STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor: 18. Informatika

Predikce sekundární struktury proteinu pomocí strojového učení

Filip Semrád

Pardubice 2025

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

PREDIKCE SEKUNDÁRNÍ STRUKTURY PROTEINU POMOCÍ STROJOVÉHO UČENÍ

PREDICTION OF PROTEIN SECONDARY STRUCTURE USING MACHINE LEARNING

AUTOR	**Filip Semrád**
ŠKOLA	**DELTA - Střední škola informatiky a ekonomie**
KRAJ	**Pardubický**
ŠKOLITEL	**RNDr. Terézia Slanináková**
OBOR	**18. Informatika**

Pardubice 2025

Prohlášení

Prohlašuji, že svou práci na téma *Predikce sekundární struktury proteinu pomocí strojového učení* jsem vypracoval/a samostatně pod vedením RNDr. Terézia Slanináková a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Dále prohlašuji, že tištěná i elektronická verze práce SOČ jsou shodné a nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a změně některých zákonů (autorský zákon) v platném znění.

V Pardubicích dne: _____

_____

Filip Semrád

Poděkování

Tímto bych chtěl poděkovat vedoucí mé odborné práce paní RNDr. Terézii Slaninákové, která mi poskytla velice cenné informace k zpracování práce.

Abstrakt

Tato práce se zaměřuje na predikci sekundární struktury proteinů pomocí strojového učení, což je klíčová oblast bioinformatiky s významnými aplikacemi v lékařském výzkumu a farmaceutickém průmyslu. Sekundární struktura proteinů, zahrnující alfa-helixy, beta-listy a smyčky, je zásadní pro pochopení jejich funkcí a interakcí. Kombinace konvolučních neuronových sítí (CNN) a rekurentních neuronových sítí (RNN) byla navržena jako efektivní metoda pro zvýšení přesnosti predikce.

Práce dosáhla významného pokroku oproti tradičním metodám. Model kombinující CNN a RNN dosáhl přesnosti až 81 % při predikci osmi tříd sekundární struktury (Q8) na testovacím datasetu CB513, což představuje zlepšení o 7 % oproti čistým CNN. Tento přístup efektivně zachycuje prostorové i sekvenční vzorce v proteinových sekvencích, přičemž si zachovává nízkou výpočetní náročnost.

Součástí projektu byla také vývoj webové aplikace, která umožňuje uživatelům snadno využívat vyvinuté modely pro predikci sekundární struktury proteinů. Aplikace zahrnuje intuitivní uživatelské rozhraní, REST API pro integraci s dalšími bioinformatickými nástroji a vizualizaci výsledků. Výsledky byly validovány na standardních datasetech a ukazují robustnost modelů napříč různými typy proteinů.

Závěrem práce zdůrazňuje potenciál kombinace CNN a RNN pro zlepšení predikce sekundární struktury proteinů a otevírá nové směry pro budoucí výzkum, včetně integrace dalších biologických parametrů, optimalizace modelů pro specifické typy proteinů a zlepšení interpretovatelnosti rozhodovacích procesů neuronových sítí. Tento projekt přispívá k pokroku v oblasti strukturní bioinformatiky a poskytuje cenný nástroj pro vědeckou komunitu. `https://github.com/Faluminus/Sec-PRED`

Obsah

Úvod

1 Uvedení do problematiky

Predikce sekundární struktury proteinů je dlouhotrvající problém v bioinformatice. Sekundární struktura proteinu, jako jsou alpha-helixy, beta-listy a smyčky, je klíčová pro funkci a interakce proteinu. Přesná predikce těchto struktur je důležitá pro pochopení biologických procesů a pro aplikaci v lékařském výzkumu.

V průběhu let došlo k významnému pokroku v metodách predikce sekundární struktury. Tradiční techniky, jako jsou Chou-Fasmanova a GOR metoda, byly postupně nahrazeny pokročilejšími přístupy využívajícími strojové učení a hluboké neuronové sítě. Současné špičkové metody dosahují přesnosti kolem 74 % při predikci osmi tříd sekundární struktury (Q8) na testovacím datasetu CB513. V této práci je navržen nový přístup k predikci sekundární struktury, který kombinuje dva typy neurálních sítí: konvoluční neuronovou síť (CNN) a rekurentní neuronovou síť (RNN), přesněji upravenou verzi zvanou Long short term memory (LSTM). Na testovacím datasetu CB513 dosahuje samotná CNN přesnosti 74 % při predikci osmi tříd sekundární struktury (Q8), zatímco kombinace LSTM a CNN zvyšuje přesnost na 81 %. Tento nárůst naznačuje efektivitu navrženého modelu v zachycování prostorových a sekvenčních vzorců v proteinových sekvencích. Model LSTM + CNN má malé množství parametrů a tak je méně výpočetně náročný.

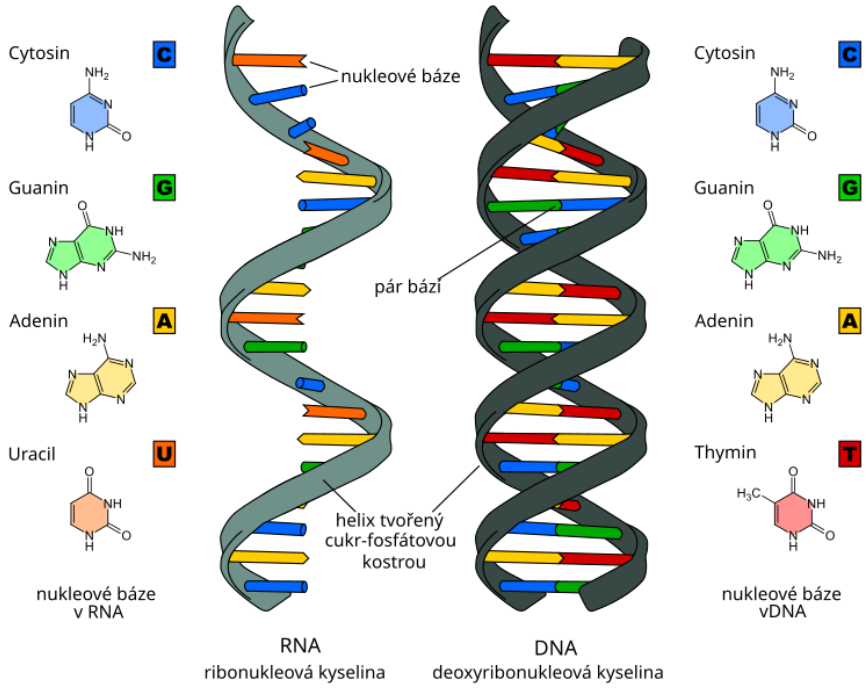
Kromě modelů byla vyvinuta webová aplikace, která umožňuje uživatelům využívat navržené modely pro predikci sekundární struktury proteinů. Backend aplikace je implementován v jazyce Python, využívající knihovny jako TensorFlow a Keras pro strojové učení a flask pro komunikaci s klientem. Frontend je postaven na frameworku Svelte, který zajišťuje rychlé a interaktivní uživatelské rozhraní. Celý projekt je dostupný jako open-source, což umožňuje komunitě vědců a vývojářů přístup k nástrojům a kódu pro další výzkum a vývoj.

Tato práce přispívá k oblasti predikce sekundární struktury proteinů prostřednictvím nových metod strojového učení a poskytováním nástroje pro širší vědeckou komunitu. Otevřený přístup k aplikaci a jejím komponentám podporuje transparentnost a spolupráci v této oblasti výzkumu.

2 Teorie vzniku proteinů

2.1 Složení a způsob "uložení" informace

Prvopočátkem proteinu je deoxyribonukleová kyselina (jednodušeji DNA), z níž se procesem zvaným proteosyntéza vytváří protein. DNA se skládá z polynukleotydového cukr-fosfátového řetězce v němž jsou jednotlivé části deoxyribózy navzájem propojeny fosfátovými skupinami. Kromě toho je na každou částici deoxyribózy připojena jedna z dusíkatých bází. Tyto báze se v kombinaci s fosforečnatou skupinou také nazývají nukleotidy a tvoří žebřiny DNA dvoušroubovice. Báze jsou sloučeniny odvozené od purinu nebo pyrimidu. K purinovým bázím patří Adenin (A) a Guanin (G). K pyrimidovým bázím naopak řadíme Thymin (T) a Cytosin(C). Nukleotidy jsou v DNA postupně poskládány v určitém pořadí a dají se interpretovat jako sekvence znaků (např. ATGTGTCA...). [**1**] [**2**]



Obrázek 1: Rna a Dna

2.2 Transkripce - přepis DNA do mRNA

První fází proteosyntézy je transkripce. V této fázi se přepisuje sekvence nukleotidů do mediatorové RNA neboli mRNA. Rybonukleová kyselina (RNA) se od DNA liší v obsahu dusíkatých bází, RNA má namísto thyminu (T) - uracil (U). [**1**] [**2**]

