Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Informatyki

Praca dyplomowa

Generacja muzyki przy pomocy dużych modeli językowych

Music generation with Lange Language Models

Autor: **Filip Ręka**
Kierunek: **Informatyka — Data Science**
Opiekun pracy: **dr hab. Maciej Smołka prof. AGH**

Kraków, 2024

Streszczenie

Praca poświęcona jest eksploracji możliwości generacji muzyki przy pomocy dużych modeli językowych (LLM). Celem badania jest analiza architektur modeli językowych i ich zastosowanie w procesie tworzenia muzyki. Praca skupia się na podobieństwach strukturalnych między muzyką a tekstem, argumentując, że te dwie formy ekspresji dzielą wspólne cechy sekwencyjne, co pozwala na adaptację modeli językowych do generacji muzycznych kompozycji. W pracy przedstawiono również przegląd istniejących formatów cyfrowej reprezentacji muzyki oraz zbiory danych, które mogą być użyte do treningu modeli generatywnych. Przez przeprowadzenie serii eksperymentów, w których wykorzystano różnorodne architektury, takie jak transformery i modele przestrzeni stanów, przeanalizowano efektywność różnych podejść w kontekście generacji muzyki. Została podjęta próba oceny wygenerowanych fragmentów muzycznych przy pomocy dostępnych modeli LLM, które są dostępne komercyjnie lub zostały wytrenowane, aby rozumiec muzykę. Praca oferuje nowe perspektywy na temat potencjału LLM w dziedzinie sztucznej kreatywności muzycznej, otwierając drogę do dalszych badań w tej fascynującej przestrzeni interdyscyplinarnej.

Abstract

The work is devoted to exploring the possibilities of music generation using large language models (LLMs). The aim of the study is to analyze the architectures of language models and their application in the process of music creation. The paper focuses on the structural similarities between music and text, arguing that these two forms of expression share common sequential features, which allows the adaptation of language models for the generation of musical compositions. The paper also provides an overview of existing formats for digital representation of music and datasets that can be used to train generative models. By conducting a series of experiments using a variety of architectures, such as transformers and state space models, the effectiveness of different approaches in the context of music generation was analyzed. An attempt was made to evaluate the generated musical fragments using commercially available LLM models or those trained to understand music. The work offers new perspectives on the potential of LLM in the field of artificial musical creativity, paving the way for further research in this fascinating interdisciplinary space.

Spis treści

5

Spis rysunków

Spis tabel

Lista kodów źródłowych

1. Wstęp

Muzyka od zawsze stanowiła istotny element ludzkiego życia, inspirując, emocjonując i łącząc ludzi na różnych poziomach. Tradycyjnie proces tworzenia muzyki był zarezerwowany dla utalentowanych muzyków i kompozytorów, którzy posiadali wyjątkowy dar jej tworzenia. Jednakże, w erze cyfrowej i rozwoju sztucznej inteligencji, pojawiają się nowe możliwości w dziedzinie generacji muzyki. Modelowanie generatywne, będące obszarem sztucznej inteligencji, pozwala na tworzenie nowych danych, w tym również muzyki, na podstawie wzorców i reguł wykrytych w zbiorze treningowym. Dynamiczny rozwój dziedziny przetwarzania języka naturalnego (*NLP*) ukazuje skuteczność tego podejścia w problemach generacji danych sekwencyjnych, do których należy właśnie tekst oraz muzyka.

1.1. Motywacja

Przez ostatnie lata widzimy szybki rozwój dużych modeli językowych (*LLM*), które w szczególnie dobry sposób radzą sobie z rozumieniem tekstu w wielu językach. Modele te uczone są na wielkich korpusach danych, przez co są w stanie nauczyć się słownictwa oraz zasad gramatyki, a nastepnie ma podstawie *promptu* udzielonego przez użytkownika wygenerować odpowiednią odpowiedź. Modele te znalazły zastosowanie w maszynowej translacji tekstu, analizie sentymentu, odpowiadaniu na pytania użytkownika czy generacji kodu na podstawie poleceń oraz kontekstu nawet dużych projektów. Muzyka swoją strukturą jest bardzo podobna do tekstu. Każda nuta, tak jak słowo, nie tylko jest zależna od tych, które występują przed nią, ale również od szerszego kontekstu utworu jak również i tonacji w której dany utwór został napisany. Wykonując to porównanie, można łatwo zauważyć, dlaczego modele analizujące tekst oraz języki pisany są potencjalnie dobrymi kandydatami do próby analizy i generacji muzyki. Obecnie w celach generacyjnych stosuje się modele o bardzo dużej ilości parametrach, przez co nie są one często dostępne dla przeciętnego użytkownika który nie posiada dedykowanego sprzętu. Dodatkowym utrudnieniem jest typowa architektura oparta o transformer, który jest bardzo dobrym wyborem w tego rodzaju zadaniach, jednak generacja danych przy jego pomocy jest wolna. Obecnie pojawiają się nowe modele do zadań *NLP*, które jeszcze nie zostały dobrze zbadane w zadaniach muzycznych a rozwiązują problemy modeli transformerowych.

1.2. Cel pracy

Celem pracy jest zbadanie architektur modeli dużych modeli językowych i zastosowania samych modeli w problemie generacji muzyki. Zostanie podjęta analiza możliwych do użycia zbiorów danych i sposobów w jaki muzyka jest w nich zapisana oraz jakie są wady i zalety każdego z nich. W tym celu zostanie wytrenowane kilka modeli o rożnych architekturach i liczbie parametrów na różnego typu danych. Praca poruszy problem weryfikacji wygenerowanych melodii i przedstawi jedno z potencjalnych rozwiązań.

1.3. Zarys pracy dyplomowej

Praca dyplomowa składa się z czterech głównych rozdziałów. W rozdziale pierwszym przedstawiono wstęp do tematu, motywację oraz cel pracy. Rozdział drugi zawiera szczegółowy opis aktualnego stanu wiedzy na temat generacji muzyki przy pomocy dużych modeli językowych. Omówiono w nim podobieństwa pomiędzy muzyką a tekstem, przegląd istniejących prób generacji muzyki, cyfrowe reprezentacje muzyki oraz różne zbiory danych używane w eksperymentach. Szczególną uwagę poświęcono architekturze transformera oraz modelom przestrzeni stanów, które stanowią podstawę dla omawianych modeli generatywnych. W rozdziale trzecim opisano metodologię przeprowadzenia eksperymentów, w tym przygotowanie danych, proces treningu modeli oraz zaprezentowano otrzymane wyniki. Przeanalizowano także różne sposoby oceny wygenerowanych melodii, w tym testowanie ich innymi dużymi modelami językowymi. Ostatni, czwarty rozdział zawiera podsumowanie wyników pracy, wnioski oraz sugestie dotyczące dalszych badań w dziedzinie generacji muzyki przy użyciu sztucznej inteligencji.

2. Opis aktualnego stanu wiedzy

2.1. Podobieństwa pomiędzy muzyką a tekstem

Tekst i muzyka wykazują fundamentalne podobieństwo strukturalne, polegając na wzorcach i sekwencjach w celu przekazywania i wywoływania emocji. Oba wykorzystują hierarchiczną organizację: zdania tworzą akapity, podczas gdy nuty i akordy tworzą melodie i harmonie. Podobnie jak tekst jest zbudowany zgodnie z prawami składni i gramatyki, muzyka przestrzega zasad rytmu i harmonii. Duże modele językowe *(LLM)* wykazały w ostatnich latach niezwykłe możliwości w zadaniach przetwarzania języka naturalnego *NLP* i całkowicie zrewolucjonizowały to pole [1, 2]. Ich sukces doprowadził do dużego napływu badań w tym kierunku. *LLM-y*, będące biegłe w rozpoznawaniu i generowaniu sekwencji tekstowych, można dostosować do generowania muzyki poprzez uczenie ich w analogiczny sposób struktur w kompozycji muzycznej. Trenując na obszernych zbiorach danych partytur muzycznych, LLM mogą przewidywać i tworzyć spójne sekwencje nut, akordów i rytmów, tworząc w ten sposób oryginalne utwory muzyczne, które zachowują integralność stylistyczną i strukturalną.

Przyczynowe modelowanie języka *Causal Language Modeling (CLM)*, znane również jako auto-regresyjne modelowanie języka, jest techniką wykorzystywaną w przetwarzaniu języka naturalnego do przewidywania następnego słowa w sekwencji na podstawie poprzednich słów. Model generuje tekst krok po kroku, gdzie przewidywanie bieżącego słowa jest uwarunkowane wcześniej wygenerowanymi słowami. Oznacza to, że model wykorzystuje własne wcześniejsze generacje jako dane wejściowe do przewidywania następnego słowa. Na każdym etapie model generuje rozkład prawdopodobieństwa nad każdym unikalnym tokenem, wskazując prawdopodobieństwo, mówiące o tym, jak prawdopodobne jest że słowo będzie następne sekwencji. Typowo w tego rodzaju modelach wystepuje pojęcia okna kontekstowego, czyli ilość wstecznych słów, które model bierze pod uwagę w celu generacji kolejnego.

Uwaga 2.1. Tutaj można podać jakiś prosty przykład w zdaniu Ala ma kota → Ala ma kota i → Ala ma kora i psa → Ala ma kota i psa imieniem ...

Poniższe równanie trochę próbuje to pokazać nie jestem pewien czy zostanie

$$
\begin{aligned}
&\text{sekwencja wejściowa: } x_1, x_2, x_3, \text{gdzie } x_i \in X \\
&P_{t1}(X) = \text{Model}(x_1, x_2, x_3) \\
&x_4 \sim P_{t_1}(X) \\
&P_{t2}(X) = \text{Model}(x_1, x_2, x_3, x_4) \\
&x_5 \sim P_{t_2}(X) \\
&\dots
\end{aligned}
\tag{2.1}
$$

Przykładowe podejście do auto-regresyjnej generacji sekwencji pokazano w równaniu 2.1. Algorytm jest agnostyczny w stosunku do typu danych, więc w zbiorze X mogą być zarówno wszystkie słowa, litery lub elementy notacji muzycznej.

2.2. Obecne próby generacji muzyki

Pierwsze próby generacji muzyki przy pomocy algorytmów można znaleźć już w roku 1994 [3]. *GenJam* to model algorytmu genetycznego, który symuluje naukę improwizacji przez początkującego muzyka jazzowego. Na użycie sieci neuronowych oraz uczenia maszynowego jakie znamy dzisiaj należało poczekać do roku 2014. Organizacją, która podjęła się tego tematu jest jeszcze obecnie działająca Magenta [4]. Projekt ten skupia się na badaniu i rozwijaniu algorytmów, które mogą wspierać artystów w tworzeniu nowych dzieł poprzez generowanie muzyki, za pomocą technik uczenia maszynowego. Magenta stworzyła różne modele generatywne, które potrafią komponować nowe utwory muzyczne. Jednym z modeli jest *MusicVAE*, który bazując na sieciach konwolucyjncyh był w stanie generować sekwencje muzyczne, oraz interpolować pomiędzy nimi [5]. Innym modelem stworzonym przez tą organizację jest *GanSynth* [6], który był w stanie generowac naturalnie brzmiąca muzykę przy pomocy modelu *GAN (generative adversarial network)*. Przełomowym momentem w rozwoju modeli generacyjnych było wprowadzenie modeli transformera w roku 2017 [7]. Doprowadziło to do istnego renesansu w kontekście generacyjnego tworzenia modeli. Rok później pojawił się model *Music Transformer*, który implementował architekturę *decoder-only* i był trenowany na zbiorze danych MAESTRO [8]. Był w stanie generować długie sekwencje muzyczne o długotrwałej strukturze. Mimo wszytko melodia generowana przez narzędzie była dość monotonna, jeśli istnieje niedostateczne dopasowanie do zbioru treningowego lub niewystarczające dane szkoleniowe. Może to ograniczyć przydatność narzędzia do złożonych zadań generowania muzyki i może wymagać dodatkowej optymalizacji lub szkolenia z większymi i bardziej zróżnicowanymi zestawami danych w celu poprawy jakości generowanej muzyki [9]. Bardzo podobnym modelem jest *Museformer* [10], który implementując inny rodzaj algorytmu *attention* w dekoderze rozwiązuje problemy swojego poprzednika z rozmiarem sekwencji. Idea polega na tym, że nie trzeba skupiać się na całej sekwencji muzycznej z tym samym poziomem ważności. W artykule zostały wyróżnione dwa rodzaje algorytmu uwagi, czyli *fine-grained attention*

(drobnoziarnista) oraz *coarse-grained attention* (gruboziarnista). Wynika to z obserwacji że dla generacji muzyki ważniejsze są bliskie sobie tokeny, niż te które znajdują się w większej odległości. Tym sposobem w zależności od tego na podstawie jakich tokenów dokonujemy obliczeń to używa się innego algorytmu.

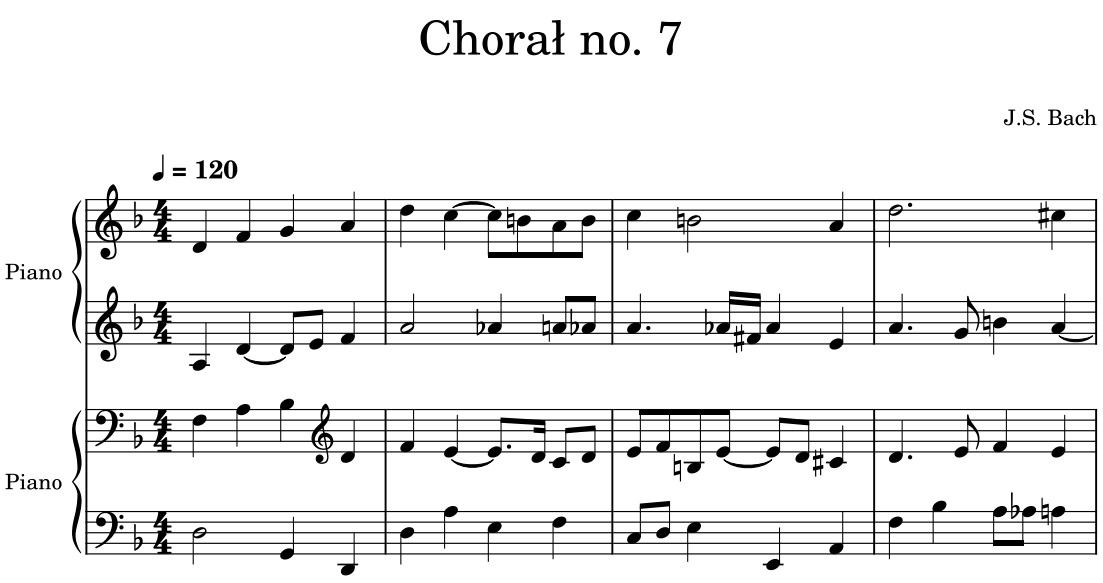
Obecnie najczęściej wykorzystywanymi modelami do generacji muzyki są modele *text-to-music*. Modele te działają na zasadzie udzielenia zapytania do modelu, który na jego podstawie generuje odpowiedni fragment. Typowe zapytanie może brzmieć na przykład "*Slow tempo, bass-and-drums-led reggae song. Sustained electric guitar. High-pitched bongos with ringing tones. Vocals are relaxed with a laid-back feel, very expressive*". Przykładem takiego modelu jest *MusicLM* stworzony przez Microsoft [11]. Zyskał on uwagę dzięki jego zdolność do tworzenia wysokiej jakości muzyki, która może trwać nawet kilka minut i generuje wysoką jakość dźwięku. Niestety model nie został udostępniony jako *open source*, natomiast zbiór danych zawierający ponad pięć tysięcy przykładów uczących wraz z ich szczegółowym opisem można pobrać ze strony *Kaggle*. Istnieją modele stworzone przez społeczność, które powstały w oparciu o zaproponowaną przez Microsoft architekturę, i są dostępne do własnego użytku. Jedną z głównych zalet *MusicLM* jest to, że może generować złożoną muzykę, która strukturą i spójnością jest podobna do muzyki tworzonej przez ludzkich kompozytorów. Oznacza to, że model ma potencjał być wykorzystany w różnych zastosowaniach, od generowania tła muzycznego do filmów i gier po wspomaganie kompozytorów w procesie twórczym.

2.3. Cyfrowa reprezentacja muzyki

W kulturze zachodniej nuty zapisuje się na pięcioliniach ułożonych jedna pod drugą. Jeśli muzyka jest wielogłosowa to pierwsze takty z każdej pięciolinii są łączone przy pomocy linii lub okalane nawiasem klamrowym. Na pięcioliniach umieszczone są takie elementy jak nuty, pauzy oznaczenia dynamiki, tempa i inne znaki dotyczące zapisu w jaki sposób należy wykonać dany utwór. Przykładowy zapis muzyki zapisano na obrazku 2.1. Przedstawiony fragment składa się z czterech głosów grających na raz oraz z czterech taktów. Na początku muzykę zapisywano na papierze, jednak po pojawieniu się komputerów, zapis nutowy jak i samo nagranie odegrania utworu mogło zostać zapisane na dysku. Nuty typowo są zapisywane jako obrazy lub pliki PDF, jednak dane zapisane w taki sposób nie nadają się do edycji lub przetwarzania przez algorytmy komputerowe. Jednym z pierwszych ustandaryzowanych formatów był *MusicXML*, który przy pomocy zwykłych tagów XML opisywał notację muzyczną.

