Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Informatyki

Praca dyplomowa

Generacja muzyki przy pomocy dużych modeli językowych

Music generation with Lange Language Models

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

Kraków, 2024

Tutaj możesz umieścić treść podziękowań. Tutaj możesz umieścić treść podziękowań. Tutaj możesz umieścić treść podziękowań. Tutaj możesz umieścić treść podziękowań. Tutaj możesz umieścić treść podziękowań.

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 [1] …

Spis treści

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

Spis rysunków

Spis tabel

Lista algorytmów

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 tworzenia dźwięków i melodii. Jednakże, w erze cyfrowej i rozwoju sztucznej inteligencji, pojawiają się nowe możliwości w dziedzinie generacji muzyki. Modelowanie generatywne, będące obszarem sztucznej inteligencji, pozwala na tworzenie nowych danych, w tym również muzyki, na podstawie wzorców i reguł wykrytych w zbiorze treningowym. Dynamiczny rozwój dziedziny przetwarzania języka naturalnego (*NLP*) ukazuje skuteczność tego podejścia w problemach generacji danych sekwencyjnych, do których należy właśnie tekst oraz muzyka.

1.1. Motywacja

Przez ostatnie lata widzimy szybki rozwój dużych modeli językowych (*LLM*), które w szczególnie dobry sposób radzą sobie z rozumieniem tekstu w wielu językach. Modele te uczone są na wielkich korpusach danych, przez co są w stanie nauczyć się słownictwa oraz zasad gramatyki, a nastepnie ma podstawie *promptu* udzielonego przez użytkownika wygenerować odpowiednią odpowiedź. Modele te znalazły zastosowanie w maszynowej translacji tekstu, analizie sentymentu, odpowiadaniu na pytania użytkownika czy nawet 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.

1.2. Cel pracy

Celem pracy jest przyjrzenie się dużym modelom językowym oraz ich zastosowaniu w celu generacji muzyki. Zawierać będzie opis modeli, które obecnie są wykorzystywane w architekturach *LLM*-ów oraz analizę odpowiadającą na pytanie dlaczego dane rozwiązanie sprawdza się w przedstawionym w pracy problemie. Nastepnie zostanie przedstawiony problem analizy

oraz generacji muzyki oraz jej możliwe cyfrowe reprezentacje. Zostaną również przedstawione dostępne zbiory danych, na podstawie których modele mogą zostać trenowane. Praca przedstawi wyniki użytych metod w celu ustalenia, który model i która reprezentacja muzyki najlepiej nadaje się do problemu przedstawionego w pracy.

1.3. Zarys pracy dyplomowej

Uwaga 1.1. Tutaj napiszę dopiero jak będzie więcej treści bo chyba troszkę bez sensu na razie pisać o strukturze jak jej jeszcze nie ma ¯_(ツ)_/¯

2. Opis aktualnego stanu wiedzy

2.1. Podobieństwa pomiędzy muzyką a tekstem

Struktura tekstu i muzyki wykazuje wiele podobieństw, co pozwala na ich analizę za pomocą podobnych narzędzi i metod. Muzyka tak jak i tekst składa się z sekwencji znaków następujących w pewnej sekwencji. W obu przypadkach pojawiają się pewne schematy, czym w przypadku muzyki jest pewna progresja akordów lub powtarzający się fragment, a w tekście pewne związki frazeologiczne, powtarzające się wyrażenia lub ściśle określone reguły ortograficzne i składniowe. Te reguły w kontekście muzycznym są odpowiednikiem harmonii oraz strukturze i poprawności zapisu muzycznego. Sam sposób generacji sekwencji rozumianych przez komputer z muzyki zostanie poruszony w rozdziale 2.2.

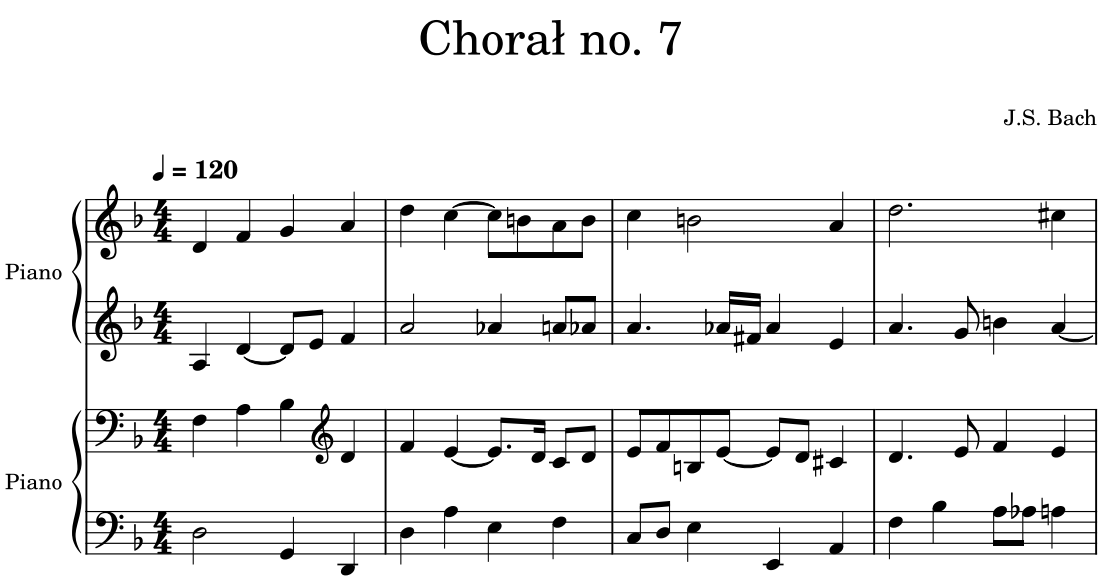
Generacja muzyki w zasadzie nie różni się od generacji tekstu. Oba modele działają autoregresyjnie. Model autoregresyjny (AR) jest modelem, który modeleuje pewne zjawisko sekwencyjne, a nastepnie iteracyjne na podstawie poprzednich sekwencji, przewiduje kolejny element, dokłada go do przewidzianej sekwencji, i ponownie "przepuszcza" ją przez model z dodatkowym elementem. Wyjściem modelu podczas jego pojedynczego kroku jest rozkład prawdopodobieństwa nad możliwymi tokenami zazwyczaj obliczanego z pomocą funkcji *softmax*.

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

Przykładowe podejście do autoregresyjnej generacji sekwencji pokazano w równaniu 2.1. 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 typowo pięciolinie są okalane przy pomocy nawiasów klamrowych.



