VIP Cheatsheet: Transformery a Velké Jazykové Modely

Afshine Amidi a Shervine Amidi
Přeložil Aleš Horák

25. července 2025

Tento tahák poskytuje přehled toho, co obsahuje kniha "Super studijní příručka: Transformery a velké jazykové modely", ve které je ∼600 ilustrací na více než 250 stranách a která se do hloubky zabývá následujícími pojmy. Další podrobnosti najdete na adrese `https://superstudy.guide`.

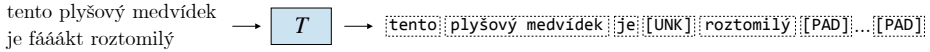
1 Základy

1.1 Tokeny

❒ **Definice** – *Token* je nedělitelná jednotka textu, například slovo, podslovo nebo znak, a je součástí předem definovaného slovníku.

Poznámka: Token neznámé slovo [UNK] představuje neznámé části textu, zatímco výplňový (padding) token [PAD] se používá k vyplnění prázdných pozic, aby se zajistila konzistentní délka vstupní sekvence.

❒ **Tokenizér** – *Tokenizér* T rozděluje text na tokeny s různou úrovní granularity.



Hlavní typy tokenizérů jsou:

Typ	Výhody	Nevýhody	Ukázka
Slovo	• Snadno interpretovatelný • Krátké sekvence	• Rozsáhlý slovník • Nezpracuje varianty slov	plyšový medvídek
Podslovo	• Využije kořeny slov • Intuitivní významové vektory	• Delší sekvence • Složitější tokenizace	plyš ##ový med ##vídek
Znak Bajt	• Žádné problémy s neznámými slovy • Malý slovník	• Mnohem delší sekvence • Vzory se obtížně interpretují, protože jsou na příliš nízké úrovni	p l y š o v ý m e d v í d e k

Poznámka: BPE (Byte-Pair Encoding) a Unigram jsou často používané podslovní tokenizéry.

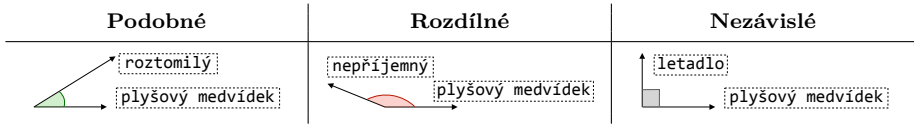
1.2 Slovní vektory

❒ **Definice** – *Slovní/větný vektor* je numerická reprezentace (významu) prvku (např. tokenu, věty) a je zadán jako vektor $x \in \mathbb{R}^n$.

❒ **Podobnost** – *Kosinová podobnost* mezi dvěma tokeny t_1, t_2 se vypočítá jako:

$$\text{podobnost}(t_1, t_2) = \frac{t_1 \cdot t_2}{||t_1||\ ||t_2||} = \cos(\theta) \in [-1, 1]$$

Úhel θ vyjadřuje podobnost dvou tokenů:

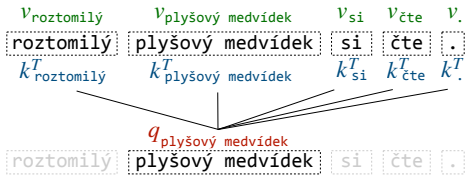


Poznámka: Approximate Nearest Neighbors (ANN) a Locality Sensitive Hashing (LSH) jsou metody, které efektivně provádějí aproximaci operace podobnosti v rozsáhlých databázích.

2 Transformery

2.1 Pozornost

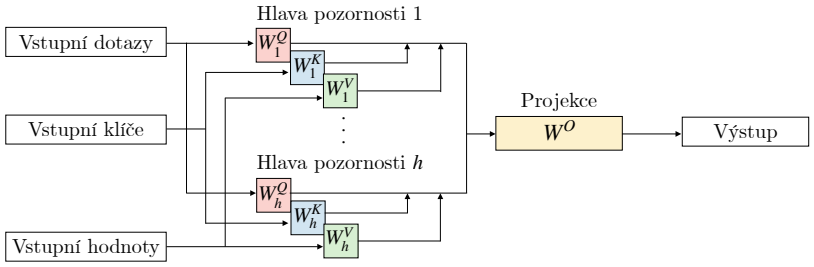
❒ **Výpočet** – Pro daný *dotaz* (query) q chceme vědět, kterému *klíči* (key) k má dotaz věnovat "pozornost" (attention) vzhledem k související *hodnotě* (value) v.



