

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گروه مهندسی نرم افزار

پایان نامه کارشناسی رشته‌ی مهندسی کامپیوتر گرایش هوش مصنوعی و نرم افزار

آموزش یک مدل زبانی کوچک برای ساخت موسیقی lo-fi

استاد راهنما:

دکتر زهرا زجاجی

دانشجو:

سهیل سلیمی

شهریور ۱۴۰۳



دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گروه مهندسی نرم افزار

پایان نامه کارشناسی رشته‌ی مهندسی کامپیوتر گرایش هوش
مصنوعی و نرم افزار

آقای سهیل سلیمی

آموزش یک مدل زبانی کوچک برای ساخت موسیقی lo-fi

در تاریخ ۱۴۰۳/۰۶/۲۰ توسط هیأت داوران بررسی و با درجه‌ی به تصویب نهایی رسید.

۱- استاد راهنمای پایان نامه دکتر زهرا زجاجی با مرتبه‌ی علمی امضاء

۲- استاد داور داخل گروه دکتر با مرتبه‌ی علمی امضاء

مهر و امضاء مدیر گروه

تقدیم به:

من، مغز متفکر پشت ایده‌ی آموزش یک مدل زبان کوچک برای تولید موسیقی لو-فای، Chat GPT، دستیار هوش مصنوعی که همیشه آماده است تا متنی تولید کند که تقریباً منطقی باشد، و LLaMA3، که بیشتر اوقات واقعاً منطقی است. بدون کمک شما، این مقاله به یک حلقه بی‌پایان از سکوت تبدیل می‌شد و جهان از صداهای شیرین و دلنشین موسیقی لو-فای تولید شده توسط این مدل محروم می‌ماند. از کمک‌هایتان متشکرم و امیدوارم آماده باشید تا به برخی از بیت‌های تولید شده توسط هوش مصنوعی گوش دهید و لذت ببرید:)

چکیده

افزایش تولید محتوا منجر به تقاضای بی‌سابقه‌ای برای موسیقی با کیفیت بالا و بدون حق کپی‌رایت برای استفاده در محتوای چندرسانه‌ای شده است. موسیقی لو-فای، با ویژگی‌های آرامش‌بخش و تسکین‌دهنده‌اش، به یکی از عناصر اصلی در تولید ویدئو، پادکست و پخش زنده تبدیل شده است. با این حال، دستیابی به موسیقی لو-فای با کیفیت بالا و بدون حق کپی‌رایت می‌تواند فرآیندی چالش‌برانگیز و پرهزینه باشد. این پروژه روشی برای تولید موسیقی لو-فای با استفاده از یک مدل زبانی کوچک ارائه می‌دهد که نیاز به یک راه‌حل مقرون‌به‌صرفه و مقیاس‌پذیر برای تولیدکنندگان محتوا را برطرف می‌کند. ما از معماری RWKV استفاده می‌کنیم، مدلی پیشرفته که کارایی یک ترانسفورمر را با انعطاف‌پذیری یک شبکه عصبی بازگشتی ترکیب می‌کند. در این پروژه، ما بر جنبه‌های فنی تولید موسیقی لو-فای تمرکز می‌کنیم و چالش‌های آموزش یک مدل برای تولید موسیقی با کیفیت بالا و طول نامحدود را بررسی می‌کنیم. ما دو مدل جداگانه، یکی برای پیانو و دیگری برای سازهای درام، آموزش دادیم تا موسیقی لو-فای را به صورت درخواستی تولید کنند. رویکرد ما به تولیدکنندگان محتوا این امکان را می‌دهد که بر روی دیدگاه خلاقانه خود تمرکز کنند، به جای اینکه زمان و منابع خود را صرف جستجوی موسیقی مناسب کنند.

کلیدواژه‌ها: ۱- تولید موسیقی لو-فای ۲- موسیقی تولید شده توسط هوش مصنوعی ۳- هوش مصنوعی مولد

فهرست مطالب

صفحه	عنوان
۱	۱: مقدمه
۱	۱-۱ پیش‌گفتار
۲	۲-۱ ساختار گزارش
۳	۲: بررسی پیشینه و ادبیات
۳	۱-۲ ادبیات موضوع
۴	۱-۱-۲ معماری RWKV
۷	۲-۱-۲ فرمت فایل MIDI
۹	۲-۲ روشهای پیشین
۹	۱-۲-۲ استفاده از معماری VAE
۱۰	۲-۲-۲ استفاده از شبکه‌های حافظه طولانی کوتاه مدت
۱۱	۳-۲-۲ جمع بندی
۱۲	۳: آموزش مدل و معماری
۱۲	۱-۳ مقدمه
۱۲	۲-۳ معماری کلی پروژه
۱۴	۳-۳ مجموعه داده ها
۱۴	۱-۳-۳ مدل پیانو
۱۴	۲-۳-۳ مدل درام
۱۵	۴-۳ تبدیل MIDI به متن
۱۵	۱-۴-۳ روش های موجود برای توکن سازی فایل های MIDI
۱۶	۲-۴-۳ کوانتایز کردن سرعت
۱۹	۳-۴-۳ تبدیل فایل های MIDI به متن
۲۰	۴-۴-۳ توکن سازی
۲۲	۵-۳ آموزش مدل
۲۲	۱-۵-۳ پارامترهای آموزش مدل
۲۵	۲-۵-۳ نحوه آموزش مدل
۲۶	۶-۳ ارزیابی عملکرد مدل
۲۶	۱-۶-۳ ارزیابی مدل پیانو
۲۶	۲-۶-۳ ارزیابی مدل درام
۲۸	۳-۶-۳ ارزیابی عملکرد مدل با معیار های موسیقی
۳۲	۷-۳ استنتاج مدل ها
۳۵	۱-۷-۳ ساخت موسیقی نهایی

صفحه	عنوان
۳۸	۳-۷-۲ جمع بندی
۴۰	۴: ساخت خط لوله متن به لو-فای
۴۰	۴-۱ معماری خط لوله.....
۴۱	۴-۱-۱ کمبودهای استفاده مستقیم از مدل MusicGen برای تولید موسیقی.....
۴۲	۴-۲ ارزیابی خط لوله
۴۴	۵: نتیجه گیری و پیشنهادها
۴۴	۵-۱ نتیجه گیری.....
۴۵	۵-۲ پیشنهادها.....
۴۶	منابع و مآخذ
۴۸	پیوست‌ها
۴۸	پ-۱ دسترسی به کد ها
۴۸	پ-۲ پارامترهای آموزش مدل ها

فهرست تصاویر

صفحه	عنوان
۵	شکل ۱-۲: معماری RWKV برای مدل های زبانی
۶	شکل ۲-۲: عناصر موجود در یک بلوک RWKV
۸	شکل ۳-۲: ساختار فایل MIDI
۱۳	شکل ۱-۳: ساختار pipeline
۲۷	شکل ۲-۳: نمودار های پیشرفت یادگیری مدل پیانو
۲۷	شکل ۳-۳: نمودار های پیشرفت یادگیری مدل درام
۳۲	شکل ۴-۳: نحوه استنتاج مدل ها برای ساخت موسیقی

فهرست جداول

عنوان	صفحه
جدول ۱-۲: مقایسه پیچیدگی استنتاج RWKV با ترانسفورمرهای مختلف	۷
جدول ۱-۳: خلاصه‌ای از مجموعه داده های درام	۱۴
جدول ۲-۳: نگاشت های فرمت MIDICompact	۱۷
جدول ۳-۳: نتایج این معیار ها برای مدل پیانو	۳۱
جدول ۱-۴: نتایج نظرسنجی ارزیابی خط لوله	۴۳

فهرست الگوریتم‌ها

صفحه	عنوان
۱۹	۱-۳ کوانتایز کردن سرعت
۲۱	۲-۳ توکن کردن فایل های MIDI
۲۴	۳-۳ آموزش مدل
۲۹	۴-۳ هماهنگی ریتم
۳۰	۵-۳ شباهت ملودی
۳۰	۶-۳ ثبات صدا
۳۱	۷-۳ هماهنگی هارمونیک
۳۴	۸-۳ استنتاج مدل ها
۳۶	۹-۳ تبدیل توکن به MIDI
۳۷	۱۰-۳ متن به MIDI message
۳۸	۱۱-۳ نحوه ساخت موسیقی نهایی

فصل اول

مقدمه

۱-۱ پیش‌گفتار

موسیقی لو-فای که با صداهای آرام خود شناخته می‌شود، در سال‌های اخیر محبوبیت زیادی پیدا کرده است. این ژانر که اغلب با لیست‌های پخش مطالعه و آرامش مرتبط است، ترکیبی از ضرب‌آهنگ‌های ملایم، صداهای محیطی و کیفیت تولید خام و متمایز را به نمایش می‌گذارد. با افزایش تعداد استریمرها و اینفلوئنسرها که به دنبال موسیقی بدون حق کپی‌رایت برای محتوای خود هستند، نیاز به آهنگ‌های لو-فای رایگان بیشتر احساس می‌شود. این مدل می‌تواند برای پخش نامحدود موسیقی لو-فای استفاده شود. ظهور هوش مصنوعی^۱ و یادگیری ماشین^۲ راه‌های جدیدی برای خلق موسیقی باز کرده است و امکان توسعه مدل‌هایی را فراهم کرده که می‌توانند به طور خودکار موسیقی لو-فای تولید کنند.

در این پروژه، فرآیند آموزش یک مدل زبان کوچک را که به طور خاص برای تولید موسیقی لو-فای طراحی شده است، بررسی می‌کنیم. روش ما شامل آموزش دو مدل جداگانه برای سازهای، پیانو و درام، هر کدام با ۱۰۰ میلیون پارامتر است. این کار به ما امکان می‌دهد تا بر جزئیات دقیق هر ساز تمرکز

^۱ Artificial intelligence

^۲ Machine learning

کنیم و تولید موسیقی با بهتری داشته باشیم.

هدف اصلی این پروژه ساخت یک مدل زبان کوچک و کارآمد برای تولید موسیقی است که می‌تواند به ویژه برای موسیقی‌دانان یا محتوا سازان^۱ و علاقه‌مندان با منابع محدود مفید باشد. ما به معماری مدل RWKV می‌پردازیم و مزایای آن در این نوع داده و مناسب بودن آن برای تولید موسیقی را نشان می‌دهیم. علاوه بر این، فرآیند جمع‌آوری داده‌ها، از جمله انتخاب و پیش‌پردازش آهنگ‌های موسیقی لو-فای برای ایجاد یک مجموعه داده آموزشی را توضیح می‌دهیم.

در پایان، هدف ما ارائه یک راهنما در مورد نحوه آموزش یک مدل زبان برای تولید موسیقی لو-فای است و پتانسیل هوش مصنوعی در حوضه تولید موسیقی و تقویت خلاقیت در موسیقی را افزایش دهیم. همچنین، کارهایی که می‌توان برای بهتر شدن مدل که شامل اضافه کردن سازهای بیشتر و برای افزایش قابلیت‌های این مدل، بررسی می‌کنیم.

۲-۱ ساختار گزارش

در این پایان‌نامه ابتدا به معرفی معماری RWKV، یک نوع معماری یادگیری ماشین مبتنی بر ترانسفورمرها است، می‌پردازیم. سپس به روش ساخت مدل خود می‌پردازیم، که با توکنایزر شروع می‌شود، که مسئول تبدیل ورودی‌های MIDI به فرمتی مناسب برای پردازش توسط مدل است. سپس فرآیند آموزش را توضیح می‌دهیم. پس از آموزش مدل، توسعه خط لوله خود برای تولید موسیقی لو-فای از متن را توضیح می‌دهیم، که شامل استفاده از مدل آموزش‌دیده برای تولید ترکیبات موسیقی از ورودی‌های متنی است. در نهایت، خلاصه‌ای از روش و مشارکت‌های کار خود ارائه می‌دهیم و به پتانسیل مدل خود برای تولید موسیقی لو-فای با کیفیت بالا می‌کنیم.

¹content creators

فصل دوم

بررسی پیشینه و ادبیات

۱-۲ ادبیات موضوع

در سال‌های اخیر، حوزه معماری‌های شبکه عصبی شاهد پیشرفت‌های چشمگیری بوده است، به ویژه در زمینه پردازش داده‌های دنباله‌ای مثل موسیقی با متن. یکی از این معماری‌ها RWKV است که ترکیبی از مزایای شبکه‌های عصبی بازگشتی و ترانسفورمرها را به کار می‌گیرد. این فصل به بررسی پیشینه و ادبیات مرتبط با معماری RWKV می‌پردازد و ویژگی‌ها و مزایای منحصر به فرد آن را نسبت به مدل‌های قبلی نشان می‌دهد. علاوه بر این، فصل به فرمت فایل MIDI که برای آموزش مدل‌های زبانی در وظایف تولید موسیقی اهمیت دارد، پرداخته و آن را با فرمت‌های دیگر مانند WAV مقایسه می‌کند.

۱-۱-۲ معماری RWKV

RWKV^۱ [۱] [۲] یک معماری شبکه عصبی است که ترکیبی از مزایای شبکه‌های عصبی بازگشتی^۲ [۳] و ترانسفورمرها^۳ [۴] را به کار می‌گیرد. این معماری برای پردازش کارآمد دنباله‌های داده طراحی شده است، مانند شبکه‌های عصبی بازگشتی، اما همچنین از قابلیت‌های پردازش موازی ترانسفورمرها بهره می‌برد. این رویکرد ترکیبی به RWKV اجازه می‌دهد تا وابستگی‌های بلندمدت در داده‌ها را حفظ کند، که این مسئله برای شبکه‌های عصبی بازگشتی‌ها به دلیل مشکلاتی مانند مشکل ناپدید شدن گرادینان چالش‌برانگیز است. در عین حال، از مقیاس‌پذیری و عملکرد ترانسفورمرها، به ویژه در طول آموزش، بهره‌مند می‌شود. به طور کلی، RWKV می‌تواند به صورت موازی مانند ترانسفورمرها آموزش ببیند اما در زمان استنتاج^۴ به صورت دنباله‌ای عمل کند، که این ویژگی آن را هم کارآمد و هم قدرتمند برای وظایف مختلف پردازش زبان طبیعی می‌سازد.