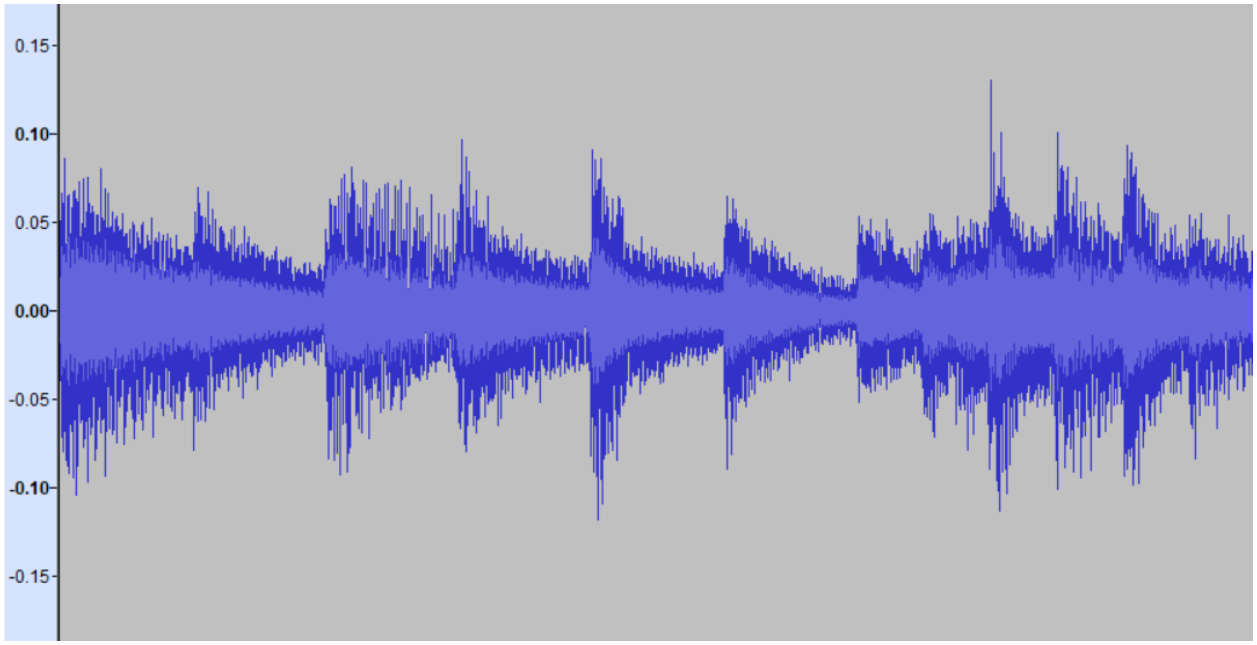
Rysunek 2.1.: Fragment chorału J.S. Bacha

Na pięciolinia umieszczone są takie elementy jak nuty, pauzy oznaczenia dynamiki, tempa i inne znaki dotyczące oznaczenia 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*) jest binarnym zapisem plików, który jest znany z tego, że jest w stanie zapisywać dźwięk nie używając żadnego algorytmu kompresji. W związku z tym rozmiary tych plików są bardzo duże, co sprawia, że ich przechowywanie ich wymaga znacznego miejsca na dysku co może być problematyczne, kiedy zbiór danych jest bardzo duży. Plik WAV jest najrzetelniejszą cyfrową reprezentacją dźwięku analogowego. Tak duży rozmiar danych zawdzięcza się temu, że dźwięk jest zapisywany z częstotliwością 44.1 kHz, czyli 44100 sampli na sekundę. Format został stworzony w roku 1991 roku, jest jednym z najbardziej rozpowszechnionym formatów i jest obsługiwany przez praktycznie każde



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

oprogramowanie edycji dźwięku. W celu wizualizacji dokonano syntezacji?? utworu przedstawionego w postaci nutowej na obrazku 2.1 w związku z czym powstała fala dźwiękowa przedstawiona na grafice 2.2.

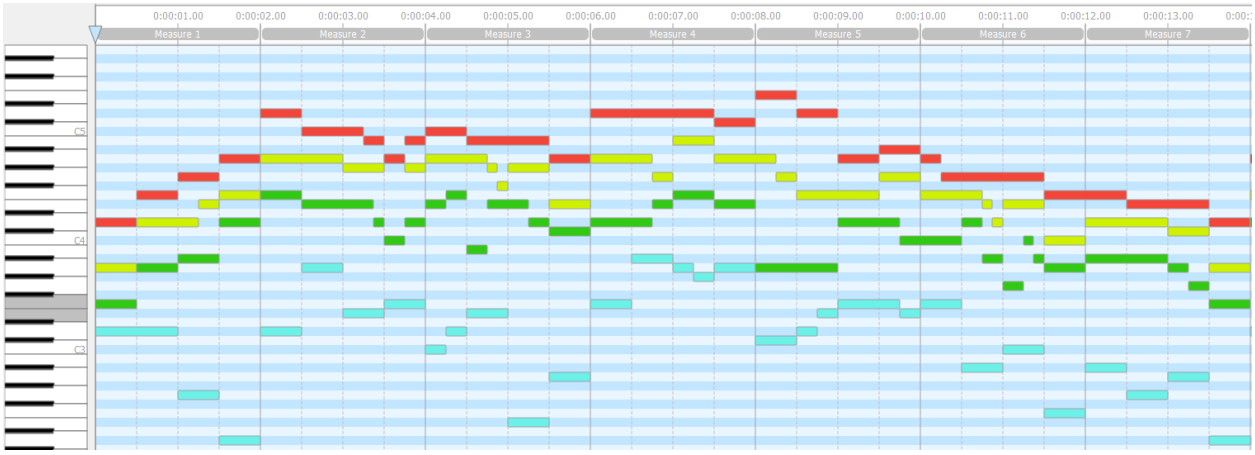
Często nie jest potrzebne przechowywanie bezstratne dźwięku, ponieważ i tak większość informacji, można zachować przy użyciu mniejszej ilości danych. Jednym z najpopularniejszych standardów kodowania stratnego jest MP3. Kompresja zmniejsza dokładność kodowania jak i również "ucina"częstotliwości, które nie są słyszalne dla człowieka. Stratne kodowanie próbuje zachować równowagę pomiędzy jakością dźwięku a rozmiarem pliku. W kontekście uczenie maszynowego, zmniejszona ilość sampli przy zachowaniu większości informacji jest zjawiskiem pożądanym, ponieważ udaje nam się przynajmniej częściowo rozwiązać problem związany z "przekleństwem wymiarowości".

