



**Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie**

**Wydział Informatyki**

**PRACA DYPLOMOWA**

**Generacja muzyki przy pomocy dużych modeli językowych**

**Music generation with Large Language Models**

<b>Autor:</b>	<b>Filip Ręka</b>
<b>Kierunek:</b>	<b>Informatyka — Data Science</b>
<b>Opiekun pracy:</b>	<b>dr hab. Maciej Smółka prof. AGH</b>

**Kraków, 2024**



*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*



## Streszczenie

Duże modele językowe (ang. *Large Language Models* **LLM**) charakteryzują się zdolnością do generacji języka oraz innych zadań w przetwarzania języka naturalnego, takich jak na przykład klasyfikacja. Zdolność tą nabierają podczas czasochłonnego oraz intensywnego obliczeniowego treningu metodami samo odraz pół-nadzorowanego, podczas którego uczą się one relacji z wielkiej ilości dokumentów tekstowych. LLMy mogą zostać wykorzystane do generacji tekstu, formy generatywnej sztucznej inteligencji, poprzez pobieranie tekstu wejściowego i wielokrotne przewidywanie kolejnego tokenu lub słowa w tekście. Strukturę muzyki można porównać struktury tekstu pisanego, gdzie każda nuta odpowiada literze lub słowu, akordy zdaniom a dłuższe i sekwencję paragrafom. Poniższa praca, zamierza zbadać możliwości generacyjne LLMów wytrenowanych na muzycznych zbiorach danych.



## **Abstract**

Abstract in English ...

# Spis treści

<b>1</b>	<b>Wstęp</b>	<b>1</b>
1.1	Motywacja . . . . .	1
1.2	Cel pracy . . . . .	2
1.3	Zarys pracy dyplomowej . . . . .	2
<b>2</b>	<b>Opis aktualnego stanu wiedzy</b>	<b>3</b>
2.1	Podobieństwa pomiędzy muzyką a tekstem . . . . .	3
2.2	Cyfrowa reprezentacja muzyki . . . . .	4
2.2.1	WAV i MP3 . . . . .	4
2.2.2	MIDI (ang. <i>Musical Instrument Digital Interface</i> ) . . . . .	5
2.2.3	Notacja ABC . . . . .	6
2.2.4	Porównanie zapisu muzycznego . . . . .	9
2.3	Zbiory danych . . . . .	12
2.3.1	Johann Sebastian Bach Chorales . . . . .	12
2.3.2	The MAESTRO v3.0 . . . . .	12
2.3.3	IrishMAN . . . . .	13
2.4	Obecne próby generacji muzyki . . . . .	13
2.5	Architekturze transformera . . . . .	15
2.5.1	Sieci RNN . . . . .	15
2.5.2	LSTM . . . . .	16
2.5.3	Algorytm uwagi (ang. <i>attention</i> ) . . . . .	17
2.5.4	Transformer . . . . .	19
2.6	Architektura <i>state space</i> . . . . .	22
2.6.1	Mamba . . . . .	22
2.6.2	Mamba . . . . .	25
<b>3</b>	<b>Przeprowadzenie eksperymentów</b>	<b>27</b>
3.1	Przygotowanie danych . . . . .	27
3.2	Trening modeli . . . . .	28
3.3	Prezentacja otrzymanych wyników . . . . .	29
3.4	Sposoby oceny wyników . . . . .	31



3.4.1	Testowanie melodii innymi LLM-ami . . . . .	32
3.4.2	Muzyczny test Turinga . . . . .	38
<b>4</b>	<b>Zakończenie</b>	<b>41</b>
	<b>Bibliografia</b>	<b>43</b>

5

Zawartość spisu treści — tytuły rozdziałów oraz ich liczba zależą od tematyki pracy — należy ustalić z opiekunem pracy.

## Spis rysunków

2.1	Fragment chorału J.S. Bacha . . . . .	5
2.2	Plik <code>.wav</code> utworzony w programie Audacity. . . . .	6
2.3	Muzyka zapisana w pliku MIDI . . . . .	7
2.4	Zapis wielu głosów w notacji ABC. . . . .	8
2.5	Prosta sekwencja muzyczna . . . . .	9
2.6	Reprezentacja muzyki w postaci tokenów . . . . .	10
2.7	Przykład utworu pochodzącego ze zbioru MAESTRO. . . . .	13
2.8	Przykład wielogłosowego fragmentu ze zbioru IrishMAN. . . . .	14
2.9	Schemat budowy fragmentu sieci RNN. . . . .	15
2.10	Komórki LSTM połączone między sobą. . . . .	16
2.11	Schemat mechanizmu uwagi oraz ich kilkukrotne złożenie. . . . .	18
2.12	Schemat transformera. . . . .	20
2.13	Algorytm uwagi z maską. . . . .	21
2.14	Schemat modelu przestrzeni stanów. . . . .	23
2.15	Schemat modelu Mamba. . . . .	25
3.1	Wykresy treningu modeli MIDI. . . . .	30
3.2	Wynik modelu. . . . .	31
3.3	Wygenerowana melodia. . . . .	31
3.4	Pytanie dla modelu ChatMusician. . . . .	34
3.5	Odpowiedź modelu ChatMusician. . . . .	35

3.6	Specjalnie stworzony niepoprawny fragment muzyki. . . . .	35
3.7	Odpowiedź modelu na fałszywy fragment. . . . .	36

## Spis tabel

2.1	Porównanie różnych zapisów muzyki . . . . .	10
3.1	GPT-2 ~6M . . . . .	29
3.2	GPT-2 ~60M . . . . .	29
3.3	Mamba ~6M . . . . .	29
3.4	Mamba ~60M . . . . .	29

## 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 szczególności 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 następnie na 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 różnych 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

**Uwaga 1.1.** Zostanie napisane na końcu jak żeby nie powtarzać się ileś razy

## 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 frazy 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 niezwykle możliwości w zadaniach przetwarzania języka naturalnego 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 autoregresyjne 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 przewidywania jako dane wejściowe do przewidywania następnego słowa. Na każdym etapie model generuje rozkład prawdopodobieństwa w słowniku, wskazując prawdopodobieństwo, że każde słowo będzie następnym słowem w sekwencji. Typowo w tego rodzaju modelach występuje pojęcie okna kontekstowego, czyli ilości 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

sekwencja wejściowa:  $x_1, x_2, x_3$ , gdzie  $x_i \in X$

$P_{t1}(X) = \text{Model}(x_1, x_2, x_3)$

$x_4 \sim P_{t1}(X)$

$P_{t2}(X) = \text{Model}(x_1, x_2, x_3, x_4)$

$x_5 \sim P_{t2}(X)$

...

Przykładowe podejście do autoregresyjnej generacji sekwencji pokazano w równaniu ?? . Algorytm jest agnostyczny w stosunku do typu danych, więc w  $X$  mogą być zarówno wszystkie słowa, litery lub elementy notacji muzycznej.

## 2.2. Cyfrowa reprezentacja muzyki

W kulturze zachodniej nuty zapisuje się na pięcioliniach ułożonych jedna pod drugą. Jeśli muzyka jest wielogłosowa to typowo dwie lub więcej pięciolin jest okalane przy pomocy nawiasów klamrowych. Na pięcioliniach umieszczone są takie elementy jak nuty, pauzy oznaczenia dynamiki, tempa i inne znaki dotyczące zapisu w jaki sposób należy wykonać dany utwór. Najpierw muzykę zapisywano na papierze, jednak po pojawieniu się komputerów, zapis nutowy jak i samo nagranie odtworzenia 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.

Przykładowy zapis muzyki zapisano na obrazku 2.1. Przedstawiony fragment składa się z czterech głosów grających na raz oraz z czterech taktów.