۱-۱-۱-۲ نحوه عملکرد RWKV

همانطور که در شکل ۱-۲ نشان داده شده است، مدل با یک لایه embedding شروع می‌شود که، پس از آن، چندین residual blocks مشابه به صورت متوالی قرار گرفته‌اند. این بلوک‌ها در شکل ۲-۲ الف نشان داده شده‌اند. پس از آخرین بلوک که در شکل ۲-۲ ب نشان داده شده است، یک سر خروجی ساده شامل یک لایه نرمال‌سازی^۵ و یک پروژکشن خطی برای تولید لاجیت‌ها^۶ جهت پیش‌بینی توکن بعدی و محاسبه‌ی خطای آنتروپی متقاطع^۷ در طول آموزش استفاده می‌شود. در حالت استنتاج، RWKV به حالت شبیه به شبکه عصبی بازگشتی تغییر می‌کند و هر توکن را به صورت جداگانه پردازش می‌کند در حالی که یک حالت^۸ را حفظ می‌کند. این پردازش حالت‌دار برای تولید توالی‌های منسجم و مرتبط از نظر مفهومی بسیار مهم است، زیرا بدون نیاز به نگاه به جلو^۹ یا ماسک^{۱۰} کردن توکن‌های آینده، ترتیب توکن‌ها را پیگیری می‌کند.

^۱Receptance Weighted Key Value

^۲Recurrent Neural Networks

^۳Transformers

^۴Inference

^۵LayerNorm

^۶Logits

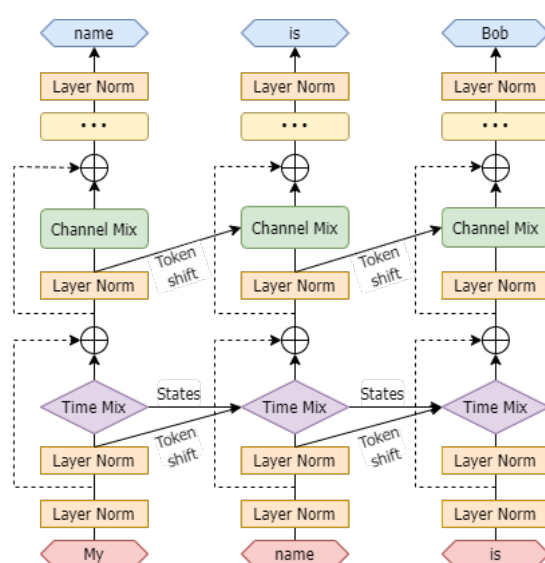
^۷Cross-entropy loss

^۸Stateful

^۹Look-ahead

^{۱۰}Masking

ماهیت حالت‌دار حالت استنتاج RWKV تضمین می‌کند که هر توکن تولید شده توسط توکن‌های قبلی اطلاع‌رسانی می‌شود و انسجام و ارتباط در توالی تولید شده حفظ می‌شود. با اجتناب از تکنیک‌های ماسکینگ RWKV فرآیند تولید را ساده می‌کند، پیچیدگی معماری را کاهش می‌دهد و کارایی را افزایش می‌دهد. این رویکرد منحصر به فرد به RWKV اجازه می‌دهد تا خروجی‌های متنی با کیفیت بالا تولید کند و آن را به یک پیشرفت قابل توجه در زمینه مدل‌های مولد تبدیل کند.

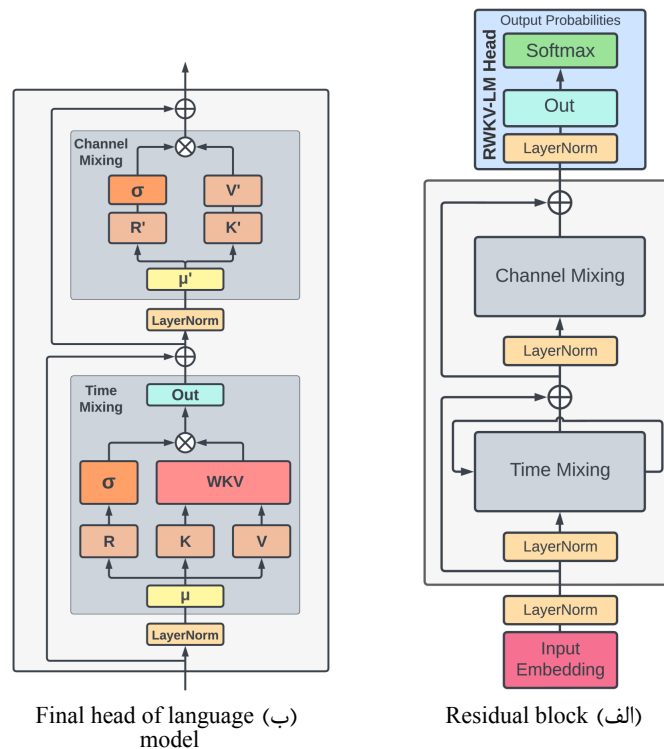


شکل ۱-۲ - معماری RWKV برای مدل‌های زبانی
گرفته شده از [۱]

۲-۱-۱-۲ RWKV در مقایسه با Transformer

در مورد آموزش یک مدل زبانی برای تولید موسیقی لو-فای، ما تصمیم گرفتیم از معماری RWKV به جای معماری ترانسفورمر استفاده کنیم. یکی از دلایل اصلی این انتخاب این است که RWKV برای کارایی بیشتر و اجرای آسان‌تر روی CPU طراحی شده است که برای پروژه ما مفید است.

با توجه به جدول ۱-۲ می‌تواند گفت RWKV می‌تواند در استنتاج بهتر از ترانسفورمرها باشد زیرا پیچیدگی محاسباتی کمتری در زمان و فضا دارد. به طور خاص، RWKV دارای پیچیدگی زمانی $O(Td)$ و پیچیدگی فضایی $O(d)$ است، در حالی که ترانسفورمرها دارای پیچیدگی زمانی $O(T^2d)$ و پیچیدگی فضایی $O(T^2 + Td)$ هستند. این بدان معناست که با افزایش طول دنباله ورودی T ، هزینه محاسباتی RWKV به صورت خطی با T افزایش می‌یابد، در حالی که هزینه محاسباتی ترانسفورمرها



شکل ۲-۲ - عناصر موجود در یک بلوک RWKV
گرفته شده از [۱]

به صورت درجه دوم با T افزایش می‌یابد. این امر RWKV را برای استنتاج و آموزش دنباله‌های طولانی کارآمدتر می‌کند.

علاوه بر این، پیچیدگی فضایی RWKV $O(d)$ است که کمتر از پیچیدگی فضایی ترانسفورمرها $O(T^2 + Td)$ است. این بدان معناست که RWKV به حافظه کمتری برای ذخیره وزن‌ها و فعال‌سازی‌های مدل نیاز دارد که می‌تواند برای آموزش و استنتاج در دستگاه‌هایی با حافظه محدود مفید باشد. به طور کلی، نیازهای محاسباتی و حافظه کمتر RWKV آن را به انتخاب کارآمدتری نسبت به ترانسفورمرها برای استنتاج، به ویژه برای دنباله‌های طولانی تبدیل می‌کند.

علاوه بر این، RWKV یک مکانیزم توجه^۱ را در خود جای داده است که به مدل اجازه می‌دهد هنگام تولید توکن بعدی، بر روی بخش‌های خاصی از دنباله ورودی تمرکز کند. این مکانیزم به RWKV امکان می‌دهد تا وابستگی‌ها و روابط بلندمدت بین توکن‌ها را به دست آورد، در حالی که همچنان کارایی یک مدل ترتیبی را حفظ می‌کند. مکانیزم توجه در RWKV به گونه‌ای طراحی شده است که از نظر

^۱ Attention mechanism

مدل	پیچیدگی زمانی	پیچیدگی فضایی
Transformer	$O(T^2d)$	$O(T^2 + Td)$
Linear Transformers [5]	$O(Td^2)$	$O(Td + d^2)$
RWKV	$O(Td)$	$O(d)$

جدول ۱-۲ - مقایسه پیچیدگی استنتاج RWKV با ترانسفورمرهای مختلف

در اینجا T طول دنباله، d بُعد ویژگی‌ها است.

محاسباتی کارآمد باشد و از ترکیبی از تبدیل‌های خطی و ضرب نقطه‌ای برای محاسبه وزن‌های توجه استفاده کند.

معماری RWKV به طور خاص برای پردازش داده‌های جریانی طراحی شده است. RWKV داده‌های ورودی را به صورت جریانی پردازش می‌کند، به طوری که هر توکن ورودی به محض ورود پردازش می‌شود، بدون نیاز به بارگذاری کل دنباله ورودی در حافظه. این به RWKV اجازه می‌دهد تا دنباله‌های ورودی بزرگ، مانند آهنگ‌ها یا فایل‌های صوتی طولانی، را بدون تمام شدن حافظه مدیریت کند. به طور کلی، معماری RWKV تعادلی بهتر بین عملکرد، کارایی و سهولت استقرار فراهم می‌کند و آن را به یک انتخاب ایده‌آل برای تولید موسیقی لو-فای تبدیل می‌کند.

۲-۱-۲ فرمت فایل MIDI

فرمت MIDI^۱ [۶] یک استاندارد فنی برای ارتباط بین ابزارهای موسیقی الکترونیکی، کامپیوترها و دیگر دستگاه‌های مرتبط با موسیقی است. برخلاف فایل‌های صوتی معمولی مانند MP3 یا WAV، فایل‌های MIDI حاوی داده‌های صوتی واقعی نیستند. آن‌ها شامل اطلاعاتی مانند نت‌های موسیقی، زمان‌بندی، مدت زمان و شدت صدا برای هر نت هستند.

این فرمت به موسیقی‌دانان و تولیدکنندگان موسیقی اجازه می‌دهد تا داده‌های موسیقی را به صورت دیجیتالی ضبط و پخش کنند و به راحتی بین نرم‌افزارها و سخت‌افزارهای مختلف به اشتراک بگذارند. به دلیل اندازه کوچک فایل‌های MIDI، انتقال و ذخیره‌سازی آن‌ها بسیار آسان است.

از دیدگاه کامپیوتری، فایل‌های MIDI به عنوان مجموعه‌ای از پیام‌های دیجیتالی ذخیره می‌شوند

¹Musical Instrument Digital Interface

که هر پیام شامل اطلاعاتی درباره نحوه پخش موسیقی است. این پیام‌ها به صورت باینری کدگذاری می‌شوند و شامل سه بخش اصلی هستند:

۱. **پیام‌های وضعیت**^۱: این پیام‌ها نوع عملیاتی که باید انجام شود را مشخص می‌کنند، مانند نواختن یک نت، تغییر شدت صدا، یا تغییر ابزار موسیقی.
۲. **پیام‌های داده**^۲: این پیام‌ها اطلاعات دقیق‌تری درباره عملیات مشخص شده در پیام‌های وضعیت ارائه می‌دهند، مانند شماره نت، شدت صدا، و مدت زمان.
۳. **زمان‌بندی**^۳: این بخش زمان دقیق اجرای هر پیام را مشخص می‌کند، که به دستگاه‌ها اجازه می‌دهد تا موسیقی را با دقت زمانی بالا پخش کنند.

مثال ۱-۲. پیام وضعیت برای نواختن نت: On Note

داده پیام می‌تواند شماره نت باشد مثل C4 یا شدت صدا ۶۴

زمان‌بندی: زمان شروع مثلاً ۵۰۰ میلی‌ثانیه پس از شروع

این پیام‌ها به ترتیب در یک فایل MIDI ذخیره می‌شوند و هنگام پخش، دستگاه‌های MIDI این پیام‌ها را تفسیر کرده و موسیقی را تولید می‌کنند. این ساختار به کامپیوترها و دستگاه‌های موسیقی اجازه می‌دهد تا به صورت هماهنگ و دقیق موسیقی را پخش کنند.

time message time message time message time message
time message time message time message time message
time message time message time message time message
time message time message time message time message
time message time message time message time message

شکل ۲-۳ - ساختار فایل MIDI

استفاده از فرمت MIDI برای آموزش مدل‌های زبانی نسبت به فرمت WAV بهتر است زیرا:

^۱Status Messages

^۲Data Messages

^۳Timing

۱. **اندازه فایل کوچکتر:** فایل‌های MIDI بسیار کوچکتر از فایل‌های WAV هستند. این امر باعث می‌شود که پردازش و انتقال داده‌ها سریع‌تر و کارآمدتر باشد.

۲. **داده‌های ساختاریافته:** فایل‌های MIDI شامل اطلاعات دقیق و ساختاریافته‌ای درباره نت‌های موسیقی، زمان‌بندی، و شدت صدا هستند. این داده‌ها به مدل‌های زبانی کمک می‌کنند تا الگوهای موسیقی را بهتر درک کنند و پیش‌بینی‌های دقیق‌تری انجام دهند.

۳. **انعطاف‌پذیری بیشتر:** با استفاده از MIDI، می‌توان به راحتی تغییرات مختلفی در موسیقی اعمال کرد، مانند تغییر تمپو، کلید، و ابزار موسیقی. این انعطاف‌پذیری به مدل‌های زبانی کمک می‌کند تا با شرایط مختلف سازگار شوند و عملکرد بهتری داشته باشند.

۴. **کاهش نویز:** فایل‌های WAV شامل داده‌های صوتی خام هستند که ممکن است نویز و اختلالات زیادی داشته باشند. در مقابل، فایل‌های MIDI تنها شامل داده‌های دیجیتالی هستند که نویز ندارند و این امر باعث می‌شود که مدل‌های زبانی با داده‌های تمیزتر و دقیق‌تری آموزش ببینند.

یک مزیت دیگر استفاده از فرمت MIDI برای آموزش مدل‌های زبانی این است که موسیقی چندلایه را به خوبی پشتیبانی می‌کند. فایل‌های MIDI می‌توانند چندین ترک^۱ را به صورت همزمان ذخیره کنند، که هر ترک می‌تواند نمایانگر یک ابزار موسیقی مختلف باشد. این ویژگی به مدل‌های زبانی اجازه می‌دهد تا تعاملات پیچیده بین سازهای مختلف را درک کنند و تحلیل کنند که چگونه این سازها با هم ترکیب می‌شوند تا یک قطعه موسیقی کامل را تشکیل دهند.