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

MIDI jest standardem, który opisuje protokół komunikacji, interfejs cyfrowy oraz złącze pozwalające połączyć ze sobą elektroniczne instrumenty, komputery oraz inne muzyczne peryferia. Pojedynczy kabel MIDI jest w stanie przekazać informacje na temat szesnastu na raz nadających kanałów z czego każdy może pochodzić z innego instrumentu. Każda interakcja z instrumentem, czyli przykładowo naciśnięty klawisz, szarpnięta struna, jest zapisana jako

"*event*", który zachowuje wartości takie jak konkretny znacznik czasu, wysokość dźwięku czy jego głośność. Dane pochodzące z urządzeń są zapisywane w specjalnym pliku, głównie o rozszerzeniu *.mid* lub *.midi*. Plik pozwala na przechowanie, rozpowszechnianie jak i również edycje dźwięków. W związku z tym, że w pliku nie jest przechowywany zapis konkretnej fali dźwiękowej zapisanej przez mikrofon, pozwala to na późniejszą zmianę np. instrumentu, który będzie odtwarzał zapisane dźwięki.

Typowym przedstawieniem zapisanych danych jest tak zwany *piano roll*, który można porównać do dwuwymiarowego układy współrzędnych, gdzie wspołrzedna horyzontalna opisuje czas, a horyzontalna konkretne dźwięki pokazane jako klawisze pianina. Przykładowe porównanie zapisu MIDI z zapisem nutowym przedstawiono na grafice 2.3 oraz 2.1.



Rysunek 2.3.: Muzyka zapisana w pliku MIDI

Warto zaobserwować, że zapis pliku w formacie MIDI umożliwia zdecydowanie większą ekspresje, 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 jest systemem zapisywania nut w postaci czystego tekstu znakami ASCII. Notacja pojawiła sie w latach 70 XX wieku w celu zapisu i nauki tradycyjnych irlandzkich melodii, a nastepnie w kolejnej dekadzie została rozwinięta przez Chrisa Walshawa, który zapisywał w niej tradycyjne melodie zanim nauczył się standardowego zachodniego zapisu nutowego. Zapis ten posłużył do stworzonego przez niego programu *abc2mtex*, który na podstawie notacji ABC generował komendy pozwalające zapis partytur w postaci *MusicTex*. Obecnie używanym standardem jest wersja z roku 2011.

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

Rysunek 2.4.: Zapis wielu głosów w notacji ABC.

Jak widać notacja ABC w dość zwięzły sposób zapisuje partyturę. Poza konkretnymi nutami oraz rytmem w tym formacie możemy zapisac również dodatkowe informacje na temat utworu. Każda linijka zaczynająca się się od znaku A-Z a następnie dwukropkiem jest tak zwanym "polem informacyjnym". W tych polach zapisywac można takie informacje jak tytuł utworu lub metrum oraz wiele innych informacji między innymi dotyczące z jakiego zbioru muzycznego pochodzi dany utwór. Wiele z tych informacji powinna być usunięta w procesie preprocessingu danych, tak aby model dostał tylko te informacje, które rzeczywiście pomogą w nauce struktury muzyki. Do takich pól należy przede wszystkim domyślna długość nuty (`L:`), metrum (`M:`) oraz tonacja (`K:`) w jakiej utwór został napisany. Każda z tych informacji jest możliwa do "odgadnięcia" przez model, jednak podając modelowi te informację wyraźnie, mamy większą kontrolę nad procesem treningu. Dodatkową zaletą podawania tych informacji podczas treningu jest możliwość podania ich jako początkowa sekwencja na podstawie której model dalej będzie próbował kończyć melodię, przez co mamy kontrolę np. nad tym w jakiej tonacji i w jakim metrum zostanie wygenerowany nasz utwór.

Notacja ABC wpiera również melodie polifoniczne przy pomocy tagów V. Ich ilość nie jest ograniczona w związku z czym można w tym zapisie jest możliwe ujęcie nawet muzyki orkiestrowej. Dość skrajnym przykładem jest zapis drugiej części VII symfonii Ludwiga van Beethovena, która składa się aż z 19 instrumentów [2].

2.2.3.1. Tokenizacja plików MIDI

Token jest odrębnym elementem, częścią sekwencji tokenów. W języku naturalnym tokenem może być znak, zaimek lub słowo. Zdanie może być następnie tokenizowane na sekwencję tokenów reprezentujących słowa i znaki interpunkcyjne. W przypadku muzyki tokeny mogą reprezentować wartości atrybutów nut (wysokość, wartość, czas trwania) lub zdarzenia czasowe. Token może przyjmować jedną z trzech form:

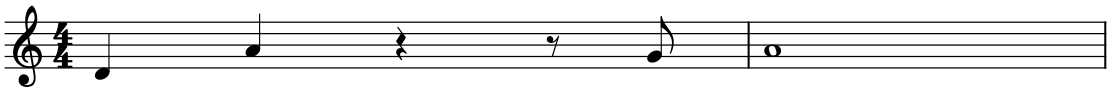
- nazwa tokenu - słowna reprezentacja *eventu* MIDI np. *Pitch_50*
- id - unikalna wartość liczbowa przypisana konkretnemu zdarzeniu
- bajt - unikalny bajt, który został przydzielony podczas treningu tokenizera

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

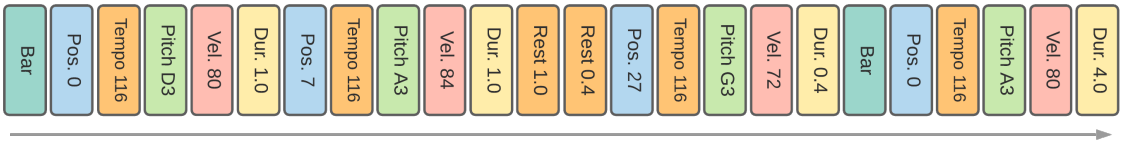
Poza tokenami, które tokenizer tworzy na podstawie pliku MIDI dodaje on również dodatkowe takie jak:

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

Istnieje wiele algorytmów tokenizacji MIDI, jednak aby przedstawić mechanizm działania, przyjrzymy się popularnemu algorytmowi 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 [3].



Rysunek 2.5.: Prosta sekwencja muzyczna



Rysunek 2.6.: Reprezentacja muzyki w postaci tokenów

2.2.4. Porównanie zapisu muzycznego

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

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

Tabela 2.1.: Porównanie różnych zapisów muzyki

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 2.2.3.1. Przy uzyciu 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 widoczna, jenak oba te formaty są zdecydowanie lepszymi rozwiązaniami w przypadku generacji notacji muzycznej (dziwne zdanie do poprawy raczej). Warto zwrócić uwagę, że gdyby w oryginalnym utworze pojawiły się powtórzenia danych sekwencji melodii notacja ABC byłaby jeszcze bardziej oszczędna w obu przypadkach 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 odtworzona przez artystę.