2.3 Translace - překlad mRNA do aminokyselinové sekvence

Následně proběhne translace, při které na přeloženou genetickou informaci do podoby mRNA nasedá jedna z 20 verzí tRNA (t - transferové), které drží jednu z 20 standardních aminokyselin. Aminokyselina se odděluje od tRNA a tvoří vazbu s předešlou aminokyselinou. Transferové RNA má na sobě takzvaný antikodon, jenž je trojice nukleotidů (např. AAA = Lysin), který je komplementární k příslušné trojici nukleotidů na mRNA, nazýváme ho kodon. Jednotlivé aminokyseliny vytvářejí vazby s předešlými aminokyselinami ve stejném pořadí jak je předepsáno v mRNA, a tak nám vznikne polypeptid, základ pro tuto práci. Následně v aminokyselinovém řetězci jednotlivé aminokyseliny vytvářejí vodíkové vazby a tvoří tak různé struktury, jako jsou například helixy, coily, strandy a více. Tomuto poskládání se říká sekundární struktura proteinu. Tato práce se zaměřuje na tzv. překlad aminokyselinové sekvence na sekvenci symbolů zastupující jednu z 8 možných struktur. [**1**] [**2**]

3 Využití predikce

Predikce sekundární struktury proteinu je možné využít v mnoha biologicky zaměřených odvětvích. Oproti experimentím postupům je predikce mnohem levnější a rychlejší alternativou, ovšem nutno podotknout, že predikce nemusí být vždy přesná a může poskytovat zavádějící informace o sekundární struktuře proteinu. Možné využití jsou ve farmacii pro tvorbu nových léků na základě poznatků o struktuře proteinu nebo v biologii při výzkumu konkrétních funkčních vlastností proteinu.

4 Přístupy a porovnání existujících aplikací

4.1 Přehled existujících metod

- **JPred**

  JPred je webová aplikace pro predikci sekundární struktury proteinů, která integruje několik predikčních metod. Využívá různé algoritmy, jako jsou statistické metody (např. Chou-Fasman a GOR), analýza nejbližšího souseda (NNSSP a PREDATOR) a neuronové sítě (PHD a NNPredict), aby dosáhl vyšší přesnosti predikcí. JPred přijímá proteinové sekvence ve formátu FASTA a poskytuje výsledky Q3 výsledky sekundární struktury proteinu. Výhodou JPred je jeho snadná dostupnost a použití bez nutnosti instalace softwaru na lokálním počítači. [**6**]

- **NetSurfP 3.0**

  NetSurfP 3.0 je nástroj pro predikci různých strukturálních vlastností proteinů, včetně sekundární struktury, solventní přístupnosti, disorderu a dihedrálních úhlů phi/psi. Tato verze využívá pokročilé proteinové jazykové modely a hluboké učení k dosažení vysoké přesnosti a rychlosti predikcí. NetSurfP 3.0 přijímá proteinové sekvence ve formátu FASTA a umožňuje zpracování velkého množství sekvencí současně, což je výhodné pro analýzu rozsáhlých datasetů. Výsledkem predikcí je sekundární struktura proteinu ve formátu Q3, relativní vystavení roztoku a další strukturální parametry. NetSurfP 3.0 je dostupný jako webový server i jako samostatný software pro lokální instalaci. [**7**]

- **Facebook ESM modely**

  Facebook ESM (Evolutionary Scale Modeling) modely představují moderní přístup k predikci proteinových struktur pomocí hlubokého učení a proteinových jazykových modelů. Tyto modely jsou trénovány na rozsáhlých databázích proteinových sekvencí a využívají evoluční informace k predikci struktury proteinů s vysokou přesností. ESM modely analyzují proteinové sekvence a předpovídají jejich sekundární a terciární struktury, a tím umožňují lepší pochopení funkce proteinů a jejich interakcí. Výhodou ESM modelů je jejich schopnost zpracovávat velké množství dat a poskytovat rychlé a přesné predikce, což je užitečné v oblastech jako je bioinformatika, biotechnologie a vývoj léčiv. [**8**]

4.2 Komparativní analýza metod

Při porovnání výše uvedených metod je důležité zvážit několik aspektů jako je, přesnost predikce, výpočetní náročnost, dostupnost a použitelnost.

- **Přesnost predikce**: Facebook ESM modely, díky využití hlubokého učení a rozsáhlých databází, často dosahují vyšší přesnosti v predikci sekundární struktury proteinů ve srovnání s tradičními metodami, jako je JPred. NetSurfP 3.0 také poskytuje přesné predikce a v některých případech může dosahovat srovnatelné přesnosti s ESM modely.

- **Výpočetní náročnost**: Hluboké neuronové sítě použité v ESM modelech jsou výpočetně náročné a vyžadují značné množství výpočetních zdrojů. NetSurfP 3.0 však využívá pokročilé proteinové jazykové modely a hluboké učení k dosažení vysoké přesnosti a rychlosti predikcí, a tím umožňuje zpracování velkého množství sekvencí současně. JPred je méně náročný na výpočetní výkon, což může být výhodné v prostředích s omezenými zdroji.

- **Dostupnost a použitelnost**: JPred a NetSurfP 3.0 jsou dostupné jako online nástroje, které mohou uživatelé snadno použít pro predikci sekundární struktury proteinů bez nutnosti instalace softwaru na lokálním počítači. NetSurfP 3.0 je navíc dostupný i jako samostatný software pro lokální instalaci, což umožňuje jeho použití v prostředích s omezeným přístupem k internetu nebo pro zpracování velkých datasetů. Facebook ESM modely mohou vyžadovat specializované znalosti a infrastrukturu pro implementaci a použití, a tím mohou omezit jejich dostupnost pro některé uživatele.

- **Funkcionalita**: NetSurfP 3.0 nabízí širší spektrum predikovaných vlastností, včetně solventní přístupnosti, disorderu a dihedrálních úhlů phi/psi, což poskytuje komplexnější pohled na strukturu a vlastnosti proteinů. JPred se zaměřuje především na predikci sekundární struktury, zatímco ESM modely mohou poskytovat informace i o terciární struktuře proteinů, a tím se stávají užitečné pro detailnější analýzu proteinových funkcí a interakcí.

V závěru lze říci, že výběr vhodné metody pro predikci sekundární struktury proteinů závisí na konkrétních potřebách a dostupných zdrojích. Jelikož cílem tohoto projektu je čistě predikce sekundární struktury proteinu a balanc mezi výpočetní náročností a přesností predikce, bude nejlepší interpretovat tuto úlohu jako čistě sekvenční nebo také jako překlad z jazyku A do jazyku B za využití co nejmenšího množství vrstev rekurentních neuronek.

5 Teorie strojového učení

5.1 Neurální sítě

Neurální sítě staví základ pro moderní metody strojového učení a mají využití v mnoha oblastech. Neurální sítě (zjednodušeně - NN) jsou složeny z takzvaných neuronek. Neuronky drží několik hodnot, které určují vztah mezi předešlými neuronkami - váhu a bias. Váha i bias jsou na začátku trénování náhodně vybrány a upravovány po čas trénování. Neurální sítě neumějí pracovat s ničím jiným než čísly, a tak veškerá vstupní data musí být upravena tak, aby se dala reprezentovat jako číslo. Mezi takové úpravy patří např. one hot encoding, nebo mapování čísla k určité třídě (např. 1 = mírně oblačno). Upravená vstupní hodnota je poté zpracována následujícími neuronkami pomocí vzorce.

$$x = vstup * vaha + bias$$

Následně má neuronka ještě jednu speciální matematickou operaci, bez které by neurální sítě byli pouze "glorifikované" lineární funkce. Tato operace se nazývá aktivační funkce a je vysvětlena v následující subsekci. [**9**]

5.2 Aktivační funkce

Aktivační funkce nám umožňují aplikovat strojové učení na data s nelineární posloupností. Jako vstup přijímají x souřadnici, která byla dopočítána pomocí základní lineární funkce neuronky a následně je vložena do programátorem zvolené aktivační funkce. Využití různých aktivačních funkcí závisí na kontextu problému a architektury modelu strojového učení. [**11**]

5.3 Relu

Relu - jedna běžně využívaných aktivačních funkcí v hlubokém učení. Rectified linear unit (Relu) je poměrně jednoduchá funkce.

$$f(x) = max(0, x) = \frac{x + |x|}{2}$$

Pokud je x menší nebo rovno nule, výstup se také rovná nule. Pokud je vstup větší než nula, tak výstup se rovná vstupní hodnotě x.

5.4 Sigmoid

Sigmoid, také nazýván jako logistická funkce. Tato typická funkce je využita ve strojovém učení, u které vypadá graf jako písmeno S.

$$\sigma = \frac{1}{1 + e^{-x}}$$

Běžně se využívá v případech, kdy je potřeba obrátit výstupy do pravděpodobnostních hodnot. Pravděpodobnostní hodnoty se pohybují mezi nulou a jedničkou.

5.5 Softmax

Softmax je aktivační funkce běžně využívána v neurálních sítích, které klasifikují dvě a více tříd najednou. Často se používá na sekvenční úlohy a je využita i v tomto projektu.

$$SoftMax_{out_1}(out) = \frac{e^{out_1}}{e^{out_1} + e^{out_2} + e^{out_3}}$$

$$SoftMax_{out_2}(out) = \frac{e^{out_2}}{e^{out_1} + e^{out_2} + e^{out_3}}$$

$$SoftMax_{out_3}(out) = \frac{e^{out_3}}{e^{out_1} + e^{out_2} + e^{out_3}}$$

Výsledně dopočítané hodnoty přes SoftMax funkci jsou pravděpodobnosti pro všechny klasifikovatelné třídy. Pohybují se od nuly až do jedné.