2.3.1. WAV i MP3

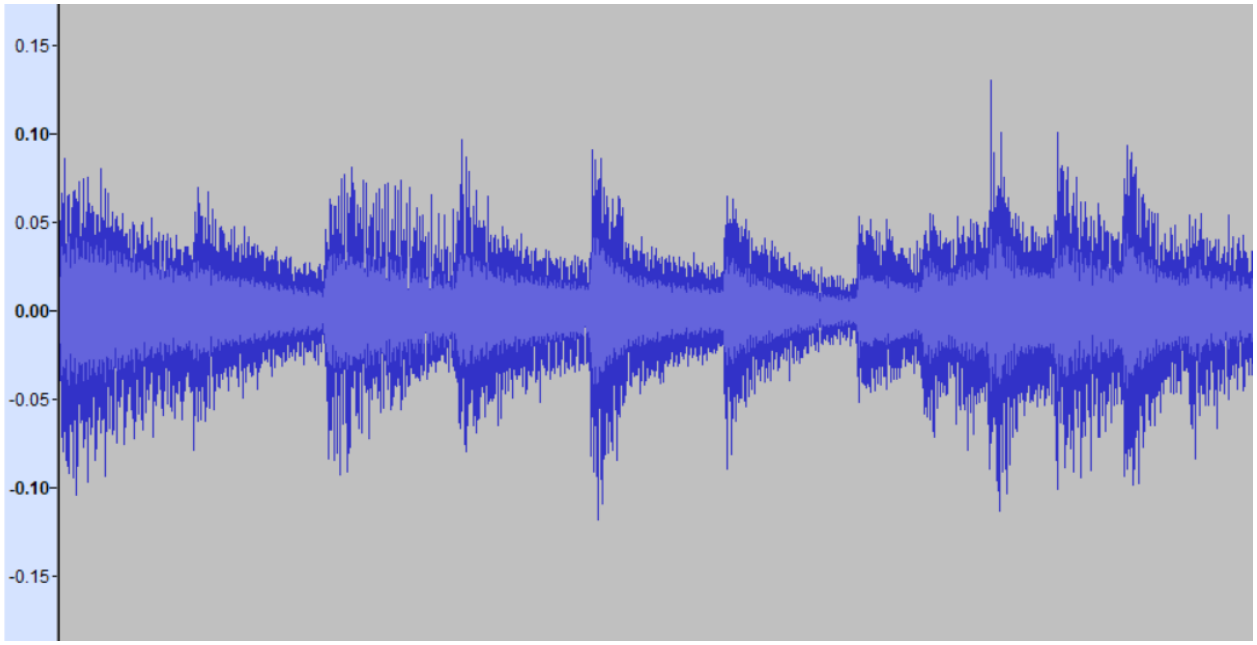
WAV *(ang. Waveform Audio Format)* to format plików binarnych, znany z możliwości zapisywania dźwięku bez użycia jakiegokolwiek algorytmu kompresji. Dzięki temu pliki WAV cha-



Rysunek 2.1.: Fragment chorału J.S. Bacha

rakteryzują się najwyższą jakością dźwięku, jednak ich rozmiary są bardzo duże, co może być problematyczne przy przechowywaniu dużych zbiorów danych na dysku. Plik WAV jest najwierniejszą cyfrową reprezentacją dźwięku analogowego. Jego duży rozmiar wynika z faktu, że dźwięk jest zapisywany z częstotliwością 44,1 kHz, czyli w każdej sekundzie w pamięci zapisywane jest 44100 próbek. Format ten został stworzony w 1991 roku i od tego czasu stał się jednym z najbardziej rozpowszechnionych formatów audio, obsługiwanym przez praktycznie każde oprogramowanie do edycji dźwięku. W celu wizualizacji dokonano elektronicznego odtworzenia utworu przedstawionego w postaci nutowej na obrazku 2.1, co zaowocowało powstaniem fali dźwiękowej, przedstawionej na grafice 2.2.

Często nie jest konieczne przechowywanie dźwięku w formacie bezstratnym, ponieważ większość informacji można zachować przy użyciu mniejszej ilości danych. Jednym z najpopularniejszych standardów kodowania stratnego jest format MP3. Kompresja zmniejsza dokładność kodowania i usuwa częstotliwości, które nie są słyszalne dla człowieka. Stratne kodowanie stara się znaleźć równowagę między jakością dźwięku a rozmiarem pliku. W kontekście uczenia maszynowego zmniejszona ilość sampli przy zachowaniu większości informacji jest zjawiskiem pożądanym. Dzięki temu przynajmniej częściowo rozwiązujemy problem związany z „przekleństwem wymiarowości”, czyli trudnością w przetwarzaniu danych o bardzo wysokiej liczbie cech (wymiarów). Stratne kodowanie pozwala na efektywniejsze zarządzanie danymi, co może być kluczowe w procesach analitycznych i modelowaniu.

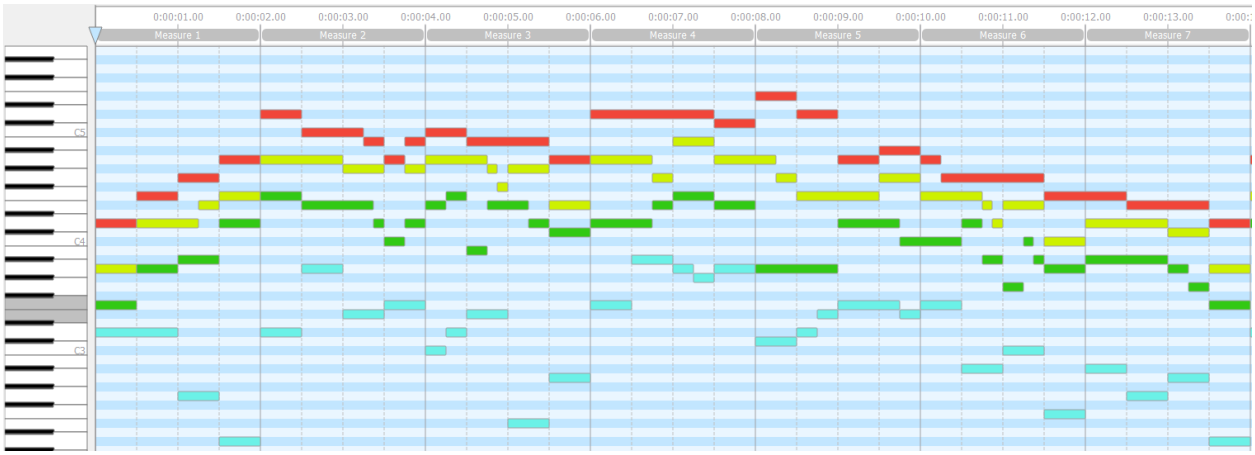


Rysunek 2.2.: Plik `.wav` otworzony w programie Audacity.

2.3.2. Format MIDI

MIDI *(ang. Musical Instrument Digital Interface)* to standard opisujący protokół komunikacji, interfejs cyfrowy oraz złącze, które umożliwiają połączenie elektronicznych instrumentów, komputerów oraz innych urządzeń muzycznych. Pojedynczy kabel MIDI może przesyłać informacje na temat szesnastu kanałów jednocześnie, z których każdy może reprezentować inny instrument. Każda interakcja z instrumentem, taka jak naciśnięcie klawisza czy szarpnięcie struny, jest zapisywana jako "zdarzenie" *(event)*, które zawiera takie dane jak znacznik czasu, wysokość dźwięku oraz jego głośność. Dane z urządzeń MIDI są zapisywane w specjalnych plikach z rozszerzeniem `.mid` lub `.midi`. Umożliwiają one przechowywanie, rozpowszechnianie oraz edytowanie dźwięków w specjalnie stworzonym do tego oprogramowaniu. Ponieważ plik MIDI nie przechowuje rzeczywistej fali dźwiękowej nagranej przez mikrofon, możliwa jest np. późniejsza zmiana instrumentu, który będzie odtwarzał zapisane dźwięki. Typowym sposobem przedstawienia danych MIDI jest tak zwany *piano roll*, który można porównać do dwuwymiarowego układu współrzędnych, gdzie na osi poziomej reprezentowany jest czas, a oś pionowa przedstawia konkretne dźwięki, pokazane jako klawisze pianina. Przykładowe porównanie zapisu MIDI z zapisem nutowym przedstawiono na grafikach 2.3 oraz 2.1.

Warto zaobserwować, że zapis pliku w formacie MIDI umożliwia zdecydowanie większą



Rysunek 2.3.: Muzyka zapisana w pliku MIDI

ekspresje, ponieważ jest to zapis odtworzenia przez artystę pewnego utworu. Ten sposób nie ogranicza muzyki sztywno do konkretnego rytmu zdefiniowanego przez autora.

2.3.3. Notacja ABC

Notacja ABC to system zapisywania nut w formie tekstowej przy użyciu znaków ASCII. Notacja ta pojawiła się w latach 70. XX wieku w celu zapisu i nauki tradycyjnych irlandzkich melodii. W kolejnej dekadzie system ten został rozwinięty przez Chrisa Walshawa, który używał go do zapisywania tradycyjnych melodii, zanim opanował standardowy zachodni zapis nutowy. Walshaw stworzył program *abc2mtex*, który na podstawie notacji ABC generował komendy umożliwiające zapis partytur w formacie *MusicTex*. Obecnie używanym standardem notacji ABC jest wersja z 2011 roku.

Obrazek 2.4 przedstawia fragmentu muzyki 2.1 zapisany w notacji ABC. Format w dość zwięzły sposób zapisuje partyturę. Poza konkretnymi nutami i rytmem, w tym formacie można zapisać również dodatkowe informacje na temat utworu. W każda linijce zaczynająca się od znaku A-Z, a następnie dwukropka, znajduje się tak zwane "pole informacyjne". W tych polach można zapisywać takie informacje jak tytuł utworu, metrum oraz wiele innych danych, w tym z jakiego zbioru muzycznego pochodzi utwór lub jaki jest jego tytuł. Wiele z tych informacji powinna zostać usunięta w procesie preprocessingu danych, aby nie zagłuszały tekstem najistotniejszych pól w utworze. Do ważnych pól należą przede wszystkim: domyślna długość nuty (`L:`), metrum (`M:`) oraz tonacja w jakiej utwór został napisany (`K:`). Chociaż model jest w stanie "odgadnąć" te informacje na podstawie dźwięków, ich wyraźne podanie daje modelowi więcej informacji na temat każdej z sekwencji. Dodatkową zaletą podawania tych informacji podczas treningu jest możliwość użycia ich jako początkowej sekwencji, na

```
X:1
T:Chorał no.7
A:J.S. Bach
Q:1/4=120
V:1
L:1/16
M:4/4
K:C clef=G2
D4F4G4A4|d4c6B2A2B2|c4B8A4|d12^c4|
V:2
L:1/16
M:4/4
K:C clef=G2
A,4D6E2F4|A8^G4A2^G2|A6^G^F^G4E4|A6G2B4A4|
V:3
F,4A,4^A,4D4|F4E7DC2D2|E2F2B,2E4D2^C4|D6E2F4E4|
V:4
L:1/16
M:4/4
K:C clef=F4
D,8G,,4D,,4|D,4A,4E,4F,4|C,2D,2E,4E,,4A,,4|F,4^A,4A,2^G,2A,4|
```

Rysunek 2.4.: Zapis wielu głosów w notacji ABC.

podstawie której model będzie dalej generował melodię, dzięki czemu można mieć kontrolę nad takimi aspektami jak tonacja i metrum utworu, co pozwala na bardziej precyzyjne kierowanie procesem generowania muzyki.

Notacja ABC wpiera również melodie polifoniczne przy pomocy tagów `V`. Ich ilość nie jest ograniczona, w związku z czym można w tym zapisie jest możliwe ujęcie nawet muzyki orkiestrowej. Dość skrajnym przykładem jest zapis drugiej części VII symfonii Ludwiga van Beethovena, która składa się aż z 19 partii instrumentów [12].

Tokenizacja plików MIDI

Tokenizacja danych jest procesem przekształcania tekstu na mniejsze jednostki, zwane tokenami, które mogą być słowami, frazami lub nawet pojedynczymi znakami. W kontekście modeli językowych, tokenizacja jest niezbędna, ponieważ modele te operują na liczbach, a nie bezpośrednio na surowym tekście. Tokenizacja umożliwia rozbicie tekstu na części,

które mogą być łatwo przetwarzane przez algorytmy komputerowe. Dzięki temu modele mogą analizować i generować tekst w sposób bardziej efektywny i precyzyjny. Używa się jej również do standardyzacji danych wejściowych, co pozwala na lepsze zrozumienie kontekstu i struktury języka przez model, a także na zmniejszenie złożoności obliczeniowej. Tokenizacja jest kluczowym krokiem, który umożliwia modelom językowym skuteczne przetwarzanie i interpretowanie dużych zbiorów danych tekstowych.

W przypadku muzyki tokeny mogą reprezentować wartości atrybutów nut (wysokość, wartość, czas trwania) lub zdarzenia czasowe. Token może przyjmować jedną z trzech form:

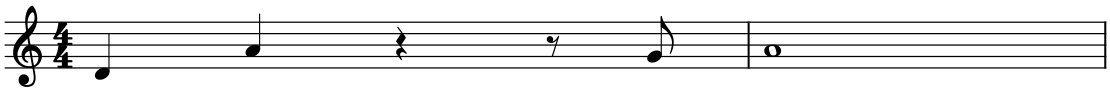
- nazwa tokenu - słowna reprezentacja *eventu* MIDI np. *Pitch_50*
- id - unikalna wartość liczbowa przypisana konkretnemu zdarzeniu
- bajt - unikalny bajt, który został przydzielony podczas treningu tokenizera

Nowe słownictwo jest zapisywane w postaci *look up table* łączącej nazwę tokenu z odpowiadającym jej id lub bajtem. Trening tokenizera polega na obliczeniu kodowania gramatykowego np. *byte pair encoding* do postaci tabelarycznej w celu wykorzystania ich w dalszym modelowaniu.

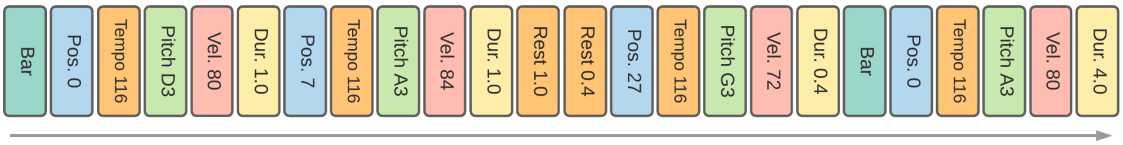
Poza tokenami, które tokenizer tworzy na podstawie pliku MIDI dodaje on również dodatkowe takie jak:

- `PAD` (*padding*) - token używany w przypadku kiedy w *batchu* długość sekwencji jest różna; w takim przypadku tym tokenem wydłuża się sekwencje aby wszystkie miały długość najdłuższej
- `BOS` (*beginning of section*) - token oznaczający początek sekwencji
- `EOS` (*end of section*) - token oznaczający koniec sekwencji

Istnieje wiele algorytmów tokenizacji MIDI, jednak aby przedstawić mechanizm działania, dokonano analizy popularnego algorytmu *REMI*. ***RE****vamped* ***MI****DI* reprezentuje zdarzenia jako sekwencje tokenów wysokości tonu, dynamiki (głośności), długości oraz czasu trwania poprzez kolejne tokeny. Token taktu oznacza początek nowego taktu, natomiast token pozycji określa miejsce zdarzenia w bieżącym takcie. Porównanie pomiędzy zapisem nutowym a jego reprezentacją w formie tokenów można zaobserwować na ilustracjach 2.5 i 2.6. Tokeny `Bar` oznaczają początek taktu, `Pos` pozycję nuty w danym takcie, `Pitch` wysokość nuty, `Vel` jej głośność a `Dur` czas jej trwania. Wiele algorytmów tokenizacji jest zaimplementowanych w bibliotece MidiTok dla języka Python [13].



Rysunek 2.5.: Prosta sekwencja muzyczna



Rysunek 2.6.: Reprezentacja muzyki w postaci tokenów

	rozmiar pliku na dysku	długość sekwencji
`wav`	3971116 B	992768
`mp3`	360951 B	992768
`midi`	1638 B	740
`abc`	953 B	562

Tabela 2.1.: Porównanie różnych zapisów muzyki

2.3.4. Porównanie zapisu muzycznego

W tabeli 2.1 zostało przedstawione porównanie zapisu całego utworu, którego fragment przedstawiono na grafice 2.1.

Jak widać zapis w postaci fali dźwiękowej zajmuje najwięcej miejsca na dysku oraz długość sekwencji, która zapisuje cały utwór jest najdłuższa. W celu uzyskania sekwencji z pliku MIDI został użyty tokenizer REMI o którym jest mowa w rozdziale 2.3.3. Podczas używania innych tokenizatorów długość sekwencji może różnić sie pomiędzy nimi. Różnica w rozmiarze na dysku oraz długością sekwencji pomiędzy zapisem MIDI oraz ABC nie jest duża, jednak warto zwrócić uwagę, że gdyby w oryginalnym utworze pojawiły się powtórzenia sekcji melodii, notacja ABC byłaby jeszcze bardziej oszczędna od plików MIDI. Powodem tego jest fakt, że notacja ABC zapisuje bezpośrednio zapis nutowy w postaci tekstowej, którego elementami są znaki repetycji. Są zapisywane znakami `:|` oraz `|:`, a w pliku MIDI cała powtarzana sekwencja musi zostać ponownie odegrana.

Do głównych zalet plików MIDI należy ich wszechstronność. Nie są one ograniczone do sztywno określonego rytmu w postaci ćwierćnut, ósemek czy szesnastek, jak w innych notacjach muzycznych, co daje zdecydowanie większe możliwości ekspresji. Dodatkowo, przy użyciu narzędzi komputerowych, bardzo łatwo można zamienić dowolny zapis nutowy (np.