### 2.2.1. 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 charakteryzują się najwyższą jakością dźwięku, jednak ich rozmiary są bardzo duże, co może być problematyczne przy przechowywaniu dużych zbiorów danych. Przechowywanie takich plików wymaga znacznej ilości miejsca 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 44100 próbek na sekundę. Format ten został stworzony w 1991 roku i od tego czasu stał się jednym z najbardziej rozpowszechnionych formatów audio,

## Chorał no. 7

J.S. Bach



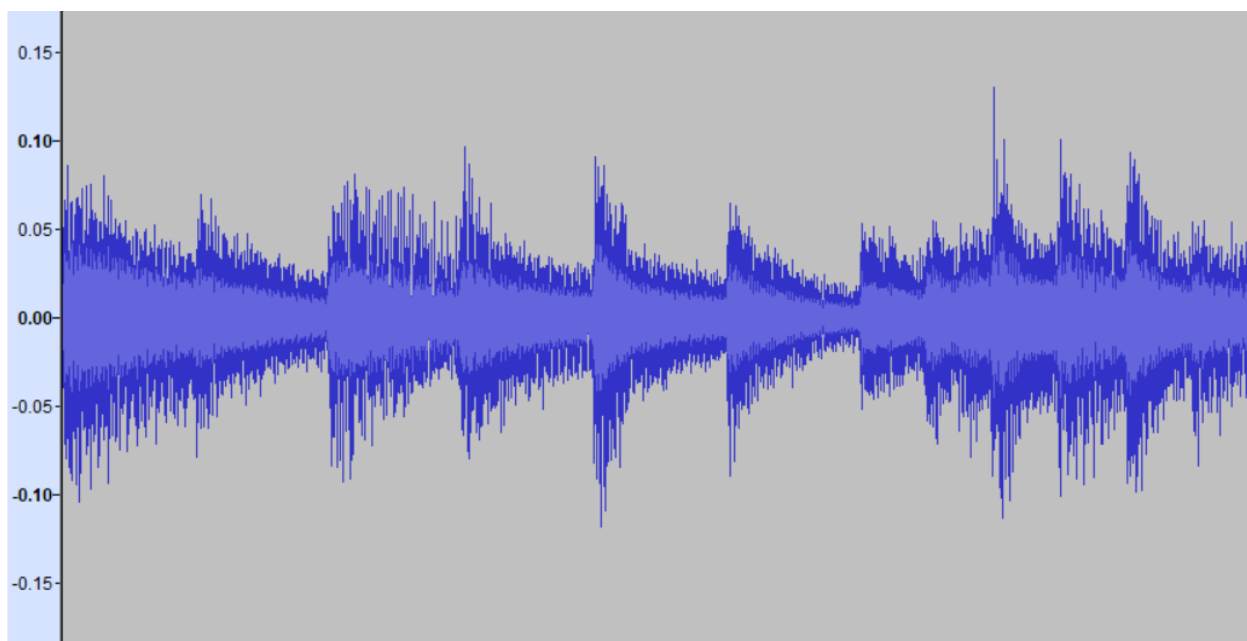
Rysunek 2.1.: Fragment chorału J.S. Bacha

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.

### 2.2.2. MIDI (ang. *Musical Instrument Digital Interface*)

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 dla szesnastu kanałów jednocześnie, z których każdy może reprezentować inny



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

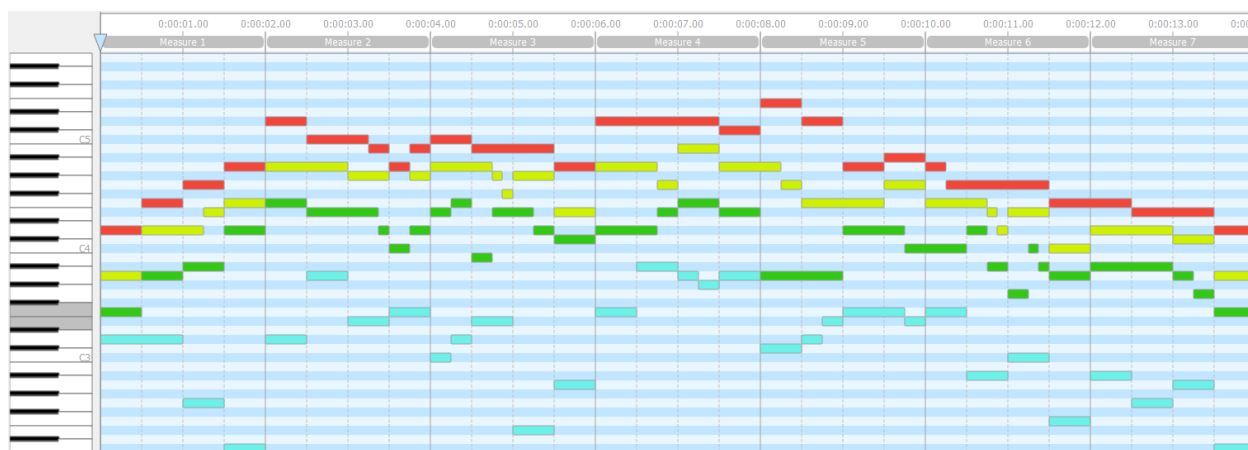
instrument. Każda interakcja z instrumentem, taka jak naciśnięcie klawisza czy szarpnięcie struny, jest zapisywana jako “zdarzenie” (*event*), które zawiera takie dane jak znacznik czasu, wysokość dźwięku oraz jego głośność. Dane z urządzeń MIDI są zapisywane w specjalnych plikach z rozszerzeniem .mid 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. Na takim wykresie oś pozioma reprezentuje czas, a oś pionowa 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 ekspresję, ponieważ jest to zapis odtworzenia przez artystę pewnego utworu. Ten sposób nie ogranicza muzyki sztywno do konkretnego rytmu zdefiniowanego przez autora.

### 2.2.3. 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





Rysunek 2.3.: Muzyka zapisana w pliku MIDI

melodii. W kolejnej dekadzie system ten został rozwinięty przez Chrisa Walshaw, 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.

Jak widać, notacja ABC w dość zwięzły sposób zapisuje partyturę. Poza konkretnymi nutami i rytmem, w tym formacie można zapisać również dodatkowe informacje na temat utworu. W każdej 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. Wiele z tych informacji powinna zostać usunięta w procesie preprocessingu danych, aby nie zagłuszały najistotniejszych pól. Do takich należą przede wszystkim: domyślna długość nuty (L:), metrum (M:) oraz tonacja (K:), w jakiej utwór został napisany. Chociaż model jest w stanie “odgadnąć” te informacje, ich wyraźne podanie daje większą kontrolę nad procesem treningu. Dodatkową zaletą podawania tych informacji podczas treningu jest możliwość użycia ich jako początkowej sekwencji, na podstawie której model będzie dalej generował melodię. Dzięki temu mamy kontrolę nad takimi aspektami jak tonacja i metrum utworu, co pozwala na bardziej precyzyjne kierowanie procesem generowania muzyki.