5.6 ArgMax

ArgMax se často využívá až po SoftMax funkci a slouží pro vytažení nejpravděpodobnější třídy z predikce. ArgMax je možné využít jako poslední aktivační funkci v neurální síti, kde pro výstup bude největší hodnota otočena na jedničku a zbytek na nulu. Avšak tímto postupem není možné přesně určit, jak si byl model "jistý"svojí predikcí.

5.7 Ztrátové funkce

Ztrátové funkce určují, jak daleko na grafu byla predikce od reálné hodnoty. Ztrátových funkcí je veliké množství a výběr záleží na kontextu a struktuře modelu. Jednou ze základních ztrátových funkcí je například SSR funkce pro lineární regresi nebo řídká křížová entropie, která se využívá pro NLP úlohy.

5.8 Klesání po gradientu

Klesání po gradientu je postupný posun po křivce vytvořené z výsledků ztrátové funkce k lokálnímu minimu. Velikost posunu po křivce je určena podle velikosti derivace pro daný bod. Příkladem může být zjištění posunu po ztrátové funkci, která vypadá následovně.

$$SSR = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Při vložení základních vah a reálných výsledků do vzorce dostaneme například

$$val_1 = (1.4 - (a + 0.64 * 0.5))^2$$

$$val_2 = (1.9 - (a + 0.64 * 2.3))^2$$

V tomto vzorci je a průsečík, který se nyní pomocí derivace a posunu k lokálnímu minimu snažíme upravit tak, abychom měli co nejpodobnější výsledky k reálné hodnotě. Za využití pravidla mocniny a řetězového pravidla se dostaneme k těmto výsledkům.

$$val_1 = -2 * (1.4 - (a + 0.64 * 0.5))$$

$$val_2 = -2 * (1.9 - (a + 0.64 * 2.3))$$

Jestliže za a dosadíme počáteční hodnotu a stanovíme učební rychlost, kterou vynásobíme výsledkem z derivované funkce, dostaneme hodnotu jenž máme od a odečíst. Učební rychlost je konstanta, která určuje jak rychle se budeme přibližovat k lokálnímu minimu.

$$a = 0$$

$$learningRate = 0.1$$

$$v = (val_1 + val_2) * learningRate$$

$$a = a - (-0.3016)$$

$$a = 0.3016$$

Tento proces opakujeme dokud se nedostaneme k opravdu nízké hodnotě derivace. V praxi je to 0.001 a méně. Stejný postup aplikujeme také pro zjištění naklonění funkce, ovšem nutno podotknout, že derivace bude vypadat trochu jinak. [**10**]

5.9 Algoritmus zpětného šíření chyby

Tento algoritmus využívá klesání po gradientu a postupně upraví všechny váhy i biasy. Prvním krokem je inicializace vah a biasů. Pro váhy se využívá mnoho metod, jednou z nich je standardní normální distribuce. Následně vyřešíme derivativy ztrátové funkce vůči každé proměnné v neurální síti. Výsledek vynásobený rychlostí učení dosadíme za proměnnou. [**12**]

5.10 Konvoluční neurální sítě

Konvoluční neurální sítě bývají běžně využívány pro klasifikaci obrázků, nebo i jiné predikční operace s obrázky. Od neurálních sítí se trochu liší, avšak využívají stejné principy. Konvoluční neurální sítě mají tzv. kernel, neboli okénko, ve kterém jsou nastaveny ze začátku hodnoty náhodně. Vypočítá se skalární součin mezi kernelem a vstupními daty, následně přičteme bias a výsledek zapíšeme do tzv. feature mapy. Kernel se poté posouvá např. o jedno pole a provede se stejný postup. Následně se běžně aplikuje Max pooling pro redukci výstupních dat, ale není to nutné. [**5**]

5.11 Rekurentní neurální sítě

Rekurentní neurální sítě, zkráceně RNN, jsou modely, jenž mají velmi dobré využití v sekvenčních úlohách. RNN jsou oproti klasické neurální síti posíleny o cyklus zpětné odezvy a dokáží se tak učit na posloupnostech jako jsou časové řady nebo jiné typy sekvencí. Cyklus zpětné odezvy přenáší informace z předchozího kroku do současného, což vytváří jakousi "paměť"sítě. [**3**] [**4**]

6 Aplikace strojového učení na predikci sekundární struktury proteinu

6.1 Přehled a fungování modelů

Predikce sekundární struktury proteinů je klíčová pro pochopení jejich funkce a interakcí. V průběhu let byly vyvinuty různé přístupy využívající strojové učení k této predikci:

- Skryté Markovovy modely (HMM): Jedny z prvních modelů aplikovaných na tento problém, dosahující přesnosti kolem 60 % v Q3 predikci.

- Umělé neuronové sítě (ANN): Postupně nahradily HMM díky své schopnosti modelovat nelineární vztahy v datech. Příkladem je nástroj PSIPRED, který využívá ANN k predikci sekundární struktury proteinů.
- Hluboké neuronové sítě (DNN): S rostoucím výpočetním výkonem se začaly využívat hluboké sítě, jako jsou rekurentní neuronové sítě (RNN) a konvoluční neuronové sítě (CNN), které zlepšily přesnost predikcí.
- Například použití LSTM a GRU paměťových buněk v RNN bylo zkoumáno pro jejich efektivitu v této oblasti.
- Celulární automaty: Alternativní přístup využívající celulární automaty byl navržen ke zvýšení rychlosti predikce, i když za cenu mírného snížení přesnosti.

6.2 Existující datasety

Pro vývoj a testování modelů predikce sekundární struktury proteinů jsou k dispozici různé veřejně dostupné datasety. Tyto datasety obvykle obsahují sekvence aminokyselin s anotovanou sekundární strukturou, získanou experimentálními metodami, jako je rentgenová krystalografie nebo nukleární magnetická rezonance. Mezi často využívané datasety patří například CB513, který obsahuje 513 proteinů s různými strukturami a CASP datasety. Tyto datasety slouží k testování predikčních modelů a umožňují objektivní srovnání jejich výkonu. Jako primární tréninkový dataset využitý v této práci je PS4 dataset, který by měl být zatím největší dataset neredundantních proteinů.

Implementace projektu

7 Tvorba a sanitizace datasetu

7.1 Tvorba datasetu

- **Základní dataset**

  Při získávání datasetu bylo na výběr několik různých proteinových databank, mezi ně například patří uniprot, PDB a AlphaFoldDB. Data jsem se rozhodl programaticky postahovat z PDB api. Jelikož jsou data experimentálně zjištěná, je tudíž menší šance na deformaci struktury proteinu, i když ne nulová. PDB má ovšem jediný problém, a tím je, že snad žádný soubor neobsahuje sekundární strukturu proteinu, nýbrž terciarní strukturu proteinu v podobě atomických koordinací. V tomto případě bylo využito programu DSSP, který se stáhl lokálně a přes Python a za využití subprocess knihovny, spouštěl v příkazovém řádku po stažení každého proteinu z api.

```
args = subprocess.run(CommandDSSP(inputFileLIN + '/'
                        + divided + '/'
                        + protein),
                        capture_output=True,
                        text=True, shell=True).stdout
                processed_dssp = ProcessDSSP(args)
```

  Program DSSP je schopný dopočítat sekundární strukturu z atomických koordinací. DSSP může vracet dvě skupiny tříd, Q3 a Q8. Q3 skupina má 3 podtřídy, helixy, coily, strandy. Q8 skupina má 8 tříd, které jsou už jenom specifičtější subtřídy Q3. Datasety na kterých pracuji, využívají Q8. Také DSSP poskytuje přidané vystavení okolnímu roztoku ke každé aminokyselině. K datasetu byli také přidány hodnoty pH a teploty v kelvinech, které se nacházely v souborech hned vedle atomických koordinací proteinu. Tyto hodnoty však po pátrání nemají na trénování pozitivní vliv, jelikož jsou získané z prostředí experimentu a nikoli organismu. V současné době se snažím shánět data, kde by k proteinu byly poskytnuty hodnoty pH a teploty, bohužel se standardně tyto informace o organismu nezachycují.

- **Phylogenic dataset**

  Phylogenický dataset tvoří primárně základ pro databázi možných organismů a peptidů jim přidělených. Byl programaticky stažen z PDB api. U tohoto datasetu nebyla potřeba žádná úprava, jelikož api obsahovala pouze data, která bylo pro tento projekt potřebná. První zamýšlenou úlohou pro kterou byl dataset vytvořen, bylo vytrénování modelu i na jménech organismů, z kterých byl protein odebrán. Avšak ke konci dataset k ničemu využit nebyl.