MusicXML lub ABC) na plik MIDI. Niestety, w drugą stronę, czyli konwersja pliku MIDI na zapis nutowy, jest znacznie trudniejsza, głównie z powodu nieokreślonego rytmu. Istnieją narzędzia, takie jak MuseScore, które potrafią wygenerować partyturę z pliku MIDI, jednak skuteczność tych narzędzi nie zawsze jest wysoka, zwłaszcza w przypadku bardziej skomplikowanych utworów, gdzie rytm i artykulacja mogą być trudne do dokładnego oszacowania. Mimo tych wyzwań, pliki MIDI pozostają niezwykle użyteczne w zastosowaniach muzycznych, od tworzenia i edytowania muzyki po jej analizę i odtwarzanie.

Zapis plik w postaci fali dźwiękowej niesie ze sobą wiele wad, jednak jest to często jedyne rozwiązanie, aby zebrać dużą ilość danych szczególnie w przypadku muzyki nowoczesnej, kiedy twórcy nie udostępniają zazwyczaj plików MIDI ani zapisu nutowego, na podstawie którego powstała dana piosenka. Z tego powodu praca z plikami MP3 lub czasochłonna translacja muzyki na inny format jest często konieczna.

Dodatkową zaletą plików dźwiękowych w zapisie cyfrowym jest dostęp do potencjalnie istniejącego zapisu odśpiewanego tekstu. Pomimo że notacja ABC pozwala na zapis tekstu w tagu `W:`, jest to jedynie zapis słowny, a nie faktyczne odśpiewanie. W związku z tym pliki WAV są bardziej odpowiednie do próby replikacji czyjegoś głosu, ponieważ zawierają rzeczywisty zapis audio, który można analizować i przetwarzać.

Główną zaletą plików ABC jest ich tekstowa reprezentacja notacji muzycznej. Z tego powodu nie wymagaja one wiedzy na temat struktury, budowy oraz standardów plików np. MIDI aby zacząć z nimi pracować. Ponieważ notacja ABC jest tekstowa, można ją łatwo integrować z narzędziami do przetwarzania tekstu. Zwięzłość zapisu jest również przyczyną dla której model uczenia maszynowego może być mniejszy w związku z mniejszą ilością tokenów w sekwencji, na podstawie której model się uczy. Zapis w postaci tekstu pozwala na bardzo proste dodanie pewnych dodatkowych danych, które można policzyć w etapie *preprocessingu*. Podejście takie zostało zaproponowane w modelu *Tunesformer* [14], który wziął przykład z artykułu CTRL [15], w którym autorzy dodawali do korpusu treningowego danych tekstowych takie tagi jak *Books*, *Horror*, *Relationships*, *Legal*. Tagi te były podawana na początku *promptu* podawanego dla modelu, przez co model miał w jasny sposób podane w jakim stylu powinien udzielić odpowiedzi. W przypadku muzyki w zapisie ABC zostały wprowadzone nowe *control codes*:

- `S:` - liczba sekcji w całym utworze; tag jest prost do policzenia ponieważ początek i koniec taktu jest jawnie oznaczany przy pomocy symboli `[|`, `||`, `|]`, `|:`, `::` i `:|`
- `B:` - liczba taktów w danej sekcji; liczy tylko wystąpienia znaku `|`
- `E:` - kontroluje poziom podobieństwa pomiędzy dwoma następującymi po sobie sekcjami; w związku z tym, że zapis melodii jest w postaci tekstu, można policzyć podobieństwo używając odległości Levenshteina używając wzoru

$$E(c,p) = 1 - \frac{lev(c,p)}{\max(|c|,|p|)} \tag{2.2}$$

gdzie c oraz p to następujące po sobie sekwencje, a $|c|$, $|p|$ to ich długości.

Policzone kody wraz z ich odpowiednim oznaczeniem dodawane są do pliku tekstowego, który zapisuje notacje. Dodanie ich nie sprawia, że notacja staje się nieprawidłowa. Tak samo jak w przypadku tagów, które opisują elementy zapisu jak tonacja czy metrum, służą one nam oraz modelowi jako wskazówki, którymi powinien się sugerować podczas generacji muzyki.

2.4. Zbiory danych

2.4.1. Johann Sebastian Bach Chorales

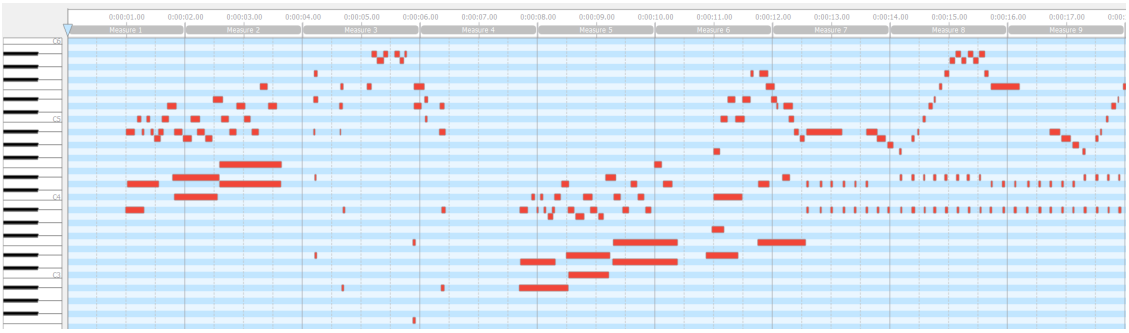
JSB Chorales [16] to zbiór krótkich, czterogłosowych utworów. Chorały zostały pierwotnie skomponowane przez Johanna Sebastiana Bacha w XVIII wieku. Napisał je najpierw biorąc wcześniej istniejące melodie ze współczesnych hymnów luterańskich, a następnie harmonizując je w celu stworzenia części dla pozostałych trzech głosów. Zbiór danych zawiera 382 takich chorałów. Utwory przed złożeniem w zbiór danych zostały transponowane do tonacji C-dur przez co stają się poostrze do analizy przez algorytmy. Dodatkowo ich rytm został znormalizowany i najmniejszą wartością jest szesnastka. Dzięki temu uproszczeniu zbiór jest bardzo łatwo konwertowalny do dowolnego zapisu które zostały zaprezentowane na grafikach 2.1 2.3 2.4.

Dodatkową zaletą takiego rytmu jest możliwość pominięcia użycia tokenizera MIDI, ponieważ wiedząc że rytm jest stały i wyznaczony przez szesnastkę, można użyć naiwnego podejścia i zapisać każdy głos jako wektor, którego wartościami jest obecnie grająca nuta. Po złożeniu wektorów w macierz 4× (długość utworu) można przejść do analizy. Wadą tego podejścia jest możliwość wystąpienia macierzy rzadkich, które często stanowią problem dla algorytmów uczenia maszynowego [17], jednak w tym zbiorze ten problem nie wystepuje.

2.4.2. The MAESTRO v3.0

Zbiór danych MAESTRO [18] powstał w współpracy z *Minnesota International Piano-e-Competition*, czyli międzynarodowym konkursem pianistycznym. Litera *e* w nazwie odnosi się do fortepianów używanych podczas konkursu którymi są Yamaha Disklavier. Jest to rodzaj instrumentu, który poza tradycyjną funkcjonalnością jest obudowany elektronicznymi sensorami, które pozwalają między innymi zapis odegranej muzyki w formacie MIDI. Z tego powodu organizacja Magenta działająca wewnątrz Google użyła zapisu melodii z wielu edycji konkursu aby stworzyć dataset zawierający około 200 godzin muzyki fortepianowej. W tych 200 godzinach znajduje się 1276 wystąpień w skład których wchodzi ponad 7 milionów nut. Dwa rodzaje zbioru jakie można pobrać to zbiór w wersji WAV oraz MIDI. Pierwszy,

z powodów które przedstawiono w tabeli 2.1 zajmuje aż 122 GB, natomiast wersja MIDI zajmuje tylko 81 MB.

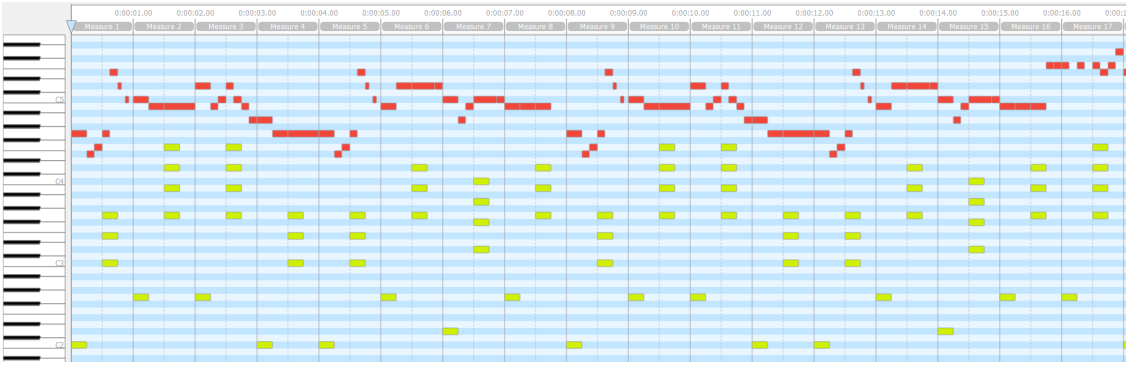


Rysunek 2.7.: Przykład utworu pochodzącego ze zbioru MAESTRO.

Porównując zapis prostego utworu widoczny na grafice 2.3 z tym na grafice 2.7 osoby nawet nie znające się na muzyce są w stanie wywnioskowac że ten utwór bardziej złożony. Jak widać odegrane nuty często nie padaja na sztywno określone rozpoczęcie taktu, przez co potencjalnie wygenerowana muzyka na podstawie tego zbioru będzie wydawała się bardziej ekspresyjna oraz “ludzka”.

2.4.3. IrishMAN

Zbiór danych IrishMAN (*ang. Irish Massive ABC Notation*) [19] zawiera 216,248 fragmentów melodii podzielonych na zbiór treningowy i walidacyjny. Melodie zebrane są ze stron takich jak `thesession.org` oraz `abcnotation.org`, które znane są z udostepniania tradycyjnej muzyki w różnych formatach. Aby zapewnić jednolitość formatowania najpierw wszystkie utwory zostały przekonwertowane do notacji MusicXML, a nastepnie do notacji ABC. Istnieją bliźniacze zbiory IrishMAN-MIDI oraz IrishMAN-XML, które zawierają te same utwory jednak zapisane w odpowiadających ich nazwą formatach. Wszystkie fragmenty muzyki należą do domeny publicznej, w związku z czym można korzystać z tych melodii bez obaw o łamanie praw autorskich. Zbiór jest o tyle ciekawy że zawiera w sobie muzykę jedno, jak i wielogłosową. Autorzy zbioru udostępnili również skrypty napisane w języku Python przy pomocy których, można samemu stworzyć własne zbiory danych na podstawie swoich plików ABC lub MusicXML. Jest to również jeden z niewielu zbiorów, który udostępnia te same fragmentu muzyki w różnych formatach, przez co nadaje się on wyjątkowo dobrze do porównywania różnych algorytmów generacyjnych.

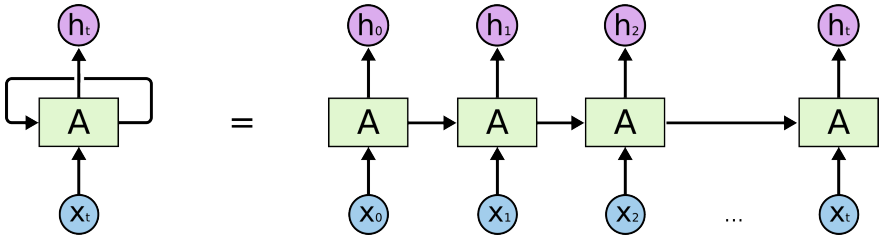


Rysunek 2.8.: Przykład wielogłosowego fragmentu ze zbioru IrishMAN.

2.5. Architektura transformera

2.5.1. Sieci RNN

Pierwszymi próbami użycia sieci neuronowych w przetwarzaniu sekwencji było użycie rekurencyjnych sieci neuronowych. Architektura ich była prosta i dodawała ona pętlę w taki sposób aby wartości mogły być przekazywane pomiędzy krokami. Rysunek 2.9 przedstawia fragment sieci oraz jej rozwinięcie [20].



Rysunek 2.9.: Schemat budowy fragmentu sieci RNN.

Prezentowane podejście jest dość proste, jednak niesie ze sobą pewne problemy.

Pierwszym z nich jest brak selekcji, które informacje są rzeczywiście istotne w danym kontekście, a które nie. W zadaniu przewidywania kolejnego słowa np. dla zdania "Kupiłem zegarek na *rękę*" nie jest wymagany żaden dodatkowy kontekst, a odległość pomiedzy ważnymi informacjami jest mała, jednak w zdaniu "Rok temu ukończyłem kurs lotnika ... Teraz staram się o licencje *pilota*" wymagany jest szersze spojrzenie na całą strukturę wypowiedzi i "przypomnienie" informacji z początku zdania. W przypadku muzyki sytuacja ma się dokładnie tak samo. Przykładowo, aby rozwiązać akord dysonansowy na konsonan-

sowy lub dominantowy na tonikę, wymagana jest jedynie informacja na temat ich dzięków, tak aby dobrać odpowiedni nowy akord, jednak jeśli w muzyce przewija się pewien temat, to wymagane jest zachowanie informacji w sieci, w jakich momentach się on pojawia, aby móc dodać go w odpowiednie miejsce. W teorii zwyczajna sieć rekurencyjna jest zdolna do zapamiętywania długich zależności, jednak w praktyce nie jest to bardzo rzadko osiągalne.

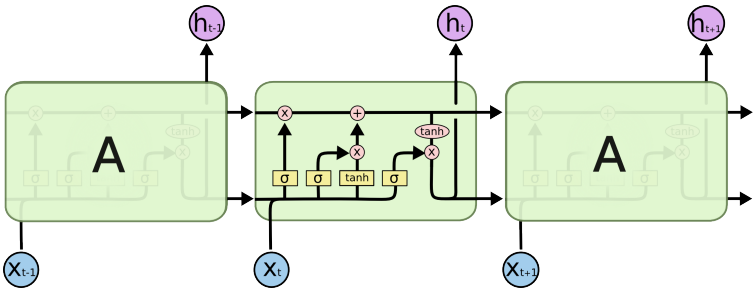
Dodatkowym problemem podczas treningu modeli jest problem zanikającego gradientu. Równanie 2.3 przedstawia równania potrzebne w celu policzenia konkretnego stanu ukrytego H_i.

$$\begin{aligned} &H_{i+1} = A(H_i, x_i) \\ &H_3 = A(A(A(H_0, x_0), x_1), x_2) \\ &\text{gdzie:} \\ &A(H, x) := Wx + ZH \\ &H_N = W^N x_0 + W^{N-1} x_1 + \dots \end{aligned} \tag{2.3}$$

Jak widać z powyższego równania, kiedy N będzie wystarczająco duże (a na pewno będzie ponieważ jest to długość okna kontekstowego) macierz wag W będzie podnoszona do bardzo wysokiej potęgi przez co jej wartości poniżej 1 będą zanikać do 0, a te będące powyżej 1 będą eksplodować w nieskończoność.

2.5.2. LSTM

Aby rozwiązać omawiane problemy sieci rekurencyjnych została zaproponowana architektura sieci nazwana *Long Short Term Memory* [21]. Ich budowa pozwala na długofalowe zapamiętywanie i przekazywanie informacji. Warstwy LSTM są zazwyczaj łączone między sobą co pozytywnie wpływa na działanie sieci.



Rysunek 2.10.: Komórki LSTM połączone między sobą.

Główną częścią budowy komórki jest górna część reprezentowana przez czarną strzałkę na obrazku 2.10, która posiadając jedynie proste operacje liniowe przekazuje z stany ukryte z

jednej części do drugiej. Dolna część komórki zawiera tak zwane bramki, które kontrolują które informacje są ważne i mają być zachowane a które można zignorować. Są one zbudowane z warstw sigmoidalnych oznaczonych na grafice literami σ. Wartościami zwrotnymi jest liczba pomiędzy 0 a 1 gdzie mała wartość znaczy niską a wysoka dużą ważność informacji. LSTM zwraca dwa wektory. Pierwszy to stan ukryty (*hidden state*), który przechowuje informacje o bieżącym stanie sieci i jest aktualizowany przy każdym kroku czasowym. To ten wektor jest bezpośrednio wykorzystywany do przewidywania kolejnych słów lub innych zadań wyjściowych. Drugi nazywany stanem komórki (*cell state*), jest dodatkowy i przechowuje informacje na dłuższą metę, przez co jest odpowiedzialny za przechowywanie bardziej trwałego kontekstu. Stan komórki jest modyfikowany przez tzw. bramki *(gate)*, które decydują, które informacje powinny być dodane, usunięte lub zaktualizowane.

Architektura takiego modelu sprawia, że jest on o wiele mniej podatny na problem zanikającego gradientu, jednak jest to sytuacja, która niestety może nadal występować [22]. Dodatkową wadą modelu LSTM jak i klasycznego RNN jest ich wysoka złożoność treningu, która występuje z powodu konieczności użycia specjalnego wariantu algorytmu wstecznej propagacji błędu nazywanego "wsteczną propagacją błędu w czacie" *backpropagation through time*, która rozwija sieci rekurencyjne, aby propagować gradienty.