Notacja ABC wspiera 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 [3].

```
1 X:1
2 T:Chorał no.7
3 A:J.S. Bach
4 Q:1/4=120
5 V:1
6 L:1/16
7 M:4/4
8 K:C clef=G2
9 D4F4G4A4|d4c6B2A2B2|c4B8A4|d12^c4|
10 V:2
11 L:1/16
12 M:4/4
13 K:C clef=G2
14 A,4D6E2F4|A8^G4A2^G2|A6^G^F^G4E4|A6G2B4A4|
15 V:3
16 F,4A,4^A,4D4|F4E7DC2D2|E2F2B,2E4D2^C4|D6E2F4E4|
17 V:4
18 L:1/16
19 M:4/4
20 K:C clef=F4
21 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.

### 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 liczbowo 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 gramatycznego 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

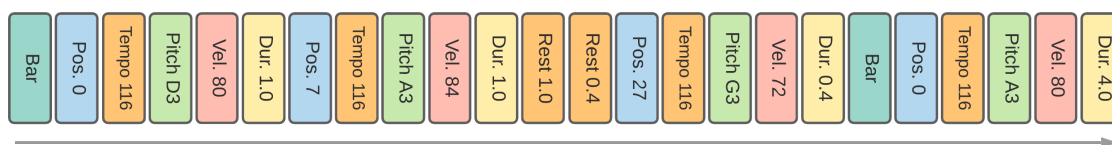
Istnieje wiele algorytmów tokenizacji MIDI, jednak aby przedstawić mechanizm działania, dokonano analizy popularnego algorytmu *REMI. REvamped MIDI (REMI)* reprezentuje zdarzenia jako sekwencje tokenów wysokości tonu, dynamiki (głośności), długości trwania oraz czasu poprzez tokeny taktu i pozycji. Token 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 [4].



Rysunek 2.5.: Prosta sekwencja muzyczna

#### 2.2.4. Porównanie zapisu muzycznego

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



Rysunek 2.6.: Reprezentacja muzyki w postaci tokenów

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

Tabela 2.1.: Porównanie różnych zapisów muzyki

Jak widać zapis w postaci fali dźwiękowej zajmuje najwięcej miejsca na dysku, jak i również 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 2.2.3. Przy użyciu innego tokenizera niż REMI, liczba powstałych tokenów może lekko różnić się od tej w tabeli. Różnica w rozmiarze na dysku oraz długością sekwencji pomiędzy zapisem MIDI oraz ABC nie jest duża. Warto zwrócić uwagę, że gdyby w oryginalnym utworze pojawiły się powtórzenia danych sekwencji melodii notacja ABC byłaby jeszcze bardziej oszczędna od plików MIDI. Powodem tego jest fakt, że notacja ABC zapisuje bezpośrednio w postaci tekstowej zapis nutowy, czyli znak powtórzenia zostanie zapisany jako :| oraz |: , a w pliku MIDI cała powtarzana sekwencja musi zostać ponownie odegrana przez artystę.

Do głównych zalet plików MIDI należy ich wszechstronność. Nie są one ograniczone do sztywno określonego rytmu, jak w innych notacjach muzycznych. Dodatkowo, przy użyciu narzędzi komputerowych, bardzo łatwo można zamienić dowolny zapis nutowy (np. MusicXML, ABC) na plik MIDI. Niestety, w drugą stronę, czyli konwersja pliku MIDI na zapis nutowy, jest znacznie trudniejsza, głównie z powodu nieokreślonego rytmu, który może być zakodowany w pliku MIDI. 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 odwzorowania. Mimo tych wyzwań, pliki MIDI pozostają niezwykle użyteczne w zastosowaniach muzycznych, od tworzenia i edytowania muzyki po jej analizę i odtwarzanie.

Plik w postaci fali dźwiękowej nie jest idealnym sposobem zapisu muzyki, jednak często jest to jedyne rozwiązanie, aby zebrać dużą ilość danych. Szczególnie w przypadku muzyki nowoczesnej twórcy zazwyczaj nie udostępniają plików MIDI ani zapisu nutowego, na podsta-

wie którego powstała dana piosenka. Z tego powodu praca z plikami MP3 lub czasochłonna translacja muzyki na inny format jest często konieczna.

Dodatkową zaletą plików z dźwiękiem w zapisie cyfrowym jest dostęp do potencjalnie istniejącego zapisu odśpiewanego tekstu. Chociaż 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 wymagają 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 krótszą 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* danych. Podejście takie zostało zaproponowane w modelu *Tunesformer*[5], który wziął przykład z artykułu CTRL [6], w którym autorzy dodawali do korpusu treningowego takie tagi jak *Books*, *Horror*, *Relationships*, *Legal*. Tagi te były podawane na początku *promptu* podawanego dla modelu, przez co model "wiedział" 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 *[|*, *||*, *l]*, *|:*, *::* i *:|*.
- *B:* - liczba taktów w danej sekcji. Liczy tylko wystąpienia znaku *|*
- *E:* - podobieństwo. 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

$$E(c, p) = 1 - \frac{lev(c, p)}{\max(|c|, |p|)} \quad (2.1)$$

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 powoduje, 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. Wartości podawane w tagach są elementami, które policzyliśmy na podstawie danych, co znaczy że model mógłby je w czasie treningu wywnioskować jednak podawanie ich w sposób *explicite* przekazuje użytkownikowi kontrolę nad tymi parametrami.

## 2.3. Zbiory danych

### 2.3.1. Johann Sebastian Bach Chorales

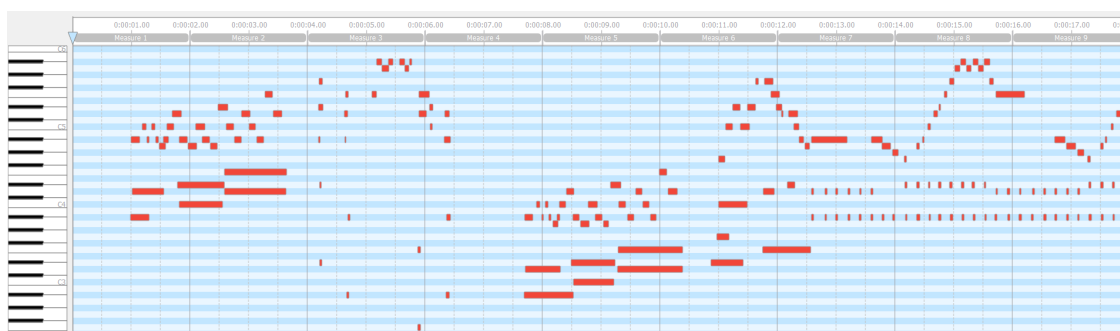
*JSB Chorales* [7] to zbiór krótkich, czterogłosowych utworów. Chorały zostały pierwotnie skomponowane przez Johanna Sebastiana Bacha w XVIII wieku. Napisał je najpierw biorąc wcześniej istniejące melodie ze współczesnych hymnów luterkańskich, a następnie harmonizując je w celu stworzenia części dla pozostałych trzech głosów. Wersja zbioru danych używana kanonicznie w kontekście uczenia się reprezentacji w kontekście uczenia się reprezentacji składa się z 382 takich chorałów. Utwory przed złożeniem w zbiór danych zostały ztransponowane 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  $4 \times \text{długość utworu}$  można przejść do analizy. Wadą tego podejścia jest możliwość wystąpienia macierzy rzadkich, które często stanowią problem dla algorytmów uczenia maszynowego (CITATION NEEDED!), jednak w tym zbiorze ten problem nie występuje.

### 2.3.2. The MAESTRO v3.0

Zbiór danych MAESTRO w wersji trzeciej [8] 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 fortepianu, 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. Jak widać zapis plików WAV jest zdecydowanie mniej oszczędny jeśli chodzi o przechowywanie danych na dysku.