۲-۲ روش‌های پیشین ۱-۲-۲ استفاده از معماری VAE

پروژه jacobz/Lofi [۷] با استفاده از معماری VAE کار مشابهی را انجام می‌دهد. استفاده از معماری RWKV، معماری که ما در این پروژه استفاده کرده‌ایم، برای ساخت موزیک لو-فای^۲ مزایای متعددی نسبت به VAE^۳ دارد:

- **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-فای که اغلب دارای

^۱Track

^۲Lo-Fi

^۳Variational Autoencoder

الگوهای تکراری و ریتمیک است، بسیار مهم است.

- **کیفیت بازسازی بهتر:** RWKV به دلیل استفاده از مکانیزم توجه، می‌تواند جزئیات بیشتری از

داده‌های ورودی را حفظ کند و بازسازی دقیق‌تری ارائه دهد.

- **انعطاف‌پذیری بیشتر:** این معماری به دلیل استفاده از مکانیزم‌های توجه^۱، 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.

پروژه jacobz/Lofi شامل چندین محدودیت است. یکی از محدودیت‌های اصلی این پروژه این است

که به دلیل استفاده از فضای نهان^۲ با ابعاد کمتر، ممکن است در بازسازی آهنگ‌های طولانی‌تر دچار

مشکل شود. این امر می‌تواند منجر به از دست رفتن جزئیات مهم و کاهش کیفیت بازسازی شود.

همچنین، VAE به دلیل استفاده از توزیع‌های احتمالاتی برای بازسازی داده‌ها، ممکن است در بازسازی

جزئیات دقیق دچار مشکل شود و کیفیت نهایی موزیک کاهش یابد.

به طور کلی، معماری RWKV به دلیل توانایی بهتر در حفظ ساختار زمانی و جزئیات داده‌ها، برای

ساخت موزیک لو-فای مناسب‌تر است. از طرف دیگر، VAE به دلیل محدودیت‌های ذاتی خود در

بازسازی آهنگ‌های طولانی و پیچیده، ممکن است کیفیت نهایی موزیک را کاهش دهد.

۲-۲-۲ استفاده از شبکه‌های حافظه طولانی کوتاه مدت

پروژه MR-KARPIN/lofi-LSTM [۸] با استفاده از شبکه‌های حافظه طولانی کوتاه مدت^۳ کار

مشابهی را انجام می‌دهد. که استفاده از معماری استفاده از معماری RWKV برای ساخت موزیک لو-

فای مزایای متعددی نسبت به شبکه‌های حافظه طولانی کوتاه مدت دارد. RWKV به دلیل استفاده

از مکانیزم‌ها توجه^۴، قادر است ساختار زمانی و توالی‌های طولانی را بهتر حفظ کند. این ویژگی برای

موزیک لو-فای که اغلب دارای الگوهای تکراری و ریتمیک است، بسیار مهم است.

از سوی دیگر، یکی از محدودیت‌های اصلی LSTM این است که به دلیل استفاده از حافظه کوتاه

مدت، ممکن است در بازسازی آهنگ‌های طولانی‌تر دچار مشکل شود. این امر می‌تواند منجر به از دست

^۱Attention mechanism

^۲Latent space

^۳Long Short-Term Memory

^۴Attention mechanism

رفتن جزئیات مهم و کاهش کیفیت بازسازی شود.

به طور کلی، معماری RWKV به دلیل توانایی بهتر در حفظ ساختار زمانی و جزئیات داده‌ها، برای ساخت موزیک لو-فای مناسب‌تر است. از طرف دیگر، LSTM به دلیل محدودیت‌های ذاتی خود در بازسازی آهنگ‌های طولانی و پیچیده، ممکن است کیفیت نهایی موزیک را کاهش دهد.

۳-۲-۲ جمع بندی

معماری RWKV یک مدل شبکه عصبی است که مزایای شبکه‌های عصبی بازگشتی و ترانسفورمرها را برای پردازش کارآمد داده‌های دنباله‌ای ترکیب می‌کند. برخلاف شبکه‌های عصبی بازگشتی که به دلیل مشکلاتی مانند ناپدید شدن گرادیان با وابستگی‌های بلندمدت مشکل دارند، این RWKV وابستگی‌ها را حفظ کرده و از قابلیت‌های پردازش موازی ترانسفورمرها بهره می‌برد. این رویکرد ترکیبی به RWKV اجازه می‌دهد تا به صورت موازی مانند ترانسفورمرها آموزش ببیند اما در زمان استنتاج به صورت دنباله‌ای عمل کند، که این ویژگی آن را بهترین گزینه برای پروژه ما می‌کند.

فصل سوم

آموزش مدل و معماری

۱-۳ مقدمه

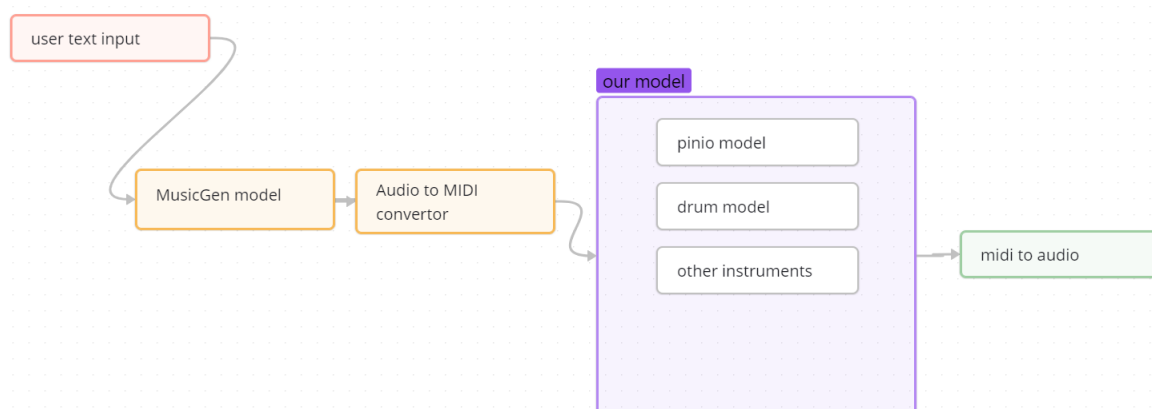
این فصل به بررسی دقیق روش‌ها و معماری‌های به‌کاررفته در پروژه ما که هدف آن آموزش یک مدل زبان کوچک است، می‌پردازد. تا درک جامعی از ساختار کلی پروژه، مجموعه داده‌های استفاده‌شده، فرآیند تبدیل فایل‌های MIDI به متن و آموزش و ارزیابی مدل ارائه دهد.

۲-۳ معماری کلی پروژه

در روش ما، هدف این است که مدل‌های جداگانه‌ای برای هر ساز که قصد استفاده در آهنگ خود داریم، آموزش دهیم. به عنوان مثال، ما مدل‌هایی برای پیانو و درام آموزش داده‌ایم. خروجی‌های این مدل‌ها سپس ترکیب می‌شوند تا ترکیب نهایی ایجاد شود.

ورودی مدل‌های ما می‌تواند یک فایل MIDI یا حتی فایل WAV باشد. اگر ورودی یک فایل WAV باشد، ابتدا با استفاده از یک مدل دیگر تبدیل به نت‌های MIDI تبدیل می‌شود. سپس این نت‌های MIDI برای پردازش به مدل‌های ما ارسال می‌شوند.

از آنجا که ما از مدل زبان RWKV استفاده می‌کنیم، نیاز به یک توکنایزر^۱ داریم تا فایل‌های MIDI را به فرمت متنی تبدیل کند که مدل بتواند آن را درک کند. توکنایزر فایل‌های MIDI را به قطعات کوچکتر و سپس به متن تبدیل می‌کند که باعث می‌شود این یکی از مهم‌ترین بخش‌های پروژه شود. این فرآیند به مدل امکان می‌دهد تا به طور مؤثر توالی‌های موسیقی را یاد بگیرد و تولید کند.



شکل ۳-۱ - ساختار pipeline

علاوه بر آموزش مدل‌ها، ما یک pipeline توسعه داده‌ایم که تجربه کاربری را بهبود می‌بخشد. این خط لوله همانطور که در ۳-۱ نشان داده شده است، ورودی متنی کاربر را دریافت کرده و آن را از طریق یک مدل تولید موسیقی (MusicGen) [۹] پردازش می‌کند. مدل MusicGen یک فایل MIDI بر اساس ورودی متنی کاربر ایجاد می‌کند. این فایل MIDI تولید شده سپس از طریق مدل‌های آموزش دیده ما برای هر ساز عبور می‌کند. در نهایت، خروجی این مدل‌ها ترکیب شده و به عنوان ترکیب نهایی موسیقی ذخیره می‌شود.

با آموزش مدل‌های جداگانه برای هر ساز و ترکیب خروجی‌های آن‌ها، می‌توانیم به یک ترکیب موسیقایی دقیق‌تر و پویاتر دست یابیم. این روش انعطاف‌پذیری و خلاقیت بیشتری در تولید موسیقی فراهم می‌کند، زیرا هر ساز می‌تواند به صورت جداگانه تنظیم شود و سپس در قطعه نهایی ادغام شود.

¹Tokenizer

۳-۳ مجموعه داده ها

۱-۳-۳ مدل پیانو

برای انجام آموزش مدل پیانو خود، ما از مجموعه داده گسترده‌ای به اسم IrishMMD [۱۰] استفاده کردیم. این مجموعه داده شامل ۲۱۶،۲۸۴ قطعه موسیقی به فرمت MIDI است. این مجموعه داده به دو بخش تقسیم شده است: ۹۹٪ (۲۱۴،۱۲۲ قطعه) برای آموزش مدل و ۱٪ (۲،۱۶۲ قطعه) برای ارزیابی آن.

قطعات موسیقی این مجموعه داده از وبسایت‌های thesession.org و abcnotation.com جمع‌آوری شده‌اند. برای اطمینان از یکپارچگی داده‌ها، ممکن است برخی از قطعات موسیقی که به صورت متن بودند به فرمت MIDI تبدیل شده باشند. همچنین، اطلاعات غیرموسیقی مانند عنوان و متن ترانه‌ها حذف شده است.

۲-۳-۳ مدل درام

مجموعه داده EGMD [۱۱]

برای پروژه خود، ما از نسخه گسترده‌تری از مجموعه داده EGMD استفاده کردیم که به عنوان **مجموعه داده گسترده EGMD** شناخته می‌شود. GMD یک مجموعه داده از اجراهای درام انسانی است که به صورت MIDI بر روی یک درام کیت الکترونیکی Roland TD-11 ضبط شده است.

بخش	تعداد توالی‌های منحصربه‌فرد	تعداد کل توالی‌ها	مدت زمان (ساعت)
آموزشی	۸۱۹	۳۵،۲۱۷	۴،۳۴۱
آزمایشی	۱۲۳	۵،۲۸۹	۹،۵۰
اعتبارسنجی	۱۱۷	۵،۰۳۱	۲،۵۲
کل	۱،۰۵۹	۴۵،۵۳۷	۵،۴۴۴

جدول ۱-۳ - خلاصه‌ای از مجموعه داده های درام

ما تقسیم‌بندی‌های آموزشی، آزمایشی و اعتبارسنجی را که در GMD وجود داشت، حفظ نکردیم و بخش های آموزشی و آزمایش را با هم یکی کردیم و برای تست فقط از داده های اعتبارسنجی استفاده کردیم و دلیل اینکار حجم کم داده ها برای آموزش بود. تعداد این داده‌ها در جدول ۱-۳ نشان داده

شده است.

۴-۳ تبدیل MIDI به متن ۱-۴-۳ روش های موجود برای توکن سازی فایل های MIDI

در نمایش موسیقی به متن، چندین فرمت مختلف وجود دارد که می توان برای نمایش اطلاعات موسیقی استفاده کرد. یکی از رایج ترین فرمت ها، نوت نویسی ABC است که یک فرمت قابل خواندن توسط انسان است و موسیقی را با استفاده از ترکیبی از حروف و نمادها برای نشان دادن ارتفاع صدا، مدت زمان و سایر عناصر موسیقی نمایش می دهد. نوت نویسی ABC به طور گسترده ای در نظریه موسیقی و آموزش موسیقی استفاده می شود.

با این حال، برای آموزش یک مدل زبانی برای تولید موسیقی لو-فای، باید تعادل بین پیچیدگی داده های ورودی و توانایی مدل در یادگیری از آن ها را در نظر بگیریم. در این مورد، ما تصمیم گرفتیم فایل های MIDI را به فرمت MIDICompact توکن سازی کنیم که یک نمایش ساده تر از اطلاعات موسیقی است.

۱-۱-۴-۳ فرمت MIDICompact

در فرمت MIDICompact، هر توکن یک رویداد موسیقی را نشان می دهد که ساختار آن به شرح زیر است:

در اینجا تجزیه و تحلیل اجزای فرمت آمده است: اگر یک نوت را به صورت

<Note>:<velocity> <Wait time>

در نظر بگیریم خواهیم داشت

- <Note>: این نشان دهنده نوتی است که نواخته می شود، جایی که <Note> یک مقدار هگزادسیمال بین ۰ و ۲۵ است. در این نمایش، هر نوت یک مقدار هگزادسیمال منحصر به فرد دارد که در جدول ۲-۳ الف نشان داده شده است.^۱

^۱ در اینجا برای سادگی از ۲۵ نوت استفاده شده ولی در کد های نوشته شده از ۱۲۸ نوت استفاده شده که کیفیت یادگیری را افزایش می دهد.

- $\langle velocity \rangle$: این یک جداکننده بین نوت و شدت صدا است. این نشان دهنده شدت صدای نوت است، جایی که $\langle velocity \rangle$ یک مقدار هگزادسیمال بین ۰ و ۱۵ است. شدت صدا به ۱۶ مقدار ممکن تقسیم می‌شود که در جدول ۳-۲ نشان داده شده است.

- $\langle Wait\ time \rangle$: این نشان دهنده یک توکن انتظار است که نشان می‌دهد چه مدت باید قبل از نواختن نوت بعدی صبر کرد. توکن انتظار به صورت یک مقدار بین ۱ ms تا 125 ms نمایش داده می‌شود.