Istnieje kilka wariantów LSTM-ów np. dwukierunkowe LSTM, które pozwalają aby informacja przepływa w dwóch kierunkach na raz a nie tylko z lewej strony do prawej [23]. Użycie takich modeli poprawia skuteczność i wyniki w stosunku do klasycznego modelu, jednak problemy z ich treningiem dalej pozostają lub nawet stają się jeszcze większe.

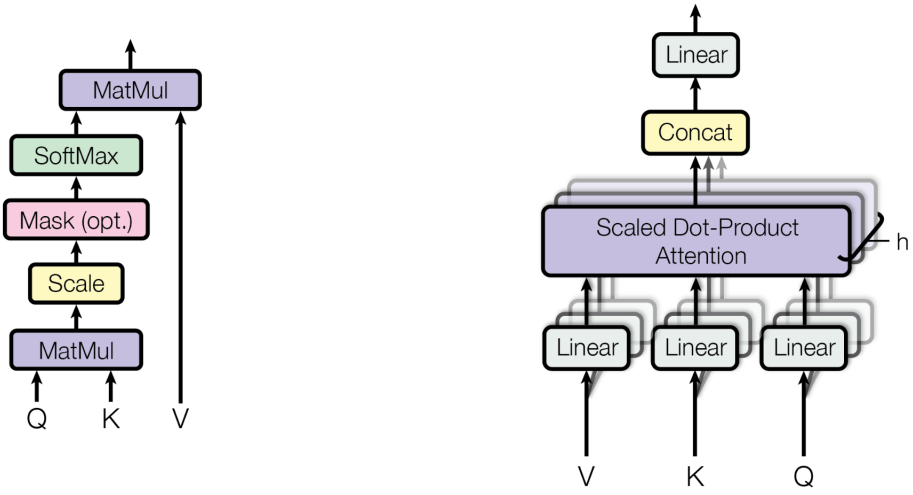
2.5.3. Algorytm uwagi (ang. *attention*)

Mechanizm uwagi (*ang. attention*) został zaproponowany w przełomowym artykule "*Attention is All You Need*" [7]. Algorytm ten działa w ustalonym oknie kontekstowym (*ang. context window*), w którym przydziela każdemu słowu wagę reprezentującą jego ważność w danym kontekście. Metoda ta imituje sposób w jaki człowiek, czytając na przykład książkę, skupia się na różnych fragmentach tekstu w danym momencie, interpretując dane słowa w kontekście innych, już przeczytanych oraz tych, które zostaną przeczytane za chwilę. Mechanizm uwagi został opracowany, aby rozwiązać problem występujący w rekurencyjnych sieciach neuronowych, gdzie "ważność" słowa była tym większa, im bliżej końca sekwencji się ono znajdowało, co sprawiało, że modele często ignorowały informacje znajdujące się na początku okna kontekstowego. Mechanizm uwagi został zaprojektowany tak, aby identyfikować najwyższe korelacje między słowami w zdaniu, zakładając, że nauczy się tych wzorców ze zbioru treningowego. Korelacja ta jest przechwytywana w wagach neuronów poprzez propagację wsteczną, albo z samonadzorowanego szkolenia wstępnego, albo z nadzorowanego *fine-tuningu*. Algorytm działa na wartościach liczbowych, więc przez analizą, każda sekwencja musi zostać osadzona (*embedded*) do wektora o określonej długości.

W kontekście muzyki problem zanikania kontekstu jest jeszcze bardziej widoczny niż w

tradycyjnych problemach przetwarzania języka naturalnego, ponieważ na początku zapisu nutowego znajdują się kluczowe dla niego informacje, takie jak metrum czy tonacja utworu, zapisane za pomocą krzyżyków lub bemoli. Z tego powodu model, analizując dalsze części sekwencji, może ignorować te istotne elementy.

Podstawowa wersja algorytmu nazywana jest *self-attention* lub *scaled dot-product attention* ponieważ sekwencja "zwraca uwagę" na samą siebie, czyli innymi słowy każdy token w sekwencji jest rozważany z każdym innym poza sobą. Zależności pomiędzy słowami w mechanizmie uwagi można przedstawić jako graf pełny skierowany, gdzie wierzchołkami są poszczególne słowa lub inne tokeny z okna czasowego, a krawędziami są wartości określające, jak istotne są dla danego słowa wszystkie inne. Jest to zdecydowanie lepsze podejście niż w to z rekurencyjnych sieciach neuronowych lub LSTM-ów, gdzie zależności dla danego słowa są przekazywane jedynie w wektorze, który kompresuje kontekst ze wszystkich poprzednich słów. Dzięki temu mechanizm uwagi pozwala na bardziej dynamiczną kontrolę i monitorowanie, jak zależności pomiędzy tokenami formują się podczas treningu.



Rysunek 2.11.: Schemat mechanizmu uwagi oraz ich kilkukrotne złożenie.

Przedstawione na rysunku 2.11 wartości K (Key), V (Value) i Q (Query) w mechanizmie *self-attention* są uzyskiwane przez przemnożenie danych wejściowych i odpowiadającym im nauczonych się macierzy wag. Dla każdego elementu sekwencji (reprezentowanego przez jego zapytanie Q), obliczane jest podobieństwo z każdym innym elementem sekwencji (reprezentowanym przez klucze K). Najczęściej używa się iloczynu skalarnego, który jest normalizowany przez wymiar osadzonego wektora (d_k) aby uzyskać współczynniki uwagi a następnie aby wartości sumowały się do jedynki używana jest funkcja softmax. Ten krok można rozumiec jako tworzenie "bazy danych" gdzie kluczami są poszczególne tokeny, a odpowiadającymi im wartościami jest ich ważność w całej sekwencji. Każda wartość V jest ważona przez

współczynniki uwagi obliczone w poprzednim kroku, co pozwala modelowi skupić się na najistotniejszych elementach sekwencji. Cały ten złożony proces sprowadza się do jednego równania zapisanego w 2.4. W algorytmie można dodać opcjonalną maskę, która umożliwia zasłonięcie niektórych wartości uwagi, dzięki czemu te tokeny nie będą na siebie wpływać. Prosty schemat tego algorytmu został pokazany na lewej stronie grafiki 2.11.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.4}$$

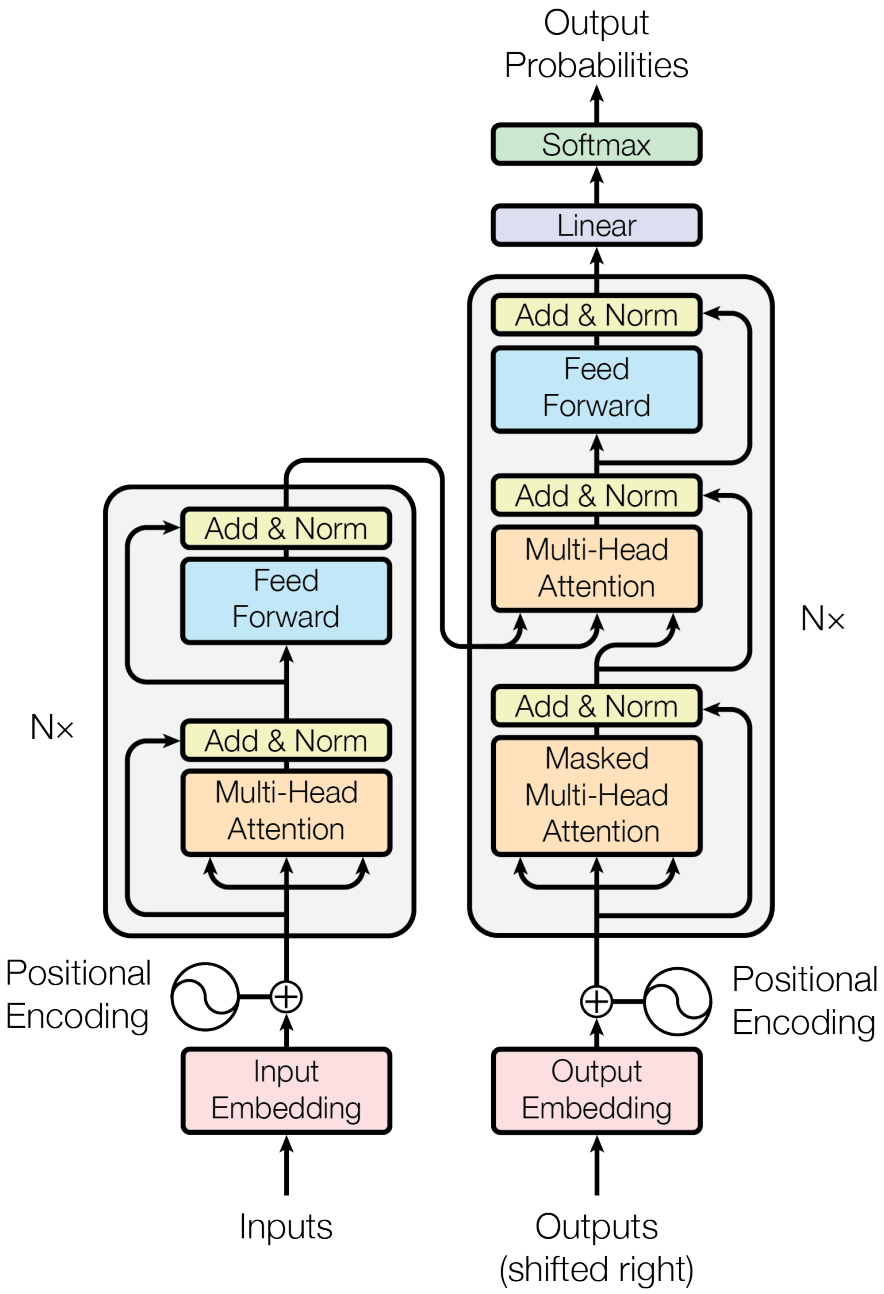
Aby zeskalować model i zwiększyć liczbę jego parametrów w celu użycia go do bardziej zaawansowanych celów, można użyć algorytmu *multi-headed attention*. Jest to kilka warstw uwagi, przed wejściem do których znajduja się warstwy liniowe. Wartości po wyjściu z algorytmu *attention* są konkatenowane, a następnie ponownie przetwarzane przez warstwę liniową. Tym sposobem, zamiast tylko jednej instancji mechanizmu uwagi, model może korzystać z wielu. Schemat takiego podejścia pokazano na prawej stronie grafiki 2.11.

2.5.4. Transformer

Wraz z wprowadzeniem algorytmu uwagi pisanego w 2.5.3 w tym samym artykule został zaproponowana architektura modelu uczenia maszynowego wykorzystująca *multi-headed attention*. Cały model składa się z dwóch części nazwanych enkoderem oraz dekoderem.

Enkoder

Zadaniem enkodera jest zapoznanie się z całym korpusem tekstu i przekazanie zebranych informacji do dekodera. Mechanizm samo-uwagi *self-attention* jest centralnym elementem każdej warstwy enkodera. Pozwala on modelowi analizować relacje między różnymi tokenami w sekwencji niezależnie od ich odległości od siebie. Warto zwrócić uwagę, że enkoder jest dwukierunkowy, czyli tokeny są w stanie "patrzeć w przód" analizowanej sekwencji. Drugi komponent każdego enkodera to sieć neuronowa typu *feed forward*, która składa się z dwóch w pełni połączonych warstw z funkcjami aktywacji *ReLU*. Każdy z dwóch głównych części składowych jest otoczony mechanizmem dodania rezydualnego, poprawiającego wydajność treningu, oraz warstwą normalizacji. Dane przed wejściem są kodowane przy pomocy tabeli kodowań do wektorów oraz, aby zapewnić modelowi informacje na temat kolejności słów we fragmencie, jest dodawany wektor kodujący ich kolejności. Omawiane w sekcji 2.5.3 pracy wartości K (klucze) oraz V (wartości) otrzymane na wyjściu enkodera zostają przekazane do dekodera.



Rysunek 2.12.: Schemat transformera.

Dekoder

Zadaniem Dekoder w architekturze transformerowej jest generowanie wyjścia na podstawie sekwencji wejściowej i dotychczas wygenerowanych tokenów wyjściowych. Podobnie jak w enkoderze, mechanizm samo-uwagi analizuje zależności między tokenami w sekwencji wyjściowej, jednak w dekoderze stosuje się maskowanie (*ang. masking*), aby zablokować dostęp do przyszłych tokenów podczas generowania bieżącego tokena. Maskowanie uniemożliwia modelowi zobaczenie tokenów, które są generowane w późniejszych krokach. Maskę można zilustrować jako macierz trójkątną górną z wartościami $-\infty$, którą następnie dodaje się do wartości uwagi co zostało pokazane na grafice 2.13. Maska zawiera te wartości z tego powodu, że jedną z operacji jaką wykonuje funkcja softmax jest e^{x_i}, co w przypadku wartości $x_i = -\infty$ da zero, co sprawi że te wartości zostaną zignorowane w dalszych operacjach modelu.

	Ala	ma	kota	i	psa
Ala	0.13	0.18	0.16	0.15	0.18
ma	0.68	0.02	0.08	0.14	0.02
kota	0.06	0.25	0.14	0.11	0.23
i	0.21	0.14	0.16	0.17	0.14
psa	0.27	0.11	0.16	0.18	0.12

$\longrightarrow$

	Ala	ma	kota	i	psa
Ala	0.13	$-\infty$	$-\infty$	$-\infty$	$-\infty$
ma	0.68	0.02	$-\infty$	$-\infty$	$-\infty$
kota	0.06	0.25	0.14	$-\infty$	$-\infty$
i	0.21	0.14	0.16	0.17	$-\infty$
psa	0.27	0.11	0.16	0.18	0.12

Rysunek 2.13.: Algorytm uwagi z maską.

Drugi mechanizm uwagi umożliwia dekoderowi skupienie się na odpowiednich częściach sekwencji wejściowej (wyjścia enkodera) podczas generowania każdego tokena wyjściowego. Proces ten jest podobny do mechanizmu samo-uwagi, ale tutaj zapytania (Q) pochodzą z poprzedniej warstwy uwagi dekodera, a klucze (K) i wartości (V) pochodzą z wyjścia enkodera. Następnie tak jak w poprzednim przypadku wystepuje warstwa *feed forward* i warstwy normalizujące. W ostatniej części modelu znajduje się warstwa liniowa, której zadaniem jest wyprodukowanie wyjścia o rozmiarze ilości wszystkich słów lub tokenów, oraz warstwa *softmax*, która przypisuje każdemu tokenowi prawdopodobieństwo bycia następnym słowem w sekwencji.

Istnieją modele transformerowe nie korzystające z całej architektury. Przykładem *encoder-only* transformer jest model BERT [24], który jest typowo wykorzystywany do zadań np.

klasyfikacji tekstu. W przypadku takiego zadania, nie potrzeba omawianej wcześniej maski, ponieważ model powinien analizować kontekst całego fragmentu w "obie strony". Wystarczy dołożyć na wyjście modelu warstwę liniową o odpowiednim rozmiarze oraz zastosować odpowiednią funkcję aktywacji, aby przerobić enkoder na model klasyfikacyjny. Analogicznie jeśli celem zadania jest wyłącznie generacja tekstu lub muzyki, enkoder nie jest koniecznie wymagany. W takim problemie należy usunąć drugą warstwę uwagi (bez maski), ponieważ wektory kluczy i wartości pochodzące z enkodera będą nieznane. Generacja będzie odbywała się jedynie na podstawie podanej sekwencji lub tokenu `BOS` (*beginning of sequence*), co da modelowi największą wolność twórczą. Rozwiązania *decoder-only* są obebnie bardzo popularne i są używane przez modele takie jak *GPT* od OpenAI. Ciekawą obserwacją jest fakt, że dekoder pozbawiony drugiej warstwy uwagi, staje się w zasadzie enkoderem z dodaną maską w pierwszej warstwie *attention*. Jest to często wykorzystywana "sztuczka", którą można wykorzystać w wielu bibliotekach implementujących gotowe części składowe modelu transformera.

Trening transformera jest możliwy do bardzo dobrego zrównoleglenia. Wynika to z faktu, że mechanizm uwagi jest w stanie niezaleznie liczyć wartości pomiędzy słowami. Przykładowo w zadaniu "*Ala ma kota.*" wartości pomiędzy *ma* i *kota* można policzyć nie znając wartości pomiędzy *Ala* i *ma*. Niestety podczas generacji kolejnych tokenów na podstawie istniejącej już sekwencji należy policzyć ponownie wartości uwagi, przez co dla sekwencji o długości n należy wykonać n^2 operacji. Jest to niestety problematyczne jeśli chcemy rozważać bardzo długie sekwencje. Omawiany wcześniej model RNN, podczas predykcji, nie ma problemów ze złożonością, ponieważ cała informacja na temat wcześniejszej sekwencji jest skompresowana w stanie ukrytym. W takim przypadku okno kontekstu jest teoretycznie nieskończone, ponieważ stan ukryty zawiera informacje o wszytkach poprzednich stanach, jednak w praktyce nigdy nie mamy dostepu do takich informacji.