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 wywnioskować że ten utwór bardziej złożony. Jak widać odegrane nuty często nie padają na sztywno określone rozpoczęcie taktu, przez co potencjalnie wygenerowana muzyka na podstawie tego zbioru będzie wydawała się bardziej



Rysunek 2.7.: Przykład utworu pochodzącego ze zbioru MAESTRO.

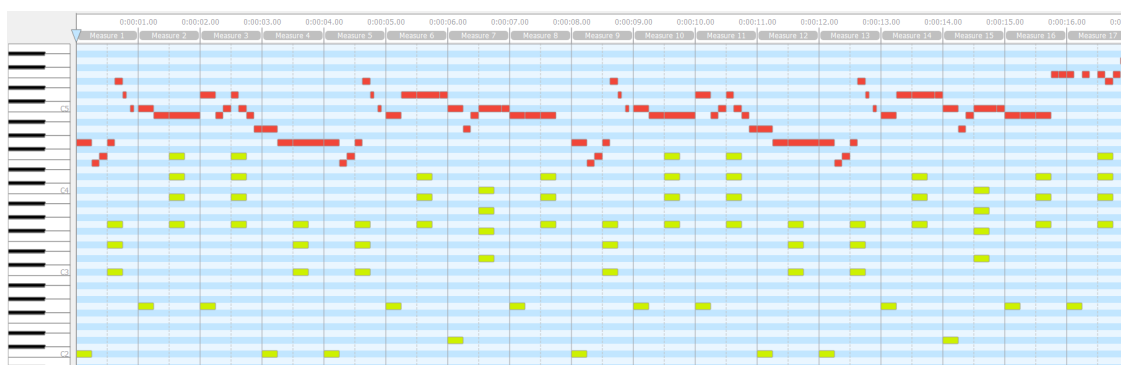
ekspresyjna oraz “ludzka”.

### 2.3.3. IrishMAN

Zbiór danych IrishMAN (*ang. Irish Massive ABC Notation*) [9] zawiera 216,248 fragmentów melodii podzielonych na zbiór treningowy i walidacyjny. Melodie zebrane są ze stron takich jak [thesession.org](http://thesession.org) oraz [abcnotation.org](http://abcnotation.org), które znane są z udostępniania tradycyjnej muzyki. Aby zapewnić jednolitość formatowania najpierw wszystkie utwory zostały przekonwertowane do notacji MusicXML, a następnie do notacji ABC. Istnieje bliźniacze zbiory IrishMAN-MIDI oraz IrishMAN-XML, które zawierają te same utwory jednak zapisane w innych formatach. Wszystkie fragmenty muzyki należą do domeny publicznej, w związku z czym bez obaw o łamanie praw autorskich można korzystać z tych melodii. 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 fragmenty muzyki w różnych formatach, przez co nadaje się on wyjątkowo dobrze do porównywania różnych algorytmów generacyjnych.

## 2.4. Obecne próby generacji muzyki

Pierwsze próby generacji muzyki przy pomocy algorytmów można znaleźć już w roku 1994 [10]. *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 [11]. Projekt ten skupia się na badaniu



Rysunek 2.8.: Przykład wielogłosowego fragmentu ze zbioru IrishMAN.

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 konwolucyjnych był w stanie generować sekwencje muzyczne, oraz interpolować pomiędzy nimi [12]. Innym modelem stworzonym przez tę organizację jest *GanSynth* [13], który był w stanie generować naturalnie brzmiącą muzykę przy pomocy modelu *GAN* (*generative adversarial network*). Przełomowym momentem w rozwoju modeli generacyjnych było wprowadzenie modeli transformera w roku 2017 [14]. 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 [15]. Był w stanie generować długie sekwencje muzyczne o długotrwałej strukturze. Mimo wszystko 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 [16]. Bardzo podobnym modelem jest *Museformer* [17], który implementując inny rodzaj algorytmu *attention* w dekodерze 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

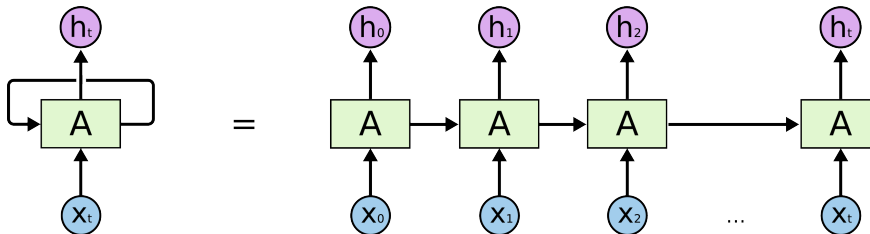


podstawie generuje odpowiedni fragment. Typowe zapytanie może brzmieć na przykład “*Slow tempo, bass-and-drums-led reggae song. Sustained electric guitar. High-pitched bongos with ringing tones. Vocals are relaxed with a laid-back feel, very expressive*”. Przykładem takiego modelu jest *MusicLM* stworzony przez Microsoft [18]. 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.

## 2.5. Architektury 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 rozwinięcie pętli.



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 pomiędzy ważnymi informacjami jest mała. Jednak dla zdania “Rok temu ukończyłem kurs lotnika ... Teraz staram się o licencję *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, wymagana jest jedynie informacja na jego temat, tak aby dobrać odpowiedni akord konsonansowy lub akord dominantowy na tonikę, 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.

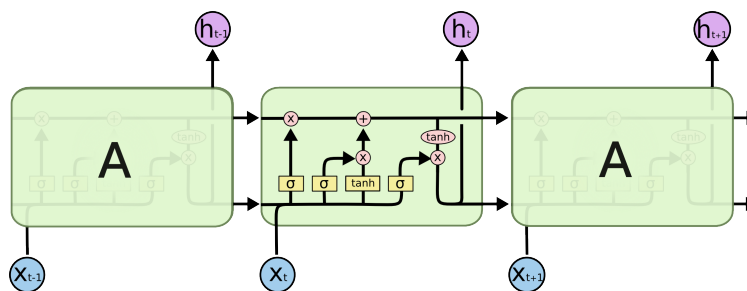
Dodatkowym problemem podczas treningu modeli jest problem zanikającego gradientu.

$$\begin{aligned}
 H_{i+1} &= A(H_i, x_i) \\
 H_3 &= A(A(A(H_0, x_0), x_1), x_2) \\
 &\text{gdzie:} \\
 A(H, x) &:= Wx + ZH \\
 H_N &= W^N x_0 + W^{N-1} x_1 + \dots
 \end{aligned} \tag{2.2}$$

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*. Ich budowa pozwala na długofalowe zapamiętywanie i przekazywanie informacji. Zazwyczaj są łączone między sobą co pozytywnie wpływa na działanie sieci.



Rysunek 2.10.: Komórki LSTM połączone między sobą.

Główną częścią budowy komórki jest górna strzałka 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 jaka które informacje są ważne i mają być zachowane a które można zignorować. Są one zbudowane z warstw sigmoidalnych

oznaczonych na grafice literami  $\sigma$ . Wartościami zwrotnymi jest liczba pomiędzy 0 a 1 gdzie mała wartość znaczy niską wagę, a wysoka dużą. 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. Dodatkową wadą modelu LSTM jak i klasycznego RNN jest ich wysoka złożoność treningu, która występuje z powodu konieczności użycia specjalnego wariantu algorytmu wstecznej propagacji błędów nazywanego “wsteczną propagacją błędów w czasie” *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. 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*)

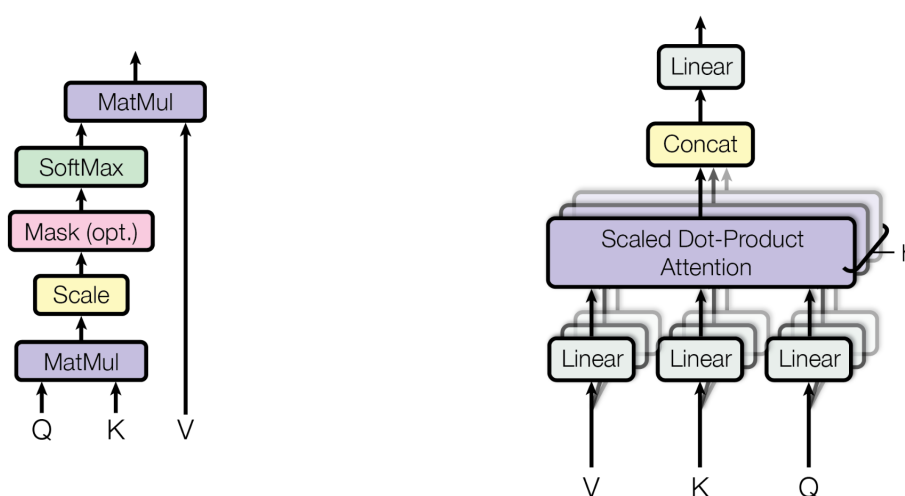
Mechanizm uwagi (ang. *attention*) został zaproponowany w artykule “Attention is All You Need” [14]. Algorytm ten działa w ustalonym oknie kontekstowym (ang. *context window*), w którym przydziela każdemu słowu wagę reprezentującą jego wagę w danym kontekście. Metoda ta imituje sposób, w jaki człowiek skupia się na różnych fragmentach tekstu w danym momencie. Mechanizm uwagi został opracowany, aby rozwiązać problem występujący w rekurencyjnych sieciach neuronowych (*RNN*), gdzie “waga” słowa była tym większa, im bliżej końca sekwencji się znajdowało. W rezultacie modele często ignorowały informacje znajdujące się na początku okna kontekstowego.