Do głównej zalety plików MIDI należy ich wszechstronność. Nie są one ograniczone do kodowania sztywno określonego rytmu tak jak w notacji muzycznej. Dodatkowo jest bardzo

łatwo przy użyciu narzędzi komputerowych zamienić dowolny zapis nutowy (MusicXML, ABC) na plik MIDI. Niestety w drugą stronę nie jest już tak łatwo, właśnie w związku z nie konkretnym rytmem, który może zostać zakodowany w MIDI. Istnieją narzędzia takie jak np. MuseScore, które jest w stanie wygenerować partyturę z pliku MIDI, jednak nie jest ono w każdym przypadku jednakowo skuteczne.

Plik w postaci fali dźwiękowej nie jest idealnym sposobem zapisu muzyki, jednak jest to często jedyne rozwiązanie aby zebrać dużą ilość danych. Szczególnie jeśli mamy do czynienia z muzyką nowoczesną zazwyczaj twórcy nie udostępniają plików MIDI lub zapisu nutowego na podstawie którego powstała dana piosenka. Z tego powodu zostaje albo praca z plikami `mp3` lub czasochłonna translacja muzyki na jej zapis w innym formacie. Dodatkowo zaletą plików z dzwiekiem w zapisie cyfrowym jest dostęp do potencjalnie istniejącego zapisu odśpiewanego tekstu. Co prawda przykładowo zapis w notacji ABC pozwala na przekazanie tekstu w tagu `W:`, jednak jest to tylko jego zapis a nie faktyczne odśpiewanie. Z tego powodu pliki `wav` nadają się np. do próby replikacji czyjegoś głosu.

Główną zaletą plików ABC jest ich tekstowa reprezentacja notacji muzycznej. Z tego powodu nie wymagaja one wiedzy na temat struktury, budowy oraz standardów plików np. MIDI aby zacząć z nimi pracować. Ponieważ notacja ABC jest tekstowa, można ją łatwo integrować z narzędziami do przetwarzania tekstu. Zwięzłość zapisu jest również przyczyną dla której model uczenia maszynowego może być mniejszy w związku z 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*[4], który wziął przykład z artykułu CTRL [5], w którym autorzy dodawali do korpusu treningowego takie tagi jak *Books*, *Horror*, *Relationships*, *Legal*. Tagi te były podawana 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 `[|`, `||`, `|]`, `|:`, `::` 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

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

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

Policzone kody wraz z ich odpowiednim oznaczeniem dodawane są do pliku tekstowego, który zapisuje notacje. Dodanie ich nie 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 uzytkownikowi kontrolę nad tymi parametrami.

2.3. Zbiory danych

2.3.1. Johann Sebastian Bach Chorales

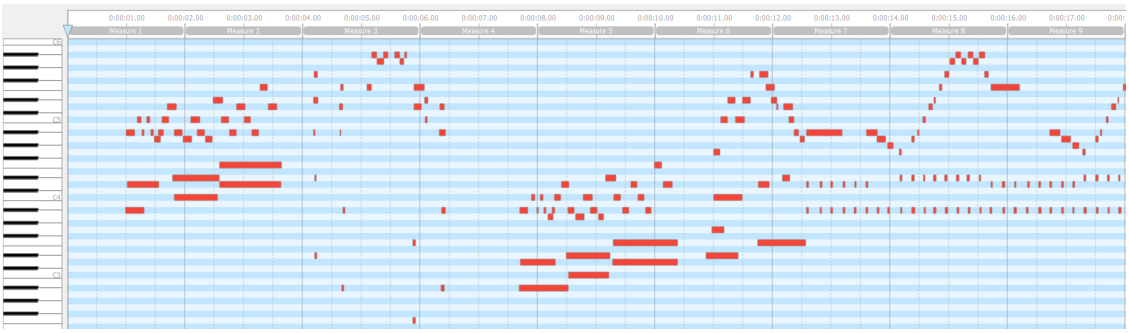
JSB Chorales [6] to zbiór krótkich, czterogłosowych utworów. Chorały zostały pierwotnie skomponowane przez Johanna Sebastiana Bacha w XVIII wieku. Napisał je najpierw biorąc wcześniej istniejące melodie ze współczesnych hymnów luterańskich, a następnie harmonizując je w celu stworzenia części dla pozostałych trzech głosów. 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× 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 wystepuje.

2.3.2. The MAESTRO v3.0

Zbiór danych MAESTRO w wersji trzeciej [7] 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.

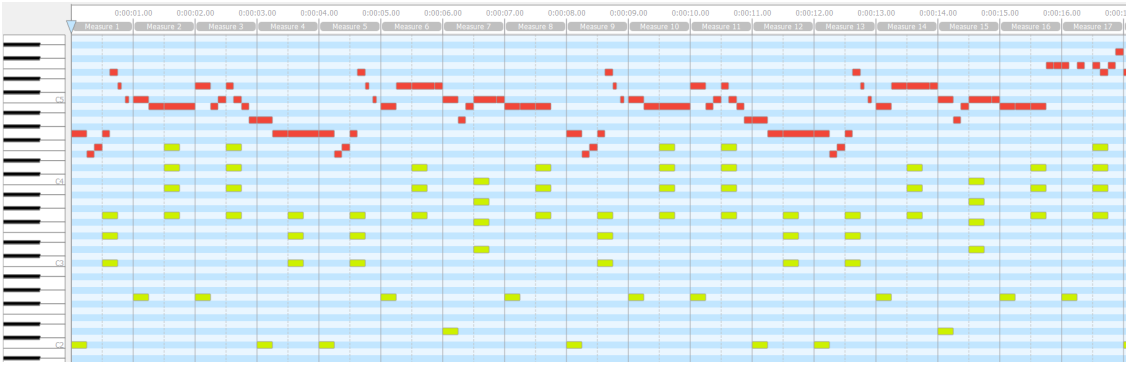


Rysunek 2.7.: Przykład utworu pochodzącego ze zbioru MAESTRO.

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

2.3.3. IrishMAN

Zbiór danych IrishMAN (*ang. Irish Massive ABC Notation*) [8] zawiera 216,248 fragmentów melodii podzielonych na zbiór treningowy i walidacyjny. Melodie zebrane są ze stron takich jak `thesession.org` oraz `abcnotation.org`, które znane są z udostepniania tradycyjnej muzyki. Aby zapewnić jednolitość formatowania najpierw wszystkie utwory zostały przekonwertowane do notacji MusicXML, a nastepnie 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 łamania 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 ustostępnia te same fragmentu muzyki w różnych formatach, przez co nadaje się on wyjątkowo dobrze do porównywania różnych algorytmów generacyjnych.