7.2 Sanitizace datasetu

Z datasetu jsem smazal všechny duplikáty a řádky, které měli prázdná pole. V prázdných polích se většinou vyskytovala hodnota NaN, kterou vytvořil program DSSP při dopočítávání sekundární struktury proteinu. Také se mnohokrát stalo, že DSSP nerozeznalo aminokyselinu a tak doplnilo hodnotu X, jelikož tato hodnota není jedna konzistentní aminokyselina a představuje může představovat šum. Rozhodl jsem se řádky s výskytem této hodnoty smazat. Z datasetu jsem také smazal řádky, kterých sekvence byly moc krátké nebo naopak dlouhé oproti zbytku.

8 Testování a využití různých metod strojového učení

8.1 Pouze konvoluční sítě (CNN)

V této sekci je model sestaven výhradně z konvolučních vrstev (CNN) pro předpověď sekundární struktury proteinů. Model využívá předzpracované tokenizované sekvence proteinů a cílové hodnoty DSSP.

Modelová architektura: Model je postaven jako sekvenční model, který obsahuje následující vrstvy a hyperparametry, které byly pečlivě vybírány tak, aby měly co nejlepší výsledky:

- **Embedding vrstva:** První vrstva modelu je Embedding vrstva, která mapuje sekvence tokenů na dense vektory. Tato vrstva používá definovanou velikost slovníku a dimenzi embeddingu, která je nastavena na 256.

- **Konvoluční vrstva:** Po embedding vrstvě následuje 1D konvoluční vrstva, která má 32 filtrů s velikostí kernelu 19. Tato vrstva slouží k extrakci lokálních vzorců v sekvenci tokenů. Aktivace je ReLU a padding je nastaven na stejný jako vstupní data, aby délka výstupu zůstala zachována.

- **Dense vrstva:** Následuje hustá vrstva s 64 neurony, která využívá ReLU aktivaci. Tato vrstva slouží k dalšímu zpracování extrahovaných rysů z konvoluční vrstvy.

- **TimeDistributed vrstva:** Na konci modelu se používá časově distribuovaná vrstva, která aplikuje hustou vrstvu na každý krok sekvence. Tato vrstva má výstup o velikosti slovníku, kde aktivace je softmax. Výstup z této vrstvy jsou finální pravděpodobnosti predikce.

Kompilace modelu: Model je kompilován s Adam optimalizátorem a ztrátovou funkcí SparseCategoricalCrossentropy, která je vhodná pro vícetřídovou klasifikaci s celočíselnými cílovými hodnotami. Jako metrika je použita accuracy.

Trénování modelu: Model je trénován na PS4 datasetu po dobu pěti epoch s validací na malém subsetu dat z trénovacího datasetu. Po dokončení trénování je model vyhodnocen na testovacím datasetu CB513 pomocí model.evaluate, na kterém dosahuje přesnosti 74 %.

Shrnutí: Tento model se zaměřuje na použití konvolučních vrstev k analýze a predikci sekundární struktury proteinů na základě jejich sekvence. Architektura je jednoduchá a efektivní pro extrakci prostorových vzorců v proteinových sekvencích.

8.2 Kombinace BiGRU a TCN (BiGRU + TCN)

V této části je popsána architektura modelu, který kombinuje obousměrnou rekurentní síť GRU (BiGRU) a konvoluční síť založenou na temporálních konvolucích (TCN) pro zpracování sekvencí aminokyselin.

- **Předzpracování a vstupní parametry:** Model pracuje s proměnnými max_seq_len (maximální délka vstupní sekvence), max_features (velikost slovníku získaná z tokenizeru), embedding_dim (rozměr vektorového prostoru pro embedding) a num_classes (počet tříd), které definují klíčové parametry modelu.

- **Embedding vrstva:** První vrstva modelu je Embedding, která mapuje celočíselné vstupy (indexy slov) do hustých vektorových reprezentací o rozměru embedding_dim. Nastavením parametru mask_zero=True se zajistí, že nulové hodnoty (padding) nebudou ovlivňovat trénování modelu.

- **BiGRU vrstva:** Další vrstvou je obousměrná GRU (Bidirectional GRU) se 128 jednotkami, která vrací sekvence (return_sequences=True). Tato vrstva využívá aktivaci relu a obsahuje dropout (dropout=0.4) pro snížení rizika přeučení. Pro regulaci vah je použita L2 regularizace (kernel_regularizer=L2(0.01)).

- **TCN vrstva:** Po BiGRU vrstvě následuje vrstva TCN (Temporal Convolutional Network) se 32 filtry a velikostí kernelu 19. Používá dilatační faktory [1, 2, 4, 8], což umožňuje modelu efektivně zachytit dlouhodobé závislosti v sekvenčních datech. Vrstva zachovává délku sekvence (return_sequences=True), čímž umožňuje efektivní propagaci informací skrz síť.

- **Optimalizace a kompilace modelu:** Model je kompilován s optimalizátorem Adam s učební rychlostí (learning_rate=0.001) a gradientním klipováním (clipnorm=0.5), které zabraňuje problémům s explodujícími gradienty. Ztrátová funkce je sparse_categorical_crossentropy a jako metrika pro sledování výkonu se používá přesnost (accuracy).

- **Trénink, evaluace a uložení modelu:** Model je trénován na PS4 datasetu a validován na jeho subsetu po dobu 3 epoch. Po trénování je model vyhodnocen na testovacím datasetu CB513 a následně uložen pro další použití.

Tato kombinace BiGRU a TCN umožňuje modelu efektivně využívat jak dlouhodobé závislosti v sekvencích (díky BiGRU), tak extrahovat klíčové lokální vzory pomocí TCN. Přesnost na testovacíh datech je 82 %, model byl vytrénován před odevzdáním soč a tak není uveden jako nejefektivnější v celé práci.

8.3 Kombinace GRU + LSTM + CNN

V této části je popsána architektura modelu, který kombinuje vrstvy GRU, LSTM a konvoluční síť (CNN) pro zpracování sekvencí aminokyselin.

- **Předzpracování a vstupní parametry:** Model pracuje s proměnnými max_seq_len (maximální délka vstupní sekvence), max_features (velikost slovníku získaná z tokenizeru), embedding_dim (rozměr vektorového prostoru pro embedding) a num_classes (počet tříd), které definují klíčové parametry modelu.

- **Embedding vrstva:** První vrstva modelu je Embedding, která mapuje celočíselné vstupy (indexy slov) do hustých vektorových reprezentací o rozměru embedding_dim. Nastavením parametru mask_zero=True se zajistí, že nulové hodnoty (padding) nebudou ovlivňovat trénování modelu.

- **LSTM vrstva:** Další vrstvou je LSTM s 128 jednotkami, která vrací sekvence (return_sequences=True). Tato vrstva využívá aktivaci relu a obsahuje dropout (dropout=0.4) pro snížení rizika přeučení. Pro regulaci vah je použita L2 regularizace (kernel_regularizer=L2(0.01)).

- **GRU vrstva:** Po LSTM vrstvě následuje GRU vrstva s 128 jednotkami, která také vrací sekvence. Používá aktivaci relu, dropout (dropout=0.4) a L2 regularizaci (kernel_regularizer=L2(0.01)).

- **CNN vrstva:** Následuje konvoluční vrstva (Conv1D) se 64 filtry a velikostí jádra 70. Používá aktivaci relu a padding nastavený na 'same', což zachovává délku sekvence. Tato vrstva umožňuje modelu efektivně extrahovat lokální vzory v datech.

- **Optimalizace a kompilace modelu:** Model je kompilován s optimalizátorem Adam s učební rychlostí (learning_rate=0.0001) a gradientním klipováním (clipnorm=0.5), které zabraňuje problémům s explodujícími gradienty. Ztrátová funkce je sparse_categorical_crossentropy a jako metrika pro sledování výkonu se používá přesnost (accuracy).

- **Trénink, evaluace a uložení modelu:** Model je trénován po dobu 10 epoch na trénovací sadě a validován na validačním datasetu. Během tréninku jsou využívány váhované třídy. Po trénování je model vyhodnocen na testovacím datasetu CB513 a následně uložen pro další použití.

Tato kombinace GRU, LSTM a CNN umožňuje modelu efektivně využívat jak dlouhodobé závislosti v sekvencích (díky LSTM a GRU), tak extrahovat klíčové lokální vzory pomocí CNN. Tento model má trénovací přesnost 83 %, validační 84 % a na testovacím datasetu má 76 %.

8.4 Kombinace GRU a CNN

V této části je popsána architektura modelu, který kombinuje rekurentní síť (GRU) a konvoluční síť (CNN) pro zpracování sekvencí aminokyselin.

- **Předzpracování a vstupní parametry:** Proměnné jako max_seq_len (maximální délka vstupní sekvence), max_features (velikost slovníku získaná z tokenizeru), embedding_dim (rozměr vektorového prostoru pro embedding) a num_classes (počet tříd) definují základní parametry modelu.

  text

- **Embedding vrstva:** První vrstva modelu je Embedding, která mapuje celočíselné vstupy (indexy slov) do hustých vektorových reprezentací o rozměru embedding_dim. Použitím parametru mask_zero=True se zajistí, že nulové hodnoty (padding) budou ignorovány při trénování.

- **GRU vrstva:** Následuje vrstva GRU s 128 jednotkami, která vrací sekvence (return_sequences=True). Tato vrstva využívá aktivaci relu a zahrnuje dropout (dropout=0.4) pro snížení rizika přeučení. Dále je aplikována L2 regularizace (kernel_regularizer=L2(0.01)) k penalizaci vysokých hodnot vah.