W kontekście muzyki problem zanikania kontekstu jest jeszcze bardziej widoczny niż w tradycyjnych problemach przetwarzania języka naturalnego (NLP), ponieważ na początku zapisu nutowego znajdują się kluczowe 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.

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 rekurencyjnych sieciach neuronowych lub LSTM-ach, 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.

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 analizę, każda sekwencja musi zostać osadzona (*embedded*) do wektora o określonej długości. W omawianym algorytmie nazywanym *self-attention* lub *scaled dot-product attention* sekwencja “zwraca uwagę” na samą siebie, czyli innymi słowy każdy token w sekwencji jest rozważany z każdym innym.



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

Wartości  $K$  (Key),  $V$  (Value) i  $Q$  (Query) w mechanizmie *self-attention* są uzyskiwane z danych wejściowych za pomocą uczenia się odpowiadającym im 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 rozumieć 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 2.3. 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.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.3)$$

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 znajdują się warstwy liniowe. Wartości po wyjściu z algorytmu *attention* są konkatelowane, 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

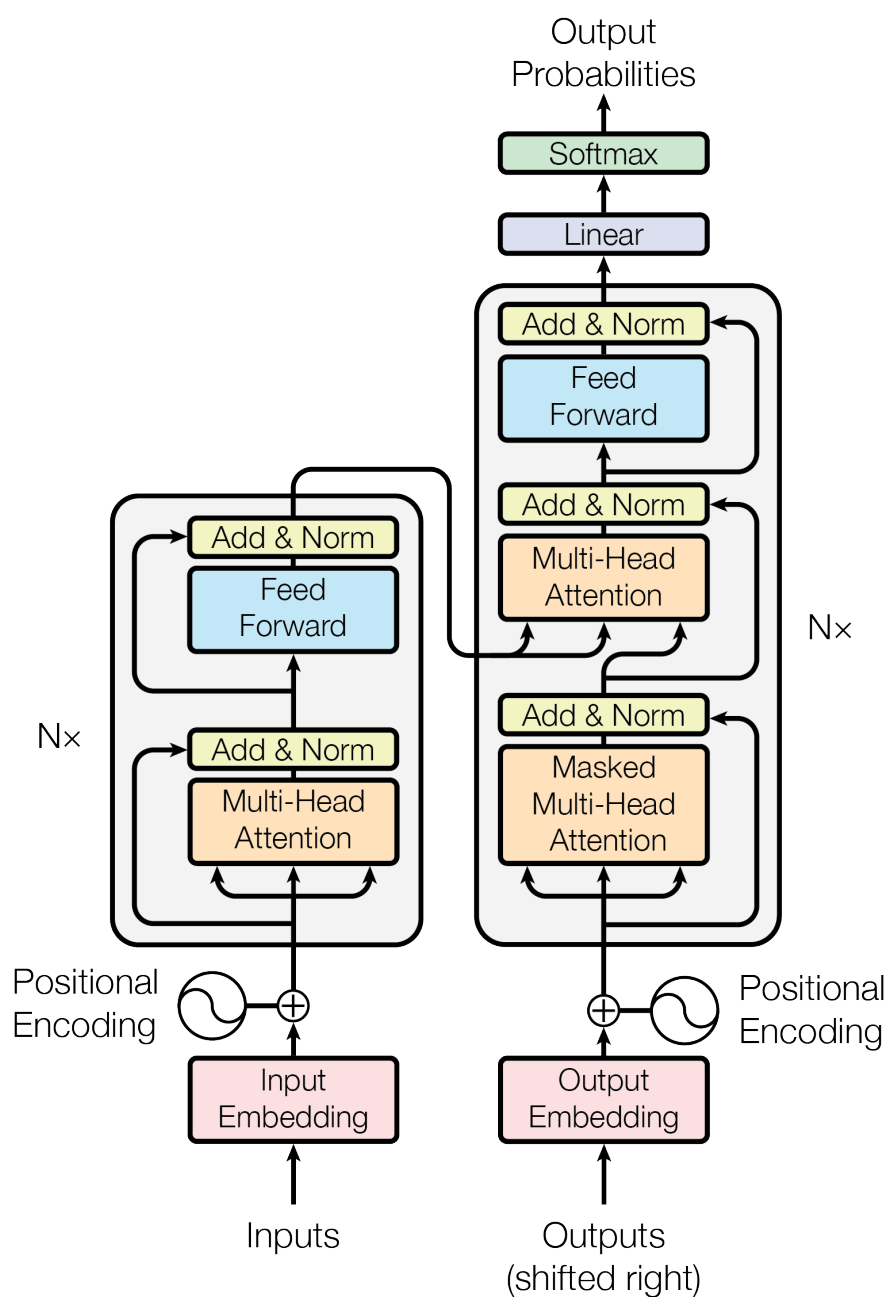
Wraz z wprowadzeniem algorytmu uwagi pisanego w 2.5.3 w tym samym artykule został zaproponowana architektura modelu, która wykorzystywała *multi-headed* algorytm uwagi. Cały model składa się z enkodera oraz dekodera.

#### 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. Drugi komponent każdej warstwy enkodera to sieć neuronowa typu *feed forward*, która składa się z dwóch w pełni połączonych warstw z funkcjami aktywacji *ReLU*. Każdy z dwóch głównych komponentów jest otoczony mechanizmem dodania rezydualnego (poprawiającego wydajność treningu) i 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.

#### Dekoder

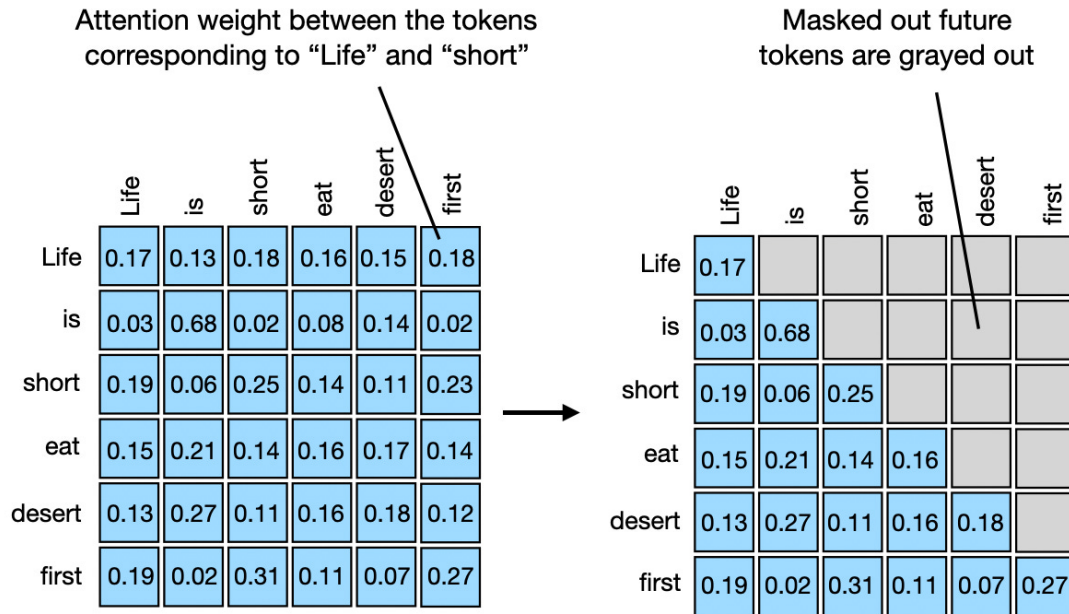
Zadaniem Dekoder w architekturze transformerowej jest generowanie wyjścia na podstawie sekwencji wejściowej i dotychczas wygenerowanych tokenów wyjściowych. Podobnie jak w enkoderze, mechanizm samo-uwagi analizuje zależności między tokenami w sekwencji wyjściowej, jednak w dekodерze stosuje się maskowanie (*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, co zostało pokazane na grafice 2.13. Drugi mechanizm



Rysunek 2.12.: Schemat transformera.

uwagi umożliwia dekodowaniu skupienie się na odpowiednich częściach sekwencji wejściowej (wyjścia enkodera) podczas generowania każdego tokena wyjściowego. Proces ten jest po-

dobny do mechanizmu samo-uwagi, ale tutaj zapytania (Q) pochodzą z poprzedniej warstwy uwagi dekodera, a klucze (K) i wartości (V) pochodzą z wyjść enkodera. Następnie tak jak w poprzednim przypadku występuje warstwa *feed forward* i warstwy normalizujące. W ostatnich warstwach 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.



Rysunek 2.13.: Algorytm uwagi z maską.

Istnieją modele transformerowe nie korzystające z całej architektury. Przykładem *encoder only transformer* jest model BERT [19], który jest typowo wykorzystywany do zadań np. klasyfikacji. W przypadku takiego zadania, nie potrzeba omawianej wcześniej maski, ponieważ model powinien analizować kontekst całego fragmentu w “obie strony”. Wystarczy dołożyć jako ostatnią warstwę 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 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 (*begin of sequence*), co da modelowi największą wolność twórczą. Rozwiązania *decoder only* są obecnie bardzo popularne i są używane przez modele takie jak GPT od OpenAI. Ciekawą obserwacją jest fakt, że dekodery pozbawione dru-

giej warstwy uwagi, staje się w zasadzie enkoderem z dodaną maską w pierwszej warstwie. Jest to często wykorzystywana “sztuczka”, którą można wykorzystać w wielu bibliotekach implementujących gotowe części składowe modeli.

Trening transformera jest możliwy do bardzo dobrego zrównoleglenia. Wynika to z faktu, że mechanizm uwagi jest w stanie niezależnie 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 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.

## 2.6. Architektura *state space*

### 2.6.1. Mamba

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. do reprezentacji ukrytej  $h(t)$
3. i przewiduje sekwencję wynikową  $y(t)$

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

$$\text{równanie stanu: } h'(t) = Ah(t) + Bx(t) \quad (2.4)$$

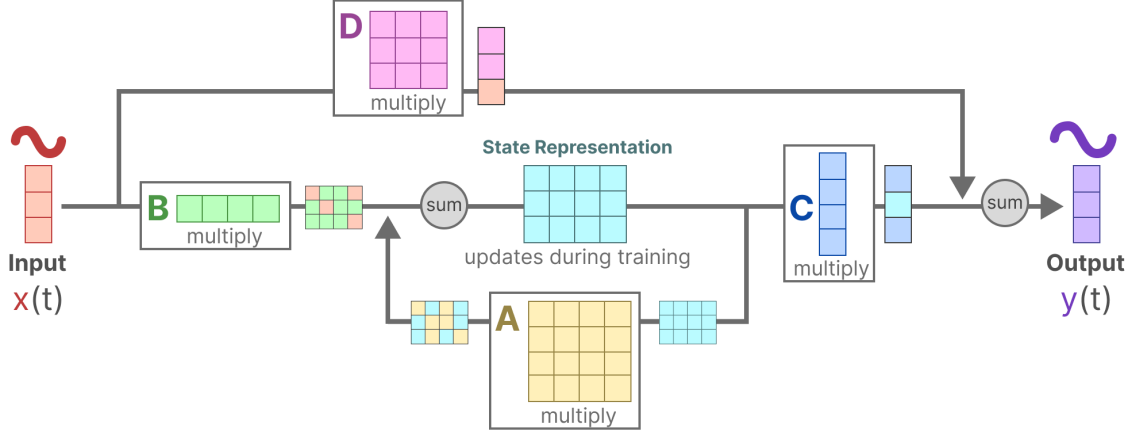
$$\text{równanie wynikowe: } y'(t) = Ch(t) + Dx(t) \quad (2.5)$$

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

Równanie stanu 2.4 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 2.5 jest opisanie jak obecny stan ukryty wpływa na sekwencje wynikowym oraz jak sekwencja wejściowa wpływa na wynik. Te informacje są zapisywane odpowiednia w



macierzy  $C$  oraz  $D$ . 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 lub połączenia typu *skip-connection* w głębokich sieciach neuronowych, często pomija się je w zapisie modelu co upraszcza zapis [20].

### Dyskretyzacja modelu

Modele tego typu działają 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) rzędu 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 2.6 oraz 2.7.

$$\text{równanie stanu: } h_k = Ah_{k-1} + Bx_k \quad (2.6)$$

$$\text{równanie wynikowe: } y_k = Ch_k \quad (2.7)$$