2.6. Modele przestrzeni stanów

2.6.1. Opis modeli *state space*

Modele przestrzeni stanów (*State Space Models* - SSM) to matematyczne modele dynamicznych systemów, które są szeroko stosowane w inżynierii, ekonomii, biologii i wielu innych dziedzinach. Modele te są szczególnie przydatne w analizie i prognozowaniu czasowych szeregów danych.
Typowo, dla każdej jednostki czasu t model:

1. mapuje sekwencje wejściową $x(t)$
2. oblicza reprezentację ukrytą $h(t)$
3. przewiduje sekwencję wynikową $y(t)$

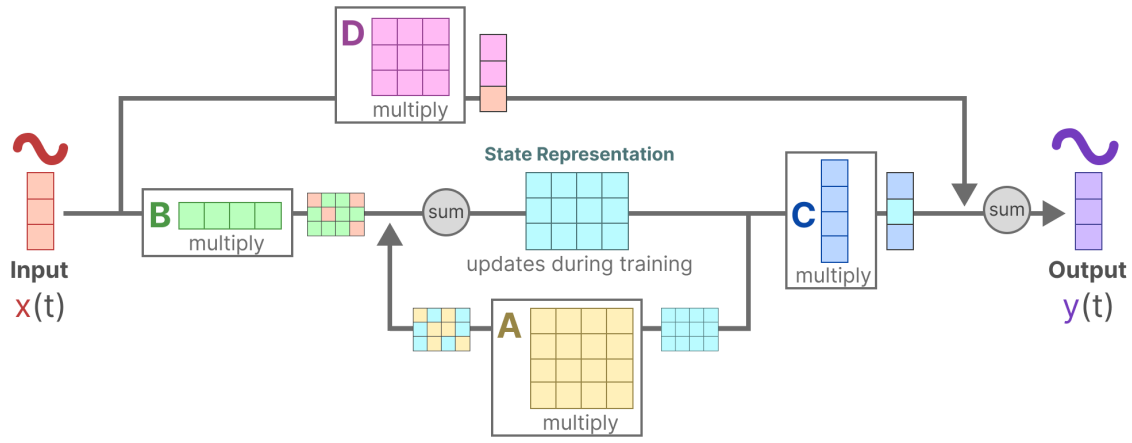
Modele SSM zakładają, że każdy system dynamiczny może zostać opisany w momencie czasu t używając dwóch równań.

$$\text{równanie stanu: } h'(t) = Ah(t) + Bx(t) \tag{2.5}$$

$$\text{równanie wynikowe: } y'(t) = Ch(t) + Dx(t) \tag{2.6}$$

Celem jest poznanie reprezentacji stanu $h(t)$ w taki sposób aby przejść z sekwencji wejściowej do sekwencji wynikowej.

Równanie stanu 2.5 opisuje w jaki sposób zmienia się stan w zależności od tego, jak dane wejściowe mają na niego wpływ. Wykonywane jest to przez macierze A oraz B. Celem równania 2.6 jest opisanie jak obecny stan ukryty wpływa na sekwencje wynikową oraz jak sekwencja wejściowa wpływa na wynik. Te informacje są zapisywane odpowiednia w macierzy C oraz D [25]. Schematyczne przedstawienie działania modelu zostało przedstawione na grafice 2.14.



Rysunek 2.14.: Schemat modelu przestrzeni stanów.

W związku z tym, że macierz D przypomina połączenia rezydualne [26] lub połączenia typu *skip-connection* w głębokich sieciach neuronowych, często pomija się je w zapisie modelu co upraszcza zapis.

Dyskretyzacja modelu

Modele tego typu działaja domyślnie w domenie danych ciągłych. Tekst lub muzyka są przykładami danych dyskretnych, przez co model musi zostać pod nie dostosowany. Aby dokonać dyskretyzacji modelu wykorzystuje się metodę eksploratora (lub interpolatora) rzędy zerowego (*ang. zero-order hold*). Metoda ta polega na podtrzymywaniu każdego otrzymanego

sygnału, dopóki nie zostanie zapisany kolejny. Czas trwania poprzedniego sygnału jest dodatkowym parametrem nazywanym krokiem. W związku z tym, że model na wyjściu zwraca sygnał ciągły, aby zamienić go na dyskretny, należy *samplować* go, z odpowiednią długością kroku. Po dyskretyzacji modelu można nazwać go modelem *sequence to sequence*. Aby lepiej zobrazować dokonane zmiany należy spojrzeć na równania 2.7 oraz 2.8.

$$\text{równanie stanu: } h_k = Ah_{k-1} + Bx_k \tag{2.7}$$

$$\text{równanie wynikowe: } y_k = Ch_k \tag{2.8}$$

Model operuje teraz na sekwencjach a nie na sygnałach. Notacja została zmieniona z t na k aby lepiej zobrazować, że obecnie model pracuje na dyskretnych znacznikach czasu. Dodatkowo w zapisie równania usunięto macierz D.

Reprezentacja konwolucyjna

Patrząc na zapis obu równań od razu można zaobserwować ich podobieństwo do rekurencyjnych sieci neuronowych. Każdy stan ukryty h zależy w głównej mierze od stanu z porzadnego kroku czasowego. *SSM-y* mogą również być reprezentowane w postaci konwolucji. W związku z tym, że tekst lub muzyka są sekwencjami jednowymiarowymi w przeciwieństwie do typowego zastosowania sieci konwolucyjnych czyli obrazów, które są dwu lub trzy wymiarowe należy zastosować kernel o wymiarze jeden. Tego rodzaju interpretacja modelu jest o tyle wygodna, że pozwala zrównoleglić trening modelu a co za tym idzie zmniejszyć jego czas [27]. Kiedy model jest już wytrenowany i gotowy do wykonywania generacji, można bez problemu przejść na interpretacje rekurencyjną.

Macierz HiPPO

Macierz A modelu jest jego jednym z najistotniejszych elementów. To od niej zależy, czy model zapamięta cały kontekst wypowiedzi czy tylko kilka ostatnich tokenów. Jest to szczególnie ważne w przypadku predykcji, ponieważ model bazuje wyłącznie na podstawie ostatniego stanu ukrytego. Aby zapewnić dobrą kompresję kontekstu stosuje się macierz HiPPO [28]. Zadaniem ***Hi**gh-order **P**olynomial **P**rojection **O**perators* jest jak najlepsze skompresowanie wejścia do wektora cech. Konstrukcje macierzy pokazano na równaniu 2.9. W tym przypadku zakładamy, że macierz jest kwadratowa.

$$A_{nk} = -\begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{dla} n > k \\ n+1 & \text{dla} n = k \\ 0 & \text{dla} n < k \end{cases} \tag{2.9}$$

Zbudowanie macierzy w taki sposób prowadzi do zdecydowanie lepszych wyników niż zainicjalizowanie jej losowo. Macierz HiPPO reprodukuje sekwencje w taki sposób, że dane

które są bliżej końca sekwencji są reprodukowane lepiej, niż te na początku. Ideą stojącą za używaniem takiego rozwiązania jest chęć zachowania jak największej liczby informacji w stanie ukrytym. Użycie współczynników wielomianów Lagrange'a pozwala aby macierz pomagała zapamiętywać długie zależności pomiędzy tokenami [29].

Modele 4S

Po dodaniu omawianych usprawnień do tradycyjnego modelu SSM (dyskretyzacja, macierz HiPPO) nowo powstały model otrzymał nazwę *Structured State Space for Sequences (4S)*. Tego rodzaju modele pozwalają na śledzenie długich zależności w sekwencji dzięki macierzy HiPPO, a możliwość ich reprezentacji w postaci rekurencyjnej i konwolucyjnej rozwiązuje problemy między innymi złożoności obliczeniowej treningu oraz infrencji modelu.

2.6.2. Mamba

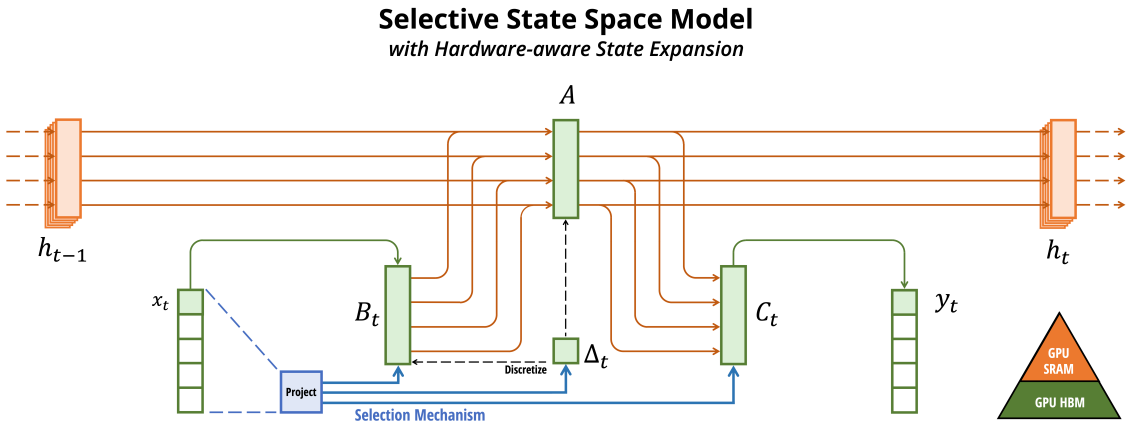
Model Mamba jest architekturą, która bazuje na modelach 4S, jednak wprowadza dodatkowe usprawnienia takie jak:

1. algorytm selektywnego skanowania (*selective scan*) pozwala skutecznie skupiać się na istotnych danych wejściowych i ignorować te nieistotne. Zdolność ta umożliwia lepszą obsługę zależności dalekiego zasięgu i filtrowanie szumów, co stanowi znaczną poprawę w stosunku do standardowych transformatorów, które traktują wszystkie dane wejściowe jako tak samo ważne.

2. algorytm *hardware-aware*, który pozwala uniknąć konieczności wykonywania rozległych operacji wejścia/wyjścia pomiędzy różnymi poziomami hierarchii pamięci GPU, co prowadzi do znacznej poprawy szybkości

Porównując grafiki 2.15 oraz 2.14 można zobaczyć wiele podobieństw. Główne elementy pojawiają się w obu modelach, jednak w przypadku Mamby jest jasno zaznaczony jest mechanizm selekcji oraz dyskretyzacja modelu oznaczona jako Δ_t. Model Mamba jest *hardware aware* co oznacza, że jest świadomy i zoptymalizowany pod kątem specyficznych właściwości i możliwości kart graficznych (GPU), na których jest on typowo uruchamiany i trenowany. Sekcje pamięci, na których są zaalokowane konkretne części modelu, na grafice są zaznaczone kolorem pomarańczowym i zielonym.

W przypadku zadań syntetycznych, takich jak kopiowanie sekwencji, Mamba nie tylko z łatwością rozwiązuje te zadania, ale może również ekstrapolować to rozwiązanie na bardzo długie sekwencje (ponad milion tokenów), gdzie ich długość znacząco przewyższa tą, na której model został wytrenowany. Mamba osiąga wydajność jakości transformatora w zadaniach modelowania języka. Zapewnia również pięciokrotnie szybszy czas generowania w



Rysunek 2.15.: Schemat modelu Mamba.

porównaniu do transformatorów o podobnej wielkości i dorównuje wydajnością transformatorom dwukrotnie większym w testach porównawczych takich jak *LAMBDA*, *Pile* lub *PIQA* [30].

3. Przeprowadzenie eksperymentów

W celu oceny skuteczności modeli generowania muzyki, przeprowadzono szereg eksperymentów, w których porównano modele transformerowe oraz Mamba. Eksperymenty te były przeprowadzone na plikach muzycznych w formatach MIDI i ABC, a modele były trenowane w dwóch wariantach wielkościowych: z około 60 milionami parametrów oraz z 6 milionami parametrów. Celem tych badań było zrozumienie, jak różne podejścia i rozmiary modeli wpływają na jakość generowanej muzyki.

3.1. Przygotowanie danych

Pliki w formacie ABC, dzięki swojej tekstowej naturze, nie wymagały skomplikowanego przetwarzania wstępnego. Format ABC zapisuje nuty i inne informacje muzyczne w postaci tekstu, co idealnie pasuje do modeli przystosowanych do pracy z sekwencjami tekstowymi.

Do tokenizacji plików MIDI zastosowano algorytm REMI o którym była mowa w rozdziale 2.3.3. Przekształca on dane muzyczne z formatu MIDI w sekwencje tokenów, które mogą być interpretowane przez modele.

Aby jeszcze bardziej zredukować złożoność i objętość danych, zastosowano technikę *Byte Pair Encoding* (BPE). BPE jest algorytmem kompresji tekstu, który jest szeroko stosowany w przetwarzaniu języka naturalnego do redukcji liczby unikalnych tokenów w sekwencjach tekstowych [31]. W kontekście przetwarzania muzyki, BPE pomaga w optymalizacji danych wejściowych, co z kolei prowadzi do skrócenia sekwencji wejściowej modelu co przekłada sę na lepszej jakości predykcje [32].
Algorytm można opisać w następujących krokach:

1. Inicializacja: Każdy unikalny token (w kontekście muzyki będzie to nuta, pauza, zmiana tempa itp.) jest traktowany jako pojedynczy symbol.

2. Zliczanie par: Algorytm identyfikuje najczęściej występujące pary sąsiadujących tokenów w sekwencji.

3. Łączenie par: Najczęściej występujące pary są łączone w jeden nowy symbol, redukując tym samym liczbę symboli w sekwencji.

4. Powtarzanie procesu: Kroki 2 i 3 są powtarzane, aż do osiągnięcia zdefiniowanego rozmiaru słownika *(vocab size)*.

Aby zilustrować działanie algorytmu BPE, można rozważyć prostą sekwencję liter `abcabcab`, na której zostanie przeprowadzony ten algorytm. Na początku należy zliczyć ilość par oraz zidentyfikować te najczęściej powtarzające się. W tym przypadku jest to para `ab`. Zostanie ona zamienione na nowy token np. `X`. Nowa sekwencja `XcXcX` zostaje poddana ponownej analizie z czego w tym przypadku wynika, że najczęstszą parą jest `Xc`. Zostanie ona zamieniona na nowy token `Y` z czego powstanie sekwencja `YYX`. W przypadku tekstu, tokenami na podstawie których obliczane są pary są kolejny bajty tekstu. W kontekście ztokenizowanego zbioru danych MIDI pary będą obliczane na podstawie tokenów takich jak: `Pitch_C3`, `Velocity_70`, `Duration_1.0`. Jeśli tego rodzaju sekwencja będzie występowała wystarczająco często, to przy użyciu BPE zostanie ona zastąpiona jednym tokenem.

W przeprowadzonych eksperymentach ustawiono rozmiar słownika na 20,000, co zapewniło odpowiednią równowagę między kompresją a zachowaniem istotnych informacji muzycznych.

3.2. Trening modeli

HuggingFace to wiodąca firma w dziedzinie przetwarzania języka naturalnego i uczenia maszynowego, znana z opracowywania narzędzi i bibliotek ułatwiających tworzenie, wdrażanie i udostępnianie modeli uczenia maszynowego. Jednym z jej flagowych produktów jest bardzo popularna biblioteka typu *open-source* Transformers, która zapewnia łatwy dostęp do szerokiej gamy wstępnie wytrenowanych modeli dla różnych zadań NLP, takich jak klasyfikacja tekstu, tłumaczenie, podsumowywanie i odpowiadanie na pytania. Biblioteka obsługuje modele takie jak BERT, GPT, T5 i wiele innych, oferując zarówno podstawową architekturę, jak i wstępnie wytrenowane wagi, co znacznie skraca czas i zasoby obliczeniowe potrzebne do dostrajania modeli. HuggingFace udostępnia wysokopoziomowe API pozwalające na tworzenie oraz trening modeli przy pomocy klas takich jak `Trainer`, `AutoModelForCausalLM` czy `ModelConfig`.

Jako reprezentant modelu transformerowego został wybrany model *GPT-2* [33]. Model został stworzony przez OpenAI i jest to obecnie ich największy model udostepniony jako *open source*. Jest oparty o architekturę *decoder-only*.

Celem trenigu modeli typu *Causal Language Model (CausalLM)* jest jak najwierniejsze przewidzenie następnego tokena w sekwencji, biorąc pod uwagę poprzednie tokeny. Podczas szkolenia tokeny wejściowe są wprowadzane do modelu, a model przewiduje rozkład prawdopodobieństwa nad wektorem tokenów, a nasępnie z takim prawdopodobieństwem losowany jest kolenjy token. Funkcja straty (*loss*) jest obliczana na podstawie przewidywań modelu i rzeczywistych tokenów docelowych, które są tokenami wejściowymi przesuniętymi o jedną pozycję. Typowo używaną funkcja jest funkcja entropii krzyżowej *cross-entropy loss*, która mierzy różnicę między dwoma rozkładami prawdopodobieństwa: prawdziwym rozkładem (rozkładem docelowym) a rozkładem przewidywanym przez model.