Pozornost jde efektivně vypočítat pomocí matic Q, K, V, které obsahují dotazy q, klíče k a hodnoty v spolu s dimenzí klíčů d_k:

$$\text{pozornost} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

❒ **MHA** – *Vrstva vícehlavá pozornost* (***M**ulti-**H**ead **A**ttention*, MHA) provádí výpočty pozornosti paralelně ve více hlavách a výsledek promítá do výstupního prostoru.

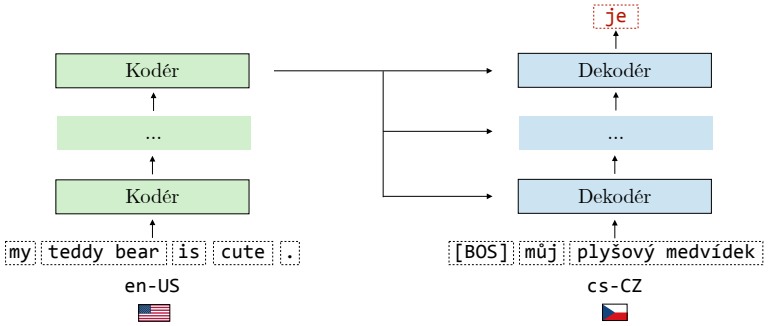


Skládá se z h hlav pozornosti a matic W^Q, W^K, W^V, které promítají vstupní data a získávají dotazy Q, klíče K a hodnoty V. Projekce se provádí pomocí matice W^O.

Poznámka: Grouped-Query Attention (GQA) a Multi-Query Attention (MQA) jsou varianty MHA, které snižují výpočetní režii sdílením klíčů a hodnot mezi hlavami pozornosti.

2.2 Architektura

❐ **Popis** – *Transformer* je referenční model, který se opírá o mechanizmus sebepozornosti (self-attention) a využívá propojení kodéru a dekodéru. Kodéry počítají vektorové reprezentace významu vstupu, které pak dekodéry používají k předpovědi dalšího tokenu v sekvenci.



Poznámka: Přestože byl transformer původně navržen jako model pro překladatelské úlohy, je nyní široce používán v mnoha dalších aplikacích.

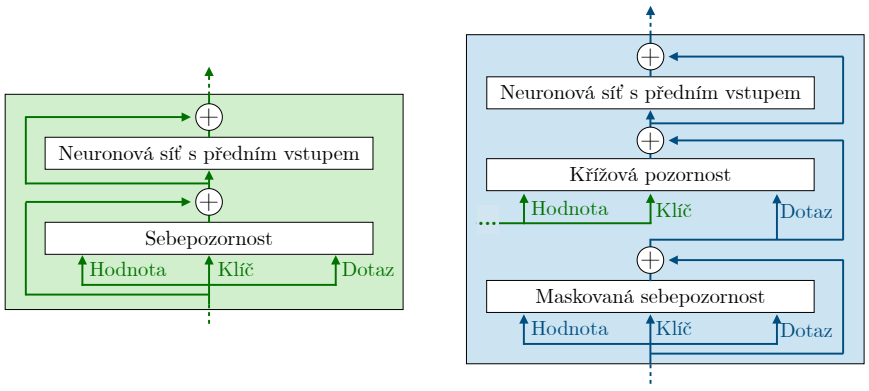
❐ **Komponenty** – *Kodér* a *dekodér* jsou dvě základní součásti transformeru a mají odlišné úlohy:

Kodér

Kódované vektory tokenů zachycují význam vstupu

Dekodér

Dekódované vektory tokenů zachycují význam vstupu i dosud predikované části výstupu

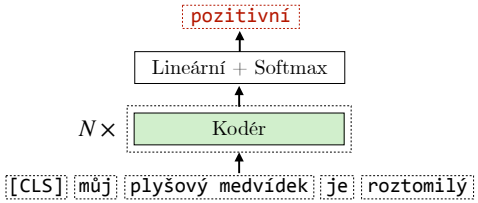


❐ **Vektory pozice** – *Vektory pozice* informují o tom, kde se token ve větě nachází, a mají stejný rozměr jako vektory tokenů. Mohou být buď definované předem nebo naučené z dat při trénování.