این فرمت اجازه می‌دهد تا اطلاعات موسیقی به صورت فشرده و کارآمد نمایش داده شود، که آن را برای استفاده در پروژه ما به خوبی عمل کند. تعداد کل توکن‌هایی که می‌تواند ساخته شود به صورت زیر است:

$$(Note * velocity) + Waittime + pad + start + end = (128 * 16) + 125 + 3 = 2176$$

مثال ۳-۱. برای مثال $\langle start \rangle$ t1 3:5 t2 10:2 t5 1:14 t3 d:11 $\langle end \rangle$ می‌تواند

خروجی الگوریتم ۳-۲ باشد.

A (نوت ۳) با شدت صدای ۵ (نسبتاً سخت) و زمان انتظار ۱

C# (نوت ۱۰) با شدت صدای ۲ (نسبتاً نرم) و زمان انتظار ۲

B (نوت ۱) با شدت صدای ۱۴ (حداکثر) و زمان انتظار ۵

A# (نوت ۱۳) با شدت صدای ۱۱ (بسیار بسیار سخت) و زمان انتظار ۳

در حالی که فرمت ABC بیانگرتر است و می‌تواند طیف گسترده‌ای از ظرافت‌های موسیقی را نمایش دهد، همچنین پیچیدگی اضافی را معرفی می‌کند که ممکن است برای پروژه ما ضروری نیست. به ویژه، فرمت ABC نیاز دارد که مدل تعداد بیشتری از توکن‌ها و روابط بین آن‌ها را یاد بگیرد که می‌تواند منجر به کاهش قابلیت تعمیم‌دهی شود.

۳-۴-۲ کوانتایز کردن سرعت

در تولید موسیقی، مقادیر سرعت^۲ اغلب پیوسته هستند، اما در تولید صدا که در بخش ۳-۷-۱ توضیح داده شده است، تنها می‌تواند از مقادیر گسسته برای تولید موسیقی استفاده شود. با کوانتایز

^۱ Millisecond

^۲ velocity

سرعت	Hex
۰-۱۰٪ (بسیار نرم)	۰
۱۱-۲۰٪ (نرم)	۱
۲۱-۳۰٪ (نسبتاً نرم)	۲
۳۱-۴۰٪ (متوسط)	۳
۴۱-۵۰٪ (نسبتاً سخت)	۴
۵۱-۶۰٪ (سخت)	۵
۶۱-۷۰٪ (بسیار سخت)	۶
۷۱-۸۰٪ (بسیار بسیار سخت)	۷
۸۱-۹۰٪ (حداکثر)	۸
۹۱-۱۰۰٪ (حداکثر)	۹
۱۰۱-۱۱۰٪ (حداکثر)	۱۰
۱۱۱-۱۲۰٪ (حداکثر)	۱۱
۱۲۱-۱۳۰٪ (حداکثر)	۱۲
۱۳۱-۱۴۰٪ (حداکثر)	۱۳
۱۴۱-۱۵۰٪ (حداکثر)	۱۴
۱۵۱-۱۶۰٪ (حداکثر)	۱۵

(ب) نگاشت سرعت

نوت	Hex
A0	۰
A#	۱
B	۲
C	۳
C#	۴
D	۵
D#	۶
E	۷
F	۸
F#	۹
G	۱۰
G#	۱۱
A	۱۲
A#	۱۳
B	۱۴
C	۱۵
C#	۱۶
D	۱۷
D#	۱۸
E	۱۹
F	۲۰
F#	۲۱
G	۲۲
G#	۲۳
A	۲۴
B	۲۵

(الف)
نگاشت
نوت ها

جدول ۳-۲ - نگاشت های فرمت MIDICompact

کردن مقادیر سرعت به تعداد ثابتی از بین‌ها^۱، سیستم می‌تواند خروجی‌ای سازگارتر و قابل پیش‌بینی‌تر تولید کند. این روش به‌ویژه در موسیقی لو-فای مفید است، جایی که هدف ایجاد صدایی سازگار است. کد یک تابع کوانتایز کردن سرعت را پیاده‌سازی می‌کند که می‌تواند خطی یا نمایی باشد. این تابع یک مقدار سرعت پیوسته را به یک بین گسسته نگاشت می‌کند، که یک تکنیک رایج در تولید موسیقی، به‌ویژه در موسیقی لو-فای است.

۳-۴-۲-۱ کوانتایز کردن خطی

در کوانتایز کردن خطی، مقدار سرعت پیوسته به تعداد ثابتی از بین‌های گسسته تقسیم می‌شود. اندازه بین با تقسیم حداکثر مقدار سرعت بر تعداد بین‌ها تعیین می‌شود. شاخص بین با تقسیم مقدار سرعت بر اندازه بین و گرد کردن به نزدیک‌ترین عدد صحیح محاسبه می‌شود.

۳-۴-۲-۲ کوانتایز کردن نمایی

در کوانتایز کردن نمایی، مقدار سرعت پیوسته با استفاده از یک تابع نمایی به یک بین گسسته نگاشت می‌شود. تابع نمایی توسط پارامتر $velocity_exp$ کنترل می‌شود که شکل منحنی را تعیین می‌کند. مقدار بالاتر $velocity_exp$ منجر به منحنی نمایی‌تر می‌شود، در حالی که مقدار پایین‌تر منجر به منحنی خطی‌تر می‌شود.

فرمول کوانتایز کردن نمایی به صورت زیر است:

$$\left\lceil \frac{velocity_events \cdot \left(\frac{velocity_exp^{\left(\frac{velocity}{velocity_events} \right)} - 1}{velocity_exp - 1} \right)}{binsize} \right\rceil$$

که در آن $velocity_events$ حداکثر مقدار سرعت، $velocity$ مقدار سرعت ورودی و $velocity_exp$ یک مقدار ثابت برای نمایی کردن تابع است.

^۱bins

۳-۲-۴-۳ چرا کوانتایز کردن نمایی؟

کوانتایز کردن نمایی در تولید موسیقی بهتر است زیرا اجازه می‌دهد صدایی ظریف‌تر و دقیق‌تر ایجاد شود. در این پروژه ما مقدار پارامتر velocity_exp را برابر 0.33 قرار دادیم.

الگوریتم ۱-۳ کوانتایز کردن سرعت

```

procedure velocity_to_bin(velocity)
    bin_size ←  $\frac{\text{velocity\_events}}{\text{velocity\_bins}-1}$ 
    if velocity_exp == 1.0 then
        bin ←  $\lceil \frac{\text{velocity}}{\text{bin\_size}} \rceil$ 
    else
        bin ←  $\lceil \left( \text{velocity\_events} \cdot \left( \left( \text{velocity\_exp}^{\frac{\text{velocity}}{\text{velocity\_events}}} - 1 \right) \right) / (\text{velocity\_exp} - 1) \right) / \text{bin\_size} \rceil$ 
    return bin

procedure bin_to_velocity(bin)
    bin_size ←  $\frac{\text{velocity\_events}}{\text{velocity\_bins}-1}$ 
    if velocity_exp == 1.0 then
        velocity ← max(0,  $\lceil \text{bin} \cdot \text{bin\_size} - 1 \rceil$ )
    else
        velocity ←  $\max(0, \lceil \text{velocity\_events} \cdot \left( \left( \frac{(\text{velocity\_exp}-1) \cdot \text{bin} \cdot \text{bin\_size}}{\text{velocity\_events}} + 1 \right) - 1 \right) \rceil$ )
    return velocity

```

۳-۴-۳ تبدیل فایل‌های MIDI به متن

توضیحات الگوریتم ۲-۳ که به تبدیل داده‌ها به متن می‌کند به صورت زیر است:

- پیش‌پردازش

□ **فیلتر کردن:** حذف پیام‌های متناشناخته برای اطمینان از پردازش فقط داده‌های MIDI

مرتبط.

□ **ادغام ترک‌ها:** اگر فایل MIDI شامل چندین ترک باشد، آن‌ها را به یک ترک واحد ادغام

می‌کنیم تا پردازش ساده‌تر شود.

- مدیریت وضعیت

□ **وضعیت کانال‌ها:** نگهداری دیکشنری‌هایی برای پیگیری وضعیت هر کانال MIDI شامل

تغییرات برنامه، حجم، بیان، نوت‌های فعال و وضعیت پدال.

□ **زمان‌بندی:** پیگیری زمان سپری شده بین رویدادهای MIDI برای نمایش دقیق زمان‌بندی

در توالی توکن‌ها.

- بافر توکن

□ **بافرینگ:** استفاده از یک بافر برای ذخیره موقت داده‌های توکن قبل از تبدیل آن‌ها به

توکن‌های رشته‌ای. این کار به مدیریت زمان‌بندی و توالی توکن‌ها کمک می‌کند.

- پردازش رویدادها

□ **رویدادهای نوت:** پردازش رویدادهای `note_on` و `note_off` برای شروع و توقف نوت‌ها،

با در نظر گرفتن سرعت، حجم و بیان.

□ **تغییرات کنترل:** پردازش پیام‌های تغییر کنترل برای به‌روزرسانی وضعیت کانال‌ها، مانند

حجم، بیان و وضعیت پدال.

- تولید توکن

□ **تبدیل توکن:** تبدیل داده‌های نوت بافر شده به توکن‌های رشته‌ای با استفاده از فرمت‌های

از پیش تعریف شده. این شامل نگاشت رویدادهای MIDI به نمایش‌های خاص توکن است.

□ **توکن‌های زمان‌بندی:** تولید توکن‌هایی که زمان سپری شده بین رویدادها را نمایش

می‌دهند تا ساختار زمانی موسیقی حفظ شود.

- ساخت خروجی

افزودن توکن‌های شروع و پایان به هر قطعه و ترکیب لیست نهایی توالی‌های توکن.

۴-۴-۳ توکن‌سازی

در کار ما، از یک روش ساده برای آماده‌سازی و توکن کردن داده از کتابخانه `Tokenizer` [۱۲]

استفاده کردیم. در اینجا توضیحات از این کار آمده است:

۴-۴-۳-۱ توکن‌سازی سریع با کتابخانه `Tokenizer`

ما از کتابخانه `Tokenizer` برای انجام توکن‌سازی سریع داده‌ها استفاده کردیم. این کتابخانه برای

پردازش مجموعه داده‌های بزرگ و تبدیل متن خام به توکن‌ها با سرعت بالا طراحی شده است. مزایای

```

1: function convert_midi_to_str(cfg, filter_cfg, mid, augment=None)
2:   Initialize state variables
3:   function flush_token_data_buffer
4:     Convert token data buffer to token data
5:     Append formatted tokens to output
6:     Clear token data buffer
7:   function consume_note_program_data(prog, chan, note, vel)
8:     if token is valid then
9:       if delta_time_ms > threshold then
10:        Check if any notes are held
11:        if no notes are held then
12:          Call flush_token_data_buffer()
13:          Append "<end>" to output
14:          Reset output and state variables
15:        Generate wait tokens and append to output
16:        Reset delta_time_ms
17:        Append token data to buffer
18:        Set started_flag to True
19:   for each msg in mid.tracks[0] do
20:     Update delta_time_ms with msg.time
21:     function handle_note_off(ch, prog, n)
22:       if pedal is on then
23:         Set pedal event
24:       else
25:         Call consume_note_program_data(prog, ch, n, 0)
26:         Remove note from channel_notes
27:     if msg.type is "program_change" then
28:       Update channel_program
29:     else if msg.type is "note_on" then
30:       if velocity is 0 then
31:         Call handle_note_off
32:       else
33:         Remove pedal event if exists
34:         Call consume_note_program_data with mixed volume
35:         Add note to channel_notes
36:     else if msg.type is "note_off" then
37:       Call handle_note_off
38:     else if msg.type is "control_change" then
39:       Update channel state based on control type
40:     else
41:       pass
42:   Call flush_token_data_buffer()
43:   Append "<end>" to output
44:   return output_list

```

استفاده از این کتابخانه شامل موارد زیر است:

- سرعت کتابخانه Tokenizer برای عملکرد بهینه‌سازی شده است و به ما امکان می‌دهد حجم زیادی از داده‌ها را در زمان کوتاهی پردازش کنیم.
- انعطاف‌پذیری این کتابخانه از استراتژی‌های مختلف توکن‌سازی پشتیبانی می‌کند و به راحتی می‌توان آن را برای نیازهای خاص پروژه سفارشی کرد.

۲-۴-۴-۳ تبدیل به فرمت JSONL

پس از توکن‌سازی، داده‌های توکن‌شده را به فرمت JSON Lines (JSONL) تبدیل کردیم. این فرمت به خصوص برای پردازش مجموعه داده‌های بزرگ مناسب است زیرا پردازش داده‌ها را خط به خط بدون نیاز به بارگذاری کل مجموعه داده در حافظه آسان می‌کند و به راحتی خوانده و نوشته می‌شود و با بسیاری از ابزارهای پردازش داده‌ها به راحتی کار کنند.

۳-۴-۴-۳ تبدیل به فرمت binidx برای آموزش سریع

برای بهینه‌سازی بیشتر فرآیند آموزش، داده‌های JSONL را به فرمت binidx تبدیل کردیم. binidx یک فرمت باینری است که چندین مزیت برای آموزش مدل‌های یادگیری ماشین ارائه می‌دهد: فرمت‌های باینری به طور کلی فشرده‌تر و سریع‌تر برای خواندن/نوشتن نسبت به فرمت‌های متنی هستند و سربار I/O را در طول آموزش کاهش می‌دهند و فرمت binidx با بسیاری از چارچوب‌های یادگیری ماشین سازگار است. ما برای تبدیل فایل‌های JSONL به فرمت binidx از بخشی از کدهای کتابخانه gpt-neox [۱۳] استفاده کردیم.

با استفاده از کتابخانه Tokenizer برای توکن‌سازی سریع و تبدیل داده‌ها به فرمت JSONL و سپس به فرمت binidx، ما به طور قابل توجهی کارایی فرآیندهای آماده‌سازی داده و آموزش را بهبود دادیم. این رویکرد به ما امکان داد تا مجموعه داده‌های بزرگ را به طور مؤثر مدیریت کنیم و زمان کلی آموزش را تسریع کنیم که منجر به توسعه کارآمدتر مدل شد.

۵-۳ آموزش مدل

۱-۵-۳ پارامترهای آموزش مدل

در این بخش، پارامترهای مورد استفاده برای آموزش مدل توضیح داده شده‌اند:

ما از معماری RWKV-6.0 [۲] استفاده کردیم. مدل ما شامل ۲۰ لایه و Embedding برابر با ۵۱۲ است.^۱ Context Length^۲ مدل برابر با ۵۱۲ است. مقدار نرخ یادگیری اولیه و نهایی برابر با 6×10^{-4} و 6×10^{-5} است. همچنین، برای محاسبه‌ی خطای از تابع آنتروپی متقاطع^۳ در طول آموزش استفاده می‌شود.