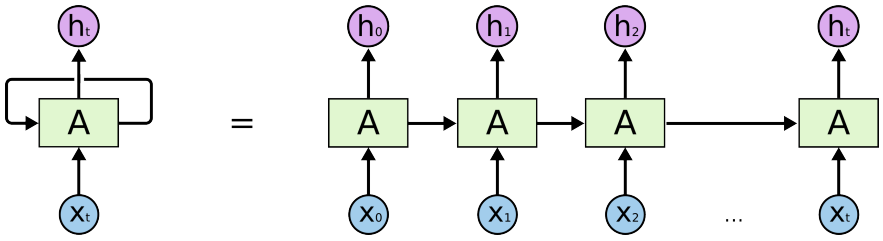


Rysunek 2.8.: Przykład wielogłosowego fragmentu ze zbioru IrishMAN.

2.4. Architekturze transformera

2.4.1. 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 pomiedzy ważnymi informacjami jest mała. Jednak dla zdania "Rok temu ukończyłem kurs lotnika ... Teraz staram się o licencje *pilota*" wymagany jest szersze spojrzenie na całą strukturę wypowiedzi i "przypomnienie" informacji z początku zdania. W przypadku muzyki sytuacja ma się dokładnie tak samo. Przykładowo, aby rozwiązać akord dysonansowy, 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.

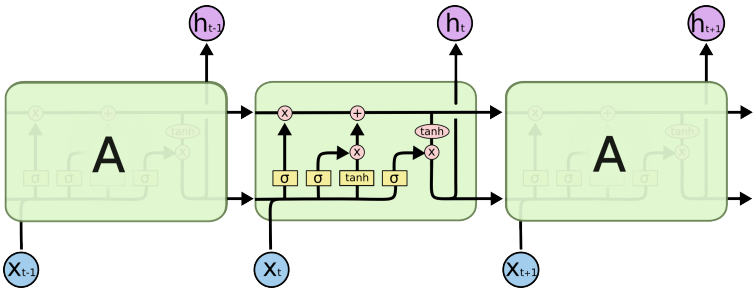
Dodatkowym problemem podczas treningu modeli jest problem zanikającego gradientu.

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

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

2.4.2. 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 σ. Wartościami zwrotnymi jest liczba pomiędzy 0 a 1 gdzie

mała wartość znaczy niską ważność, 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 błędu nazywanego "wsteczną propagacją błędu w czacie" *backpropagation through time*, która rozwija sieci rekurencyjne, aby propagować gradienty.

Istnieje kilka wariantów LSTM-ów np. dwukierunkowe LSTM, które pozwalają aby informacja przepływa w dwóch kierunkach na raz a nie tylko z lewej strony do prawej. 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.

Uwaga 2.1. Można cos więcej opisać ale czy jest po co? Jak będzie brakowało stron a będzie czas to się dopisze ¯_(ツ)_/¯Plus trzeba dodać cytowania

2.4.3. Algorytm uwagi (ang. *attention*)

Mechanizm uwagi (*ang. attention*) został zaproponowany w artykule "Attention is All You Need" [1]. Algorytm ten działa w ustalonym oknie kontekstowym (*ang. context window*), w którym przydziela każdemu słowu wagę reprezentującą jego ważność w danym kontekście. Metoda ta imituje sposób, w jaki człowiek skupia się na różnych fragmentach tekstu w danym momencie. Mechanizm uwagi został opracowany, aby rozwiązać problem występujący w rekurencyjnych sieciach neuronowych (*RNN*), gdzie "ważność"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

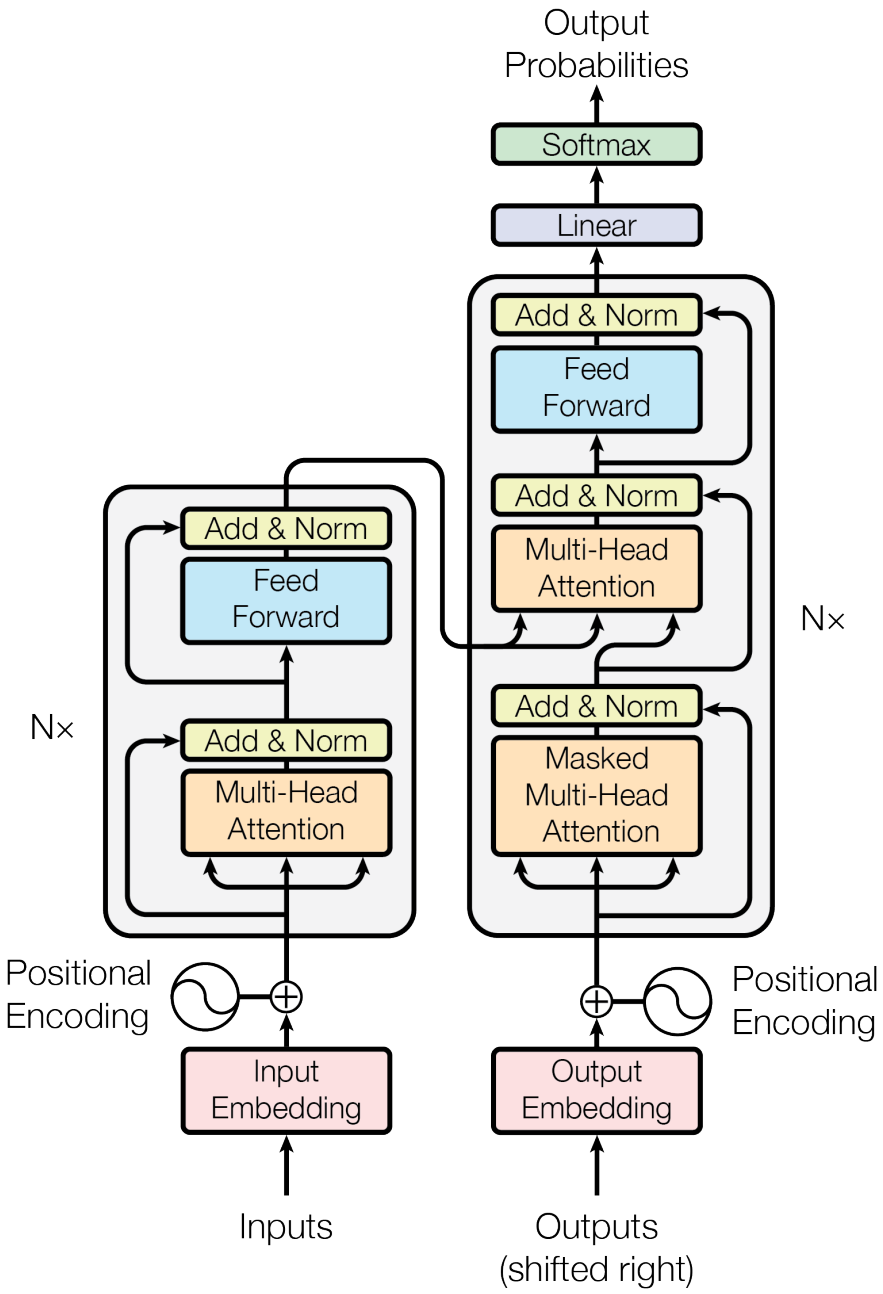
krawędziami są wartości określające, jak istotne są dla danego słowa wszystkie inne. Jest to zdecydowanie lepsze podejście niż w rekurencyjnych sieciach neuronowych lub LSTM-ach, gdzie zależności dla danego słowa są przekazywane jedynie w wektorze, który kompresuje kontekst ze wszystkich poprzednich słów. Dzięki temu mechanizm uwagi pozwala na bardziej dynamiczną kontrolę i monitorowanie, jak zależności pomiędzy tokenami formują się podczas treningu.