- **Konvoluční (CNN) vrstva:** Poté následuje vrstva Conv1D se 64 filtry a velikostí kernelu 70. Tato vrstva používá aktivaci relu a padding nastavený na same, což zajišťuje, že výstupní sekvence bude mít stejnou délku jako vstupní. Stejně jako GRU vrstva, i tato vrstva využívá L2 regularizaci pro prevenci přeučení.

- **Optimalizace a kompilace modelu:** Model je kompilován pomocí optimalizátoru Adam s nízkou učební rychlostí (learning_rate=0.0001) a gradientním klipováním (clipnorm=0.5), které pomáhá předcházet problémům s explodujícími gradienty. Jako ztrátová funkce se používá sparse_categorical_crossentropy. Pro sledování výkonu modelu se měří přesnost (accuracy).

- **Trénink, evaluace a uložení modelu:** Model je trénován na PS4 datasetu a validován na jeho subsetu, po dobu deseti epoch. Během tréninku jsou využívány vážené třídy. Po tréninku se model vyhodnocuje na CB513 testovacím datasetu, na kterém dosahuje přesnosti okolo 76 %.

Tato kombinace GRU a CNN umožňuje modelu nejprve zachytit dlouhodobé závislosti v sekvenčních datech (díky GRU) a následně extrahovat lokální vzory pomocí konvoluční vrstvy. Takový přístup vede ke zlepšení výkonu v predikci sekundární struktury proteinu s nárůstem přesnosti oproti čisté CNN o 4 %.

8.5 Kombinace rekurentních (LSTM) a konvolučních sítí (CNN) (LSTM + CNN)

V této části je popsána architektura modelu, který kombinuje rekurentní síť (LSTM) a konvoluční síť (CNN) pro zpracování sekvencí aminokyselin.

- **Předzpracování a vstupní parametry:** Proměnné jako max_seq_len (maximální délka vstupní sekvence), max_features (velikost slovníku získaná z tokenizeru), embedding_dim (rozměr vektorového prostoru pro embedding) a num_classes (počet tříd) definují základní parametry modelu.

- **Embedding vrstva:** První vrstva modelu je Embedding, která mapuje celočíselné vstupy (indexy slov) do hustých vektorových reprezentací o rozměru embedding_dim. Použitím parametru mask_zero=True se zajistí, že nulové hodnoty (padding) budou ignorovány při trénování.

- **LSTM vrstva:** Následuje vrstva LSTM s 128 jednotkami, která vrací sekvence (return_sequences=True). Tato vrstva využívá aktivaci relu a zahrnuje dropout (dropout=0.4) pro snížení rizika přeučení. Dále je aplikována L2 regularizace (kernel_regularizer=L2(0.01)) k penalizaci vysokých hodnot vah.

- **Konvoluční (CNN) vrstva:** Poté následuje vrstva Conv1D se 64 filtry a velikostí kernelu 70. Tato vrstva používá aktivaci relu a padding nastavený na same, což zajišťuje, že výstupní sekvence bude mít stejnou délku jako vstupní. Stejně jako LSTM vrstva, i tato vrstva využívá L2 regularizaci pro prevenci přeučení.

- **Optimalizace a kompilace modelu:** Model je kompilován pomocí optimalizátoru Adam s nízkou učební rychlostí (learning_rate=0.0001) a gradientním klipováním (clipnorm=0.5), které pomáhá předcházet problémům s explodujícími gradienty. Jako ztrátová funkce se používá sparse_categorical_crossentropy. Pro sledování výkonu modelu se měří přesnost (accuracy).

- **Trénink, evaluace a uložení modelu:** Model je trénován na PS4 datasetu a validován na jeho subsetu datasetu, po dobu 10 epoch. Během tréninku jsou využívány ováhované třídy. Po tréninku se model vyhodnocuje na CB513 testovacím datasetu, na kterém doshuje přesnosti okolo 81 %.

Tato kombinace LSTM a CNN umožňuje modelu nejprve zachytit dlouhodobé závislosti v sekvenčních datech (díky LSTM) a následně extrahovat lokální vzory pomocí konvoluční vrstvy. Takový přístup vede ke zlepšení výkonu v predikci sekundární struktury proteinu s nárustem přesnosti oproti čisté CNN o 7 %.

9 Architektura webové aplikace

9.1 Backend

Backend je složen z několika pomyslných částí, z vstupního bodu, ovladače a modelu a separátní části, která řeší zpracování dat z queue a jejich zpětné nahrání do redis databáze. Backend je napsán v jazyku Python a využívá knihovny Flask pro práci s rest api a knihovnu Tensorflow pro načtení vytrénovaného modelu a následnou predikci dat. Celá architektura je monolitní, avšak je očekáváno, že projekt bude přepsán do mikroservisní architektury.

9.1.1 Vstupní bod

Vstupní bod aplikace je definován v souboru, který spouští Flask server a nastavuje různé konfigurační a aplikační parametry. Tento soubor načítá všechny potřebné moduly a inicializuje aplikaci, aby mohla začít splňovat požadavky API.

Zde je podrobný popis klíčových částí:

1. **Inicializace aplikace**:

```
app = Flask(__name__)
cors = CORS(app)
api = Api(app)
```

   Na začátku je vytvořena instance třídy Flask, která bude sloužit jako základ pro webový server. Dále je povoleno CORS (Cross-Origin Resource Sharing) pro aplikaci, což umožňuje, aby aplikace komunikovala s jinými doménami. Instance Api je pak přiřazena k aplikaci pro podporu RESTful API.

2. **Swagger pro dokumentaci API**:

```
app.config['SWAGGER'] = {
    'title': 'My API',
    'uiversion': 3
}
swagger = Swagger(app)
```

Použití knihovny Flasgger slouží k automatickému generování dokumentace API v Swagger UI, což poskytuje přehledné a interaktivní prostředí pro testování API metod. V tomto případě je nastaven název API a verze UI.

3. **Definice zdrojů API**:

```
    api.add_resource(Welcome, '/api')
    api.add_resource(DoPrediction, '/api/do-prediction')
    api.add_resource(GetById, '/api/get-by-id/<id>')
    api.add_resource(GetAllProteins, '/api/get-all-cached-proteins')
```

Aplikace využívá Flask-RESTful k definování různých endpointů (zdrojů) API. Každý endpoint je připojen ke konkrétní třídě, která implementuje logiku pro zpracování požadavků. Například:

- /api vede na třídu Welcome, která obsluhuje úvodní požadavky.
- /api/do-prediction spustí proces predikce.
- /api/get-by-id/<id> vrátí data na základě konkrétního ID.
- /api/get-all-cached-proteins vrátí všechny cachované proteiny.

4. **Spuštění front-endového serveru**:

```
if __name__ == '__main__':
    queue_handler = QueueHandler()
    queue_thread = threading.Thread(target=queue_handler, args=())
    queue_thread.start()
    app.run(host='0.0.0.0', port=5001, debug=True)
```

Při spuštění aplikace se inicializuje QueueHandler, který zajišťuje zpracování úkolů v pozadí. Tento handler je spuštěn v samostatném vlákně (queue_thread), což umožňuje asynchronní zpracování úloh, aniž by to ovlivnilo výkon hlavního API serveru. Aplikace se spustí na adrese 0.0.0.0 na portu 5001, což umožňuje přístup z jakéhokoli počítače v síti.

Tato část kódu tedy slouží jako centrální bod, který iniciuje server, nastavuje API a spouští další vlákna pro asynchronní úkoly. Pro produkční nasazení by měla být nastavena možnost debug=False.

9.1.2 Kontroler

Kontroler je zodpovědný za obsluhu logiky pro zpracování požadavků API, ověřování vstupních dat a volání metod modelu, které se zabývají předpovědí a správou cache. V následujícím textu jsou podrobně popsány jednotlivé části kódu kontroleru.

1. **Inicializace**:

```
class Controller:
    def __init__(self):
        self.amino_acids = [
            "A", "R", "N", "D", "C",
            "Q", "E", "G", "H", "I",
            "L", "K", "M", "F", "P",
            "S", "T", "W", "Y", "V"
        ]
        self.model = ModelA()
```

Kontroler je inicializován s definovaným seznamem amino_acids, který obsahuje všechny platné aminokyseliny. Dále je vytvořena instance modelu ModelA, který bude použit pro kontrolu cache a predikce.

2. **Kontrola dat**:

```
def __data_check(self, data):
    if 'AC' not in data.keys():
        output_error = "Unknown variable provided"
        return make_response(jsonify({"output": output_error}), 400)
    for i, ac in enumerate(data['AC']):
        if ac.upper() not in self.amino_acids:
            output_error = f"Unknown amino acid type used on char:{i}"
            return make_response(jsonify({"output": output_error}), 422)
    return None
```

Funkce __data_check kontroluje vstupní data. Prvně ověřuje, zda obsahují klíč AC (seznam aminokyselin). Pokud klíč neexistuje, vrací chybu s kódem 400. Dále kontroluje každou aminokyselinu v seznamu, jestli je platná. Pokud nalezne neplatnou aminokyselinu, vrátí chybu s kódem 422 a informací o konkrétním místě chyby v seznamu.