در مدل RWKV، پارامتر head_size_a اندازه سر توجه در مکانیزم توجه چندسری^۴ را کنترل می‌کند. ما $head_size_a = 64$ را انتخاب کردیم زیرا این مقدار به مدل اجازه می‌دهد تا به تعداد بیشتری از عناصر ورودی به طور همزمان توجه کند، که برای پروژه ما که نیاز به یادآوری نوت‌های قبلی دارد میتواند به خوبی عمل کند و یک هارمونی بهتری داشته باشد.

در این پروژه ما از کتابخانه DeepSpeed نیز استفاده کردیم توضیحاتی درباره پارامترهای DeepSpeed آمده است:

پیکربندی بهینه‌ساز به گونه‌ای تنظیم شده است که از بهینه‌ساز Adam با نرخ یادگیری که به مقدار مشخصی مقداردهی اولیه شده است، استفاده کند. بهینه‌ساز Adam دارای مجموعه‌ای از ابرپارامترها، از جمله مقادیر بتا و یک مقدار کوچک برای پایداری عددی است که مشخص شده‌اند. پیکربندی زمان‌بند به گونه‌ای تنظیم شده است که از زمان‌بند نرخ یادگیری کاهش گرم‌شونده استفاده کند. این زمان‌بند دارای تعداد کل مراحل است که مدت زمان فرآیند آموزش را تعریف می‌کند. در طول دوره گرم‌شدن، نرخ یادگیری از یک مقدار حداقل شروع شده و طی تعداد مشخصی از مراحل به مقدار حداکثر افزایش می‌یابد.

پیکربندی دقت مختلط به گونه‌ای تنظیم شده است که دقت bfloat16 (bf16) یا float16 (fp16) را فعال کند، که می‌تواند با کاهش استفاده از حافظه و نیازهای محاسباتی مدل، سرعت آموزش را بهبود بخشد. مقدار دقیق این ابرپارامترها در پیوست پ-۲ آمده است.

```
1: Function save_pth(dd, ff)
2: torch.save(dd, ff)
3: Class ResetValDataloader(Callback)
4: Function on_validation_start(trainer, pl_module)
5: trainer.reset_val_dataloader(pl_module)
6: Class TrainCallback(Callback)
7: Function __init__(self, args)
8: self.args = args
9: Function on_train_batch_start(self, trainer, pl_module, batch, batch_idx)
10: Adjust learning rate based on global step and schedule
11: for param_group in trainer.optimizers[0].param_groups do
12:     param_group['lr'] = lr * param_group['my_lr_scale'] if args.layerwise_lr > 0 else
        lr
13: trainer.my_lr = lr
14: Function on_train_batch_end(self, trainer, pl_module, outputs, batch, batch_idx)
15: Log metrics and update loss
16: Function on_train_epoch_start(self, trainer, pl_module)
17: Update dataset attributes
18: Function on_train_epoch_end(self, trainer, pl_module)
19: if trainer.is_global_zero and (args.epoch_save > 0 and trainer.current_epoch %
    args.epoch_save == 0) or (trainer.current_epoch == args.epoch_count - 1) then
20:     save_pth(pl_module.state_dict(), f'args.proj_dir/self.prefix_args.epoch_begin +
    trainer.current_epoch.pth')
```

۲-۵-۳ نحوه آموزش مدل

الگوریتم ۳-۳ برای مدیریت و بهینه‌سازی فرآیند آموزش مدل با استفاده از PyTorch Lightning [۱۴] طراحی شده است.

ابتدا، تابع `save_pth` که برای ذخیره دیکشنری حالت مدل در یک مسیر فایل مشخص استفاده می‌شود. این قابلیت برای ایجاد نقاط بازرسی در طول آموزش ضروری است و به مدل اجازه می‌دهد تا ذخیره و بعداً بارگذاری شود. این امر تضمین می‌کند که فرآیند آموزش می‌تواند از آخرین حالت ذخیره شده در صورت وقفه‌ها از سر گرفته شود و پیشرفت‌های حاصل شده در طول آموزش حفظ شود. کلاس `TrainCallback` با آرگومان‌های مختلفی که فرآیند آموزش را کنترل می‌کنند، مقداردهی اولیه می‌شود. این شامل تنظیم برنامه‌های نرخ یادگیری، لاگ‌گیری و سایر پارامترهای خاص آموزش است. در متد `on_train_batch_start`، نرخ یادگیری به صورت پویا بر اساس مرحله فعلی آموزش تنظیم می‌شود. برنامه نرخ یادگیری به دو مرحله اصلی تقسیم می‌شود: مرحله گرم‌کردن^۱ و مرحله کاهش. در مرحله گرم‌کردن، نرخ یادگیری به تدریج از یک مقدار کوچک به نرخ یادگیری اولیه افزایش می‌یابد. این کار به فرآیند آموزش در مراحل اولیه کمک زیادی می‌کند. در مرحله کاهش، نرخ یادگیری بر اساس کاهش خطی یا نمایی تنظیم می‌شود. این تنظیم پویا نرخ یادگیری به بهینه‌سازی فرآیند آموزش و بهبود همگرایی کمک می‌کند.

لاگ‌گیری و نظارت گسترده بخش‌های جدایی‌ناپذیر این بخش هستند. متدهای `on_train_batch_start` و `on_train_batch_end` شامل مکانیزم‌های لاگ‌گیری دقیقی برای دیدن پیشرفت آموزش هستند. این شامل لاگ‌گیری نرخ یادگیری و `Loss Function`، پیگیری تعداد توکن‌های پردازش شده در هر ثانیه و لاگ‌گیری به سرویس‌های مانند `Weights & Biases (wandb)` [۱۵] برای پیگیری آزمایش‌ها است.

متدهای `on_train_epoch_start` و `on_train_epoch_end` وظایفی را که باید در ابتدای هر دوره آموزش و پایان آن انجام شوند، مدیریت می‌کنند. این شامل تنظیم پارامترهای مجموعه داده

^۱ارتفاع ۵۱۲ و عرض ۲۰

^۲Context Length بی‌نهایت فقط در هنگام اجرای مدل معنا می‌دهد.

^۳cross-entropy loss

^۴multi-head attention mechanism

^۱warm up

مانند دوره واقعی و ذخیره نقاط بازرسی مدل در فواصل مشخص یا در پایان آموزش است. ذخیره منظم نقاط بازرسی مدل تضمین می‌کند که فرآیند آموزش می‌تواند از آخرین حالت ذخیره شده از سر گرفته شود و یک محافظت در برابر وقفه‌ها فراهم می‌کند.

۳-۵-۱ کتابخانه‌ها استفاده شده

چندین کتابخانه در این روش برای ساده‌سازی فرآیند آموزش استفاده می‌شوند. کتابخانه torch [۱۶] قابلیت‌های اصلی برای ساخت و آموزش شبکه‌های عصبی را فراهم می‌کند. PyTorch Lightning [۱۴] فرآیند آموزش را با انتزاع کدهای تکراری ساده می‌کند، حلقه‌های آموزش، اعتبارسنجی و تست را از طریق کلاس Trainer مدیریت می‌کند و از کلاس‌های Callback برای افزودن رفتار سفارشی در مراحل مختلف آموزش استفاده می‌کند. کتابخانه wandb برای لاگ‌گیری و پیگیری آزمایش‌ها استفاده می‌شود.

۳-۶ ارزیابی عملکرد مدل

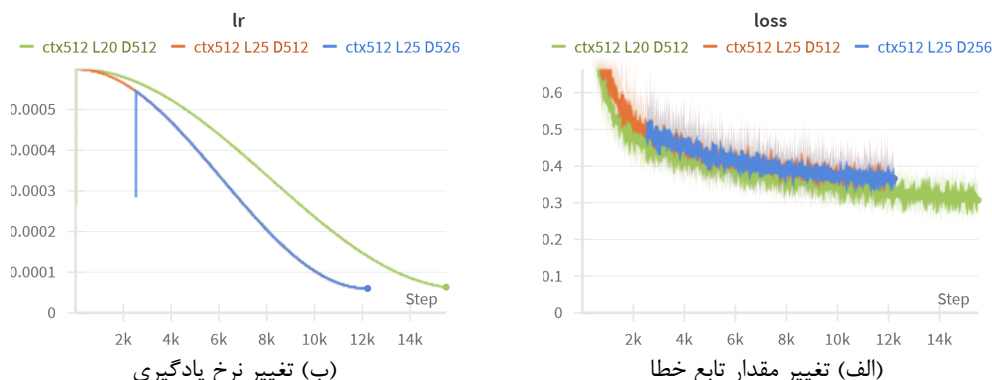
۳-۶-۱ ارزیابی مدل پیانو

با توجه به شکل ۳-۲ می‌توان گفت که مدل سبز^۱ به دلیل شروع با مقدار اولیه کمتری از خطا، عملکرد بهتری دارد. این موضوع می‌تواند به دلیل پیش‌آموزش مؤثرتر یا وزن‌های اولیه بهتر باشد. برخلاف مدل‌های قرمز و آبی که کاهش سریعی در خطا نشان می‌دهند و سپس به یک سطح ثابت می‌رسند، مدل سبز به تدریج و به طور پیوسته کاهش می‌یابد. این نشان‌دهنده یک فرآیند یادگیری پایدارتر است که خطر بیش‌برازش را کاهش می‌دهد و اطمینان می‌دهد که مدل به داده‌های جدید بهتر تعمیم می‌یابد. مدل‌های قرمز و آبی، در حالی که بهبودهای اولیه سریعی نشان می‌دهند، تمایل دارند در کمینه‌های محلی گیر کنند که عملکرد بلندمدت آن‌ها را محدود می‌کند.

۳-۶-۲ ارزیابی مدل درام

مطابق با نمودار ۳-۳ می‌توان گفت که مدل بنفش^۲ با مقدار خطای اولیه کمتری شروع می‌شود و به تدریج در طول زمان کاهش می‌یابد. این نشان‌دهنده یک فرآیند یادگیری پایدار است. کاهش تدریجی

^۱در اینجا مدل سبز به مدل ctx512 L20 D512 اشاره می‌کند. که مدل نهایی انتخاب شده برای ارزیابی است.
^۲در اینجا مدل بنفش به مدل ctx512 L20 D512 اشاره می‌کند. که مدل نهایی انتخاب شده برای ارزیابی است.

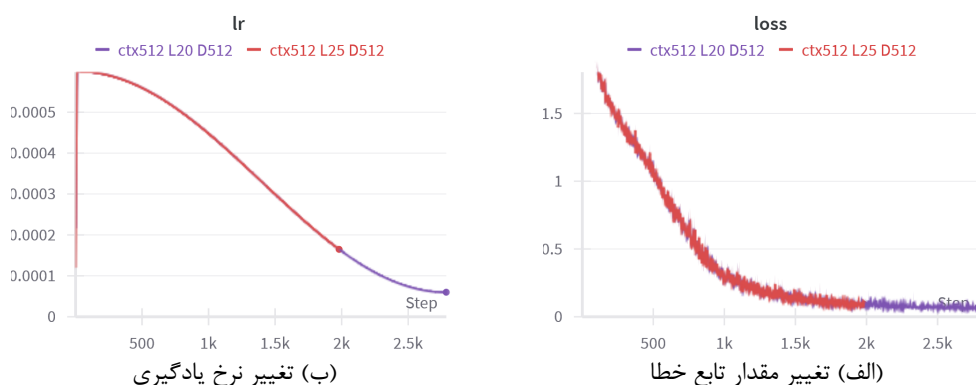


(ب) تغییر نرخ یادگیری

(الف) تغییر مقدار تابع خطا

شکل ۳-۲ - نمودارهای پیشرفت یادگیری مدل پیانو

نمودار سبز: context length ۵۱۲ با ۲۰ لایه و بُعد ۵۱۲، نمودار نارنجی: context length ۵۱۲ با ۲۵ لایه و بُعد ۵۱۲، نمودار آبی: context length ۵۱۲ با ۲۵ لایه و بُعد ۲۵۶



(ب) تغییر نرخ یادگیری

(الف) تغییر مقدار تابع خطا

شکل ۳-۳ - نمودارهای پیشرفت یادگیری مدل درام

نمودار قرمز: context length ۵۱۲ با ۲۰ لایه و بُعد ۵۱۲، نمودار بنفش: context length ۵۱۲ با ۲۵ لایه و بُعد ۵۱۲

مقدار خطا نشان می‌دهد که مدل در حال یادگیری و تطبیق خوب است که این یک نشانه مثبت است. با تنظیمات بیشتر و دوره‌های آموزشی اضافی، مدل بنفش پتانسیل دستیابی به عملکرد حتی بهتر را دارد. ولی به احتمال زیاد ادامه آموزش این مدل باعث overfit شدن خواهد شد زیرا حجم داده‌های دارم خیلی بالا نیست و ادامه بیش از این احتمالاً باعث overfit می‌شود.

پارامترهای اولیه آموزش و ابرپارامترهای مدل زبان کوچک تا حد زیادی با کمک‌های جامعه سازندگان RWKV تعیین شدند، به ویژه از طریق به اشتراک‌گذاری تجربیات و تخصص جمعی آن‌ها در سرور Discord RWKV. PENG Bo، که همچنین خالق معماری RWKV است، با سخاوت آزمایش‌های قبلی

و بینش‌های خود را به اشتراک گذاشت و به ما اجازه داد تا از دانش آن‌ها بهره‌مند شویم و از مشکلات رایج در آموزش مدل‌های زبان کوچک اجتناب کنیم. با راهنمایی آن‌ها، ما توانستیم مقادیر بهینه برای نرخ یادگیری، ابعاد مدل و سایر ابرپارامترها را سریعتر شناسایی کنیم که به طور قابل توجهی فرآیند آموزش را تسریع کرد و عملکرد کلی مدل را بهبود بخشید.