2.4.4. Budowa transformera

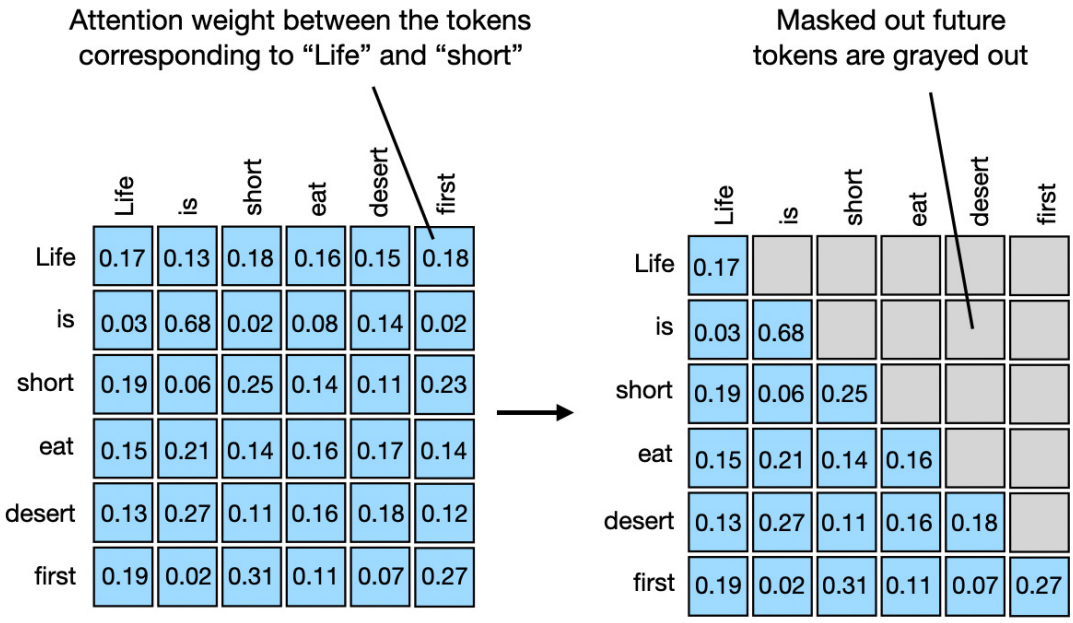
Wraz z wprowadzeniem algorytmu uwagi pisanego w 2.4.3 w tym samym artykule został zaproponowana architektura modelu, która wykorzystywała *multi-headed* algorytm uwagi. Cały model składa się z enkodera oraz dekodera.

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.

Zadaniem Dekoder w architekturze transformerowej jest generowanie wyjścia na podstawie sekwencji wejściowej i dotychczas wygenerowanych tokenów wyjściowych. Podobnie jak w enkoderze, mechanizm samo-uwagi analizuje zależności między tokenami w sekwencji wyjściowej, jednak w dekoderze stosuje się maskowanie (*ang. masking*), aby zablokować dostęp do przyszłych tokenów podczas generowania bieżącego tokena. Maskowanie uniemożliwia modelowi zobaczenie tokenów, które są generowane w późniejszych krokach. Maskę można zilustrować jako macierz trójkątną, co zostało pokazane na grafice 2.12. Drugi mechanizm uwagi umożliwia dekoderowi skupienie się na odpowiednich częściach sekwencji wejściowej (wyjścia enkodera) podczas generowania każdego tokena wyjściowego. Proces ten jest podobny do mechanizmu samo-uwagi, ale tutaj zapytania (`Q`) pochodzą z poprzedniej warstwy uwagi dekodera, a klucze (`K`) i wartości (`V`) pochodzą z wyjść enkodera. Następnie tak jak w poprzednim przypadku wystepuje 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 2.11.: Schemat transformera.



Rysunek 2.12.: Algorytm uwagi z maską.

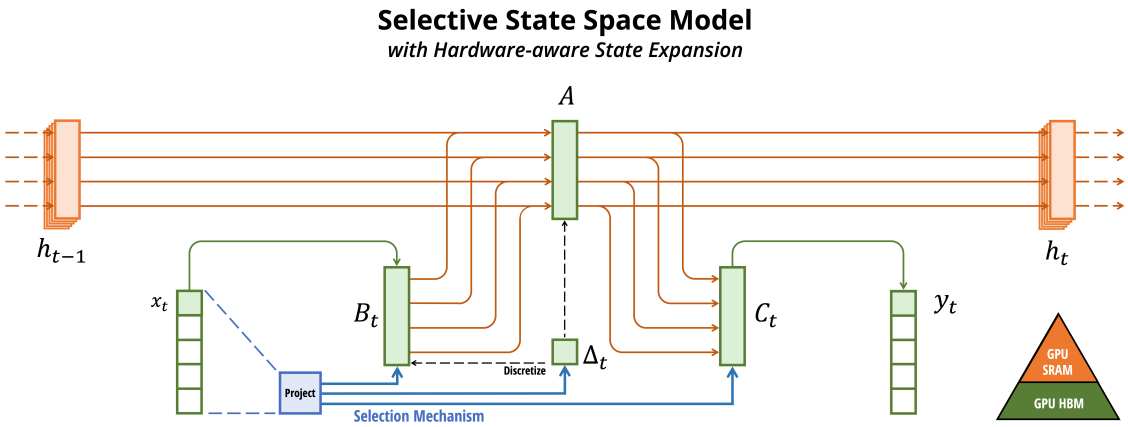
2.5. Architektura *state space*

2.5.1. Mamba

2.5.2. Tutaj się rozdrobnić trzeba

Uwaga 2.2. Tytuł oraz strukturę rozdziału należy ustalić z opiekunem pracy.

Aktualny stan wiedzy, na dany temat, na podstawie dostępnej literatury naukowej oraz specjalistycznej.



Rysunek 2.13.: Schemat modelu Mamba.

3. Propozycja rozwiązania

4. Przeprowadzenie eksperymentów

Uwaga 4.1. Tytuł oraz strukturę rozdziału należy ustalić z opiekunem pracy.