3. **Predikce**:

```
def do_prediction(self, data):
    data = json.loads(data.decode('utf-8'))
    status = self.__data_check(data)
    if status is not None:
        return status
    amino_acids = data['AC']
    status, secondary_structure = self.model.check_cache_record(
    amino_acids=amino_acids
    )
    if status:
        return make_response(jsonify(secondary_structure), 202)

    id = self.model.add_empty_cache_record(amino_acids)
    self.model.push_to_queue(id, json.dumps(amino_acids))
    return make_response(jsonify({"AC": amino_acids,
    "PENDING": True,
    "ID": id}), 202)
```

Funkce do_prediction provádí celkový proces predikce. Nejprve dekóduje a načte data z příchozího požadavku. Poté provede kontrolu dat pomocí funkce__data_check. Pokud jsou data v pořádku, kontroluje, zda již existuje cache pro danou sekvenci aminokyselin. Pokud ano, vrátí uloženou sekundární strukturu s kódem 202 (Accepted). Pokud cache není dostupná, vytvoří nový záznam v cache a přidá požadavek do fronty pro další zpracování.

4. **Získání dat podle ID**:

```
def get_by_id(self, id):
    status, data = self.model.check_cache_record(id)
    print(status)
    print(data)
    if status:
        return make_response(jsonify(data), 200)
    return make_response(jsonify(data))
```

Funkce get_by_id umožňuje získat data podle specifického ID. Nejprve kontroluje, zda existuje záznam v cache pro dané ID. Pokud je záznam k dispozici, vrátí jej s kódem 200 (OK). V opačném případě vrátí chybovou odpověď.

Kontroler se tedy stará o logiku ověřování vstupních dat, interakci s modelem a správu předpovědí. Funkce v kontroleru zajišťují správu požadavků API, kontrolu cache a správu úkolů v pozadí pro asynchronní zpracování predikcí.

9.1.3 Model

Třída ModelA zajišťuje veškerou logiku spojenou se správou cache v Redis databázi a frontou úkolů pro predikce. Model obsahuje metody pro přidávání záznamů do cache, ověřování existujících záznamů a správu asynchronního zpracování úkolů. V následujícím textu jsou podrobně popsány jednotlivé metody třídy.

1. **Inicializace**:

```
class ModelA:
    def __init__(self):
        self.redis_db = RedisSingleton()
        return None
```

Při inicializaci třídy ModelA je vytvořena instance RedisSingleton, která zajišťuje připojení k Redis databázi. Tato databáze se používá pro ukládání dat souvisejících s predikcemi a frontami úkolů.

2. **Přidání prázdného záznamu do cache**:

```
def add_empty_cache_record(self, amino_acid):
    uuid_bytes = uuid.uuid4().bytes
    id = base64.urlsafe_b64encode(uuid_bytes).decode("utf-8").rstrip("=")
    self.redis_db.hset(id, mapping={
        "pending": "True",
        "aminoAcids": "empty",
        "secondaryStructureCONV": "empty",
        "secondaryStructureLSTM": "empty",
    })
    self.redis_db.expire(id, 86400)
    self.redis_db.set(amino_acid, id)
    self.redis_db.expire(amino_acid, 86401)
    return id
```

Funkce add_empty_cache_record generuje unikátní ID pro nový záznam v cache, používá UUID a kódování base64 pro vytvoření ID. Tento záznam je následně uložen v Redis databázi, kde jsou nastaveny výchozí hodnoty pro různé klíče, jako je pending, aminoAcids, secondaryStructureCONV a secondaryStructureLSTM. Záznam má také nastavený čas vypršení na 24 hodin. ID aminokyseliny je přiřazeno tomuto záznamu, čímž se vytváří spojení mezi aminokyselinou a jejím záznamem v cache.

3. **Přidání úkolu do fronty**:

```
    def push_to_queue(self, id, aminoAcids):
        data = json.dumps([id, aminoAcids])
        self.redis_db.rpush("queue", data)
```

Funkce push_to_queue přidává úkol (reprezentovaný jako seznam obsahující ID záznamu a sekvenci aminokyselin) do fronty Redis pomocí příkazu rpush. Tento úkol bude následně zpracován asynchronně.

4. **Kontrola záznamu v cache**:

```
def check_cache_record(self, id=None, amino_acids=None):
    if amino_acids is not None:
        id = self.redis_db.get(amino_acids)
        if id is None:
            return False, None
    pending = self.redis_db.hget(id, "pending")
    if pending == None:
        return False, {'PENDING': None, 'ERROR': True}
    if pending == "True":
        return False, {"PENDING": True, 'ERROR': False}
    amino_acids = self.redis_db.hget(id, "aminoAcids")
    secondary_structure_conv = self.redis_db.hget(id,
    "secondaryStructureCONV")
    secondary_structure_lstm = self.redis_db.hget(id,
    "secondaryStructureLSTM")
    if isinstance(id, bytes):
        id = bytes.decode(id)
    return True, {
        "PENDING": False,
        "AC": bytes.decode(amino_acids),
        "SSCONV": bytes.decode(secondary_structure_conv),
        "SSLSTM": bytes.decode(secondary_structure_lstm),
        "ID": id,
        'ERROR': False
    }
```

Funkce check_cache_record provádí kontrolu existujícího záznamu v cache podle ID nebo sekvence aminokyselin. Pokud je zadána sekvence aminokyselin, hledá se ID odpovídající této sekvenci v Redis db. Pokud není záznam v cache nalezen, vrací se odpověď s hodnotou None. Dále se kontroluje stav pending, který určuje, zda je záznam stále ve zpracování. Pokud je záznam dokončen, funkce vrátí informace o aminokyselinách a sekundárních strukturách (CONV a LSTM). Pokud je záznam stále ve stavu pending, vrátí příslušnou informaci.

Třída ModelA se tedy stará o správu dat v Redis databázi, kontrolu cache pro predikce sekundární struktury proteinů. Pomocí těchto metod aplikace udržuje efektivní správu predikcí a minimalizuje zátěž serveru při opakovaných požadavcích.

9.1.4 Queue Handler

Třída QueueHandler zajišťuje správu fronty úkolů pro predikce sekundární struktury proteinů. Její hlavní úkol spočívá ve zpracování úkolů, které jsou přiřazeny k modelům, a jejich následném vyhodnocení pomocí konkrétního modelu pro predikci.

V následujícím textu jsou podrobně popsány hlavní komponenty třídy a její metody:

1. **Inicializace třídy Inference**: Třída Inference je zodpovědná za predikce na základě zvoleného modelu. Při inicializaci se načítají modely LSTM a CNN, stejně jako tokenizer z knihovny transformers.

```
class Inference():
    def __init__(self):
        self.tokenizer = AutoTokenizer.from_pretrained(
        "facebook/esm2_t6_8M_UR50D"
        )
        self.conv = tf.keras.models.load_model(
        './prediction_models/convolutionalSecPRED.keras'
        )
        self.lstm = tf.keras.models.load_model(
        './prediction_models/lstmSecPRED-PREDICTION.keras'
        )
```

2. **Predikce pomocí LSTM modelu**: Metoda _predict_lstm tokenizuje vstupní sekvenci aminokyselin a použije LSTM model k predikci sekundární struktury.

```
def _predict_lstm(self, ac):
    tokenized_input = self.tokenizer(ac, return_tensors="tf")["input_ids"]
    predictions = self.lstm.predict(tokenized_input)
    predicted_labels = np.argmax(predictions, axis=-1)
    return self._process_output(predicted_labels)
```

Vstupní sekvence aminokyselin (ac) je tokenizována a převedena na vstupy pro LSTM model. Predikce jsou provedeny na základě těchto vstupů, a výstupy jsou následně pomocí argmax funkce získány.

3. **Predikce pomocí konvolučního modelu**: Podobně jako v předchozím případě, metoda _predict_conv provádí predikci pomocí cnn modelu.

```
def _predict_conv(self, ac):
    tokenized_input = self.tokenizer(ac, return_tensors="tf")["input_ids"]
    predictions = self.conv.predict(tokenized_input)
    predicted_labels = np.argmax(predictions, axis=-1)
    return self._process_output(predicted_labels)
```

Tento model používá stejný tokenizér pro vstupy, ale provádí predikci pomocí konvoluční neuronové sítě, což vede k mírně jiným výsledkům než LSTM model.

4. **Zpracování výstupu predikce**: Metoda _process_output dekóduje výstupy z modelů a filtruje nežádoucí znaky, jako jsou symboly <, >, c, l, s, e, o, s a mezery, aby získala čistý výstup.

```
def _process_output(self, predicted_labels):
    output = self.tokenizer.decode(predicted_labels[0])
    filtered_vals = ['<', '>', 'c', 'l', 's', 'e', 'o', 's', ' ']
    cleared_output = ""
    for x in output:
        if x not in filtered_vals:
            cleared_output += x
    return cleared_output
```

Tento krok zajišťuje, že výstup modelu je vyčištěn od nežádoucích znaků a zůstane pouze relevantní predikce sekundární struktury.