۳-۶-۳ ارزیابی عملکرد مدل با معیارهای موسیقی

برای ارزیابی عملکرد مدل زبان کوچک ما در تولید موسیقی لو-فی، از مجموعه‌ای از معیارهای ارزیابی استفاده می‌شود که کیفیت‌های مختلف موسیقی مانند ریتم، ملودی و توالی را ارزیابی می‌کنند. بخش‌های زیر معیارهای مورد استفاده برای ارزیابی عملکرد مدل را به تصویر می‌کشند. اکثر این معیارها در مقاله [۱۷] معرفی شده‌اند. و ما آن‌ها را با کتابخانه music21 [۱۸] پیاده‌سازی کردیم.^۱

۱-۳-۶-۳ هماهنگی ریتم (Rhythm Consistency)

هماهنگی ریتم یک متریک است که به بررسی نوسان یا یکنواختی طول نت‌ها در یک قطعه موسیقی می‌پردازد. این متریک نشان می‌دهد که قطعه‌های موسیقی چه میزان یکنواختی و یا چه میزان نوسانی دارند.

فرمول و محاسبه

$$RC = \frac{\mu}{\mu + SD}$$

فرمول هماهنگی ریتم به صورت زیر است:

در این فرمول:

• SD معیار انحراف معیار طول نت‌ها است

• μ میانگین طول نت‌ها است

الگوریتم ۳-۴ نشان دهنده نحوه انجام این کار برای یک فایل MIDI است.

این فرمول برای نرمال‌سازی معیار نقطه کمره طول نت‌ها توسط میانگین، امکان مقایسه و تفسیر

بهتر هماهنگی ریتم را فراهم می‌کند. نتیجه به صورت یک عدد بین ۰ و ۱ است:

• هماهنگی ریتم = ۱ نشان دهنده هماهنگی کامل (همه نت‌ها طول یکسانی دارند)

• هماهنگی ریتم = ۰ نشان دهنده عدم هماهنگی کامل (نت‌ها طول‌های بسیار متفاوت دارند)

^۱ کد پایتون این الگوریتم‌ها را می‌توانید در پیوست پ-۱ مشاهده کنید.

Require: MIDI file

Ensure: rhythm consistency measure

Parse the MIDI file using a converter to obtain a score

Extract notes from the score

Extract durations of notes

Calculate the mean duration of notes

Calculate the variance of note durations

Calculate the standard deviation of note durations as the square root of the variance

Return the standard deviation of note durations

• هماهنگی ریتم = ۵.۰ نشان دهنده هماهنگی متوسط (نت ها طول هایی یکنواخت اما با کمی نوسان دارند)

۲-۳-۶-۳ شباهت ملودی (Melodic Similarity Metric)

شباهت ملودی، یک روش برای اندازه گیری شباهت بین دو ملودی بر اساس ترتیب نت هایشان است. این متد ساده و مستقیماً از نسبت نت های مشابه دو ملودی برای محاسبه شباهت استفاده می کند. فرمول:

شباهت ملودی = (تعداد نت های مشابه) / (اندازه کوتاه ترین ملودی)

در این فرمول: تعداد نت های مشابه، تعداد نت هایی است که در همان موقعیت در دو ملودی وجود دارند. اندازه کوتاه ترین ملودی، طول کوتاه ترین ملودی بین دو ملودی است. الگوریتم ۵-۳ نشان دهنده نحوه انجام این کار برای یک فایل MIDI است.^۱

۳-۳-۶-۳ ثبات صدا (Tonal Stability Metric)

ثبات صدا، یک متد برای اندازه گیری ثبات صدا یک ملودی است. این متد از تعداد تغییرات کلید در یک ملودی برای محاسبه ثبات صدا استفاده می کند. فرمول:

ثبات صدا = ۱ - (تعداد تغییرات کلید)

به عبارت دیگر، هرچه تعداد تغییرات کلید در یک ملودی کمتر باشد، ثبات صدا آن بیشتر است.

الگوریتم ۶-۳ نشان دهنده نحوه انجام این کار برای یک فایل MIDI است.^۱

Require: MIDI file, Reference MIDI file

Ensure: melodic similarity measure

- Parse the input MIDI file using a converter to obtain a score
- Parse the reference MIDI file using a converter to obtain a reference score
- Extract generated melody from the score
- Extract reference melody from the reference score
- Extract pitch sequences from the generated and reference melodies
- Calculate the number of matching pitches between the generated and reference pitch sequences
- Calculate the similarity as the ratio of matching pitches to the length of the shorter pitch sequence
- Return the melodic similarity measure

Require: MIDI file

Ensure: tonal stability measure

- Parse the MIDI file using a converter to obtain a score
- Extract key changes from the score
- Count the number of key changes
- Return the number of key changes

۳-۶-۴ هماهنگی هارمونیک (Harmonic Coherence Metric)

هماهنگی هارمونیک یک متد برای اندازه گیری هماهنگی هارمونیک یک ملودی است. این متد از نسبت هارمونی سازگار (ملایمت) و غیرسازگار (ناهم خوانی) در یک ملودی برای محاسبه هماهنگی هارمونیک استفاده می کند.

فرمول:

هماهنگی هارمونیک = (نسبت ملایمت + نسبت ناهم خوانی)

نسبت ملایمت: نسبت تعداد هارمونی های سازگار به کل تعداد هارمونی ها نسبت ناهم خوانی: نسبت

تعداد هارمونی های غیرسازگار به کل تعداد هارمونی ها

به عبارت دیگر، هرچه نسبت ملایمت در یک ملودی بیشتر باشد، هماهنگی هارمونیک آن بیشتر

است. الگوریتم ۳-۷ نشان دهنده نحوه انجام این کار برای یک فایل MIDI است.^۱

برای کد های الگوریتم های گفته می توانید به پ-۱ مراجعه کنید.


```

1: procedure analyzeHarmonicCoherence(midi_file)
2:   Parse MIDI file:  $score \leftarrow converter.parse(midi\_file)$ 
3:   Analyze key:  $key\_analysis \leftarrow score.analyze('key')$ 
4:   Chordify score:  $chords \leftarrow score.chordify()$ 
5:   Get chord list:  $chord\_list \leftarrow [ch \text{ for } ch \text{ in } chords.recurse().Chord]$ 
6:   Count consonant chords:
7:      $consonance\_count \leftarrow \sum_{ch \in chord\_list} 1 \text{ if } ch.isConsonant() \text{ else } 0$ 
8:   Calculate ratios:  $dissonance\_count \leftarrow len(chord\_list) - consonance\_count$ 
9:    $total\_chords \leftarrow len(chord\_list)$ 
10:   $consonance\_ratio \leftarrow consonance\_count / total\_chords$ 
11:   $dissonance\_ratio \leftarrow dissonance\_count / total\_chords$ 
12:  return  $consonance\_ratio, dissonance\_ratio$ 

```

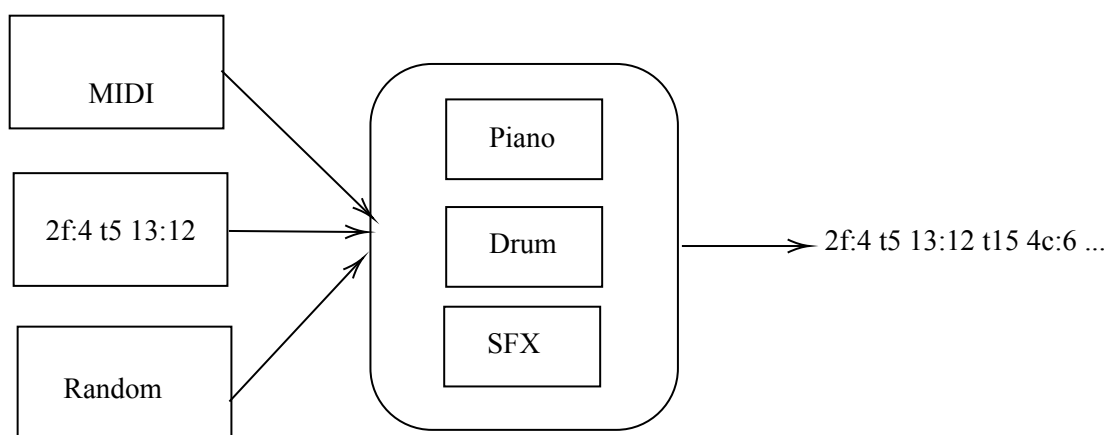
جدول ۳-۳ - نتایج این معیار ها برای مدل پیانو

مقدار	متد
۳۴.۰	هماهنگی ریتم
۸۷.۰	هماهنگی هارمونیک (ملایمت)
۱۳.۰	هماهنگی هارمونیک (ناهم خوانی)
۲.۰	تغییرات کلید

۵-۳-۶-۳ نتایج این معیار ها برای مدل پیانو

تحلیل جدول ۳-۳ به صورت زیر است:

- هماهنگی ریتم مقدار ۳۴.۰ نشان دهنده این است که ریتم ملودی دارای تناوب های نسبتاً مشابه است، اما همچنان دارای برخی از تغییرات و نوسانات است.
 - هماهنگی هارمونیک مقدار ۸۷.۰ نشان دهنده این است که هارمونی های ملودی در مجموع سازگار هستند و دارای هماهنگی بالایی هستند.
 - تغییرات کلید مقدار ۲.۰ نشان دهنده این است که ملودی تقریباً هیچ تغییراتی در کلید ندارد و در کلید ثابت باقی می ماند.
- ملودی های ساخته شده توسط مدل دارای هماهنگی بالایی در هارمونی و ریتم است، اما هنوز دارای بعضی از ناهمسانی ها و عدم هماهنگی است که می تواند بهبود یابد.



شکل ۳-۴ - نحوه استنتاج مدل ها برای ساخت موسیقی

۷-۳ استنتاج مدل ها

همانطور که در شکل ۳-۴ نمایش داده شده است، می تواند از سه نوع ورودی مختلف که موسیقی خود را تولید کند. برای ورودی MIDI، فایل ابتدا طبق مواردی که در بخش ۳-۴ گفته شد توکنایزر می شود و به مدل داده می شود. برای حالت رندم هم چند نوت تصادفی با فاصله زمانی های تصادفی می سازیم و آن را به مدل می دهیم. در نهایت مدل ما یک خروجی از نوت ها و فاصله زمانی آن ها می دهد که میتواند آن ها را پردازش کرد. این روند به طور خلاصه در الگوریتم ۳-۸ آمده است. لازم به ذکر است که نحوه استنتاج RWKV به طور دقیق تر در بخش ۲-۱-۱-۱ آمده است.

در اینجا توضیحی درباره هایپرپارامترهای استفاده شده در موقع استنتاج مدل ها آمده است. این هایپرپارامترها می توانند برای دستیابی به تعادل مطلوب بین انسجام و نوآوری در موسیقی تولید شده تنظیم شوند:

۱. Temperature

- **هدف:** کنترل تصادفی بودن موسیقی تولید شده.
- **محدوده:** معمولاً بین ۰ تا ۱.
- **تأثیر بر خروجی:** مقدار Temperature بالاتر به این معنی است که مدل احتمال بیشتری دارد خروجی های جدید و غیرمنتظره تولید کند، در حالی که مقدار Temperature پایین تر به این معنی است که مدل احتمال بیشتری دارد خروجی هایی مشابه داده های آموزشی تولید کند. این به این دلیل است که Temperature احتمال انتخاب یک توکن از توزیع

خروجی را کنترل می‌کند. وقتی Temperature بالا است، مدل احتمال بیشتری دارد توکن‌هایی از انتهای توزیع انتخاب کند که منجر به خروجی‌های تصادفی‌تر و جدیدتر می‌شود.

۲. Top_p

- **هدف:** کنترل احتمال انتخاب یک توکن از بین توکن‌های با بیشترین احتمال در توزیع خروجی.

- **محدوده:** معمولاً بین ۰ تا ۱.

- **تأثیر بر خروجی:** مقدار Top-p بالاتر به این معنی است که مدل احتمال بیشتری دارد توکن‌هایی از بخش با بیشترین احتمال توزیع انتخاب کند که منجر به خروجی‌های روان‌تر و منسجم‌تر می‌شود. مقدار Top-p پایین‌تر به این معنی است که مدل احتمال بیشتری دارد توکن‌هایی از بخش با احتمال کمتر توزیع انتخاب کند که منجر به خروجی‌های جدیدتر و غیرمنتظره‌تر می‌شود.

۳. Top_k

- **هدف:** کنترل تعداد توکن‌هایی که هنگام انتخاب توکن بعدی در دنباله خروجی در نظر گرفته می‌شوند.

- **محدوده:** معمولاً یک عدد صحیح کوچک (مثلاً ۸، ۱۰، ۲۰).

- **تأثیر بر خروجی:** مقدار Top-k بالاتر به این معنی است که مدل احتمال بیشتری دارد دامنه وسیع‌تری از توکن‌های ممکن را هنگام انتخاب توکن بعدی در نظر بگیرد که منجر به خروجی‌های متنوع‌تر و جالب‌تر می‌شود.

۴. Alpha_decay

- **هدف:** کنترل نرخ کاهش جریمه برای انتخاب توکنی که در بین توکن‌های با بیشترین احتمال Top-p نیست.

- **محدوده:** معمولاً بین ۰ تا ۱.

- **تأثیر بر خروجی:** مقدار Alpha_decay بالاتر به این معنی است که جریمه برای انتخاب توکنی که در بین توکن‌های با بیشترین احتمال Top-p نیست، به آرامی کاهش می‌یابد.

که منجر به خروجی‌های روان‌تر و منسجم‌تر می‌شود. مقدار Alpha_decay پایین‌تر به این معنی است که جریمه برای انتخاب توکنی که در بین توکن‌های با بیشترین احتمال Top-p نیست، سریع‌تر کاهش می‌یابد که منجر به خروجی‌های جدیدتر و غیرمنتظره‌تر می‌شود.