W poniższych tabelach zostały pokazane konfiguracje modeli jak i liczba parametrów

trenowanych modeli:

Tabela 3.1.: Parametry modelu GPT-2

Parametr	model 6M	model 60M
`n_embed`	128	512
`n_layer`	2	6
`n_head`	1	4
`n_inner`	256	2048

Tabela 3.2.: Parametry modelu Mamba

Parametr	model 6M	model 60M
`hidden_size`	128	512
`state_size`	4	8
`num_hidden_layers`	8	12

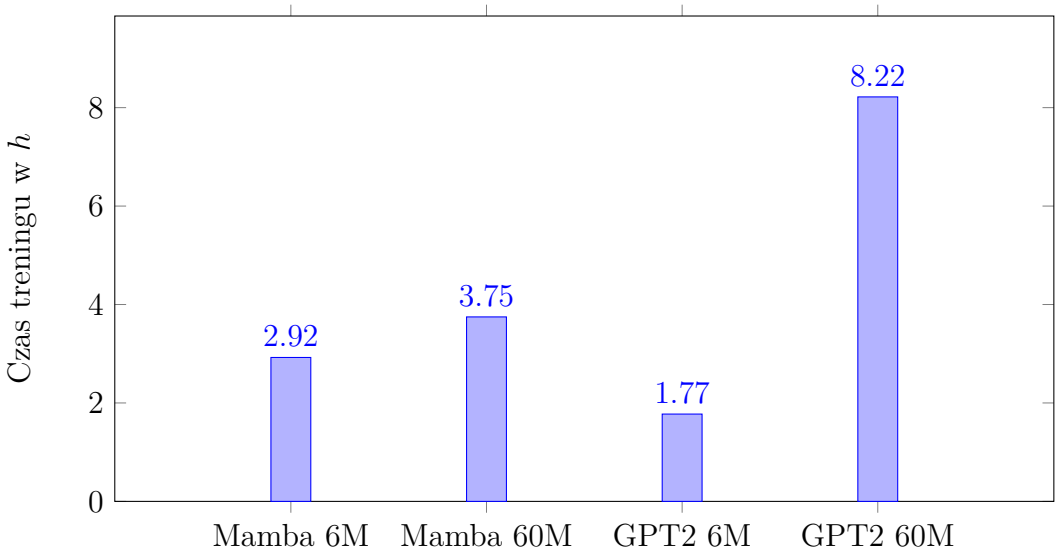
Parametry modelu GPT-2:

- `n_embed` - wymiarowość osadzeń oraz stanów ukrytych
- `n_layer` - liczba warstw ukrytych w transformerze
- `n_head` - liczba "głów" w warstwach *multi-headed attention* trasformera
- `n_inner` - wymiar środkowych warstw typu *feed-foreward*

Parametry modeli Mamba:

- `hidden_size` - wymiarowość osadzeń oraz stanów ukrytych
- `state_size` - rozmair stanów w modelu przestrzeni stanów
- `num_hidden_layers` - ilość warstw ukrytych modelu

Trening modeli odbywał się na karcie graficznej NVIDIA Tesla A100. Czas trenigu modeli przedstawia wykres 3.1. Każdy model były trenowane przez około 800 tysiecy kroków, jednak ich dokłada liczba różniła się pomiędzy eksperymentami. Aby lepiej przedstawić różnice w czasie trenigu na wykresie przedstawiono tylko czas pierwszych 100 tysiecy. Modele trenowane na plikach ABC, wymagały zdecydowanie krótszego czasu trenigu, co wynikało z mniejszej ilości unikatowych tokenów w stosunku do reprezentacji MIDI.

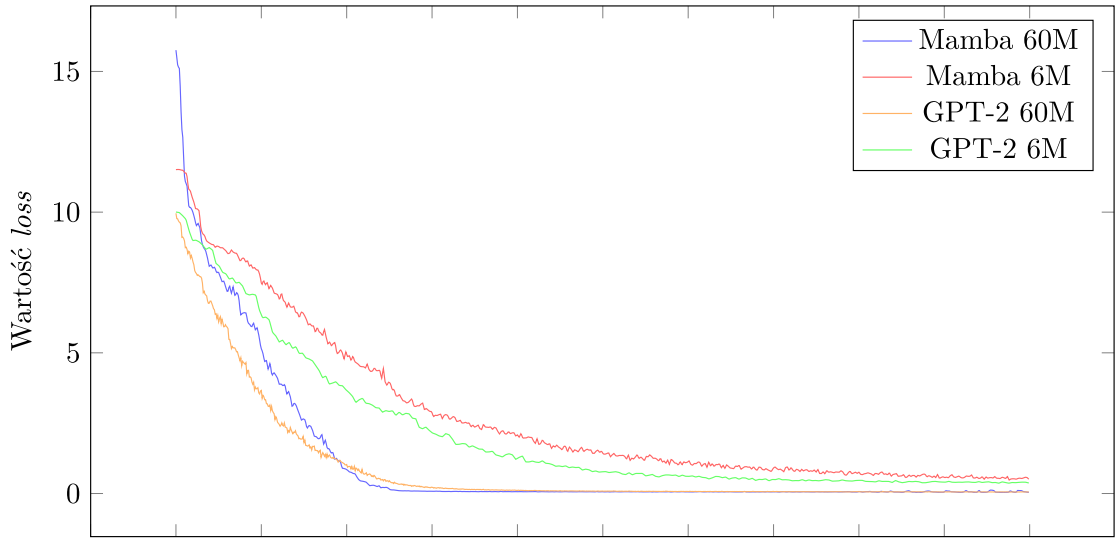


Rysunek 3.1.: Czas trenigu 100 tysięcy kroków modeli.

Porównując czasy treningu obu architektur można dojść do następujących wniosków. Dość oczywistą obserwacją jest fakt, że trening modeli mniejszych jest krótszy niż tych większych. Pomimo tego, że model Mamba powinien trenować się szybciej niż modele transformerowe, nie jest to prawdą dla mniejszego modelu. Model transformerowy GPT2 6M trenował się o 39.4% szybciej niż jego odpowiednik w konkurencyjnej architekturze Mamba 6M. Różnica w szybkości treningu jest zdecydowanie bardziej widoczna w dużych modelach, gdzie model transformerowy GPT2 60M trenował się prawie 120% dłużej niż model Mamba 60M o tej samej liczbie parametrów. Na podstawie tych obserwacji można stwierdzić, że w kontekście dużych modeli, architektura Mamba jest bardziej efektywna pod względem czasu treningu niż architektura GPT2.

Na grafice 3.2 przedstawiono *loss* trenigu modeli. W celu dokonania lepszej wizualizacji ograniczono ilość kroków do 250 tysięcy. Modele z większą liczbą parametrów (60M) szybciej osiągnęły niższą funkcję straty w porównaniu do modeli z mniejszą liczbą parametrów (6M). Dzieje się tak, ponieważ większe modele mają lepszą zdolność do reprezentowania skomplikowanych zależności w danych przy pomocy swoich parametrów dzięki czemu mogą lepiej dopasować się do zbioru treningowych, co prowadzi do szybszej redukcji funkcji straty. Pomimo tego, że duże modele osiągnęły niższe wartości szybciej, wszystkie modele wytrenowały się do podobnego poziomu.

Na grafice 3.3 przedstawiono czas generacji konkretnej liczby tokenów przez modele. Najszybciej działa najmniejszy model GPT2, gdzie dla każdej wartości generuje sekwencje poniżej jednej sekundy. Porównując duże modele, widać że Mamba radzi sobie lepiej, szczególnie dla dłuższych sekwenji, działając prawie dwa razy szybciej. Dużym zasokoczeniem jest gwał-



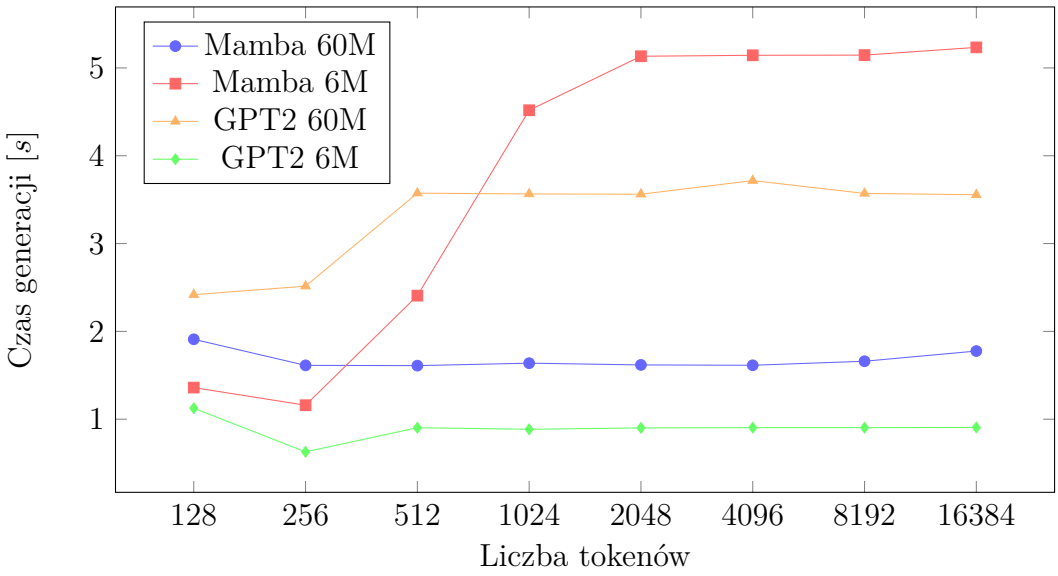
Rysunek 3.2.: Wykresy treningu pierwszych 250 tysięcy kroków modeli.

towny wzrost czasu dla modelu Mamba 6M.

Przyglądając mały oraz duzy model Mamby, można zauważyć że różnica między nimi nie jest tak duża jak w przypadku modeli GPT2. Może wynikać to z tego, że model ten w swojej architekturze jest zoptymializowany pod bycie bardzo dużym. Najmniejszy model jaki został wytrenowany i udostępniony przez twórców miał 128 milionów parametrów.

3.3. Prezentacja otrzymanych wyników

Przedstawienie muzyki w pracy stanowi wyzwanie w porównaniu na przykład do wizualnych form sztuki, takich jak obrazy. Muzyka jest medium audytywnym, które oddziałuje na słuch i emocje poprzez dźwięki i rytm, co sprawia, że jej pełne doświadczenie jest trudne do przekazania za pomocą samego tekstu. Opisy muzyczne, notacja i partytury mogą dostarczyć informacji o strukturze i elementach utworu, ale nie oddają one w pełni jego brzmienia i atmosfery. W przeciwieństwie do obrazów, które można bezpośrednio zobaczyć i ocenić na papierze, muzyka wymaga aktywnego odtwarzania, by w pełni ją zrozumieć i docenić. W pracy zostaną przedstawione generacje w dwojaki sposób. W zależności od tego, czy model był wytrenowany na plikach MIDI lub notacji ABC, będzie przedstawiony wynik modelu w formie tekstowej lub *piano roll*. Dodatkowo do każdego z tych przykładów zostanie dodana wygenerowana partytura, jednak należy pamiętać, że w związku z tym, że pliki MIDI są zapisem odegrania muzyki a nie samej jej notacji, reprezentacja plików MIDI na pięciolinii potrafi nie być idealna co jest spowodowane możliwymi nieregularnościami w rytmie.



Rysunek 3.3.: Czas generacji tokenów przez modele.

Na grafice 3.4 przedstawiono wynik modelu GPT-2 60M w wersji ABC. Część oznaczoną kolorem niebieskim należy traktowac jako sekwencje wejściową dla modelu, na której podstawie model generuje kolejne tokeny. Dodatkowo należy zwrócić uwagę na podane kody kontrolne `S`, `B` oraz `E`. Reprezentacje otrzymanej muzyki zapisanej na pięciolinii przedstawia grafika 3.5.

Należy zwrócić uwagę, że model został poproszony o wygenerowanie melodii w tonacji C-dur (`K:C`), jednak jak można zaobserwować na obu reprezentacjach, model zaczął dodawać do dźwięków `c` krzyzyki (`^c` w notacji ABC). Prawdopodobnie wzięło się to z faktu, że podane jako początek sekwencji dźwięki `d`, nie są typowym rozpoczęciem melodii w tonacji C-dur. Model oczekiwał podania toniki (pierwszego stopnia skali) ponieważ takie fragmenty są najbardziej popularne i dominowały zbiór uczący.

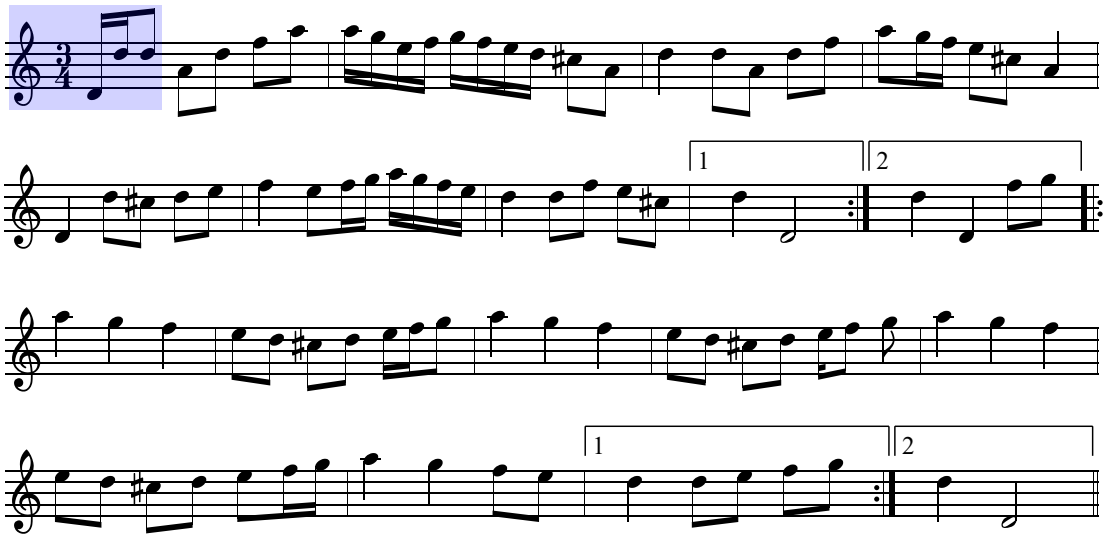
Generacja plików MIDI jest bardziej skomplikowanym procesem w przeciwieństwie to tego z plików ABC, gdzie wystarczy przy pomocy klawiatury wprowadzić odpowiednie nuty. W tym przypadku, predykcje są powiązane z tokenizatorem, który zamienia konkretne dźwięki na tokeny, które są rozumiane przez model. Najprostrzym sposobem, jest stworznie początku sekwencji przy pomocy dowolnego edytora plików MIDI, a następnie otwarcie ich przy pomocy odpowiedniej bilbioteki oraz zamianę na tokeny przy pomocy tego samego tokenizatora, który został użyty podczas treningu modelu.

Przykładowe predykcje stworzone przy pomocy modelu Mamba 6M można zobaczyć na grafikach 3.6 oraz 3.7.

Różnicą, która jest widocza pomiędzy precykcjami 3.6 oraz 3.5 jest sposób generacji po-

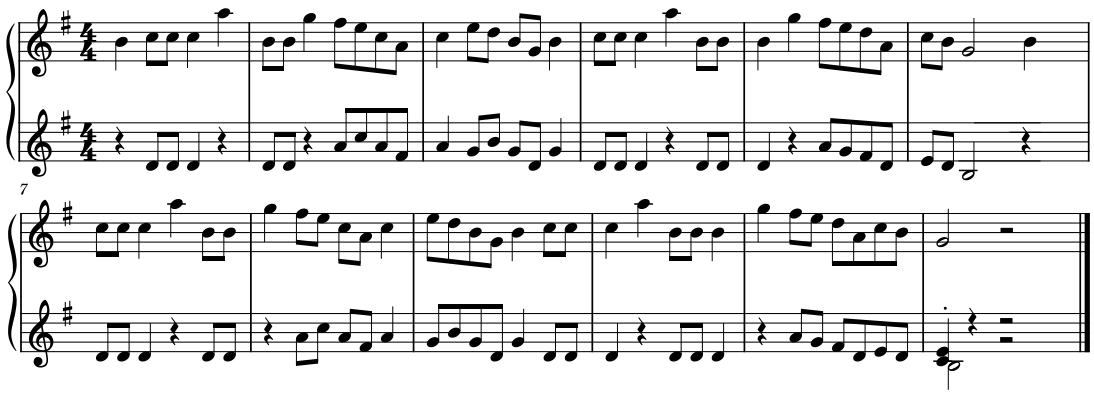
```
X:1
S:2
B:9
E:4
L:1/8
M:3/4
K:C
D/2d/2d Ad fa | a/g/e/f/ g/f/e/d/ ^cA | d2 dA df | ag/f/ e^c A2 | D2 d^c
de | f2 ef/g/ a/g/f/e/ | d2 df e^c |1 d2 D4 :| 2 d2 D2 fg |: a2 g2 f2 | ed
^cd e/f/g | a2 g2 f2 | ed ^cd e/f g | a2 g2 f2 | ed ^cd ef/g/ | a2 g2 fe
|1 d2 de fg :|2 d2 D4 ||
```

Rysunek 3.4.: Wynik modelu.

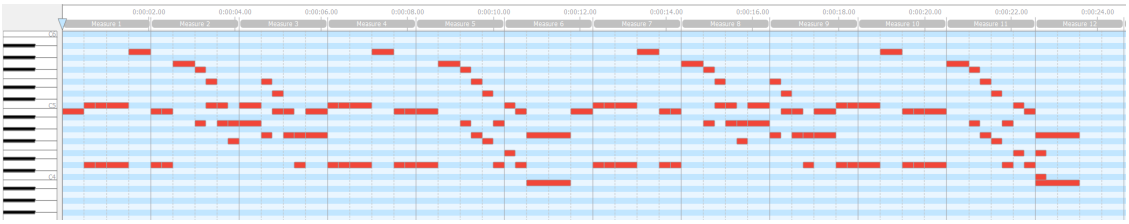


Rysunek 3.5.: Wygenerowana melodia ABC.

wtórzeń w muzyce. W notacji ABC odbywa się to przy pomocy znaków notacji muzycznej, które mówią o tym jakie fragmenty mają zostać powtórzone. W przypadku predykcji plików MIDI, w drugiej linijce wygenerowanego fragmentu występuje powtórzenie pierwszej linijki (poza ostatnim taktem), które mogłby zostać zapisane na pięciolini przy pomocy odpowiednich znaków. Modelowi udało się zapamiętać i poprawnie odtworzyć wcześniej wygenerowaną sekwencję, co nie zawsze może się wydarzyć.