5. **Predikce na základě vybraného modelu**: Hlavní metoda predict volí, zda použít LSTM nebo konvoluční model na základě parametru model_type.

```
def predict(self, ac, model_type="lstm"):
    if model_type == "lstm":
        return self._predict_lstm(ac)
    else:
        return self._predict_conv(ac)
```

Na základě zadaného typu modelu (LSTM nebo konvoluční) se používá odpovídající metoda pro predikci. Tato flexibilita umožňuje použít různé modely pro různé scénáře.

Třída Inference je klíčová pro provádění predikcí sekundární struktury proteinů pomocí dvou různých modelů strojového učení (LSTM a konvoluční neuronové sítě). Pomocí této třídy se může aplikace rozhodnout, jaký model použít, a spustit predikci na základě zadané sekvence aminokyselin.

9.2 Frontend

9.3 Vizualizace sekundární struktury proteinu

9.3.1 Q3 vizualizace

Tato sekce popisuje implementaci standardní vizualizace sekundární struktury proteinu, tzv. Q3 vizualizaci, která je součástí finální verze projektu. Vizualizace je generována na frontendu pomocí frameworku Svelte a dynamicky vykresluje SVG prvky, jež reprezentují různé typy sekundárních struktur proteinu.

1. Inicializace a příprava dat Na začátku komponenty se načítají vstupní data, konkrétně posloupnost aminokyselin a jejich sekundární struktury, které jsou předávány jako vlastnosti (props). Dále jsou definovány konstanty, např. délka řádku (rowLen) a vertikální inkrement (incr). Počet řádků (height) je spočítán jako:

```
const rowLen = 80;
const incr = 80;
const height = Math.ceil(aminoAcid.length / rowLen);
```

Tento výpočet zajistí, že se celá sekvence rozdělí do řádků vhodné délky pro zobrazení.

2. Definice pomocných funkcí Implementace využívá několik pomocných funkcí, které usnadňují tvorbu SVG elementů:

- **text:** Tato funkce generuje SVG textový element, který zobrazuje jednotlivé aminokyseliny. Je nastavena specifická velikost písma a pozice textu tak, aby byl výstup čitelný.

```
let text = (x, aminoAcid) => {
  return `<text style="font-size: 13.75px !important;"
  x="${x}" pointer-events="none" y="18" text-anchor="middle">
  ${aminoAcid}
  </text>`;
};
```

- **helix, coil a sheet:** Tyto funkce generují SVG elementy pro různé typy sekundárních struktur.
  - Funkce helix vytváří odkaz na předdefinovanou SVG ikonu, která reprezentuje helix. Ikona se vykresluje pomocí tagu <use>.

```
let helix = (helixSymbol, x) => {
  return `<use xlink:href="${helixSymbol}"
  transform="translate(${x}, 0)">
  </use>`;
};
```

  - Funkce coil generuje obdélník, který symbolizuje smyčku (coil) v proteinu.

```
let coil = (width, x) => {
  return `<rect width="${width}" height="3.6" y="7.2"
  x="${x}" fill="#cc3399">
  </rect>`;
};
```

– Funkce sheet vytváří obdélník s definovanou výškou a barvou, který reprezentuje beta-list (sheet).

```
let sheet = (width, x) => {
  return `<rect width="${width}" height="10" y="3.6"
  x="${x}" fill="#660099">
  </rect>`;
};
```

- **defs:** Funkce defs definuje opakovaně používané SVG prvky, jako jsou klipy a předdefinované tvary (např. ikony pro různé typy helixů). Tyto prvky jsou poté referencovány pomocí tagu use.

```
let defs = (id) => {
  return `<defs>
    <g id="icon-ss-Hb-8eaa">
      <path d="M 7.05 0.5 H 2.35 Q 0 0.5, -7.05 17.5
      H -2.35 Q 0 17.5, 7.05 0.5" fill="#ff6600">
      </path>
    </g>
    ...
    <clipPath id=${id}>
      <rect x="0" y="0" width="940" height="18"></rect>
    </clipPath>
  </defs>`;
};
```

3. Generování vizualizace po řádcích Pro zajištění čitelného rozložení je vizualizace rozdělena do řádků. Hlavní funkcí pro tento účel je rowBlocks. Tato funkce postupně prochází sekvenci aminokyselin a sekundárních struktur, rozdělí je do segmentů dle hodnoty rowLen a pro každý řádek vygeneruje dva typy dat:

- Textové zobrazení aminokyselin, které je řešeno funkcí getAminoAcids. Tato funkce umisťuje jednotlivé znaky na specifické pozice v rámci řádku.

- Vizuální reprezentaci sekundární struktury, kterou vytváří funkce makeVisualization. Tato funkce iteruje přes znaky sekundární struktury a podle jejich hodnoty (např. 'H' pro helix, 'B' nebo 'E' pro beta list, 'T','S','C' pro coil) volá odpovídající funkce pro vytvoření SVG elementu.

Ukázka funkce rowBlocks:

```
function rowBlocks(incr) {
  let data = [];
  for (let i = 0; i < height; i++) {
    let rest = rowLen > aminoAcid.length - (i * rowLen) ?
               aminoAcid.length - (i * rowLen) : rowLen;
    let insides = [getAminoAcids(i * rowLen, i * rowLen + rest),
                   makeVisualization(i * rowLen, i * rowLen + rest)];
    data.push(insides);
  }
  return data;
}
```

Funkce getAminoAcids nastaví textové elementy:

```
function getAminoAcids(from, to) {
  let insides = [];
  let x = 5.875;
  for (let i = from; i < to; i++) {
    insides.push(text(x, aminoAcid[i]));
    x += 11.75;
  }
  return insides;
}
```

Funkce makeVisualization poté zpracuje sekundární strukturu a vybere odpovídající vizualizaci:

```
function makeVisualization(from, to) {
  let visRow = '';
  let insides = [];
  for (let i = from; i < to; i++) {
    visRow += secondaryStructure[i] === 'H' ? '1' :
  (secondaryStructure[i] === 'B' || secondaryStructure[i] === 'E')
  ? '2' :
  (secondaryStructure[i] === 'T' || secondaryStructure[i] === 'S'
  || secondaryStructure[i] === 'C')
  ? '3' :
  '0';
  }
  let x = 5.875;
  for (let i = 0; i < to; i++) {
    if (visRow[i] === '1') {
      insides.push(helix(i % 2 === 0 ? '#icon-ss-Hb-8eaa'
      : '#icon-ss-Hf-8eaa', x));
    }
    if (visRow[i] === '2') {
      insides.push(sheet(11.75, x - 5.875));
    }
    if (visRow[i] === '3') {
      insides.push(coil(11.75, x - 5.875));
    }
    x += 11.75;
  }
  return insides;
}
```

4. Reaktivita a dynamická aktualizace Komponenta využívá reaktivní mechanismy Svelte, aby se vizualizace aktualizovala při změně vstupních dat. Díky reaktivnímu bloku $effect se při změně proměnné secondaryStructure znovu spustí generování vizualizace:

```
$effect(() => {
  if (secondaryStructure !== previousMyProp) {
    dataArr = rowBlocks(incr);
    previousMyProp = secondaryStructure;
  }
});
```

Navíc se při inicializaci komponenty pomocí funkce onMount zavolá generování řádků:

```
onMount(() => {
  dataArr = rowBlocks(incr);
});
```

Shrnutí Implementace Q3 vizualizace poskytuje standardizovanou, přehlednou a dynamickou reprezentaci sekundární struktury proteinu. Díky pečlivě navrženým pomocným funkcím a reaktivním mechanismům Svelte JS se jakékoliv změny ve vstupních datech okamžitě projeví ve vizuálním výstupu. Tento přístup umožňuje interaktivitu a efektivní prezentaci výsledků přímo ve webovém prohlížeči.

9.3.2 Q8 Vizualizace

Třída SecVis je navržena pro vizualizaci různých typů sekundárních struktur proteinů. Implementuje metody pro zobrazení struktur, jako jsou alfa-helixy, beta-bridges, beta-ladders, G-helixy a další. Každý typ struktury je reprezentován specifickým geometrickým útvarem na základě matematických vzorců, což umožňuje vizualizaci prostorových vlastností proteinů.

Metody vizualizace Třída SecVis obsahuje různé metody pro vizualizaci, které generují 2D reprezentace jednotlivých sekundárních struktur proteinů. Tyto metody zahrnují:

1. **AlphaHelix**: Vytváří sinusoidu reprezentující alfa-helix. Počet vln v sinusoide je určen parametrem num_waves. Každá vlna odpovídá jednomu otočení helixu.

2. **BetaBridge**: Znázorňuje dvě paralelní čáry spojené tečkovanými čarami, které symbolizují vodíkové můstky.

3. **BetaLadder**: Reprezentováno jako šipkový tvar, kde se jednotlivé beta-listy propojují pomocí šipky směřující k dalšímu prvku struktury.

4. **GHelix a PiHelix**: Podobně jako AlphaHelix, ale s upraveným počtem vln tak, aby odpovídaly specifickým vlastnostem G-helixu a Pi-helixu.

5. **HydrogenBondedTurn a Bend**: Vizualizace ohybů řetězce založené na vodíkových můstcích.

Každá z těchto metod generuje souřadnice.