الگوریتم ۳-۸ استنتاج مدل‌ها

```

1: function generateTheSong(ctx, lenOfOut, modelPath, temperature)
2:   model  $\leftarrow$  RWKV(model=modelPath, strategy="cpu fp32")
3:   pipeline  $\leftarrow$  PIPELINE(model, os.environ["tokenizer"])
4:   data  $\leftarrow$  []
5:   args  $\leftarrow$  PIPELINE_ARGS( temperature, top_p=0.8, top_k=8, alpha_decay=0.997,
   token_stop=["<end>"], chunk_len=512)
6:   while len(data)  $\leq$  lenOfOut do
7:     pipeline.generate(ctx, args=args, callback=lambda token: data.append(token))
8:     ctx  $\leftarrow$  " ".join(map(str, data[:5]))
9:     print(len(data))
10:  return data
11: function mixTheSongs(song_len, ctx, out_len, te=1.0)
12:  pinio,      drum       $\leftarrow$  generateTheSong(ctx,      out_len,
   os.environ["PIANO_MODEL_PATH"], te), generateTheSong(ctx, out_len,
   os.environ["DRUM_MODEL_PATH"], te)

13: arg_parser  $\leftarrow$  argparse.ArgumentParser()
14: arg_parser.add_argument("-c", "--ctx", help="Enter the Context")
15: arg_parser.add_argument("-m", "--midi", help="Enter the Midi file path", type=str)
16: arg_parser.add_argument("-t", "--out_len", help="Enter the Length of unique output",
   default=100, type=int)
17: arg_parser.add_argument("-st", "--song_len", help="Enter the Length of song len", de-
   fault=60, type=int)
18: arg_parser.add_argument("-te", "--temperature", help="Enter the temperature of
   model", default=1.0, type=float)
19: args  $\leftarrow$  arg_parser.parse_args()
20: context  $\leftarrow$  ""
21: if args.midi is not None then
22:   with open(args.midi, 'rb') as f:
23:     context  $\leftarrow$  convert_midi_bytes_to_str(None, ("", f.read()))[1][0]
24: else if args.ctx is not None then
25:   context  $\leftarrow$  args.ctx
26: else
27:   context  $\leftarrow$  f'random.randint(50,90):x:random.randint(1,16):x      tran-
   dom.randint(20,80) " " * 2
28: mixTheSongs(args.song_len, context, args.out_len, args.temperature)

```

۳-۷-۱ ساخت موسیقی نهایی

در کد ما که در الگوریتم ۳-۱۱ نشان داده شده است، برای تولید موسیقی لو-فای، چندین روش به کار گرفته شده است تا یک قطعه موسیقی منحصر به فرد و هماهنگ ایجاد شود. این فرآیند با تولید یک تمپوی^۱ تصادفی آغاز می‌شود که به هر قطعه موسیقی تنوع و یگانگی می‌بخشد. هسته اصلی تولید موسیقی شامل تبدیل خروجی مدل به فایل‌های MIDI با استفاده از تابع `convert_str_to_midi` است که مربوط به بخش ۳-۴ است که در الگوریتم ۳-۹ و الگوریتم ۳-۱۰ نشان داده شده است. این فایل‌های MIDI سپس با استفاده از FluidSynth، یک کتابخانه تبدیل MIDI، که برای تولید صدا به فونت‌های صوتی متکی است، به فایل‌های صوتی استفاده می‌شوند. فونت‌های صوتی، مانند فونت مشخص شده در کد ما (`OmegaGMGS2.sf2`)، مجموعه‌ای از نمونه‌های صوتی هستند که صدای سازهای واقعی و با کیفیت بالا را فراهم می‌کنند.

استفاده از فونت‌های صوتی چندین مزیت دارد. آن‌ها صدای سازهای واقعی را فراهم می‌کنند که کیفیت موسیقی تولید شده را افزایش می‌دهد. همچنین امکان سفارشی‌سازی را فراهم می‌کنند و به شما اجازه می‌دهند فونت‌های صوتی را انتخاب کنید که با سبک و احساس مورد نظر موسیقی مطابقت داشته باشند. یکنواختی در کیفیت صدا نیز یکی دیگر از مزایا است که اطمینان می‌دهد موسیقی دارای صدای یکنواختی است.

پس از تبدیل فایل‌های MIDI به فایل‌های صوتی، ما از `pydub` برای بارگذاری و دستکاری این قطعات صوتی استفاده می‌کنیم. این شامل رول پیانو اصلی، لوپ‌های درام که توسط مدل‌های ما ساخته شده است و `sfx sound` است. یک `sfx sound` تصادفی انتخاب می‌شود تا به موسیقی بافت و تصادفی بودن اضافه کند. توالی درام به طور مشابه با ملودی اصلی پردازش می‌شود تا هماهنگی با موسیقی تولید شده را تضمین کند. قطعات صوتی مختلف سپس برای ایجاد لایه‌دار روی هم قرار می‌گیرند. حجم رول پیانو تنظیم می‌شود و لوپ‌های `sfx sound` و درام برای مطابقت با مدت زمان رول پیانو تکرار می‌شوند. قطعه موسیقی نهایی با تکرار قطعات مخلوط شده به طول مورد نظر گسترش می‌یابد و با یک افکت محو شدن در پایان، یک پایان نرم ایجاد می‌شود. همچنین، ما در پروژه خود فقط از ساز پیانو استفاده نکردیم. علاوه بر پیانو، یک موسیقی دیگر نیز تولید می‌کنیم که توسط یک ساز تصادفی اجرا می‌شود.

^۱ در MIDI، تمپو به سرعت اجرای یک قطعه موسیقی اشاره دارد که معمولاً بر حسب ضرب در دقیقه (BPM) اندازه‌گیری می‌شود و مدت زمان هر نت ساز را بر حسب میکروثانیه تعیین می‌کند.

```

1: Input: utils, token, state, channel, end_token_pause
2: Output: Iterator of (Optional MIDI Message, DecodeState)
3: if state is None then
4:     Initialize state
5: token ← token.strip()
6: if token is empty or starts with "<" then
7:     yield (None, state) return
8: if token is "<end>" then
9:     Update state with end_token_pause
10:    if utils.cfg.decode_end_held_note_delay ≠ 0.0 then
11:        for (channel, note), start_time in state.active_notes do
12:            Convert delta_accum to ticks
13:            Remove (channel, note) from active_notes
14:            yield (note_off message, state)
15:    yield (None, state) return
16: if token is a wait token then
17:     Update state with wait token delta
18:     if utils.cfg.decode_end_held_note_delay ≠ 0.0 then
19:         for (channel, note), start_time in state.active_notes do
20:             if note held too long then
21:                 Convert delta_accum to ticks
22:                 Remove (channel, note) from active_notes
23:                 yield (note_off message, state) return
24: else
25:     (bin, note, velocity) ← utils.note_token_to_data(token)
26:     Convert delta_accum to ticks
27:     if velocity > 0 then
28:         if utils.cfg.decode_fix_repeated_notes and (channel, note) in active_notes then
29:             Remove (channel, note) from active_notes
30:             yield (note_off message, state)
31:         Add (channel, note) to active_notes
32:         yield (note_on message, state)
33:     else
34:         Remove (channel, note) from active_notes
35:         yield (note_off message, state)
36: yield (None, state)

```

```
1: Input: utils, data, channel
2: Output: Iterator of MIDI Messages
3: state ← None
4: for token in data.split(" ") do
5:     for msg, new_state in token_to_midi_message(utils, token, state, channel) do
6:         state ← new_state
7:         if msg is not None then
8:             yield msg
```

این موسیقی تولید شده ممکن است همیشه کیفیت بالایی نداشته باشد زیرا مدل ما برای آن نوع ساز خاص آموزش ندیده است. با این حال، گاهی اوقات نتیجه‌های بسیار خوبی به دست می‌آید که حتی ما را شگفت‌زده می‌کند. این تنوع در استفاده از سازها به پروژه ما جذابیت و پویایی بیشتری می‌بخشد و به ما امکان می‌دهد تا با صداها و ترکیب‌های مختلف آزمایش کنیم.

برای دسترسی به کدها و اجرا آنها به پیوست پ-۱ مراجعه کنید.

```

1: procedure GenMusic(data, dataDrum)
2:   tempo ← 5 18) random.randint(14,                                ▷ Set tempo
3:   convert_str_to_midi(__join(data), tempo)                          ←
   save(relpath('./soundFont/mdiOut.mid'))                            ▷ Generate MIDI file
4:   fs ← FluidSynth(relpath('./soundFont/SGM-v2.01-NicePianosGuitarsBass-V1.2.sf2))
   ▷ Load sound font
5:   fs.midi_to_audio(relpath('./soundFont/mdiOut.mid'), relpath('./soundFont/output.flac'))
   ▷ Convert MIDI to audio
6:   pianoRoll ← AudioSegment.from_file(relpath('./soundFont/output.flac')) ▷ Load
   piano roll
7:   fillName ← random.choice(os.listdir(relpath('./loops/vinyl')))    ▷ Select fill name
8:   fill ← AudioSegment.from_file(relpath('./loops/vinyl/fillName'))    ▷ Load fill
9:   convert_str_to_midi(__join(data), 400, 9)                          ←
   save(relpath('./soundFont/mdiOutDrum.mid'))                        ▷ Generate drum MIDI file
10:  fs.midi_to_audio(relpath('./soundFont/mdiOutDrum.mid'), relpath('./soundFont/output.flac'))
   ▷ Convert drum MIDI to audio
11:  drum ← AudioSegment.from_file(relpath('./soundFont/output.flac')) ▷ Load drum
12: procedure mix_lines(music_len)
13:  pianoRoll ← pianoRoll + 10                                          ▷ Adjust piano roll
14:  fill ← fillmath.ceil(pianoRoll.duration_seconds/fill.duration_seconds) ▷ Repeat
   fill
15:  drum ← drummath.ceil(pianoRoll.duration_seconds/drum.duration_seconds) ▷
   Repeat drum
16:  music ← pianoRoll.overlay(fill - 10).overlay(drum + 3)             ▷ Mix audio
17:  music ← musicmath.ceil(music_len/music.duration_seconds)           ▷ Repeat music
18:  music ← music.fade_out(2SECOND)                                     ▷ Fade out music
19:  return music

```

۳-۷-۲ جمع بندی

در این فصل، ما به بررسی جامع و دقیق اجزای مختلف پروژه پرداختیم. ابتدا، معماری کلی پروژه را معرفی کردیم که به عنوان پایه و اساس تمامی مراحل بعدی عمل می‌کند. سپس، به مجموعه داده‌ها پرداختیم و مدل‌های پیانو و درام را بررسی کردیم.

در بخش تبدیل MIDI به متن، روش‌های مختلف توکن‌سازی فایل‌های MIDI و تبدیل آن‌ها به متن را مورد بحث قرار دادیم. همچنین، فرآیند توکن‌سازی سریع با استفاده از کتابخانه Tokenizer و تبدیل به فرمت‌های JSONL و binidx برای آموزش را توضیح دادیم.

در نهایت، آموزش مدل و پارامترهای مربوط به آن را بررسی کردیم و به ارزیابی عملکرد مدل پرداختیم. این ارزیابی‌ها نشان داد که مدل‌های ما توانسته‌اند با دقت و کارایی بالا به اهداف تعیین شده

دست یابند.

فصل چهارم

ساخت خط لوله متن به لو-فای

۱-۴ معماری خط لوله

این خط لوله^۱ که در شکل ۱-۳ نمایش داده شده است، شامل چند مرحله ای برای تولید موسیقی است که از مدل های مختلف و تکنیک ها برای تولید موسیقی با کیفیت بالا از متون بنام استفاده می کند. این روش را می توان به چهار مرحله اصلی تقسیم کرد:

- تولید موسیقی از متن
- تبدیل نت به MIDI
- پیش بینی نت ها
- پردازش پس از تولید موسیقی

در مرحله اول، از مدل Musicgen [۹]، برای تولید موسیقی از متن استفاده می کنیم، که یک مدل از پیش آموزش داده شده^۲ را برای تولید موسیقی است. مدل، یک دنباله از مقادیر صوتی را بر اساس متن ورودی تولید می کند و صوت تولید شده را به عنوان فایل WAV ذخیره می کند.

^۱pipeline

^۲pre-trained

در مرحله دوم، از یک مدل دیگری با نام basic pitch [۱۹] برای پیش بینی نت های صوتی تولید شده استفاده می کنیم. مدل پیش بینی نت های صوتی، فایل صوتی را تحلیل می کند و نت ها را استخراج می کند که سپس برای تولید فایل MIDI استفاده می شود.

در مرحله سوم، از الگوریتم ۲-۳ استفاده می کنیم تا فایل MIDI را به یک متن قابل فهم برای مدل خود تبدیل کنیم.

در مرحله چهارم، از الگوریتم ۱۱-۳ استفاده می کنیم تا آهنگ نهایی خود را بسازیم. فایل صوتی تولید شده سپس با استفاده از IPython display module پخش می شود و کاربر می تواند صوت تولید شده را بشنود.

برای دسترسی به کد ها و اجرا آنها به پیوست پ-۱ مراجعه کنید.

۱-۱-۴ کمبودهای استفاده مستقیم از مدل MusicGen برای تولید موسیقی

شاید این سوال ایجاد شود چرا باید مدل قوی مانند Musicgen با این مدل ها pipe شود تا بتواند این موسیقی ساده را تولید کند. مگر به تنها قادر به انجام اینکار نیست ؟ جواب این سوال این است که با اینکه مدل Musicgen، مانند بسیاری از مدل های تولید موسیقی دیگر، یک سیستم پیچیده است که می تواند موسیقی با کیفیت بالا تولید کند، اما همچنین دارای محدودیت هایی است. در اینجا چند دلیل وجود دارد که چرا استفاده مستقیم از آن برای تولید موسیقی ممکن است ایده آل نباشد:

- کیفیت پایین و نویز: همانطور که اشاره کردید، خروجی مدل Musicgen می تواند نویزی و با کیفیت پایین باشد. این به این دلیل است که مدل بر روی مقدار زیادی داده آموزش دیده است که شامل موسیقی نویزی و با کیفیت پایین است. مدل این الگوها را یاد می گیرد و می تواند منجر به خروجی نویزی و با کیفیت پایین شود.
- سختی در تغییر و ویرایش موسیقی تولید شده: مدل Musicgen موسیقی را به صورت فایل WAV تولید می کند که یک فرمت باینری است. این باعث می شود تغییر و ویرایش موسیقی تولید شده دشوار باشد. سازندگان آهنگ ممکن است بخواهند یک نت، آکورد یا ملودی خاص را تغییر دهند، اما انجام این کار با یک فایل WAV چالش برانگیز است.
- انعطاف پذیری محدود در سبک و ژانر موسیقی: مدل Musicgen بر روی یک مجموعه داده خاص آموزش دیده است، به این معنی که درک محدودی از سبک ها و ژانرهای مختلف موسیقی دارد.