Rysunek 3.6.: Wygenerowana melodia MIDI zapisana w notacji muzycznej.



Rysunek 3.7.: Wygenerowana melodia MIDI.

W pracy zostały przedstawione fragmenty, które zostały wybrane przez autora jako wyjątkowo dobre i brzmiące wiarygodnie, jednak nie wszystkie predykcje należą do tej grupy. Część predykcji jest nieudanymi próbami modelu, do generacji muzyki, która zwyczajnie nie brzmi dobrze. Część fragmentów kończy się za szybko, ponieważ model dodał w nich znak końca sekwencji w niepoprawnym miejscu lub fragmenty zawierają zdecydowanie zbyt dużą liczbę pauz następujących po sobie, co wprowadza dość niezręczną ciszę w wygenerowanych fragmentach. Zdecydowanie większa część udanych generacji modelu powstała na podstawie plików ABC, co jest następstwem prostrzej stuktury tych plików. Również w przypadku tego formatu, co jest szczególnie widoczenie, kiedy model jest niedotrenowany jest on w stanie wygenerować sekwencje, które nie są zgodne z notacją, przez co nie są one możliwe do odsłuchania w dostępnym oprogramowaniu. Takie fragmenty, należy pomijać, lub próbować poprawiać je ręcznie.

3.4. Sposoby oceny wyników

Celem porównania otrzymanych wyników z dowolnych modeli, jest ocena jakości ich pracy. W kontekście modeli dyskryminujących (ang. *discriminative model*), na przykład klasyfikatorów bądź modeli regresyjnych, ocena ich jakości jest dość prosta, ponieważ istnieje predefiniowana, prawdziwa wartość w zbiorze testowym, którą model próbuje przewidzieć. W takim przypadku ewaluacja jakości modelu to porównanie prawdziwych danych, z tymi przewidzianymi przez model. Porównanie odbywa się przez policzenie metryk np. dla modelu klasyfikującego celności, precyzji, *F1-score* lub dla modelu regresyjnego MSE lub R^2. W przypadku modeli generacyjnych celem treningu jest jak najlepsza aproksymacja rozkładu prawdopodobieństwa danych $P(X_{data})$ lub w przypadku danych oznaczonych łączny rozkład $P(X, Y)$. W przypadku dużej wymiarowości danych, obliczenie obiektywnych metryk takich jak *log-likelihood* lub dywergencja Kullbacka-Leiblera (*KLD*) często jest nieobliczalne. Dla człowieka weryfikacja wyników modeli generacyjnych takich jak *text-to-speech* lub *text-to-image* jest trywialnym zadaniem, jednak nie jest to oczywiste zadanie algorytmiczne. Często odwołuje się do subiektywnych metryk takich jak (*MOS ang. mean opinion score*), które oblicza się jako średnią opinię np. w skali od 1-5 braną z zazwyczaj niewielkiej grupy ludzi. Niestety metryka ta często nie jest wiarygodna ze względu na jej subiektywnośś oraz niewielką próbę badawczą. Aby wyeliminować te wady serwisy takie jak *HuggingFace* udostępniają narzędzia dla członków społeczności, które pozwalają na ranking modeli, z nadzieją że zgodnie z prawem wielkich liczb, przy wystarczającej liczbie odpowiedzi, uda się otrzymać w miarę obiektywną ocenę. Przykładowym narzędziem tego rodzaju jest *The TTS Arena*[34], która pozwala na ranking modeli *text-to-speech*.

W przypadku muzyki, istnieje możliwość aby zweryfikować poprawność wygenerowanych sekwencji odwołując się do teorii oraz harmonii muzyki. Istnieje kilka narzędzi takich jak *Chordify*, *Hooktheory* lub *Sibelius*, które pozwalają na analizę harmoniczną utworów, dzieki czemu autorzy mogą w prosty sposób analizować i dobierać progresję danej melodii. Niestety większość takich narzędzi jest płatna i nie pozwala na zautomatyzowaną analizę wielu plików. Dodatkowym problemem jest w analizie harmonicznej, szczególnie prowadzonej przez algorytmy, jest rozróżnienie harmonii wertykalnej oraz horyzontalnej. Rozróżnienie to zostało wytłumaczone przez Jacoba Colliera, kilkukrotnego laureata nagród *Grammy*, którego zdaniem akord, który nawet zagrany sam brzmi niepoprawnie, w odpowiednim kontekście i przez odpowiednią progresję w kolejnych fragmentach muzyki, może mieć nadany sens, przez co cała sekwencja nabiera muzycznego piękna [35].

Brak obiektywnych metryk, którymi można się posłużyć podczas ewaluacji modelu jest dodatkowym problemem w momencie *fine-tunigu* modelu. Ciężko jest wybrać odpowiednie rozwiązanie architektoniczne lub odpowiednie hiperparametry, kiedy jedyną miarą jakości wygenerowanych danych jest subiektywna opinia programisty lub grupy osób wybranych jako wyrocznia.

3.4.1. Testowanie melodii innymi LLM-ami

Aby zweryfikować jakość predykcji, można zastosować podejście polegające na wytrenowaniu innego modelu na danych muzycznych. Tym sposobem powstanie model, który będzie w stanie ocenić oraz udzielić konstruktywnej krytyki na temat przedstawionego mu fragmentu muzycznego.

ChatMusician

ChatMusician to otwarty model językowy, który integruje zdolności muzyczne [36]. Model ten został zaprojektowany na bazie modelu LLaMA2 z 7 miliardów parametrów, jest trenowany i dostrajany z wykorzystaniem notacji muzycznej ABC, traktując muzykę jako drugi język. Dzięki temu ChatMusician jest w stanie rozumieć i generować muzykę za pomocą tekstowego tokenizatora, bez konieczności stosowania zewnętrznych struktur neuronowych czy tokenizatorów stworzonych specjalnie dla notacji muzycznej. *ChatMusician* może komponować dobrze zorganizowane, pełnometrażowe utwory muzyczne na podstawie tekstu, akordów, melodii, motywów i form muzycznych, przewyższając wyniki modelu *GPT-4*. Model ten osiąga lepsze wyniki niż *LLaMA2* i *GPT-3.5* w zadaniach związanych z rozumieniem teorii muzyki, co potwierdzono w benchmarku *MusicTheoryBench*, zaprojektowanym do oceny zdolności modeli do rozumienia i rozumowania muzycznego. Model został udostepniony dla użytku publicznego na platformie *HuggingFace*.

Model jest potencjalnym rozwiązaniem problemu z oceną wygenerowanej muzyki. Będąc wytrenowanym na prawdziwych fragmentach muzycznych z całego świata i wielu kultur oraz tekstu zawierającego teorię muzyki, powinien w dość obiektywny sposób oceniać fragmenty melodii. Pytanie zadane do modelu brzmi następująco: “Wygenerowałem muzykę w notacji ABC. Czy mógłbyś ją przeanalizować i powiedzieć, czy brzmi naturalnie? W szczególności chciałbym prosić o opinię na temat ogólnej muzykalności, poczucia harmonii i tego, czy melodia i progresje akordów są spójne i przyjemne. Oto jej notacja:”. W związku z tym, że model działa tylko dla języka angielskiego należy przetłumaczyć pytanie na ten własnie język.

Zapytanie o wygenerowany fragment zapisany na partyturze 3.5 przedstawiono na 3.8.

Model dość przychylnie spojrzał na ten wygenerowany fragment, jednak jego odpowiedzi nie są specyficzne o czym można się przekonać czytając odpowiedź na grafice 3.9. Nie podejmuje on analizy konkretach fragmentów i dodatkowo udziela informacji o użytych wielu instrumentach, co nie występuje w przedstawionym mu fragmencie. W kontekście harmonii używa słów takich jak *well-crafted*, *coherent*, *pleasing*. Słowa te są bardzo ogólne i nie stanowią dobrej analizy fragmentu muzycznego. Dodatkowo w opinii udzielonej przez model występują powtorzenia, które dodatkowo zmniejszają wiarygodność oceny. DOdatkowo model nie udzielił żadnej negatywnej odpowiedzi lub nie zasugerował możliwości zmiany.

Aby upewnić się czy udzielania wyłącznie pozytywnych informacji nie jest domyślnym

```
PROMPT: I have some music in ABC notation that I generated. Couldyou
analyze it and tell me if it sounds natural? Specifically, I would like
your feedback on the overall musicality, the sense of harmony, and whether
the melody and chord progressions are coherent and pleasing. Here is the
ABC notation of the music:
X:1
L:1/8
M:3/4
K:C
D/2d/2d Ad fa | a/g/e/f/ g/f/e/d/ ^cA | d2 dA df | ag/f/ e^c A2 |
D2 d^c de | f2 ef/g/ a/g/f/e/ | d2 df e^c |1 d2 D4 :|2 d2 D2 fg |:
a2 g2 f2 | ed ^cd e/f/g | a2 g2 f2 | ed ^cd e/f g | a2 g2 f2 |
ed ^cd ef/g/ | a2 g2 fe |1 d2 de fg :|2 d2 D4 ||
```

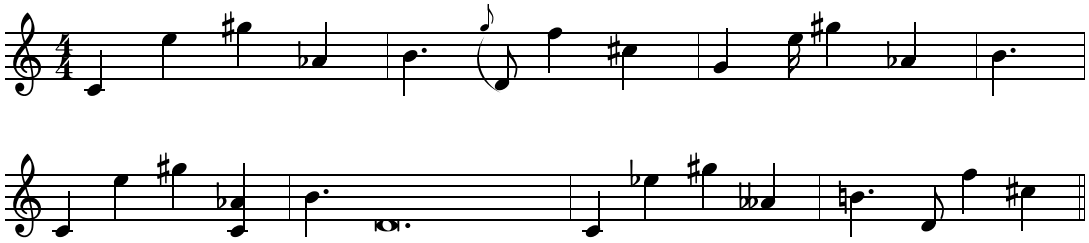
Rysunek 3.8.: Pytanie dla modelu ChatMusician.

```
ANSWER: Overall Musicality: 1. The piece has a pleasant and flowing rhythm
with a steady tempo. It features a mix of different instruments andvoices,
creating an interesting texture. The melodies are catchy and memorable,
making the piece enjoyable to listen to. 2. Sense of Harmony: The
harmonies in this piece are well-crafted and create a rich and full sound.
The use of various chords and progressions adds depth and complexity to
the music. The harmonic choices are appropriate for the style and genre of
the piece. 3. Melody and Chord Progressions: The melody and chord
progressions in this piece are coherent and pleasing. They follow a
logical progression and fit together well. The melodic lines are
well-written and expressive, adding to the emotional impact of the piece.
The chord progressions provide a solid foundation for the melodies and add
interest and variety to the composition.In conclusion, this piece of music
has a strong sense of musicality, harmony, and coherence. The melodies and
chord progressions are well-constructed and contribute to the overall
appeal of the piece. It is a successful example of its genre and style.
```

Rysunek 3.9.: Odpowiedź modelu ChatMusician.

zachowaniem modelu należy poprosić model o ocenę fragmentu, który został specjalnie przygotowany, aby nie brzmiał dobrze dla ludzkiego ucha. Zapis tego fragmentu został zapisany

na grafice 3.10. Dodatkowo, pomimo tego że zapis jest poprawny w notacji ABC, niektóre takty zawierają więcej lub mniej dźwięków niż jest to przewidziane w metrum $\frac{4}{4}$. Odpowiedź modelu przedstawiono we fragmencie 3.11.



Rysunek 3.10.: Specjalnie stworzony niepoprawny fragment muzyki.

```
ANSWER: Overall Musicality: The piece has a pleasant and flowing
quality to it, with a mix of different rhythms and tempos. The use of
dynamics and articulations adds depth and interest to the composition.
2. Sense of Harmony: The harmonies used in this piece are mostly consonant
and create a sense of stability and balance. There are occasional
dissonances that add tension and color to the music. 3. Melody and Chord
Progressions: The melodies in this piece are well-crafted and follow a
logical progression. They are easy to sing or play and have a clear
sense of direction. The chord progressions are also coherent and support
the melodies effectively. In conclusion, this piece has a good overall
musicality, sense of harmony, and coherence between its melody and
chord progressions. It is an enjoyable and engaging composition.
```

Rysunek 3.11.: Odpowiedź modelu na fałszywy fragment.

Obie opinie są bardzo do siebie podobne. Okazuje się, że model nie jest w stanie dobrze rozpoznawać to, czy muzyka jest rzeczywiście przyjemna dla ucha czy nie. Nie wychwycił problemów z rytmem, które nie jest jednakowe dla każdego taktu oraz nie wyraził żadnej konstruktywnej krytyki. Niestety model *ChatMusician* nie jest dobrym rozwiązania jeśli chodzi o ocenianie muzyki.

GPT-4

Aby porównać możliwości oceny modeli LLM nalezy użyć silniejszego modelu. *GPT-4* to model, który obecnie najlepiej wypada na praktycznie każdym benchmarku [37]. W porównaniu do swojego poprzednika, *GPT-3*, charakteryzuje się większą liczbą parametrów, co pozwala mu na lepsze rozumienie i generowanie skomplikowanych tekstów. GPT-4 potrafi również wykazywać większą spójność i koherencję w dłuższych wypowiedziach, lepiej interpretować kontekst oraz odpowiadać na bardziej złożone pytania. W związku z tym, że model nie jest *open source* jego dokłada konfiguracja nie jest znana, jednak hipotetyzuje się, że ma 1.67 trylionów parametrów [38]. Nie był on trenowany bezpośrednio na danych muzycznych, jednak przez swój rozmiar, który jest potencjalnie aż o ponad 2000 razy większy, powinien posiadać przynajmniej podstawową znajomość muzyczną. Wadą tego rozwiązania jest fakt, że *GPT-4* jest dostępny jedynie po wykupieniu subskrypcji na stronie OpenAI, która kosztuje 20$ na miesiąc.

Do modelu zostało przekazane takie samo pytanie jak w wyżej opisanym przypadku *ChatMusician*.

```
To evaluate the musicality and harmony of your ABC notation, I will provide an analysis based on the key, melody, rhythm, and chord progressions.
Here is a detailed breakdown of your piece:

### Key and Meter
- **Key:** C major
- **Meter:** 3/4 (Waltz time)

### Melody and Harmony
1. **Opening Phrase:**
   - `D/2d/2d Ad fa | a/g/e/f/ g/f/e/d/ ^cA`
     - This phrase introduces a mix of stepwise motion and leaps, giving a dynamic start. The alternation between the upper and lower notes
     creates a call-and-response effect. The use of the raised 4th degree (^c) adds a touch of chromaticism.

2. **Middle Phrase:**
   - `d2 dA df | ag/f/ e^c A2`
     - The repetition of 'd' with variations in intervals suggests a thematic consistency. The leap to 'f' and back to 'd' maintains interest.
     The movement towards 'A' anchors the phrase back to the tonic's dominant, providing harmonic stability.

3. **Transition:**
   - `D2 d^c de | f2 ef/g/ a/g/f/e/`
     - The use of the D minor chord (D-F-A) transitioning to E minor (E-G-B) and then back to fragments of D minor adds tension and resolution.

4. **Closing Phrase:**
   - `d2 df e^c |1 d2 D4 :|2 d2 D2 fg`
     - This phrase returns to the tonic D and introduces a cadential figure (D-F-E-^C). The first ending resolves to the tonic D major chord,
     while the second ending moves into a continuation with 'fg', creating a sense of forward motion.

### Second Part
1. **Opening Phrase:**
   - `a2 g2 f2 | ed ^cd e/f/g`
     - This section starts with a descending sequence, providing contrast to the first part. The melodic minor descent with a chromatic touch
     (e-^c-d) maintains interest.

2. **Middle Phrase:**
   - `a2 g2 f2 | ed ^cd e/f g`
     - The repetition reinforces the theme while subtle variations in articulation (slurs) add expressiveness.

3. **Transition and Closing:**
   - `a2 g2 f2 | ed ^cd ef/g/ | a2 g2 fe`
     - The alternation between stepwise motion and leaps ensures coherence and variety. The repeated 'a-g-f' figure creates familiarity, while
     'e-^c-d ef/g/' keeps it fresh.

4. **Final Cadence:**
   - `1 d2 de fg :|2 d2 D4 ||`
     - The final cadence resolves nicely with the repeating 'de fg' figure leading to the tonic 'D' again, ensuring a satisfying conclusion.

### Overall Musicality
- **Melodic Coherence:** The melody has a clear direction with a good mix of stepwise motion and leaps, maintaining interest.
```

```
- **Harmony:** The harmony is mostly implied through the melodic line with occasional chromatic touches adding tension and resolution. The
frequent returns to tonic and dominant notes (C and G) provide a sense of harmonic stability.
- **Rhythmic Flow:** The piece maintains a consistent waltz rhythm (3/4 time), which gives it a dance-like quality, adding to its musicality.

### Conclusion
The piece sounds natural and coherent, with a pleasing melody and harmonic progression. The use of chromatic notes and variations in articulation
adds interest and depth. It effectively balances repetition with variation, ensuring both familiarity and novelty for the listener.
```

Jak można zauważyć ocena fragmentu muzyki przez *GPT-4* jest dość długa oraz bardzo dokładana. Podczas analizy rozbija muzyke na główne fragmenty, a następnie każdy z tych fragmentów poddaje szczegółowej analizie rozdzielając je dalej na początek, środek, przejście i zakończenia. Dla każdej z wyróżnionych części udziela konkretnej analizy harmonicznej, pokazując progresie akordów używając technicznego słownictwa jak dominanta lub tonika. Rozpoznaje kierunkowość muzyki, czyli to czy melodia jest opadająca czy rosnąca. Model jest w stanie dać opinię na temat rytmu oraz tego w jaki sposób melodia komponuje się z rytmem. Na końcu wypowiedzi udziela podsumowania. Dodatkową zaletą, jest fakt, że odpowiedź jest udzielona w formacie *markdown* przez co tekst jest podzielony na logiczne fragmenty, a kluczowe słowa są oznaczone pogrubioną czcionką.