Poznámka: Rotary Position Embeddings (RoPE) jsou populární a efektivní variantou, která rotuje vektory dotazů (query) a klíčů (key) tak, aby zahrnovaly informace o relativní poloze.

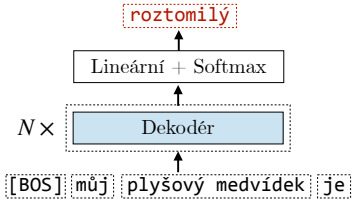
2.3 Varianty

❐ **Pouze kodér** – ***B**idirectional **E**ncoder **R**epresentations from **T**ransformers* (BERT) je model založený na transformerech, který se skládá z vrstev kodérů, na jejichž vstupu je text a výstupem jsou významové vektory, které jde později použít v navazujících klasifikačních úlohách.



Na začátek sekvence je přidán token `[CLS]`, který zachycuje význam věty. Vektor `[CLS]` tokenu se často používá v navazujících úlohách, jako třeba analýza sentimentu.

❐ **Pouze dekodér** – ***G**enerative **P**re-trained **T**ransformer* (GPT) je model založený na autoregresivním transformeru, který se skládá z vrstev dekodérů. Na rozdíl od modelu BERT a jeho odvozenin se v GPT ke všem problémům přistupuje jako k problémům typu text-text.



Většina současných nejmodernějších velkých jazykových modelů je založena na architektuře dekodéru, jako třeba série GPT, LLaMA, Mistral, Gemma, DeepSeek a další.

Poznámka: Modely kodér-dekodér, jako například model T5, jsou také autoregresivní a mají mnoho společných znaků s modely samotného dekodéru.

2.4 Optimalizace

❐ **Aproximace pozornosti** – Výpočty pozornosti jsou v $\mathcal{O}(n^2)$, což může být nákladné s rostoucí délkou sekvence n. Existují dvě hlavní metody aproximace výpočtů:

- *Řídkost*: Sebepozornost (self-attention) se nepočítá pro celou sekvenci, ale pouze mezi relevantnějšími tokeny.

- *Low-rank*: Vzorec pozornosti je zjednodušený jako součin matic nižších hodností (rank), což snižuje výpočetní zátěž.

❒ **Flash attention** – *Flash attention* (*blesková pozornost*) je přesná metoda, která optimalizuje výpočty pozornosti chytrým využitím hardwaru GPU a využívá rychlou paměť SRAM (***S**tatic **R**andom-**A**ccess **M**emory*) pro maticové operace před zápisem výsledků do pomalejší paměti HBM (***H**igh **B**andwidth **M**emory*).

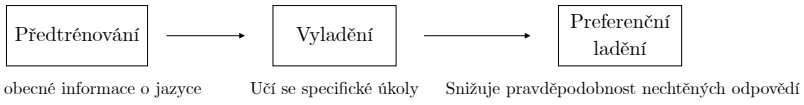
Poznámka: V praxi to snižuje spotřebu paměti a zrychluje výpočty.

3 Velké jazykové modely

3.1 Popis

❒ **Definice** – *Velký jazykový model* (***L**arge **L**anguage **M**odel*, LLM) je model založený na transformerech se skvělými schopnostmi zpracování přirozeného jazyka. Je "velký" v tom smyslu, že obvykle obsahuje miliardy parametrů.

❒ **Životní cyklus** – LLM se trénuje ve třech krocích: předtrénování, vyladění a preferenční ladění.

