) \
         * (music_len // self.RandRoll.duration_seconds).fade_out(2000)
```

خط لوله متن به Lo-fi



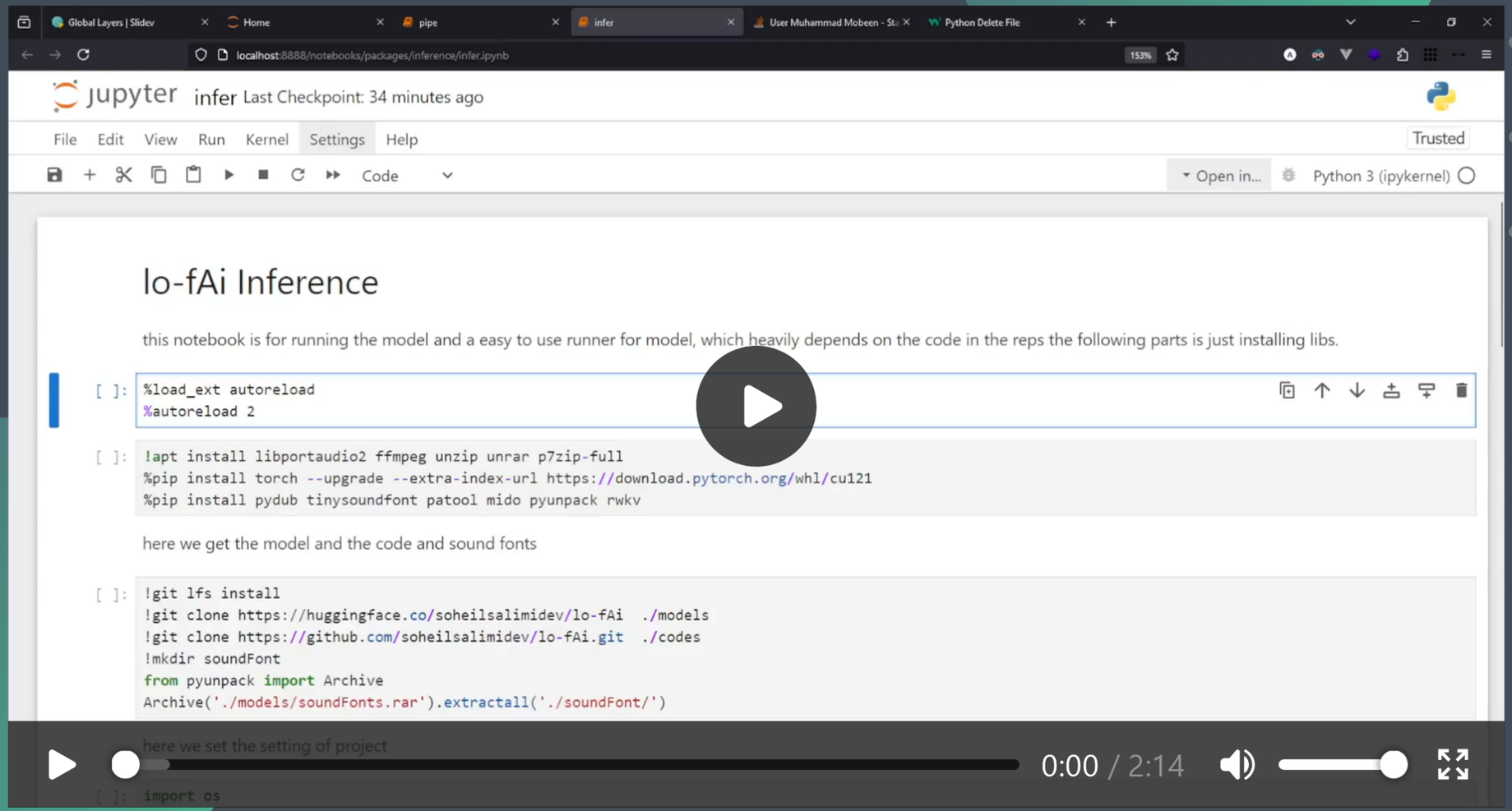
ارزیابی خط لوله متن به Lo-fi

برای ارزیابی این خط لوله ما از پنج نفر نظرسنجی کردیم. نتایج به شکل زیر است:

سوال	پاسخ
وضوح صدای تولید شده	20% (یک نفر) (خیلی بلند بودن یک ساز و خیلی آرام بودن ساز دیگر)آن را ضعیف ارزیابی کردند و به ناهمخوانی درجه صدا ها اشاره کردند.
کیفیت موسیقی تولید شده	40% (دو نفر) احساس کردند که صدا عمق و احساس کافی ندارد و آن را تخت توصیف کردند.
تطابق با ورودی‌ها	همه نظر دهندگان به عدم تطابق بین انتظارات و خروجی اشاره کردند.

1.00

نمایش پروژه



نتیجه‌گیری و پیشنهادها

در این پروژه، ما روشی برای تولید موسیقی لو-فای با استفاده از مدل‌های زبان کوچک ارائه کردیم. مدل ما توانست آهنگ‌های لو-فای منحصر به فرد و جذاب تولید کند اما در تبدیل توصیفات متنی به موسیقی با چالش‌هایی روبرو شد. دلایل اصلی این چالش‌ها، پیچیدگی ترجمه ویژگی‌های صوتی به متن و وابستگی بیش از حد به کیفیت داده‌های متنی است. با این حال، نتایج ما پتانسیل این روش را نشان داد. برای بهبود عملکرد مدل، پیشنهاد می‌شود که پردازش سازها به صورت جامع و استفاده از مدل‌های Mixture of experts [9] مورد بررسی قرار گیرد. این رویکردها می‌توانند به مدل کمک کنند تا روابط پیچیده بین سازها را درک کرده و موسیقی واقعی‌تر و منسجم‌تری تولید کند.

منابع

- [1] B. Peng et al., ‘RWKV: Reinventing RNNs for the Transformer Era’, 2023.
- [2] H. M. de Oliveira and R. C. de Oliveira, ‘Understanding MIDI: A Painless Tutorial on Midi Format’, arXiv preprint arXiv:1705. 05322, 2017.
- [3] J. Zhang, ‘Lofi: ML-supported lo-fi music generator’.
- [4] A. Vaswani et al., ‘Attention Is All You Need’, 2017.
- [5] S. Wu, X. Li, F. Yu, and M. Sun, ‘TunesFormer: Forming Irish Tunes with Control Codes by Bar Patching’, in Proceedings of the 2nd Workshop on Human-Centric Music Information Retrieval 2023 co-located with the 24th International Society for Music Information Retrieval Conference (ISMIR 2023), Milan, Italy, November 10, 2023, 2023, vol. 3528.
- [6] L. Callender, C. Hawthorne, and J. Engel, ‘Improving Perceptual Quality of Drum Transcription with the Expanded Groove MIDI Dataset’, arXiv [cs.SD]. 2020.
- [7] B. Peng et al., ‘Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence’, arXiv preprint arXiv:2404. 05892, 2024.

منابع

- [8] M. S. Cuthbert and C. Ariza, ‘Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data’, in ISMIR, 2010, pp. 637–642.
- [9] W. Cai, J. Jiang, F. Wang, J. Tang, S. Kim, and J. Huang, ‘A survey on mixture of experts’, arXiv preprint arXiv:2407. 06204, 2024.

Thank You

Hope you have good day