Aby eksperyment był wiarygodny nalezy również przedstawić modelowi fragment "złej" muzyki.

```
Based on the ABC notation provided, here is an analysis of the music in terms of musicality, harmony, and melody coherence:

### Overall Musicality
The notation seems to follow a structure that could be pleasing. The use of different note values and ornamentations like the grace note `{g}`
add interest to the melody. The notation implies a mix of staccato and legato sections, which can contribute to a dynamic musical flow.

### Harmony
The harmony is somewhat implied rather than explicitly stated. The melody suggests movement between different tonal centers, but the use of
accidentals (like `^g`, `_A`, `^c`, etc.) makes the harmony more complex. It moves away from a strictly diatonic C major scale, introducing
chromaticism which can add color but might sound unusual without proper context or accompanying chords.

### Melody Coherence
The melody uses a variety of intervals and note values, which gives it a sense of motion. However, the frequent use of accidentals (`^g`, `_A`,
`^c`, `__A`, `=B`) can make it sound less coherent if not handled carefully. These accidentals can create tension and release, which is good for
musical interest but must be resolved correctly to sound pleasing.

### Specific Sections
- **Measure 1**: Starts with a simple C major outline but immediately introduces accidentals (`^g`, `_A`), which suggest movement towards
different tonal areas.
- **Measure 2**: The grace note `{g}` and the intervals used (`D1 f2 ^c2`) create a surprising twist.
- **Measure 3**: The use of `G2 e1/2 ^g2 _A2` suggests a continuation of the chromatic movement, making it intriguing.
- **Measure 4**: The abrupt stop with `B3` could serve as a point of tension or punctuation.
- **Measure 5**: A return to `C2 e2 ^g2 [_A2C]` reintroduces some stability but with the added complexity of accidentals.
- **Measure 6**: `B3 D22` maintains the movement and suggests a possible resolution.
- **Measure 7**: The phrase `C2 _e2 ^g2 __A2` is quite chromatic and could be disorienting without a clear harmonic context.
- **Measure 8**: The closing `=B3 D1 f2 ^c2` should ideally resolve the chromatic tension but continues to add accidentals, leaving a sense of
unresolved tension.

### Conclusion
The melody and chord progressions can be coherent and pleasing if the chromaticism and accidentals are used intentionally to create tension and
resolution. It may sound natural in a modern or experimental context but could be challenging in a more traditional tonal framework. To improve
musicality, ensure that the chromatic notes resolve in a way that feels intentional, and consider adding harmonic support to clarify the
underlying chord progressions.
```

W przypadku tego fragmentu muzycznego model w dość niepewny sposób udziela odpowiedzi. Często używa takich wyrażeń jak "struktura **może** być przyjemna", "...**może** wpływać na dynamikę", "**niespodziewany zwrot** muzyki". Sugeruje to że model rozumie dość nienaturalną strukturę przekazanej mu melodii. Dokonuje również analizy konkretnych taktów,

gdzie gość krytycznie wypowiada się na ich temat. Oferuje również konstruktywną krytykę, zauważając, że częste stosowanie akcydensów i fraz chromatycznych może wywoływać poczucie dezorientacji i nierozwiązanego napięcia. Gwałtowne zatrzymania i złożone interwały są doceniane za ich potencjał dodawania zainteresowania, ale są również podkreślane jako punkty, które mogłyby skorzystać z wyraźniejszego harmonicznego rozwiązania. Ogólnie rzecz ujmując modelowi udało się zwrócić uwagę na szczególne fragmenty w melodii i poddać je krytycznej analizie, która nie była wyłacznie przychylna.

Okazało się, że model *GPT-4*, pomimo tego że nie był specjalnie wyszkolony do tego celu, lepiej przeanalizował muzykę, wykazując zdolność do precyzyjnej identyfikacji i oceny harmonicznych struktur w utworach. Fragment, który był specjalnie przygotowany jako niepoprawny harmonicznie, został skrytykowany za brak spójności i niezgodność z klasycznymi zasadami harmonii. Natomiast model *ChatMusician*, który również przeanalizował te same fragmenty, przedstawił bardzo podobną ocenę dla obu, zarówno poprawnego, jak i niepoprawnego harmonicznie, co wskazywało na jego ogólną analizę, nieodróżniającą istotnych niuansów harmonicznych.

Istnieje technika nazwana *RLHF* (*reinforcement learning with human feedback*), która jest wykorzystywane w trenowaniu lub *fin-tuningu* modeli sztucznej inteligencji [39]. Polega ona na zbieraniu danych od ludzi na temat działania modelu i wykorzystaniu tych danych do poprawy jego wydajności. W praktyce wygląda to tak, że model generuje odpowiedzi na różne zapytania, a ludzie oceniają te odpowiedzi pod kątem ich poprawności, użyteczności, spójności czy innych kryteriów jakościowych. Na podstawie tych ocen model jest wzmacniany, aby generować coraz lepsze odpowiedzi w przyszłości. Przedstawione wyniki analizy modelu *GPT-4* mogą być brane pod uwagę jako dobra opinia na temat tego, czy dany wygenerowany fragment jest poprawny czy nie. Tym sposobem istnieje szansa na rozwój tego rodzaju sposobu dotrenowywania modeli generacyjnych. Problemem jest obecnie koszt używania modelu od OpenAI, który posiada dzienne limity, które co prawda można zwiększyć za określoną sumę. Niestety nie jest to obecnie dobrze skalujące się rozwiązanie szczególnie dla użytku niekomercyjnego. Mimo wszytko jest to możliwy sposób *fine-tuningu* modeli generatywnych muzyki, pod warunkiem odpowiedniej inżynierii *promtu* modelu nauczyciela.

3.4.2. Muzyczny test Turinga

Test Turinga dla muzyki, analogicznie do klasycznego Testu Turinga, jest próbą oceny, czy maszyna może wykazać się poziomem inteligencji muzycznej, który jest nieodróżnialny od ludzkiego. Pomysł ten pojawił się po raz pierwszy w 1988 roku w *Computer Music Journal* jako forma identyfikacji inteligencji maszynowej w muzyce [40]. Test ten polega na tym, że maszyna i człowiek uczestniczą w sesji improwizacyjnej (*jam session*), a zadaniem oceniającego jest rozpoznanie, który z uczestników jest maszyną. Maszyna musi być w stanie wykryć rytm, rozpoznać, co gra człowiek, i odpowiedzieć na to w czasie rzeczywistym, tworząc improwizowane "solówki", która swoją strukturą będzie niejako "odpowiedzią" na

to, co zagrał człowiek.

Oryginalny Test Turinga, zaproponowany przez Alana Turinga w 1950 roku, polegał na tym, że sędzia prowadził rozmowę z dwoma podmiotami, jednym ludzkim i jednym maszynowym, bez wiedzy, który jest który [41]. Jeśli sędzia nie mógłby odróżnić maszyny od człowieka na podstawie odpowiedzi, maszyna przechodziła test, wykazując inteligencję na poziomie ludzkim. Test Turinga dla muzyki rozszerza tę koncepcję na kontekst muzyczny, gdzie zamiast rozmowy mamy interaktywną improwizację muzyczną. Kluczowym elementem jest tutaj nie tylko wynik, ale również proces tworzenia muzyki w czasie rzeczywistym, który musi być na tyle ludzki, aby oceniający nie mógł odróżnić maszyny od człowieka. Obecnie każdy model, który generuje muzykę, skupia się na wyniku, a nie na procesie. Sztuczna inteligencja musiałaby być w stanie reagować na zmiany w muzyce w czasie rzeczywistym, co wymaga nie tylko analizy dźwięku, ale również interpretacji wizualnych sygnałów i dopasowania do tempa i stylu gry człowieka. Jest to znacznie bardziej skomplikowane niż generowanie statycznych utworów muzycznych. Czynniki te sprawiają, że chociaż modele generacyjne są w stanie generowac muzykę, przejście muzycznego Testu Turinga, który wymaga głębokiej interakcji i ludzkiego zrozumienia procesu twórczego, pozostaje wyzwaniem trudnym do osiągnięcia.

Uwaga 3.1. Tutaj dopiszę zdanie czy dwa z linkiem do playlisty na youtube albo coś takiego, żeby można było posłuchać modeli.

4. Zakończenie

Omawiane w pracy modele przeznaczone do generacji i analizy teksty, sprawdziły się również w analogicznych zastosowaniach w kontekście muzyki. Mocno uznane modele transformerowe dość dobrze radzą sobie z tymi zagadnieniami, natomiast poprzez intensywne badania w tematach NLP pojawiają się nowe rozwiazania, na przykład modele Mamba, które równie dobrze, jak nawet nie lepiej dają sobie rade z generacją muzyki. Poruszone porównanie zapisu ABC oraz MIDI pozwala wybrać pasujące dane i model do potrzeb użytkownika rozwiązanie w taki sposób, który będzie dla niego zadowalający.

Przedstawione metody generacji muzyki mogą być dalej w rozwijane głównie poprzez zwiększanie liczby parametrów modeli. Im większy model, tym jest on w stanie generować bardziej skomplikowane oraz innowacyjne melodie. Należy zwrócić uwagę również na zróżnicowanie danych treningowych. Większe modele mogą "uczyć się na pamięć" przedstawionych im danych, przez co dywersyfikacja oraz gromadzenie większej ilości danych jest istotnym elementem w kontekście poprawienie jakości modelu. Dodatkowo praca poryszyła dość innowacyjne rozwiązanie oceny muzyki przy pomocy LLM-ów. Rozwiązanie to może zostać dodane jako sposób końcowego dotrenowywania parametrów modelu przy pomocy technich uczenia ze wzmacnianiem, gdzie to nie człowiek, a algorytm dostarcza feedback na temat wygenerowanych fragmentów.

Przedstawione rozwiązania mogą prowadzieć do stworzenia pluginów lub podobnych narzędzi do programów takich jak *FL Studio* lub *MuseScore*, które pozwala generatywne kończenie muzyki, która jest tworzona przez kompozytora. Podobne rozwiązania zostały wprowadzone przez Adobe w programie Photoshop, gdzie osoba mogła zanaczyć dany fragment obrazku, a następnie poprosić AI, aby dokonało pewnych korekt według polecenia. Można wyobrazić sobie, że na przykałd we wcześniej wymienionym *FL Studio*, artysta gra fragment muzyki, a następnie dostanie kilka podpowiedzi, w jaki sposób mółby wzbogacić lub kontunuować tworzony utwór. W taki sam sposób osoba tworząca zapis muzyczny w MuseScore, mogłaby dostawać propozycje od modelu, który dostaje automatycznie wygenrowaną notacje muzyczną w formacie ABC na podstawie innego zapisu. Tym sposobem, twórcy mogliby wspierać swoją kreatywność przy pomocy metod sztucznej inteligencji.

We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Center: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2022/015801

Bibliografia

[1] Daniel Adiwardana i in. *Towards a Human-like Open-Domain Chatbot*. 2020. arXiv: 2001.09977 [cs.CL].

[2] Anton Chernyavskiy, Dmitry Ilvovsky i Preslav Nakov. „Transformers: "The End of History" for Natural Language Processing?" W: *Machine Learning and Knowledge Discovery in Databases. Research Track*. Red. Nuria Oliver i in. Cham: Springer International Publishing, 2021, s. 677–693. ISBN: 978-3-030-86523-8.

[3] John Biles. „GenJam: A Genetic Algorithm for Generating Jazz Solos". W: lip. 1994.

[4] Google AI. *Magenta*. URL: https://magenta.tensorflow.org/.

[5] Adam Roberts i in. *A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music*. 2019. arXiv: 1803.05428 [cs.LG].

[6] Jesse Engel i in. *GANSynth: Adversarial Neural Audio Synthesis*. 2019. arXiv: 1902.08710 [cs.SD].

[7] Ashish Vaswani i in. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[8] Cheng-Zhi Anna Huang i in. *Music Transformer*. 2018. arXiv: 1809.04281 [cs.LG].

[9] Yueyue Zhu i in. *A Survey of AI Music Generation Tools and Models*. 2023. arXiv: 2308.12982 [cs.SD].

[10] Botao Yu i in. *Museformer: Transformer with Fine- and Coarse-Grained Attention for Music Generation*. 2022. arXiv: 2210.10349 [cs.SD].

[11] Andrea Agostinelli i in. *MusicLM: Generating Music From Text*. 2023. arXiv: 2301.11325 [cs.SD].

[12] Steve Allen. *Beethoven Symphony No. 7, Movement 2 in ABC*. URL: https://www.ucolick.org/~sla/abcmusic/sym7mov2.html.

[13] Nathan Fradet i in. „MidiTok: A Python package for MIDI file tokenization". W: *Extended Abstracts for the Late-Breaking Demo Session of the 22nd International Society for Music Information Retrieval Conference*. 2021. URL: https://archives.ismir.net/ismir2021/latebreaking/000005.pdf.

[14] Shangda Wu i in. *TunesFormer: Forming Irish Tunes with Control Codes by Bar Patching*. 2023. arXiv: 2301.02884 [cs.SD].

[15] Nitish Shirish Keskar i in. „CTRL - A Conditional Transformer Language Model for Controllable Generation". W: *arXiv preprint arXiv:1909.05858* (2019).

[16] Darrell Conklin. *Bach Chorales*. UCI Machine Learning Repository. URL: https://doi.org/10.24432/C5GC7P.

[17] I. S. Duff, A. M. Erisman i J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, sty. 2017. ISBN: 9780198508380. DOI: 10.1093/acprof:oso/9780198508380.001.0001. URL: https://doi.org/10.1093/acprof:oso/9780198508380.001.0001.

[18] Curtis Hawthorne i in. „Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset". W: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=r1lYRjC9F7.

[19] Shangda Wu i in. „TunesFormer: Forming Irish Tunes with Control Codes by Bar Patching". W: *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*. Red. Lorenzo Porcaro, Roser Batlle-Roca i Emilia Gómez. T. 3528. CEUR Workshop Proceedings. CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3528/paper1.pdf.

[20] Christopher Olah. *Understanding LSTM Networks*. 27 sierp. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[21] Sepp Hochreiter i Jürgen Schmidhuber. „Long Short-term Memory". W: *Neural computation* 9 (grud. 1997), s. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[22] Sepp Hochreiter. „The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions". W: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6 (kw. 1998), s. 107–116. DOI: 10.1142/S0218488598000094.

[23] Zhiyong Cui i in. *Deep Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction*. 2019. arXiv: 1801.02143 [cs.LG].

[24] Jacob Devlin i in. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[25] Maarten Grootendorst. *A Visual Guide to Mamba and State Space Models*. 19 lut. 2024. URL: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state.

[26] Kaiming He i in. „Deep Residual Learning for Image Recognition". W: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, s. 770–778. DOI: 10.1109/CVPR.2016.90.

[27] Albert Gu i in. „Combining Recurrent, Convolutional, and Continuous-time Models with Linear State Space Layers". W: *Advances in Neural Information Processing Systems*. Red. M. Ranzato i in. T. 34. Curran Associates, Inc., 2021, s. 572–585.

[28] Albert Gu i in. *HiPPO: Recurrent Memory with Optimal Polynomial Projections*. 2020. arXiv: 2008.07669 [cs.LG].

[29] Aaron Voelker, Ivana Kajić i Chris Eliasmith. „Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks". W: *Advances in Neural Information Processing Systems*. Red. H. Wallach i in. T. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/952285b9b7e7a1be5aa7849f32ffff05-Paper.pdf.

[30] Albert Gu i Tri Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. 2023. arXiv: 2312.00752 [cs.LG].

[31] Philip Gage. „A new algorithm for data compression". W: *C Users J.* 12.2 (lut. 1994), s. 23–38. ISSN: 0898-9788.

[32] Nathan Fradet i in. *Byte Pair Encoding for Symbolic Music*. 2023. arXiv: 2301.11975 [cs.LG].

[33] Alec Radford i in. „Language Models are Unsupervised Multitask Learners". W: (2019).

[34] *The TTS Arena*. URL: https://huggingface.co/blog/arena-tts.

[35] Jacob Collier. *That's not a wrong note, you just lack confidence.* URL: https://www.youtube.com/watch?v=meha_FCcHbo.

[36] Ruibin Yuan i in. *ChatMusician: Understanding and Generating Music Intrinsically with LLM*. 2024. arXiv: 2402.16153 [cs.SD].

[37] OpenAI i in. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL].

[38] Maximilian Schreiner. *GPT-4 architecture, datasets, costs and more leaked*. 11 lip. 2023.

[39] Long Ouyang i in. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL].

[40] Christopher Ariza. „The Interrogator as Critic: The Turing Test and the Evaluation of Generative Music Systems". W: *Computer Music Journal* 33.2 (2009), s. 48–70. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/40301027 (term. wiz. 30. 05. 2024).

[41] A. M. TURING. „I.—COMPUTING MACHINERY AND INTELLIGENCE". W: *Mind* LIX.236 (paź. 1950), s. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf. URL: https://doi.org/10.1093/mind/LIX.236.433.