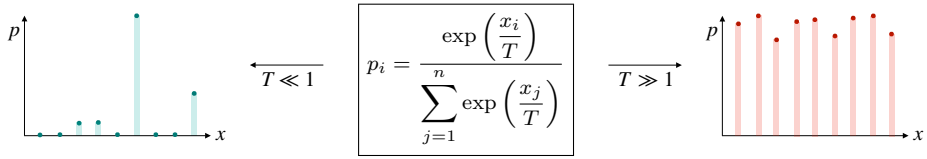


Vyladění a preferenční ladění jsou post-tréninkové přístupy, jejichž cílem je přizpůsobit model k provádění určitých úkolů.

3.2 Dotazování

❒ **Délka kontextu** – *Délka kontextu* modelu je maximální počet tokenů, které se vejdou na vstup. Obvykle se pohybuje od desítek tisíc až po miliony tokenů.

❒ **Výběr tokenů pro generování** – Generované tokeny jsou vzorkovány z predikované pravděpodobnostní distribuce pi, která je řízena hyperparametrem teplota T.



Poznámka: Vysoké teploty vedou ke kreativnějším výstupům, zatímco nízké teploty vedou k determinističtějším výstupům.

❒ **Myšlenkový postup** – *Myšlenkový postup* (***C**hain-**o**f-**T**hought*, CoT) je proces uvažování, při kterém model rozkládá složitý problém na řadu mezikroků. To pomáhá modelu generovat správnou konečnou odpověď. *Myšlenkový strom* (***T**ree **o**f **T**houghts*, ToT) je pokročilejší verze CoT.

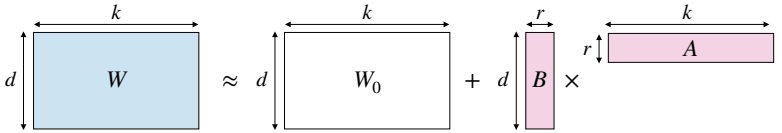
Poznámka: Sebekonzistence (self-consistency) je metoda, která agreguje odpovědi napříč cestami uvažování CoT.

3.3 Vyladění

❒ **SFT** – *Vyladění s dohledem* (***S**upervised **F**ine**T**uning*, SFT) je post-tréninkový přístup, který přizpůsobuje chování modelu konečné úloze. Opírá se o kvalitní vstupně-výstupní páry sladěné s úlohou.

Poznámka: Pokud se data SFT týkají instrukcí nebo dialogu, pak se tento krok nazývá "instrukční ladění" (instruction tuning).

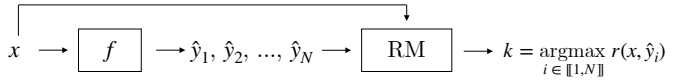
❒ **PEFT** – ***P**arameter-**E**fficient **F**ine**T**uning* (PEFT) je kategorie metod, které se používají k efektivnímu provádění SFT. Zejména ***Lo**w-**R**ank **A**daptation* (LoRA) aproximuje trénovatelné váhy W tím, že zafixuje W_0 a místo toho se učí matice A, B s nízkou hodností:



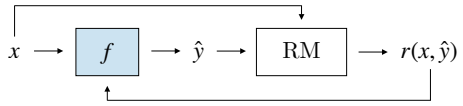
Poznámka: Mezi další techniky PEFT patří ladění prefixů a vkládání adaptérové vrstvy.

3.4 Preferenční ladění

❒ **Model odměn** – *Model odměn* (***R**eward **M**odel*, RM) je model, který předpovídá, jak dobře výstup $\hat{y}$ odpovídá požadovanému chování vzhledem ke vstupu x. Vzorkování ***B**est-**o**f-**N*** (BoN), nazývané také rejection sampling, je metoda, která používá model odměn k výběru nejlepší odpovědi z N vygenerovaných.



❒ **Zpětnovazební učení** – *Zpětnovazební učení* (***R**einforcement **L**earning*, RL) je přístup, který využívá RM a aktualizuje model f na základě odměn za jeho generované výstupy. Pokud je RM založen na lidských preferencích, nazývá se tento proces ***R**einforcement **L**earning from **H**uman **F**eedback* (RLHF).