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 poprzedniego 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 1. Tego rodzaju interpretacja modelu jest o tyle wygodna, że pozwala zrównoleglić trening modelu a co za tym idzie zmniejszyć jego czas [21]. Kiedy model jest już wytrenowany i gotowy do wykonywania generacji, można bez problemu przejść na interpretację 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 [22]. Zadaniem *High-order Polynomial Projection Operators* jest jak najlepsze skompresowanie wejścia do wektora cech. Konstrukcje macierzy pokazano na równaniu 2.8. W tym przypadku zakładamy, że macierz jest kwadratowa.

$$A_{nk} = - \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{dla } n > k \\ n+1 & \text{dla } n = k \\ 0 & \text{dla } n < k \end{cases} \quad (2.8)$$

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 [23].

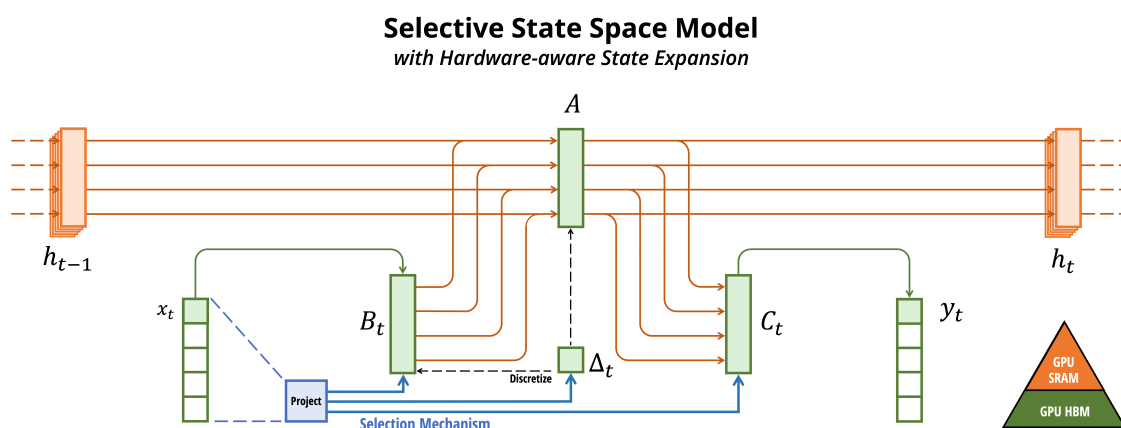
### Modele 4S

Po dodanie omawianych usprawnień (dyskretyzacja, macierz HiPPO) do tradycyjnego modelu SSM nowo otrzymany model dostał 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 oraz konwolucyjnej rozwiązuje problemy między innymi złożoności obliczeniowej treningu i infrencji modelu.

## 2.6.2. Mamba

Mamba [24] jest architekturą modelu, 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. W szczególności, skaluje się ona liniowo wraz z długością sekwencji



Rysunek 2.15.: Schemat modelu Mamba.

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, a kolorami jest zaznaczona sekcja pamięci GPU na której są zaalokowane konkretne części modelu.

W przypadku zadań syntetycznych, takich jak kopiowanie, Mamba nie tylko z łatwością rozwiązuje te zadania, ale może również ekstrapolować rozwiązania 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 porównaniu do transformatorów o podobnej wielkości i dorównuje wydajnością transformatorom dwukrotnie większym w testach porównawczych takich jak LAMBDA, Pile lub PIQA.



## 3. Przeprowadzenie eksperymentów

W celu oceny skuteczności modeli generowania muzyki, przeprowadzono szereg eksperymentów, w których porównano modele transformerowe oraz model 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 (*REpresentation of Musical Information*) o którym była mowa w rozdziale 2.2.3. Przekształca on dane muzyczne z formatu MIDI w sekwencje tokenów, które mogą być interpretowane przez modele tekstowe.

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 [25]. W kontekście przetwarzania muzyki, BPE pomaga w optymalizacji danych wejściowych, co z kolei może prowadzić do lepszej wydajności modeli.

1. Inicjalizacja: 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 *Byte Pair Encoding*, można rozważyć prostą sekwencję tekstową `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 zamieniona 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 kolejne 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 (*NLP*) i uczenia maszynowego, znana z opracowywania narzędzi i bibliotek ułatwiających tworzenie, wdrażanie i udostępnianie modeli uczenia maszynowego. Jednym z jej flagowych produktów jest biblioteka Transformers, bardzo popularna biblioteka typu *open-source*, 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* [26]. Model został stworzony przez OpenAI i jest to obecnie ich największy model udostępniony jako *open source*. Jest oparty o architekturę *decoder-only*.

Trening modeli typu *Causal Language Model (CausalLM)* jest trenowany w celu przewidywania następnego tokena w sekwencji, biorąc pod uwagę poprzednie tokeny. Podczas szkolenia tokeny wejściowe są wprowadzane do modelu, a model przewiduje rozkład prawdopodobieństwa następnego tokena. Funkcja straty (*loss*) jest obliczana na podstawie przewidywań modelu i rzeczywistych tokenów docelowych, które są tokenami wejściowymi przesuniętymi o jedną pozycję. Typowo używaną funkcją 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 ich parametrów. Parametry modelu GPT-2:

Tabela 3.1.: GPT-2 ~6M

Parametr	wartość
n_embed	512
n_layer	6
n_head	4
n_inner	2048

Tabela 3.2.: GPT-2 ~60M

Parametr	wartość
n_embed	128
n_layer	2
n_head	1
n_inner	256

Tabela 3.3.: Mamba ~6M

Parametr	wartość
hidden_size	128
state_size	4
num_hidden_layers	8

Tabela 3.4.: Mamba ~60M

Parametr	wartość
hidden_size	512
state_size	8
num_hidden_layers	12

- n\_embed - Dimensionality of the embeddings and hidden states.
- n\_layer - Number of hidden layers in the Transformer encoder.
- n\_head - Number of attention heads for each attention layer in the Transformer encoder.
- n\_inner - Dimensionality of the inner feed-forward layers.

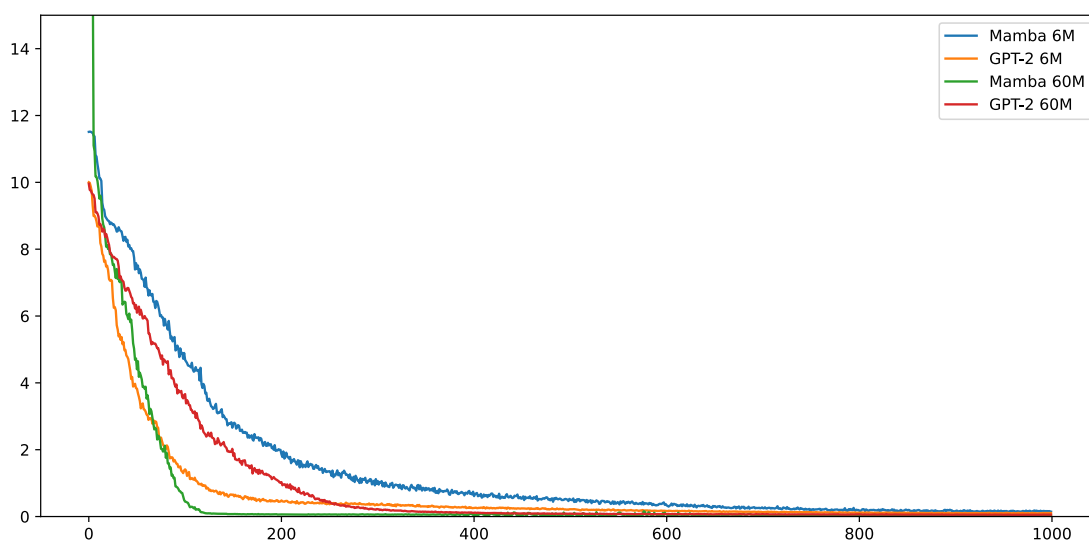
Parametry modeli Mamba:

- hidden\_size - Dimensionality of the embeddings and hidden states.
- state\_size - Shape of the state space latents.
- num\_hidden\_layers - Number of hidden layers in the model.

**Uwaga 3.1.** W tym miejscu będą wykresiki treningu modeli, i tego rodzaju rzeczy, czas treningu, ilość epok i tak dalej + opis czym są te parametry w tabelkach

### 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



Rysunek 3.1.: Wykresy treningu modeli MIDI.

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.

Na grafice 3.2 przedstawiono wynik modelu [DO OPISANIA JAKIEGO MODELU]. Część oznaczoną kolorem niebieskim należy traktować jako sekwencję wejściową dla modelu, na której podstawie model generuje kolejne tokeny. Dodatkowo należy zwrócić uwagę na podane kody kontrolne w linijce 2, 3 i 4. Reprezentacje otrzymanej muzyki zapisanej na pięciolinii przedstawia grafika 3.3.

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 krzyżyki ( $\text{^c}$  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.