Implementace třídy SecVis Třída SecVis poskytuje rozhraní pro vizualizaci sekundárních struktur. Používá metody pro generování souřadnic, které jsou převáděny do vizuální podoby. Základní atributy třídy zahrnují:

- **self.alpha_helix_waves** - Počet vln alfa-helixu.
- **self.beta_bridge_dotted_lines_spacing** - Rozestup mezi tečkovanými čarami v beta-bridge.
- **self.beta_ladder_arrow_height, self.beta_ladder_arrow_width** - Parametry určující velikost šipek u beta-ladderu.

Příklad implementace metody pro kreslení alfa-helixu:

```
def AlphaHelix(self, num_waves: int) -> tuple[np.ndarray]:
    d_y = self.y_axis / 2
    xy_coordinations = []
    for x_pos in range(self.end * 10, (self.end + self.x_axis) * 10):
        x_pos = x_pos / self.density
        y_pos = np.sin(x_pos / (self.x_axis / (2 * math.pi * num_waves)))
        * d_y + d_y
        xy_coordinations.append([x_pos, y_pos])
    return xy_coordinations
```

Podobně jsou implementovány i další metody pro generování vizualizací jednotlivých sekundárních struktur.

Použité vizualizace v projektu Ve finální implementaci projektu byla využita pouze jedna z těchto vizualizačních metod. Jelikož primárně pracujeme se standardizovanými vizualizacemi pro účely analýz a prezentací, byla zvolena metoda Q3 vizualizace. Tato metoda poskytuje přehledný a vyvážený pohled na sekundární strukturu proteinů, což umožňuje snadnou interpretaci dat. Q8 vizualizace, která poskytuje různé variace v geometrických tvarech (např. šipky pro beta-ladder struktury nebo sinusoidy pro helixy), nejsou ve finální verzi projektu použity.

Vzhledem k tomu, že Q3 vizualizace zajišťuje stabilní a univerzálně akceptovanou reprezentaci sekundárních struktur, byla vybrána jako jediná vizualizace použitá ve finálním projektu. Tato vizualizace je generována na frontendu, což umožňuje interaktivitu a snadnější úpravy při prezentaci výsledků.

10 Konečný stav a hodnocení projektu

10.1 Funkčnost a dosažené výsledky

Projekt úspěšně dosáhl stanovených cílů v oblasti predikce sekundární struktury proteinů. Kombinace modelů konvolučních neuronových sítí (CNN) a long short term memory(LSTM) umožnila významné zvýšení přesnosti predikce na 81 %, což představuje podstatný pokrok oproti tradičním metodám. Tento výsledek překonal původní očekávání a potvrdil efektivitu zvolené strategie.

Vyvinutá webová aplikace poskytuje uživatelům intuitivní rozhraní pro využití těchto pokročilých modelů. Aplikace umožňuje snadné nahrávání proteinových sekvencí, rychlé zpracování dat a přehlednou vizualizaci výsledků predikce. Implementace REST API zajišťuje flexibilitu a možnost integrace s jinými bioinformatickými nástroji.

10.2 Zhodnocení efektivity a kvality řešení

Efektivita řešení byla zajištěna několika klíčovými faktory. Optimalizace architektury neuronových sítí vedla ke snížení výpočetní náročnosti při zachování vysoké přesnosti predikce. Využití technik jako je batch normalization a dropout přispělo k lepší generalizaci modelů a snížení rizika přetrénování.

Kvalita predikce byla důkladně ověřena na standardních testovacích datasetech CB513 a CASP, kde modely dosáhly vysoké přesnosti. Tyto výsledky byly konzistentní napříč různými typy proteinů, což potvrzuje robustnost vyvinutého řešení. Navíc, použití křížové validace během trénování zajistilo spolehlivost modelů na nových, dosud neviděných datech.

Open-source přístup k aplikaci podporuje transparentnost a spolupráci v rámci vědecké komunity.

10.3 Porovnání s očekávanými výsledky

Očekávané výsledky byly významně překonány díky kombinaci modelů CNN a RNN. Původní cíle zahrnovaly dosažení přesnosti 74 % v predikci sekundární struktury, což bylo považováno za ambiciózní vzhledem k state-of-the-art výsledkům v době zahájení projektu. Výsledky však ukázaly, že zvolená kombinace modelů umožnila zvýšení přesnosti na 81 %.

11 Závěr a doporučení pro budoucí výzkum

11.1 Shrnutí klíčových zjištění

Klíčovým zjištěním projektu je, že kombinace modelů CNN a LSTM je vysoce efektivní pro predikci sekundární struktury proteinů. Tato synergie umožňuje zachytit jak lokální vzory v sekvenci aminokyselin (prostřednictvím CNN), tak dlouhodobé závislosti v proteinové struktuře (prostřednictvím RNN). Vyvinutá aplikace poskytuje uživatelům výkonný nástroj pro rychlou a přesnou analýzu proteinových sekvencí.

11.2 Kritické zhodnocení projektu

Projekt přinesl významné výsledky v oblasti predikce sekundární struktury proteinů, avšak některé aspekty zůstávají otevřené pro další zlepšení. Jedním z hlavních omezení je dostupnost komplexních dat o pH a teplotě organismů, ze kterých proteiny pocházejí. Tyto informace by mohly dále zvýšit přesnost predikcí.

Dalším aspektem, který vyžaduje pozornost, je interpretovatelnost modelů. Přestože dosahují vysoké přesnosti, jejich vnitřní fungování zůstává do značné míry "černou skříňkou". Rozvoj metod pro vizualizaci a interpretaci rozhodovacích procesů v neuronových sítích by mohl poskytnout cenné poznatky o principech skládání proteinů.

11.3 Závěrečná slova

Tento projekt významně přispěl k pokroku v oblasti predikce sekundární struktury proteinů a ukázal potenciál pokročilých metod strojového učení v bioinformatice. Vyvinutý nástroj poskytuje cenné informace pro výzkumníky v oblasti strukturní biologie, biochemie a farmaceutického vývoje.

Budoucí výzkum by měl pokračovat v rozvoji těchto metod, s důrazem na integraci různých zdrojů biologických dat. Tímto způsobem může strojové učení nejen zlepšit naše prediktivní schopnosti, ale také přispět k hlubšímu porozumění základním principům skládání proteinů a jejich funkcí.

Závěrem lze říci, že tento projekt položil kvalitní základ pro další inovace v oblasti strukturní bioinformatiky. Pokračující výzkum a vývoj v této oblasti má potenciál přinést revoluční objevy v pochopení a manipulaci proteinových struktur, s dalekosáhlými důsledky pro medicínu, biotechnologie a základní vědecký výzkum.

12 Seznam literatury

Odkazy

[1] KOČÁREK, Eduard. Genetika: obecná genetika a cytogenetika, molekulární biologie, biotechnologie, genomika. Biologie pro gymnázia. Praha: Scientia, 2004. ISBN 80-7183-326-6.

[2] ROSYPAL, Stanislav. Nový přehled biologie. Praha: Scientia, 2003. ISBN 80-7183-268-5.

[3] Recurrent Neural Networks. Dostupné z: `https://www.youtube.com/watch?v=AsNTP8Kwu80&t=206s`, [cit. 2025-03-27].

[4] Lipton, Zachary. (2015). A Critical Review of Recurrent Neural Networks for Sequence Learning.

[5] Purwono, Purwono & Ma'arif, Alfian & Rahmaniar, Wahyu & Imam, Haris & Fathurrahman, Haris Imam Karim & Frisky, Aufaclav & Haq, Qazi Mazhar Ul. (2023). Understanding of Convolutional Neural Network (CNN): A Review. International Journal of Robotics and Control Systems. 2. 739-748. 10.31763/ijrcs.v2i4.888.

[6] Cuff, J.A. & Clamp, Michele & Siddiqui, Asim & Finlay, M & Barton, Geoffrey. (1998). Jpred: A Consensus Secondary Structure Prediction Server. Bioinformatics (Oxford, England). 14. 892-3. 10.1093/bioinformatics/14.10.892.

[7] Høie, Magnus & Kiehl, Erik & Petersen, Bent & Nielsen, Morten & Winther, Ole & Nielsen, Henrik & Hallgren, Jeppe & Marcatili, Paolo. (2022). NetSurfP-3.0: accurate and fast prediction of protein structural features by protein language models and deep learning. Nucleic Acids Research. 50. 10.1093/nar/gkac439.

[8] ESM models https://github.com/facebookresearch/esm

[9] But how do neural networks work ? Dostupné z: https://www.youtube.com/watch?v=aircAruvnKk [cit. 2025-03-27]

[10] Gradient descent. Dostupné z: https://en.wikipedia.org/wiki/Gradient_descent [cit. 2025-03-27]

[11] Activation functions in neural networks.
Dostupné z : https://www.geeksforgeeks.org/activation-functions-neural-networks/ [cit. 2025-03-27]

[12] Algoritmus zpětného šíření chyby. Online. 2023.
Dostupné z: https://cs.wikipedia.org/wiki/Algoritmus_zp%C4%9Btn%C3%A9ho_%C5%A1%C3%AD%C5%99en%C3%AD_chyby. [cit. 2025-03-27].