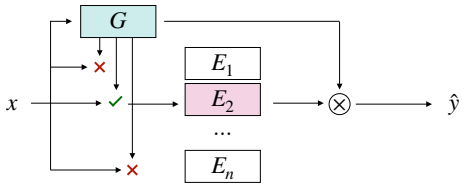


***P**roximal **P**olicy **O**ptimization* (PPO) je populární RL algoritmus, který motivuje k vyšším odměnám a zároveň udržuje model v blízkosti základního modelu, aby se zabránilo nabourání odměn (reward hacking).

Poznámka: Existují také přístupy s dohledem, jako je Direct Preference Optimization (DPO), které kombinují RM a RL v jednom kroku s dohledem.

3.5 Optimalizace

❒ **Směs expertů** – *Směs expertů* (***M**ixture **o**f **E**xperts*, MoE) je model, který v době inference aktivuje pouze část svých neuronů. Je založen na bráně G a expertech $E_1, ..., E_n$.



$$\hat{y} = \sum_{i=1}^{n} G(x)_i E_i(x)$$

LLM založené na MoE používají tento mechanizmus bran ve svých vrstvách neuronových sítí s předním vstupem (FFNN).

Poznámka: Trénování LLM založeného na MoE je notoricky náročné, jak je zmíněno v článku LLaMA, jehož autoři se rozhodli tuto architekturu nepoužít i přes její efektivitu při inferenci.

❒ **Destilace** – *Destilace* (*distillation*) je proces, při kterém se (malý) studentský model S trénuje na výstupech predikce (velkého) modelu učitele T. Trénuje se pomocí ztrátové funkce KL divergence:

$$\text{KL}(\hat{y}_T || \hat{y}_S) = \sum_i \hat{y}_T^{(i)} \log\left(\frac{\hat{y}_T^{(i)}}{\hat{y}_S^{(i)}}\right)$$

Poznámka: Trénovací výstupy (labels) jsou považovány za "měkké" hodnoty, protože představují pravděpodobnosti tříd.

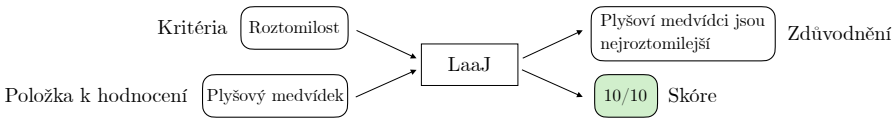
❒ **Kvantizace** – *Kvantizace modelu* je kategorie technik, která snižuje přesnost vah modelu a zároveň omezuje dopad přesnosti na výsledný výkon modelu. V důsledku toho se snižuje paměťová náročnost modelu a zrychluje se inference.

Poznámka: QLoRA je běžně používaná kvantizační varianta LoRA.

4 Aplikace

4.1 LLM jako posuzovatel

❒ **Definice** – LLM jako posuzovatel (***L**LM-**a**s-**a**-**J**udge*, LaaJ) je metoda, která využívá LLM k hodnocení zadaných výstupů podle určitých kritérií. Za zmínku stojí fakt, že LaaJ je schopna generovat i zdůvodnění svého skóre, což napomáhá jeho interpretovatelnosti.



Na rozdíl od metrik z doby před používáním LLM, jako je ***R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation* (ROUGE), LaaJ nepotřebuje žádný referenční text, což tuto metodu činí vhodnou pro vyhodnocení jakéhokoli druhu úlohy. LaaJ vykazuje silnou korelaci s lidským hodnocením zejména tehdy, když se opírá o velký výkonný model (např. GPT-4), protože k dobrému výkonu potřebuje schopnosti uvažování (reasoning).

Poznámka: LaaJ je užitečná pro provádění rychlých kol hodnocení, ale je důležité sledovat shodu mezi výstupy LaaJ a lidskými hodnoceními, aby se zajistilo, že nedochází k žádným odchylkám.