```

1 X:1
2 S:2
3 B:9
4 E:4
5 L:1/8
6 M:3/4
7 K:C
8 D/2d/2d Ad fa | a/g/e/f/ g/f/e/d/ ^cA | d2 dA df |
9 ag/f/ e^c A2 | D2 d^c de | f2 ef/g/ a/g/f/e/ | d2 df e^c | 1 d2 D4 :|
10 2 d2 D2 fg |: a2 g2 f2 | ed ^cd e/f/g | a2 g2 f2 | ed ^cd e/f g |
11 a2 g2 f2 | ed ^cd ef/g/ | a2 g2 fe | 1 d2 de fg :| 2 d2 D4 ||

```

Rysunek 3.2.: Wynik modelu.



Rysunek 3.3.: Wygenerowana melodia.

### 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 subiektywności 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*[\[27\]](#), 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, dzięki 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 harmoniczej, 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 [\[28\]](#).

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 można stworzyć 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 [29]. 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 udostępniony 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 właśnie język.

Zapytanie o wygenerowany fragment zapisany na partyturze 3.3 przedstawiono na 3.4.

Model dość przychylnie spojrzał na ten wygenerowany fragment, jednak jego odpowiedzi nie są specyficzne o czym można się przekonać czytając odpowiedź na grafice 3.5. Nie podejmuje on analizy konkretnych 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 powtórzenia, 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 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 3.6. Dodatkowo, pomimo tego że zapis jest poprawny w notacji ABC, niektóre takty zawierają więcej lub mniej dźwięków niż jest to przewidziane w metrum<sup>4</sup>. Odpowiedź modelu przedstawiono we fragmencie 3.7.

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

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

Rysunek 3.4.: Pytanie dla modelu ChatMusician.

problemów z rytmem, które nie jest jednakowe dla każdego taktu oraz nie wyraził żadnej konstruktywnej krytyki. Niestety model *ChatMusician* nie jest dobrym rozwiązaniem jeśli chodzi o ocenianie muzyki.

## GPT-4

Aby porównać możliwości oceny modeli LLM należy użyć silniejszego modelu. *GPT-4* to model, który obecnie najlepiej wypada na praktycznie każdym benchmarku [30]. 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 dokładna konfiguracja nie jest znana, jednak hipotetyzuje się, że ma 1.67 trylionów parametrów [31]. 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*.

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

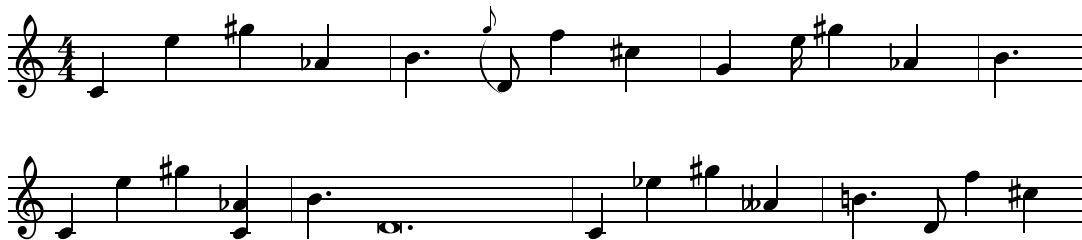
**ANSWER:** Overall Musicality: 1. The piece has a pleasant and flowing rhythm with a steady tempo. It features a mix of different instruments and voices, creating an interesting texture. The melodies are catchy and memorable, making the piece enjoyable to listen to.

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 3.5.: Odpowiedź modelu ChatMusician.



Rysunek 3.6.: Specjalnie stworzony niepoprawny fragment muzyki.

```

3  ### Key and Meter
4  - **Key:** C major
5  - **Meter:** 3/4 (Waltz time)
6
7  ### Melody and Harmony
8  1. **Opening Phrase:**
9    - `D/2d/2d Ad fa | a/g/e/f/ g/f/e/d/ ~cA`
10   - This phrase introduces a mix of stepwise motion and leaps, giving a dynamic start. The alternation between the upper and lower notes
11     creates a call-and-response effect. The use of the raised 4th degree (~c) adds a touch of chromaticism.
12
13  2. **Middle Phrase:**
14   - `d2 dA df | ag/f/ e~c A2`

```

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

Rysunek 3.7.: Odpowiedź modelu na fałszywy fragment.