سازندگان آهنگ ممکن است بخواهند موسیقی را در یک سبک یا ژانر خاص ایجاد کنند، اما مدل Musicgen ممکن است نتواند موسیقی‌ای تولید کند که انتظارات آنها را برآورده کند.

به همین دلیل است که روش که ما توصیف کردیم، که از مدل Musicgen برای تولید موسیقی استفاده می‌کنیم و سپس از یک مدل جداگانه برای پیش‌بینی گام صدای تولید شده و تبدیل آن به MIDI استفاده می‌کند و در نهایت ساخت موسیقی لو-فای با یک مدل دیگر می‌تواند ایده خوبی باشد. این رویکرد مزایای زیر را می‌دهد:

۱. بهبود کیفیت و کاهش نویز در موسیقی تولید شده

۲. کنترل بیشتر بر فرآیند تولید موسیقی

۳. تغییر و ویرایش آسان‌تر موسیقی تولید شده

۲-۴ ارزیابی خط لوله

متن کاربران اغلب شامل نکات احساسی و انتظاراتی است که به سختی می‌توان آن‌ها را به صورت ریاضی اندازه‌گیری کرد. وقتی کاربران درخواست‌های خود را برای تولید موسیقی ارائه می‌دهند حال و هوا، جو و تأثیر احساسی مورد نظر خود را با متن می‌خواهند منتقل کنند. معیارهای ریاضی می‌توانند جنبه‌هایی مانند دقت نت‌ها یا زمان‌بندی را اندازه‌گیری کنند، اما در ثبت تأثیر احساسی و ارزش زیبایی‌شناختی موسیقی نمی‌توانند خیلی خوب عمل کنند. پس برای اینکه بتوانیم این pipeline را ارزیابی کنیم یک نظر سنجی انجام دادیم.

ما یک نظرسنجی با پنج کاربر، که به طوری حرفه‌ای در زمینه ساخت این نوع موسیقی هستند یا شنونده این نوع موسیقی هستند، برای ارزیابی اثربخشی خط لوله تولید موسیقی خود انجام دادیم. این نظرسنجی شامل سوالاتی در مورد کیفیت صدا، رضایت کاربر از خروجی موسیقی بود. متأسفانه، نتایج امیدوارکننده نبود. در جدول ۴-۱ خلاصه‌ای از بازخوردهای دریافت شده آمده است:

همه نظر دهنده‌گان اشاره کردند که موسیقی تولید شده با ورودی‌های داده شده به خوبی تطابق ندارد، که منجر به عدم تطابق بین انتظارات و خروجی شد.

ایده تبدیل متن به آهنگ لو-فای (lo-fi) ممکن است به دلیل وجود افکت‌های مختلف و نودهای متفاوت و همچنین عدم توانایی گفتار در تشبیه دقیق آن‌ها، به خوبی عمل نکند. موسیقی لو-فای شامل

جدول ۴-۱ - نتایج نظرسنجی ارزیابی خط لوله

سوال	پاسخ
وضوح صدای تولید شده	۲۰٪ (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	۴۰٪ (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.

عناصر مختلفی مانند افکت‌های صوتی، تغییرات در ریتم و تمپو، و استفاده از صداهاى محیطی است که به سختی می‌توان آن‌ها را به صورت متنی توصیف کرد.

علاوه بر این، نودهای موسیقی لو-فای معمولاً دارای حس و حال خاصی هستند که به سختی می‌توان آن‌ها را با کلمات بیان کرد. این نودها ممکن است شامل حس آرامش، نوستالژی، یا حتی غم باشند که انتقال این احساسات از طریق متن به موسیقی نیازمند درک عمیق و دقیق از موسیقی و احساسات انسانی است.

بنابراین، به دلیل پیچیدگی‌های احساسی و توصیف درست احساسات موجود در موسیقی لو-فای، تبدیل متن به آهنگ لو-فای ممکن است نتواند به طور کامل و دقیق این عناصر را بازتاب دهد. و مدل بتواند موسیقی تولید کند که مطابق ورودی باشد که کاربر وارد کرده است. همچنین این نتیجه در پروژه jacz/Lofi [۷] نیز به دست آمده است.^۱

^۱نتیجه این کار در <https://github.com/jacz/Lofi/tree/main/model#lyrics2lofi> قابل مشاهده است.

فصل پنجم

نتیجه‌گیری و پیشنهادها

۱-۵ نتیجه‌گیری

در این پروژه، ما یک روش برای تولید موسیقی لو-فای با استفاده از یک مدل زبان کوچک ارائه کرده‌ایم. مدل ما که بر روی چند مجموعه داده مختلف و ملودی‌های موسیقی لو-فای آموزش دید، که نتایج امیدوارکننده‌ای در ایجاد آهنگ‌های منحصر به فرد و جذاب با ملودی‌های دلپذیر نشان داد. همچنین معیارهای ارزیابی، از جمله نمره نوآوری و نمره شباهت ملودی، نشان می‌دهند که مدل ما می‌تواند به طور مؤثر آهنگ‌های جدید لو-فای تولید کند که از آهنگ‌های موجود متمایز هستند و در عین حال ساختار موسیقایی منسجمی را حفظ می‌کنند.

با این حال، ما همچنین مشاهده کردیم که خط لوله ما با چالش‌های قابل توجهی در ساخت موسیقی لو-فای از طریق توصیفات متنی مواجه است. روش فعلی به شدت به کیفیت و انسجام داده‌های متنی متکی است که می‌تواند منجر به عدم تطابق بین موسیقی تولید شده و زیبایی‌شناسی مورد نظر شود. این موضوع دشواری‌های ترجمه ویژگی‌های پیچیده صوتی به نمایه متنی را برجسته می‌کند، که یک چالش رایج در تولید موسیقی با مدل‌های هوش مصنوعی است.

با وجود این چالش‌ها، نتایج ما پتانسیل استفاده از مدل‌های زبان کوچک برای تولید موسیقی لو-فای را نشان می‌دهد. کارهای آینده باید بر بهبود خط لوله متن به موسیقی تمرکز کنند، احتمالاً از طریق استفاده از تکنیک‌های پیشرفته‌تر پردازش زبان طبیعی یا ادغام ویژگی‌های صوتی در توصیف متنی بتوان نتیجه بهتری را کسب کرد.

۲-۵ پیشنهادها

یکی از کارهایی که می‌توان انجام داد، پردازش جامع سازها است. روش فعلی ما هر ساز را به صورت جداگانه پردازش می‌کند که می‌تواند منجر به یک ترکیب غیرطبیعی شود. در آینده، می‌توانیم همه سازها را به صورت یکجا پردازش کنیم. این رویکرد به مدل اجازه می‌دهد تا روابط بین سازها را یاد بگیرد و موسیقی واقعی‌تر و منسجم‌تری تولید کند. این هدف می‌تواند با استفاده از تکنیک‌های پردازش چند ساز یا بهره‌گیری از یک معماری پیشرفته‌تر که تعاملات بین سازها را به تصویر می‌کشد، محقق شود.

یک رویکرد جایگزین برای آموزش یک مدل زبان جداگانه برای هر ساز، آموزش یک ترکیبی از کارشناسان^۱ [۲۰] است. این نوع مدل می‌تواند برای درک روابط بین سازهای مختلف آموزش داده شود و موسیقی‌ای تولید کند که تعاملات بین آن‌ها را در نظر بگیرد.

با استفاده از مدل MoE، می‌توانیم از نقاط قوت چندین مدل بهره‌برداری کنیم و موسیقی‌ای تولید کنیم که هماهنگ‌تر و منسجم‌تر باشد. این رویکرد پتانسیل بهبود کیفیت کلی موسیقی تولید شده را دارد و درک دقیق‌تری از روابط بین سازهای مختلف ارائه می‌دهد.

¹Mixture of Experts

- [1] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, S. Biderman, H. Cao, X. Cheng, M. Chung, M. Grella, K. K. GV, X. He, H. Hou, J. Lin, P. Kazienko, J. Koccon, J. Kong, B. Koptyra, H. Lau, K. S. I. Mantri, F. Mom, A. Saito, G. Song, X. Tang, B. Wang, J. S. Wind, S. Wozniak, R. Zhang, Z. Zhang, Q. Zhao, P. Zhou, Q. Zhou, J. Zhu, and R.-J. Zhu, "Rwkv: Reinventing rnns for the transformer era," 2023.
- [2] B. Peng, D. Goldstein, Q. Anthony, A. Albalak, E. Alcaide, S. Biderman, E. Cheah, T. Ferdinan, H. Hou, P. Kazienko, *et al.*, "Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence," *arXiv preprint arXiv:2404.05892*, 2024.
- [3] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," 2018.
- [4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.
- [5] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," in *International conference on machine learning*, pp.5156–5165, PMLR, 2020.
- [6] H. M. de Oliveira and R. de Oliveira, "Understanding midi: A painless tutorial on midi format," *arXiv preprint arXiv:1705.05322*, 2017.
- [7] J. Zhang, "Lofi: Ml-supported lo-fi music generator,"
- [8] MR-KARPIN, "Github - mr-karpin/lofi-lstm: Bachelor's final project," 2023.
- [9] J. Copet, F. Kreuk, I. Gat, T. Remez, D. Kant, G. Synnaeve, Y. Adi, and A. Défossez, "Simple and controllable music generation," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.
- [10] 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* (L. Porcaro, R. Batlle-Roca, and E. Gómez, eds.), vol.3528 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023.
- [11] L. Callender, C. Hawthorne, and J. Engel, "Improving perceptual quality of drum transcription with the expanded groove midi dataset," 2020.
- [12] A. Moi and N. Patry, "HuggingFace's Tokenizers," Apr. 2023.
- [13] A. Andonian, Q. Anthony, S. Biderman, S. Black, P. Gali, L. Gao, E. Hallahan, J. Levy-Kramer, C. Leahy, L. Nestler, K. Parker, M. Pieler, J. Phang, S. Purohit, H. Schoelkopf, D. Stander, T. Songz, C. Tigges, B. Thérien, P. Wang, and S. Weinbach, "GPT-NeoX: Large Scale Autoregressive Language Modeling in PyTorch," 9 2023.

- [14] W. Falcon and The PyTorch Lightning team, “PyTorch Lightning,” Mar. 2019.
- [15] L. Biewald, “Experiment tracking with weights and biases,” 2020. Software available from wandb.com.
- [16] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” in *NIPS-W*, 2017.
- [17] Z. Xiong, W. Wang, J. Yu, Y. Lin, and Z. Wang, “A comprehensive survey for evaluation methodologies of ai-generated music,” *arXiv preprint arXiv:2308.13736*, 2023.
- [18] M. S. Cuthbert and C. Ariza, “Music21: A toolkit for computer-aided musicology and symbolic music data.,” in *ISMIR* (J. S. Downie and R. C. Veltkamp, eds.), pp.637–642, International Society for Music Information Retrieval, 2010.
- [19] R. M. Bittner, J. J. Bosch, D. Rubinstein, G. Meseguer-Brocal, and S. Ewert, “A lightweight instrument-agnostic model for polyphonic note transcription and multi-pitch estimation,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, (Singapore), 2022.
- [20] 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.

پیوست‌ها

پ-۱ دسترسی به کد ها

می‌توانید کد استفاده شده در این پروژه را در آدرس زیر دسترسی پیدا کنید:

<https://github.com/soheilsalimidev/lo-fAi>

برای اجرای مدل می‌توانید از نوت بوک زیر استفاده کنید.

<https://colab.research.google.com/drive/1mc6A0XWgNbJECqkoU0uCluzjcSxVfyuY?usp=sharing>

برای اجرای مدل با pipeline می‌توانید از نوت بوک زیر استفاده کنید.

<https://github.com/soheilsalimidev/lo-fAi/blob/main/packages/pipe/pipe.ipynb>

پ-۲ پارامترهای آموزش مدل ها

```
cd ./RWKV-LM;
MODEL_TYPE="x060" # x060 => rwkv0e.6-
N_LAYER="20"
N_EMBD="512"
CTX_LEN="512"
PROJ_DIR="./"
M_BSZ="24"
LR_INIT="6e"4-
LR_FINAL="6e"5-
GRAD_CP=
EPOCH_SAVE=10
N_NODE=1
GPU_PER_NODE=1
DS_BUCKET_MB=2
VOCAB_SIZE=2176
```

```
python train.py --wandb "li-fAi" --proj_dir $PROJ_DIR --type $MODEL_TYPE \
--ctx_len $CTX_LEN --epoch_count 999999 --epoch_begin 0 \
--data_file "text_document" --my_exit_tokens 34234278 --magic_prime 66851 \
--num_nodes $N_NODE --micro_bsz $M_BSZ --n_layer $N_LAYER \
--n_embd $N_EMBD --pre_ffn 0 --head_qk 0 \
--lr_init $LR_INIT --lr_final $LR_FINAL --warmup_steps 10 \
--beta1 9.0 --beta2 99.0 --adam_eps 1e8- --data_type "binidx" \
--vocab_size $VOCAB_SIZE \
--weight_decay 001.0 --epoch_save $EPOCH_SAVE --head_size_a 64 \
--accelerator gpu --devices $GPU_PER_NODE --precision bf16 \
--strategy deepspeed_stage_2 --grad_cp $GRAD_CP \
--enable_progress_bar True
```


Abstract

The rise of content creation has led to an unprecedented demand for high-quality, copyright-free music for use in multimedia content. Lo-fi music, with its calming and soothing properties, has become a staple in video, podcast, and live stream production. However, obtaining high-quality, copyright-free lo-fi music can be a challenging and costly process. This project presents a novel approach to generating lo-fi music using a small language model, addressing the need for a cost-effective and scalable solution for content creators. We utilize the RWKV architecture, a state-of-the-art model that combines the efficiency of a transformer with the flexibility of a recurrent neural network. In this project, we focus on the technical aspects of generating lo-fi music, exploring the challenges of training a model to produce high-quality music of unlimited length. We trained two separate models, one for piano and one for drum instruments, to generate lo-fi music on demand. Our approach enables content creators to focus on their creative vision, rather than spending time and resources searching for suitable music.

Keywords: 1- generate lo-fi music 2- music generating ai 3- generate ai



University of Isfahan
Faculty of Computer Engineering

BS Thesis

Train a small language model for generating lo-fi music
Supervisor:

Dr. Zahra Zojaji

By:
Soheil Salimi

August 2024