❒ **Předpojatost modelu** – Modely LaaJ se mohou "chovat předpojatě" (biased):

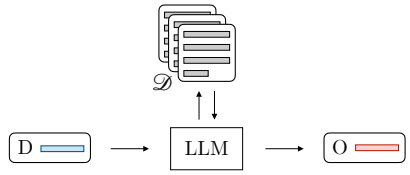
	Předpojatost k pozici	**Předpojatost k délce textu**	**Předpojatost k vlastnímu výstupu**
Problém	Upřednostňuje první pozici v párových srovnáních	Upřednostňuje obsáhlejší texty	Upřednostňuje výstupy, které generuje sám
Řešení	Zprůměrování metriky na náhodně vybraných pozicích	Přidání penalizace za délku výstupu	Použití posuzovatele sestaveného z jiného základního modelu

Řešením těchto problémů může být vyladění vlastního LaaJ, což však vyžaduje velké úsilí.

Poznámka: Výše uvedený seznam předpojatostí není úplný.

4.2 RAG

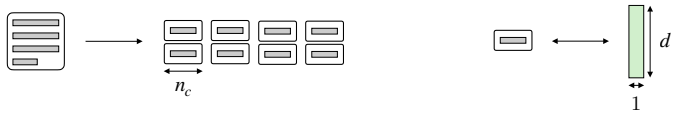
❒ **Definice** – *Generování založené na vyhledávání* (***R**etrieval-**A**ugmented **G**eneration*, RAG) je metoda, která umožňuje LLM přistupovat k relevantním externím znalostem pro zodpovězení zadané otázky. To je užitečné zejména v případě, že chceme zahrnout informace z doby po ukončení procesu předtrénování LLM.



Pro danou bázi znalostí $\mathscr{D}$ a dotaz vybere vyhledávač nejrelevantnější dokumenty, následně rozšíří dotaz o relevantní informace a vygeneruje výstup.

Poznámka: Fáze výběru nejrelevantnějších dokumentů se obvykle opírá o vektorové reprezentace z kódovacích (encoder-only) modelů.

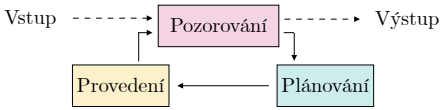
❒ **Hyperparametry** – Báze znalostí $\mathscr{D}$ se inicializuje rozdělením dokumentů na části o velikosti n_c a jejich převedení na vektory dimenze $\mathbb{R}^d$.



4.3 Agenti

❒ **Definice** – *Agent* je systém, který autonomně sleduje cíle a plní úkoly jménem uživatele. Může k tomu používat různé řetězce volání LLM.

❒ **ReAct** – *Uvažuj + proveď* (***Re**ason + **Act***, ReAct) je postup, který umožňuje více řetězců volání LLM k dokončení složitých úloh:



Tento postup se skládá z následujících kroků:

- *Pozorování*: Syntéza předchozích akcí a explicitní vyjádření toho, co je v současné době známo.
- *Plánování*: Podrobný popis úkolů, které je potřeba splnit, a nástrojů, které se mají použít.
- *Provedení*: Vykonání akce prostřednictvím rozhraní API nebo vyhledání relevantních informací v bázi znalostí.

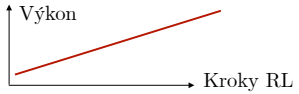
Poznámka: Evaluace agentního systému je náročná. Přesto ji lze provádět jak na úrovni komponent prostřednictvím lokálních vstupů a výstupů, tak na úrovni systému prostřednictvím řetězců volání.

4.4 Argumentační modely

❒ **Definice** – Argumentační (reasoning) model je model, který při řešení složitějších úloh v matematice, programování a logice vychází z vlastních argumentací založených na CoT. Mezi příklady argumentačních modelů patří série o společnosti OpenAI, DeepSeek-R1 a Gemini Flash Thinking společnosti Google.

Poznámka: DeepSeek-R1 explicitně vypisuje svou argumentaci mezi značkami `<think>`.

❒ **Škálování** – K posílení argumentačních schopností se používají dva typy škálovacích metod:

	Popis	**Schéma**
Škálování v průběhu trénování	Spustit RL na delší dobu, aby se model naučil vytvářet argumentace ve stylu CoT předtím, než poskytne odpověď.	
Škálování v průběhu testování	Nechat model déle přemýšlet před poskytnutím odpovědi pomocí slov pro vynucení argumentace, jako třeba "Počkat, ...".