```
14 - The repetition of 'd' with variations in intervals suggests a thematic consistency. The leap to 'f' and back to 'd' maintains interest.
15 The movement towards 'A' anchors the phrase back to the tonic's dominant, providing harmonic stability.
16
17 3. **Transition:**
18 - `D2 d^c de | f2 ef/g/ a/g/f/e/`
19 - 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.
20
21 4. **Closing Phrase:**
22 - `d2 df e^c |1 d2 D4 :|2 d2 D2 fg`
23 - 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,
24 while the second ending moves into a continuation with 'fg', creating a sense of forward motion.
25
26 ### Second Part
27 1. **Opening Phrase:**
28 - `a2 g2 f2 | ed ^cd e/f/g`
29 - This section starts with a descending sequence, providing contrast to the first part. The melodic minor descent with a chromatic touch
30 (e-^c-d) maintains interest.
31
32 2. **Middle Phrase:**
33 - `a2 g2 f2 | ed ^cd e/f g`
34 - The repetition reinforces the theme while subtle variations in articulation (slurs) add expressiveness.
35
36 3. **Transition and Closing:**
37 - `a2 g2 f2 | ed ^cd ef/g/ | a2 g2 fe`
38 - The alternation between stepwise motion and leaps ensures coherence and variety. The repeated 'a-g-f' figure creates familiarity, while
39 'e-^c-d ef/g/' keeps it fresh.
40
41 4. **Final Cadence:**
42 - `1 d2 de fg :|2 d2 D4 ||`
43 - The final cadence resolves nicely with the repeating 'de fg' figure leading to the tonic 'D' again, ensuring a satisfying conclusion.
44
45 ### Overall Musicality
46 - **Melodic Coherence:** The melody has a clear direction with a good mix of stepwise motion and leaps, maintaining interest.
47 - **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 muzykę na główne fragmenty, a następnie każdy z tych

fragmentów poddaje szczegółowej analizie rozdzielać 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 należy również przedstawić modelowi fragment “złej” muzyki.

```

1 Based on the ABC notation provided, here is an analysis of the music in terms of musicality, harmony, and melody coherence:
2
3 ### Overall Musicality
4 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.
5
6 ### Harmony
7 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.
8
9 ### Melody Coherence
10 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.
11
12 ### Specific Sections
13 - **Measure 1**:: Starts with a simple C major outline but immediately introduces accidentals (`~g`, `~A`), which suggest movement towards
  different tonal areas.
14 - **Measure 2**:: The grace note `{g}` and the intervals used (`D1 f2 ~c2`) create a surprising twist.
15 - **Measure 3**:: The use of `G2 e1/2 ~g2 _A2` suggests a continuation of the chromatic movement, making it intriguing.
16 - **Measure 4**:: The abrupt stop with `B3` could serve as a point of tension or punctuation.
17 - **Measure 5**:: A return to `C2 e2 ~g2 [_A2C]` reintroduces some stability but with the added complexity of accidentals.
18 - **Measure 6**:: `B3 D22` maintains the movement and suggests a possible resolution.
19 - **Measure 7**:: The phrase `C2 _e2 ~g2 _A2` is quite chromatic and could be disorienting without a clear harmonic context.
20 - **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.
21
22 ### Conclusion
23 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 wyrażen jak “struktura **może** być przyjemna”, “...**może** wpływać na dynamikę”, “**niespodziewany zwrot** muzyki”. Sugeruje to że model rozumie dość nie-naturalną 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 wyłącznie 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 wykorzystywana w trenowaniu lub *fin-tuningu* modeli sztucznej inteligencji [32]. 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 posiadaienne limity, które co prawda można zwiększyć za określoną sumę. Niestety nie jest to obecnie dobrze skalujące się rozwiązanie szczególnie dla użytku niekomercyjnego. Mimo wszystko jest to możliwy sposób *fine-tuningu* modeli generatywnych muzyki, pod warunkiem odpowiedniej inżynierii *promptu* 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 [33]. 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 “odповідzią” 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 [34]. 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 generować muzykę, przejście muzycznego Testu Turinga, który wymaga głębokiej interakcji i ludzkiego zrozumienia procesu twórczego, pozostaje wyzwaniem trudnym do osiągnięcia.



## 4. Zakończenie

1. Podsumowanie.
2. Możliwości dalszego rozwoju.
3. Potencjalne obszary zastosowania pracy.



# Bibliografia

- [1] Daniel Adiwardana i in. *Towards a Human-like Open-Domain Chatbot*. 2020. arXiv: [2001.09977](https://arxiv.org/abs/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] Steve Allen. *Beethoven Symphony No. 7, Movement 2 in ABC*. URL: <https://www.ucolick.org/~sla/abcmusic/sym7mov2.html>.
- [4] 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>.
- [5] Shangda Wu i in. *TunesFormer: Forming Irish Tunes with Control Codes by Bar Patching*. 2023. arXiv: [2301.02884](https://arxiv.org/abs/2301.02884) [cs.SD].
- [6] Nitish Shirish Keskar i in. „CTRL - A Conditional Transformer Language Model for Controllable Generation”. W: *arXiv preprint arXiv:1909.05858* (2019).
- [7] Darrell Conklin. *Bach Chorales*. UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C5G>
- [8] 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>.
- [9] 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>.
- [10] John Biles. „GenJam: A Genetic Algorithm for Generating Jazz Solos”. W: *lip*. 1994.
- [11] Google AI. *Magenta*. URL: <https://magenta.tensorflow.org/>.

- [12] Adam Roberts i in. *A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music*. 2019. arXiv: [1803.05428 \[cs.LG\]](#).
- [13] Jesse Engel i in. *GANSynth: Adversarial Neural Audio Synthesis*. 2019. arXiv: [1902.08710 \[cs.SD\]](#).
- [14] Ashish Vaswani i in. *Attention Is All You Need*. 2023. arXiv: [1706.03762 \[cs.CL\]](#).
- [15] Cheng-Zhi Anna Huang i in. *Music Transformer*. 2018. arXiv: [1809.04281 \[cs.LG\]](#).
- [16] Yueyue Zhu i in. *A Survey of AI Music Generation Tools and Models*. 2023. arXiv: [2308.12982 \[cs.SD\]](#).
- [17] Botao Yu i in. *Museformer: Transformer with Fine- and Coarse-Grained Attention for Music Generation*. 2022. arXiv: [2210.10349 \[cs.SD\]](#).
- [18] Andrea Agostinelli i in. *MusicLM: Generating Music From Text*. 2023. arXiv: [2301.11325 \[cs.SD\]](#).
- [19] Jacob Devlin i in. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: [1810.04805 \[cs.CL\]](#).
- [20] 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](#).
- [21] 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. URL: [https://proceedings.neurips.cc/paper\\_files/paper/2021/file/05546b0e38ab9175cd9Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2021/file/05546b0e38ab9175cd9Paper.pdf).
- [22] Albert Gu i in. *HiPPO: Recurrent Memory with Optimal Polynomial Projections*. 2020. arXiv: [2008.07669 \[cs.LG\]](#).
- [23] 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](https://proceedings.neurips.cc/paper_files/paper/2019/file/952285b9b7e7a1be5aa7849f32ffff05-Paper.pdf).
- [24] Albert Gu i Tri Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. 2023. arXiv: [2312.00752 \[cs.LG\]](#).
- [25] Philip Gage. „A new algorithm for data compression”. W: *C Users J*. 12.2 (lut. 1994), s. 23–38. ISSN: 0898-9788.
- [26] Alec Radford i in. „Language Models are Unsupervised Multitask Learners”. W: (2019).
- [27] *The TTS Arena*. URL: <https://huggingface.co/blog/arena-tts>.

- [28] Jacob Collier. *That's not a wrong note, you just lack confidence*. URL: [https://www.youtube.com/watch?v=meha\\_FCcHbo](https://www.youtube.com/watch?v=meha_FCcHbo).
- [29] Ruibin Yuan i in. *ChatMusician: Understanding and Generating Music Intrinsically with LLM*. 2024. arXiv: [2402.16153 \[cs.SD\]](#).
- [30] OpenAI i in. *GPT-4 Technical Report*. 2024. arXiv: [2303.08774 \[cs.CL\]](#).
- [31] Maximilian Schreiner. *GPT-4 architecture, datasets, costs and more leaked*. 11 lip. 2023.
- [32] Long Ouyang i in. *Training language models to follow instructions with human feedback*. 2022. arXiv: [2203.02155 \[cs.CL\]](#).
- [33] 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).
- [34] 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](https://doi.org/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>.