4.1. Opis *pipeline-u*

Tutaj zamierzam opisać w jaki sposób modele zostały stworzone, jakie bilbioteki zostały użyte, jaki sprzęto został użyty podczas treningu

4.2. Porównanie architektur użytych modeli

4.3. Prezentacja otrzymanych wyników

4.4. Porównanie 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*[9], która pozwala na ranking modeli *text-to-speech*.

W przypadku muzyki, istnieje możliwość aby zweryfikować poprawność wygenerowanych sekwencji odwołując się do teorii oraz harmonii muzyki. Istnieje kilka narzędzi takich jak *Chordify*, *Hooktheory* lub *Sibelius*, które pozwalają na analizę harmoniczną utworów, dzieki czemu autorzy mogą w prosty sposób analizować i dobierać progresję danej melodii. Niestety większość takich narzędzi jest płatna i nie pozwala na zautomatyzowaną analizę wielu plików. Dodatkowym problemem jest w analizie harmonicznej, szczególnie prowadzonej przez algorytmy, jest rozróżnienie harmonii wertykalnej oraz horyzontalnej. Rozróżnienie to zostało wytłumaczone przez Jacoba Colliera, kilkukrotnego laureata nagród *Grammy*, którego zdaniem akord, który nawet zagrany sam brzmi niepoprawnie, w odpowiednim kontekście i przez odpowiednią progresję w kolejnych fragmentach muzyki, może mieć nadany sens, przez co cała sekwencja nabiera muzycznego piękna[10].

Uwaga 4.2. W mojej pracy prawdopodobnie zostanie zastosowane podejście MOS dla dość niewielkiej grupy ludzi, jednak jeśli triale oprogramowania pozwola, spróbuję przynajmniej sprawdzić czy taka analiza pozwala na jakieś sensowne obliczenie metryki

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.

5. Zakończenie

Uwaga 5.1. Tytuł oraz strukturę rozdziału należy ustalić z opiekunem pracy.

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

Dodatek A.

Typowe elementy składowe pracy dyplomowej z informatyki

A.1. Tabele

Uwaga A.1.

- Każda tabela powinna być opisana w treści pracy.
- Podpis ma być przed tabelą.

W tabeli A.1 przedstawiono wyniki pomiarów.

Tabela A.1.: Pomiary zużycia energii elektrycznej.

L.p.	**Wartość**
1	12345,6789
	45,89
2	45,678901

Jeżeli tabela zawiera dużą liczbę wierszy i może nie zmieścić się na stronie — patrz tabela A.2 — skorzystaj z pakietu *longtable* [11].

Tabela A.2.: Tabela, która zawiera dużą liczbę wierszy.

	1	2	3	4	5	6	7	8	
Student 1									

	1	2	3	4	5	6	7	8	
Student 2									
Student 3									
Student 4									
Student 5									
Student 6									
Student 7									
Student 8									
Student 9									

Tabele, w których występuje długi tekst, a co za tym idzie może się on nie zmieścić — musi zostać zawinięty, z pomocą przychodzi środowisko 'tabularx' [12] — patrz tabela A.3.

Tabela A.3.: Tabela zawierająca długi tekst.

Wpis wielokolumnowy!		TRZY	CZTERY
jeden	Szerokość tej kolumny zależy od szerokości tabeli.	trzy	Kolumna czwarta będzie zachowywać się w taki sam sposób jak druga kolumna o tej samej szerokości.

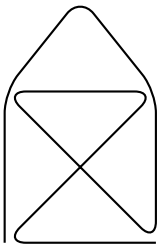
A.2. Rysunki

Uwaga A.2.

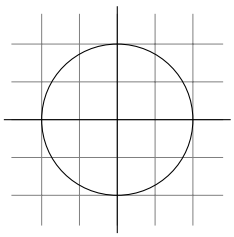
- Rysunki powinny być przerysowane samodzielnie albo używane tylko te, których twórcy zezwolili na ich rozpowszechnianie oraz kopiowanie, czyli np. rysunki objęte licencją Creative Commons.
- Każdy rysunek powinien być opisany w treści pracy.

A.2.1. Wewnętrzne

Klasa *agh-wi*, automatycznie, dołącza pakiet *TikZ* [13] — dostarcza on komend pozwalających na tworzenie grafik. Przykładowe grafiki pokazano na rysunku A.1 oraz A.2.



Rysunek A.1.: Prosty rysunek *TikZ*.



Rysunek A.2.: Bardziej złożony rysunek *TikZ*.

Oprócz rysunków eksponowanych możliwe jest tworzenie grafik będących częścią zdania.

TikZ pozwala również na kreślenie po powierzchni strony, np. możemy narysować strzałki pomiędzy elementami strony.

Ułamek — patrz wzór A.1 — składa się z:

- Licznika
- Mianownika

$$a = \frac{x + y}{y - z} \tag{A.1}$$

A.2.2. Zewnętrzne

Oczywiście możliwe jest również dołączanie rysunków zewnętrznych — pakiet *graphicx* [14] pozwala na wstawianie grafik zapisanych w plikach: '.png', '.jpg' oraz '.pdf'. Rysunek A.3 wstawiono przy użyciu tego pakietu.

Rysunek A.3.: Logo Wydziału Informatyki.

A.3. Kody źródłowe

Najpopularniejszymi pakietami, które umożliwiają składanie kodów źródłowych programów, są:

listings [15] — kod źródłowy jest formatowany bezpośrednio przez LaTeX-a — nie jest używany żaden, zewnętrzny, formater kodu.

Kod źródłowy A.1: Przykładowy kod źródłowy sformatowany za pomocą pakietu 'listings'.

```
1 /* Pierwszy program w C++ */
2
3 #include <iostream>
4
5 int main() {
6 std::cout << "Hello World!";
7 return 0;
8 }
```

minted [16] — formatuje kod źródłowy przy użyciu biblioteki języka Python o nazwie *Pygments* [17].

Kod źródłowy A.1.: Przykładowy listing sformatowany za pomocą pakietu 'minted'.

```
1 /* Pierwszy program w C++ */
2
3 #include <iostream>
4
5 int main() {
6 std::cout << "Hello World!";
7 return 0;
8 }
```

Uwaga A.3.

- Podpis ma być przed kodem źródłowym.
- Proszę używać tylko jednego z tych pakietów; w przeciwnym razie otrzymasz taki efekt, jak w przykładowej pracy — obydwa listingi mają ten sam numer.

Kod źródłowy w C++ sformatowany przy użyciu pakietu *listings*, pokazano na listingu A.1; sformatowany przy użyciu pakietu *minted*, pokazano na listingu A.1.

A.4. Algorytmy

Pakiet *algorithm2e* [18] to jeden z kilku, które pozwalają zapisywać algorytmy w formie pseudokodu — patrz algorytm 1.

Uwaga A.4. Podpis ma być przed algorytmem.

Algorytm 1: Disjoint decomposition.

input : A bitmap Im of size $w \times l$
output: A partition of the bitmap

```
1  special treatment of the first line;
2  for i ← 2 to l do
3  |  special treatment of the first element of line i;
4  |  for j ← 2 to w do
5  |  |  left ← FindCompress(Im[i, j − 1]);
6  |  |  up ← FindCompress(Im[i − 1, ]);
7  |  |  this ← FindCompress(Im[i, j]);
8  |  |  if left compatible with this then // O(left,this)==1
9  |  |  |  if left < this then Union(left,this);
10 |  |  |  else Union(this,left);
11 |  |  end
12 |  |  if up compatible with this then                    // O(up,this)==1
13 |  |  |  if up < this then Union(up,this);
   |  |  |  // this is put under up to keep tree as flat as possible
14 |  |  |  else Union(this,up);
   |  |  |  // this linked to up
15 |  |  end
16 |  end
17 |  foreach element e of the line i do FindCompress(p);
18 end
```

A.5. Wzory

LaTeX bardzo dobrze sprawdza się w przypadku prac dyplomowych zawierających wzory matematyczne[1].

[1] W przypadku złożonych wzorów warto zastosować pakiet *amsmath* [19].

A.5.1. Przykłady

Wzór $E = mc^2$ jest częścią zdania.

$$\left|\sum_{i=1}^{n} a_i b_i\right| \le \left(\sum_{i=1}^{n} a_i^2\right)^{1/2} \left(\sum_{i=1}^{n} b_i^2\right)^{1/2} \tag{A.2}$$

Wartości zmiennej opisano wzorem A.3.

$$x = \begin{cases} y & \text{dla } y > 0 \\ \frac{z}{y} & \text{dla } y \le 0 \end{cases} \tag{A.3}$$

Wzór A.4 to wzór wielowierszowy.

$$\begin{aligned} 2x^2 + 3(x-1)(x-2) &= 2x^2 + 3(x^2 - 3x + 2) \\ &= 2x^2 + 3x^2 - 9x + 6 \\ &= 5x^2 - 9x + 6 \end{aligned} \tag{A.4}$$

Uwaga A.5. Należy używać tylko dwóch rodzajów wzorów:

1. „W linii”.
2. Eksponowane, numerowane.

A.6. Twierdzenia i podobne struktury

Twierdzenie nr 1 opublikował, w roku 1691, francuski matematyk Michel Rolle.

Twierdzenie 1 (Rolle’a) *Jeśli dana funkcja f:* $\mathbb{R} \to \mathbb{R}$ *jest:*

1. *ciągła w przedziale* $[a, b]$
2. *jest różniczkowalna w przedziale* (a, b)
3. *na końcach przedziału* $[a, b]$ *przyjmuje równe wartości:* $f(a) = f(b)$,

to w przedziale (a, b) *istnieje co najmniej jeden punkt* c *taki, że* $f'(c) = 0$.

Teraz coś z informatyki …

Definicja 1 *Bit to najmniejsza jednostka informacji w komputerze.*

Definicja 2 *Bajtem nazywamy ciąg ośmiu bitów.*

Uwagi Autora

- Aktualna wersja klasy jest dostępna pod adresem https://github.com/polaksta/LaTeX/tree/master/agh-wi[1].
- Skoro Twoja praca dyplomowa powstała w LaTeXu, to zachęcam Cię również do przygotowania prezentacji (na obronę pracy magisterskiej) w tym języku. Najpopularniejszą klasą do tworzenia tego typu dokumentów jest *beamer* [20].
- Pod adresem https://github.com/polaksta/LaTeX/tree/master/beamerthemeAGH[2] możesz znaleźć, stworzony przeze mnie, nasz uczelniany szablon dla prezentacji LaTeX Beamer.
- Treść wszystkich rozdziałów tej, przykładowej, pracy dyplomowej znajduje się w jednym pliku — **nie jest to polecane rozwiązanie**. W przypadku pisania własnej pracy warto umieścić zawartość każdego z rozdziałów w osobnych plikach, a następnie dołączać je do dokumentu głównego — patrz opis na stronie https://www.dickimaw-books.com/latex/thesis/html/include.html.
- Jeżeli pewne elementy mają być wyróżniane w jednakowy sposób, to proponuję nie używać bezpośredniego stylowania, tzn.

```
\colorbox{red!50}{jednakowy} \colorbox{red!50}{sposób}
```

  ale zdefiniować własną komendę stylującą, np. `\alert`,

```
\newcommand{\alert}[1]{\colorbox{red!50}{#1}}
```

  a następnie użyć jej w dokumencie.

```
\alert{jednakowy} \alert{sposób}
```

  Dzięki temu, jeżeli będziesz chciał / chciała zmienić sposób stylowania tych elementów, np. niebieskie tło zamiast czerwonego, to wystarczy zmodyfikować, tylko, definicję komendy, zamiast zastępować, w tekście pracy dyplomowej, wybrane (niekoniecznie wszystkie!) wystąpienia tekstu `red`, tekstem `blue`.

[1] W przypadku Overleaf-a jest ona pod adresem https://www.overleaf.com/read/fnvcvqjyrbyw#5ac622
[2] W przypadku Overleaf-a jest on pod adresem https://www.overleaf.com/read/fkjdthnbrfhj#9c6184

Stanisław Polak

Bibliografia

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

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

[3] 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.

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

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

[6] Darrell Conklin. *Bach Chorales*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5G

[7] 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.

[8] 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.

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

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

[11] *The longtable package*. URL: http://mirrors.ctan.org/macros/latex/required/tools/longtable.pdf.

[12] *The tabularx package*. URL: http://mirrors.ctan.org/macros/latex/required/tools/tabularx.pdf.

[13] *The TikZ and PGF Packages*. URL: http://mirrors.ctan.org/graphics/pgf/base/doc/pgfmanual.pdf.

[14] *Packages in the 'graphics' bundle*. URL: http://mirrors.ctan.org/macros/latex/required/graphics/grfguide.pdf.

[15] *The Listings Package*. URL: http://mirrors.ctan.org/macros/latex/contrib/listings/listings.pdf.

[16] *The minted package: Highlighted source code in LaTeX*. URL: http://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf.

[17] *Strona WWW biblioteki „Pygments"*. URL: https://pygments.org/.

[18] *algorithm2e.sty — package for algorithms*. URL: http://mirrors.ctan.org/macros/latex/contrib/algorithm2e/doc/algorithm2e.pdf.

[19] *User's Guide for the amsmath Package*. URL: http://mirrors.ctan.org/macros/latex/required/amsmath/amsldoc.pdf.

[20] *The beamer class*. URL: http://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf.