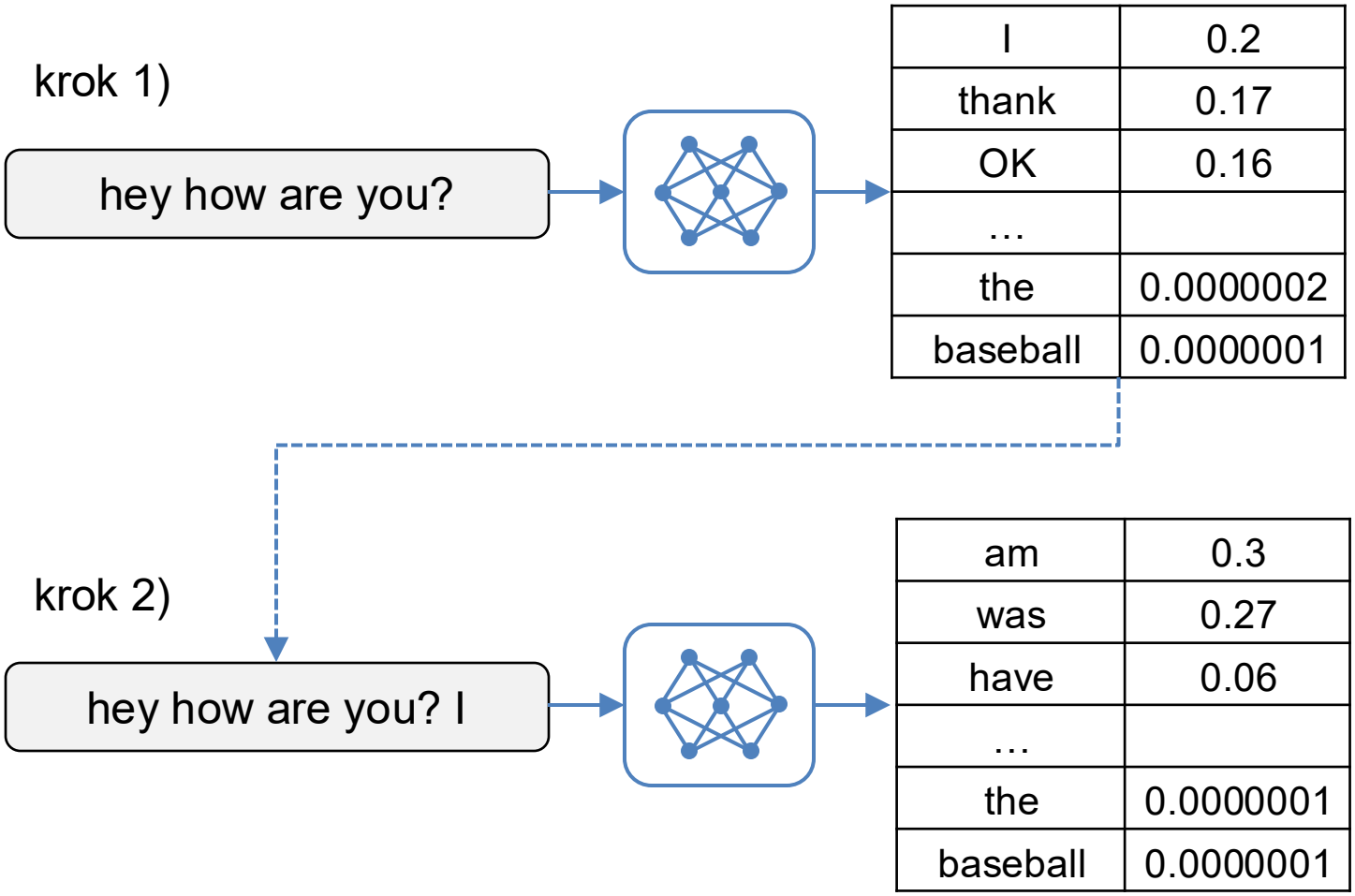
TEXT MINING 3

Objavovanie znalostí v textoch

Peter Bednár

Jazykové modely

Jazykové modely



Aké úlohy vie „riešiť“ jazykový model?

- **Vstup**: Kto bol Vincent Van Gogh?
- **Výstup**: Vincent Van Gogh bol holandský maliar 19 storočia.

- **Vstup**: Aký je sentiment vety „s ubytovaním sme boli spokojní“?
- **Výstup**: Veta má pozitívny sentiment.

- **Vstup**: Prelož vetu „kde je múzeum“ do taliančiny.
- **Výstup**: Dov'è il museo.

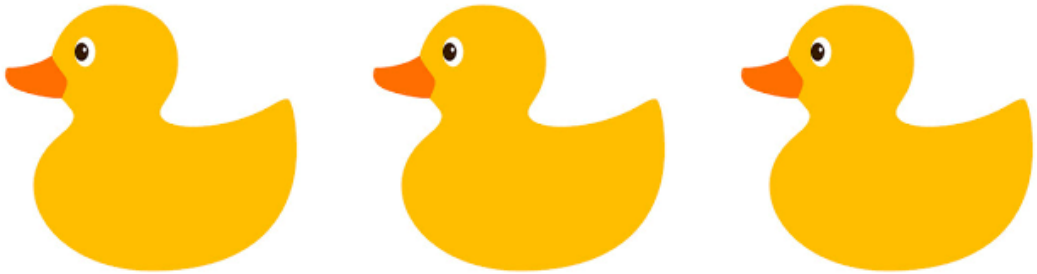
Aké úlohy vie „riešiť“ jazykový model?

- **Vstup**: Aké je ďalšie číslo v postupnosti 1 14 27 40?
- **Výstup**: Ďalšie číslo je 53, pretože rozdiel medzi nasledujúcimi číslami je vždy 13.

- **Vstup**: Idú dve kačiatka pred kačiatkom, dve kačiatka za kačiatkom a jedno kačiatko v strede. Koľko ide kačiatok?
- **Výstup**: ?

Aké úlohy vie „riešiť“ jazykový model?

- **Vstup**: Idú dve kačiatka pred kačiatkom, dve kačiatka za kačiatkom a jedno kačiatko v strede. Koľko ide kačiatok?
- **Výstup**: Idú 3 kačiatka.

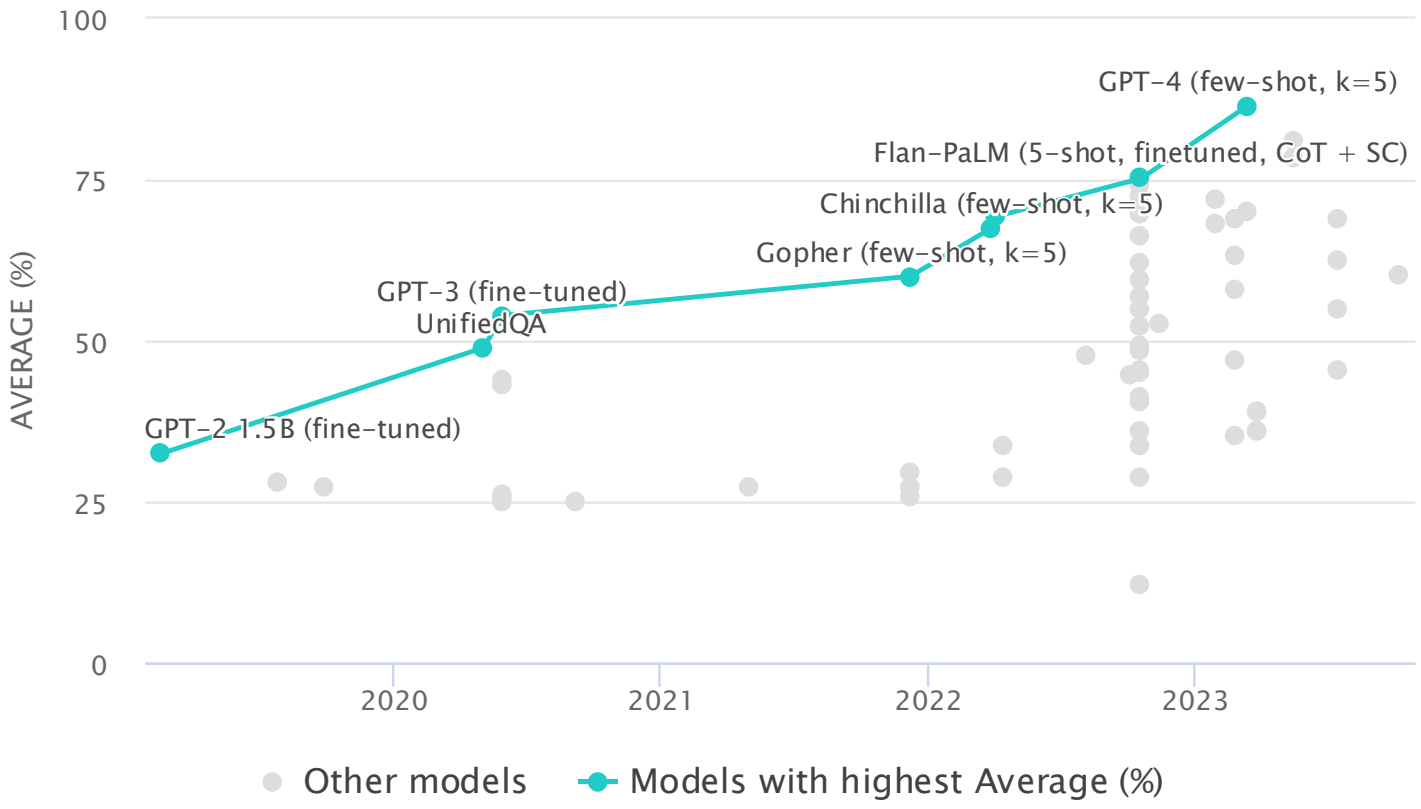


Testovanie jazykových modelov

- HellaSwag
  - Úlohy, ktoré vyžadujú jednoduché uvažovanie (*common sense*), jednoduché pre človeka ale zložité pre predchádzajúce UI systémy
- MMLU
  - 57 úloh zameraných na všeobecné znalosti z rôznych oblastí ako je napr. matematika, história, počítačové vedy, právo, náboženstvá atď.
- TruthfulQA
  - Sklony generovať nepravdy, ktoré sa často vyskytujú online

- Zjednotené metódy testovania
  - HuggingFace Open LLM Leaderboard
  - Stanford HELM

Testovanie jazykových modelov

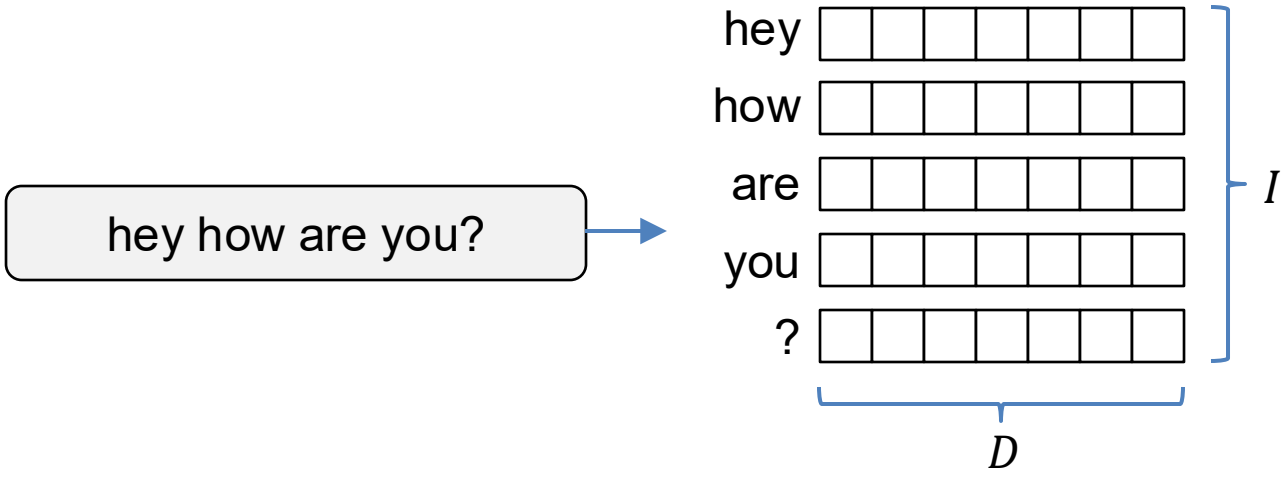


Transformer

Transformer

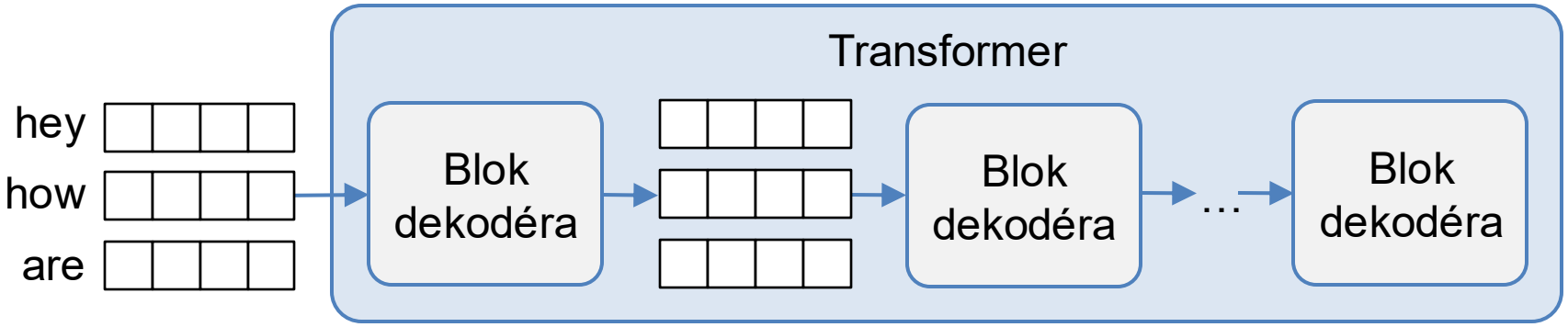
Tokenizácia a Vektorová reprezentácia slov

- Text sa rozdelí na tokeny
  - Slovník častých slov a podreťazcov
  - *Elvégezhetitek* | *El* | *#vé* | *#ge* | *#zhet* | *#ite* | *#k* |
  - *Aannialiqpasaarama iksivaassuujaluarnikumut*
- Každému tokenu je priradený číselný vektor – *embedding*
  - Parametre, ktoré sa menia pri učení



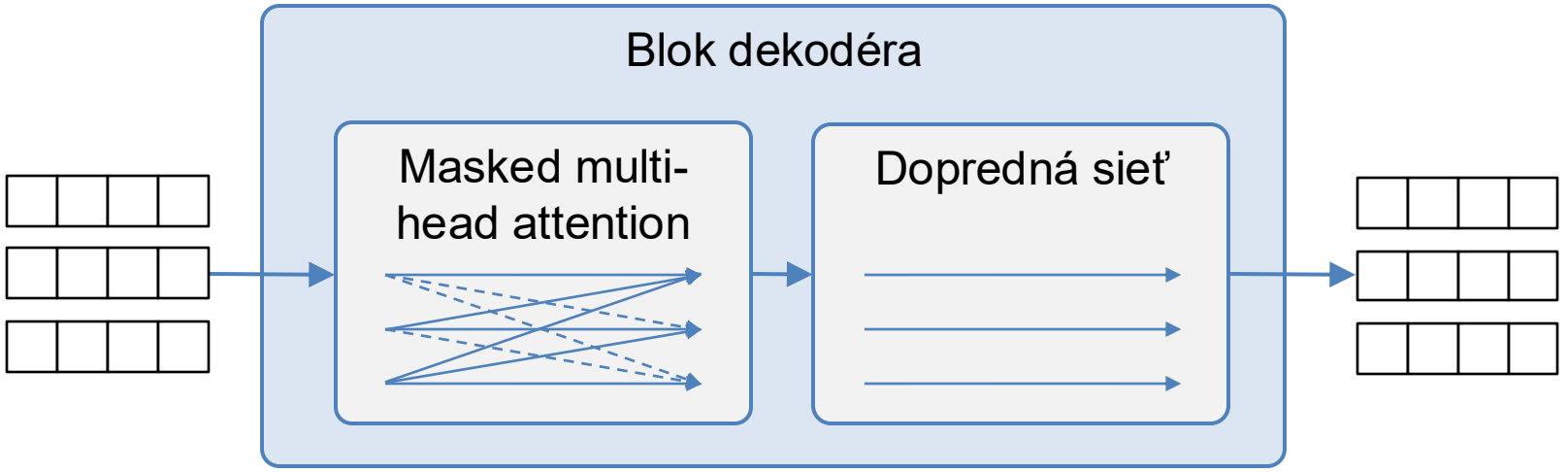
Transformer

- Hlboká sieť zložená z blokov, ktoré dekódujú vstupnú maticu tokenov



Dekodér

- Samo-pozornostný model, ktorý sa učí vzťahy medzi tokenmi na vstupe + dopredná sieť aplikovaná na každý token



Samo-pozornostný model

- Pre každý vstupný token $\boldsymbol{x}_i$ je výstup vektor, ktorý kombinuje vektorovú reprezentáciu daného tokenu so všetkými tokenmi na vstupe

$$sa[\mathbf{x}_i] = \sum_{j=0}^{N} a_{i,j} \boldsymbol{\Phi}_v \mathbf{x}_j$$

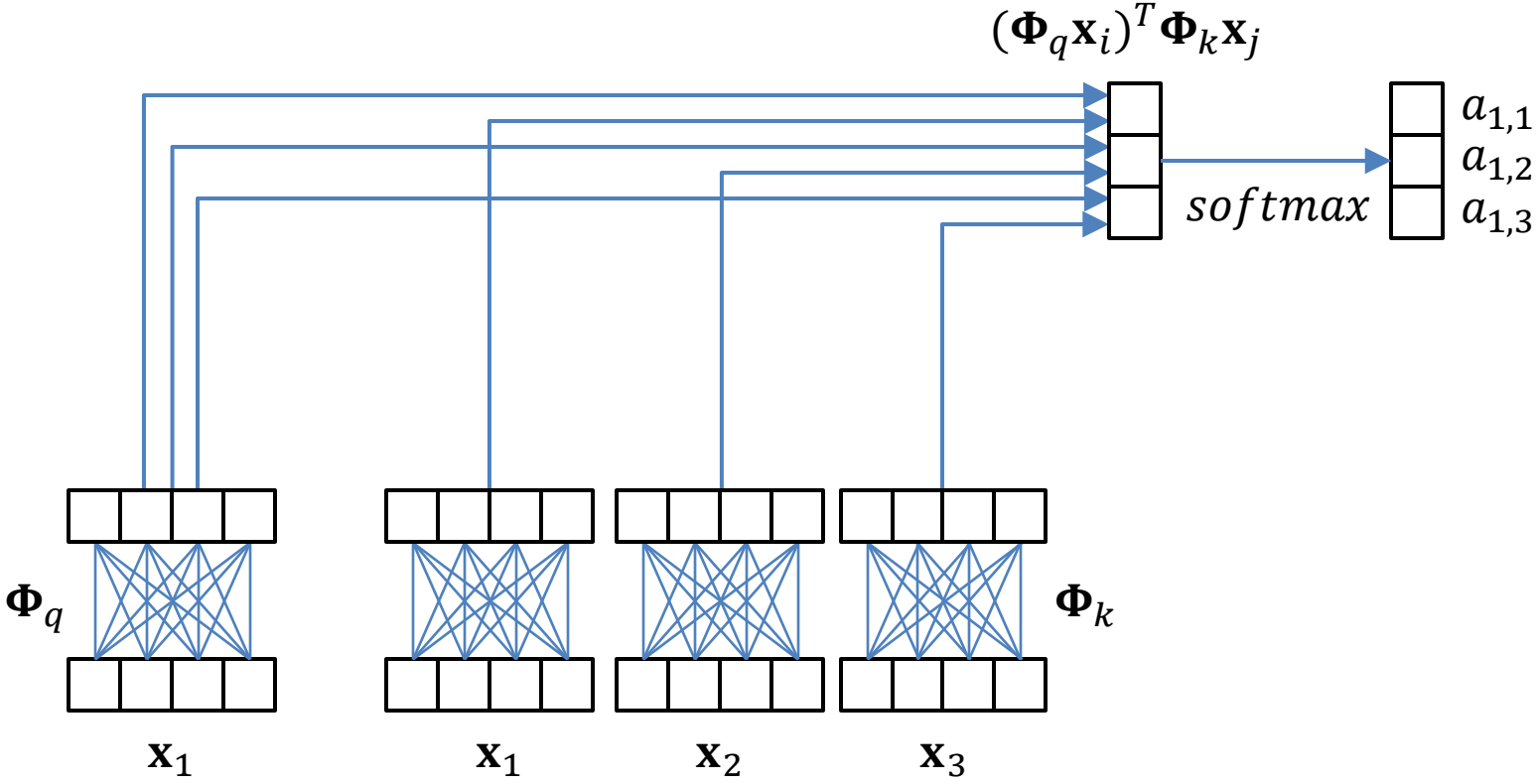
1. Lineárna transformácia pomocou matice $\boldsymbol{\Phi}_v$ (rovnaká pre každý vstupný token)
2. Vážená suma pre všetky tokeny, kde váha $a_{i,j}$ vyjadruje ako veľmi token j súvisí s tokenom i – tzv. *pozornosť* i na j

Samo-pozornostný model

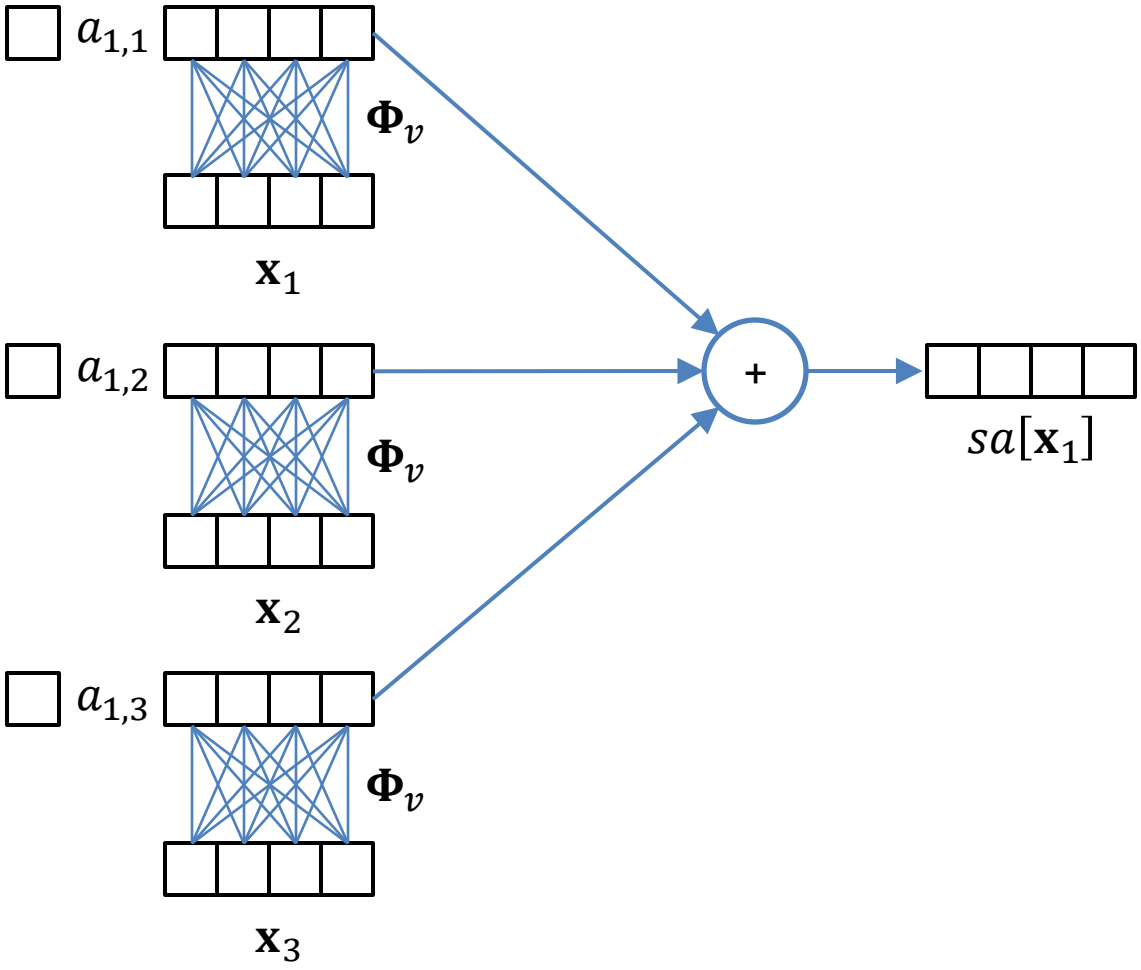
- Pozornosť $a_{i,j}$ medzi dvoma tokenmi závisí od podobnosti ich vektorov
  - Skalárny súčin
- Podobnosť vypočítame až po lineárnom transformovaní vektorov $\mathbf{x}_i$ a $\mathbf{x}_j$ pomocou matíc $\mathbf{\Phi}_q$ a $\mathbf{\Phi}_k$
- Chceme aby pozornosti tvorili distribúciu (rozsah v intervale 0-1, suma pre všetky tokeny rovná 1)
  - Softmax funkcia

$$a_{i,j} = softmax\left[(\mathbf{\Phi}_q \mathbf{x}_i)^T \mathbf{\Phi}_k \mathbf{x}_j\right]$$

Samo-pozornostný model



Samo-pozornostný model



Rozšírenia pozornostného modelu

- Normalizácia
- Skalárny súčin vektorov môže nadobúdať veľkú hodnotu čo môže spôsobiť numerické problémy pri učení (veľmi malé hodnoty gradientu) – normalizácia, kde d_q je počet stĺpcov $\mathbf{\Phi}_q$ a $\mathbf{\Phi}_k$

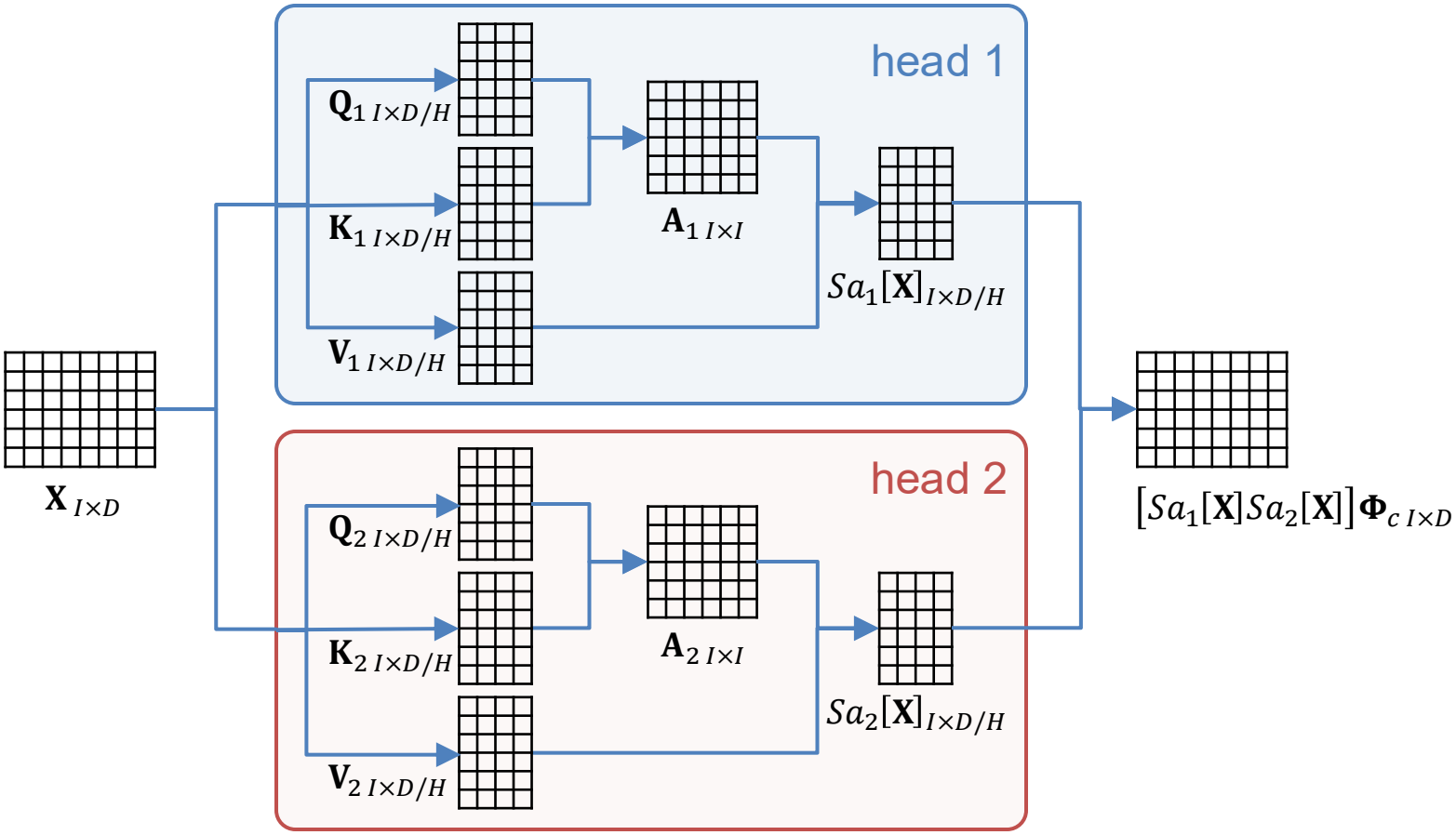
$$a_{i,j} = softmax\left[\frac{(\mathbf{\Phi}_q \mathbf{x}_i)^T \mathbf{\Phi}_k \mathbf{x}_j}{\sqrt{d_q}}\right]$$

Rozšírenia pozornostného modelu

- Multi-head attention
- Rozdelíme vstupné vektory $\mathbf{x}_i$ na rovnako veľké časti a pre každú časť h - *head* budeme mať samostatné parametre $\mathbf{\Phi}_{q,h}$, $\mathbf{\Phi}_{k,h}$ a $\mathbf{\Phi}_{v,h}$
- Výstupné matice spojíme do jednej a lineárne transformujeme
- Maticový zápis:

$$MhSA(\mathbf{X}) = \left[Sa_1[\mathbf{X}]Sa_2[\mathbf{X}] \ldots Sa_H[\mathbf{X}]\right]\mathbf{\Phi}_c$$

Multi-head attention



Pozičné kódovanie

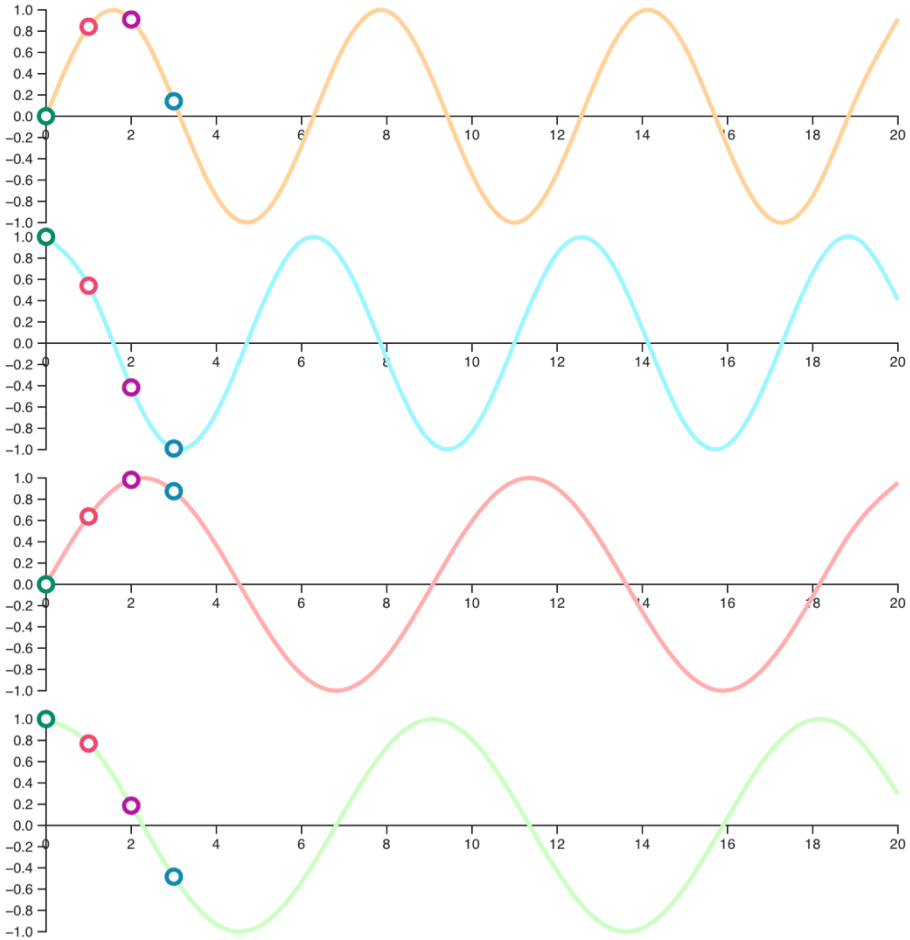
- Samotná pozornostná vrstva nerozlišuje poradie tokenov
- K vstupnému vektoru pre každý token $\boldsymbol{x}$ pripočítame vektor, ktorý kóduje jeho pozíciu v texte:

$$PE(pos)_{2j} = sin\left(\frac{pos}{1000^{2j/I}}\right)$$

$$PE(pos)_{2j+1} = cos\left(\frac{pos}{1000^{2j/I}}\right)$$

- kde j je zložka vektora a I je rozmer vstupu (maximálna dĺžka vstupnej sekvencie tokenov)

Pozičné kódovanie

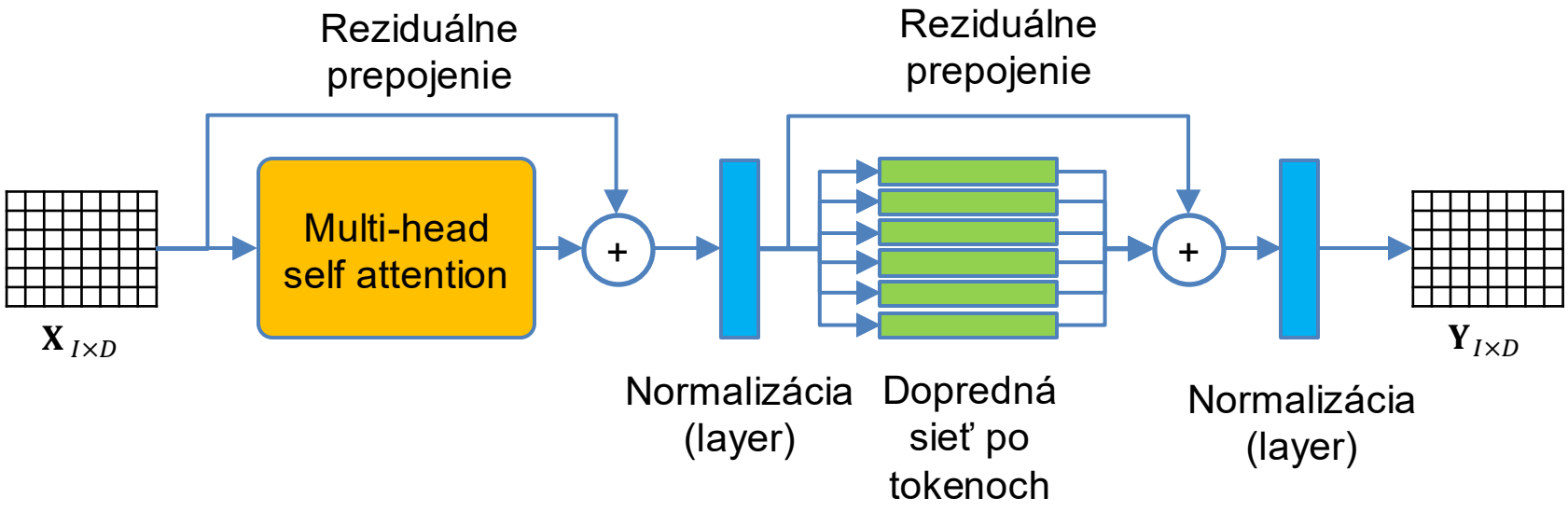


	j=0	j=1	j=2	j=3
0	0.00	1.00	0.00	1.00
1	0.84	0.54	0.63	0.77
2	0.90	-0.41	0.98	0.18
3	0.90	-0.41	0.98	0.18

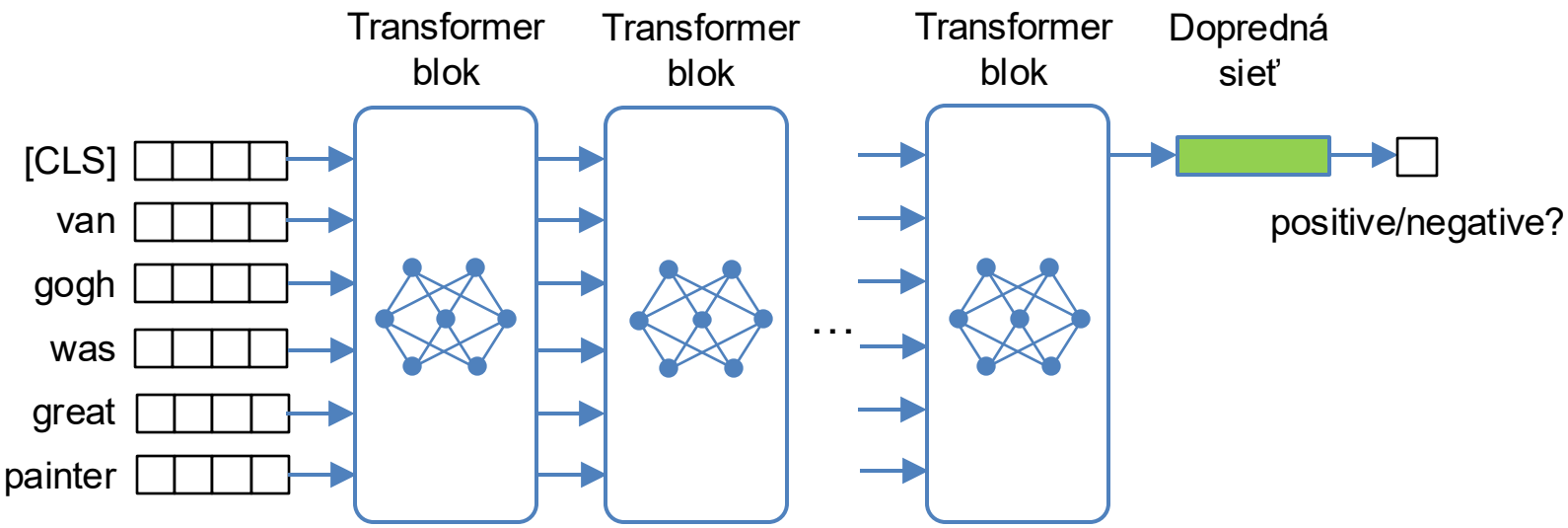
...

Transformer blok

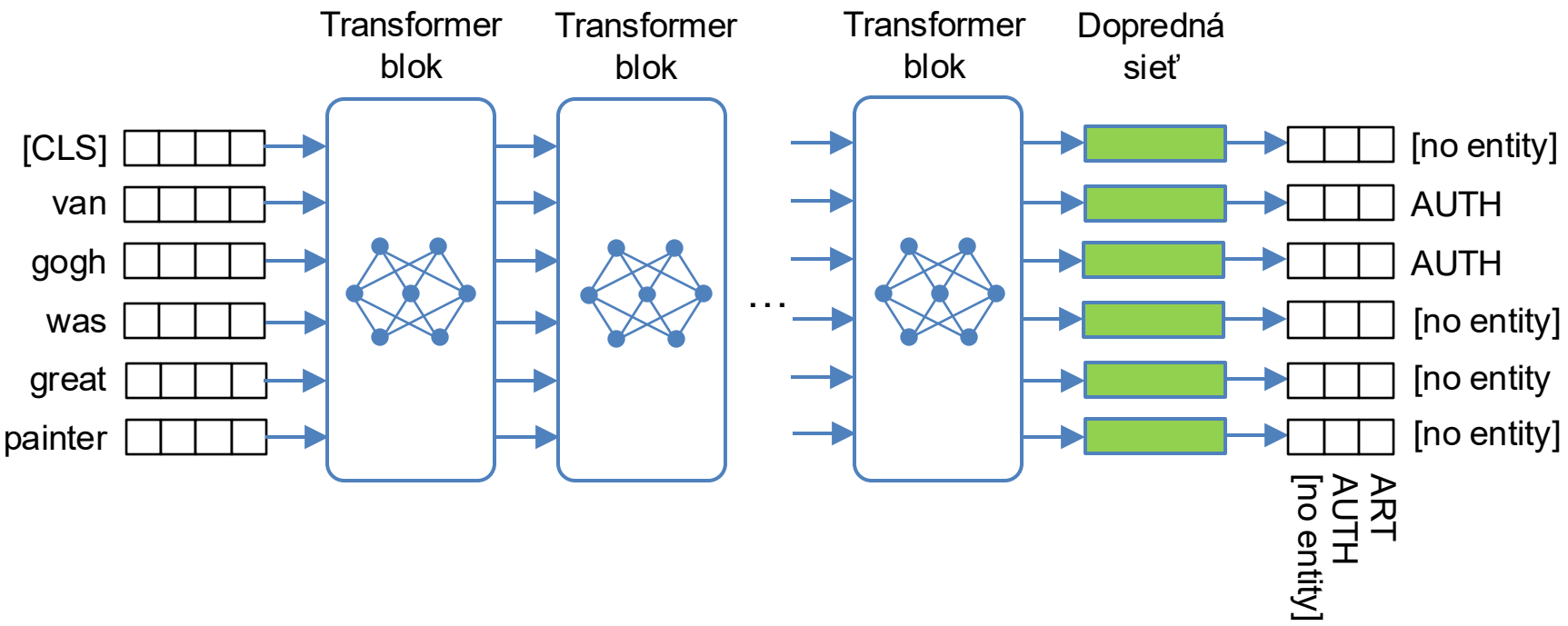
- *Multi-head self attention* vrstva + reziduálne prepojenia + normalizácia + dopredná sieť (ktorá má na vstupe jednotlivé vektory reprezentujúce tokeny)



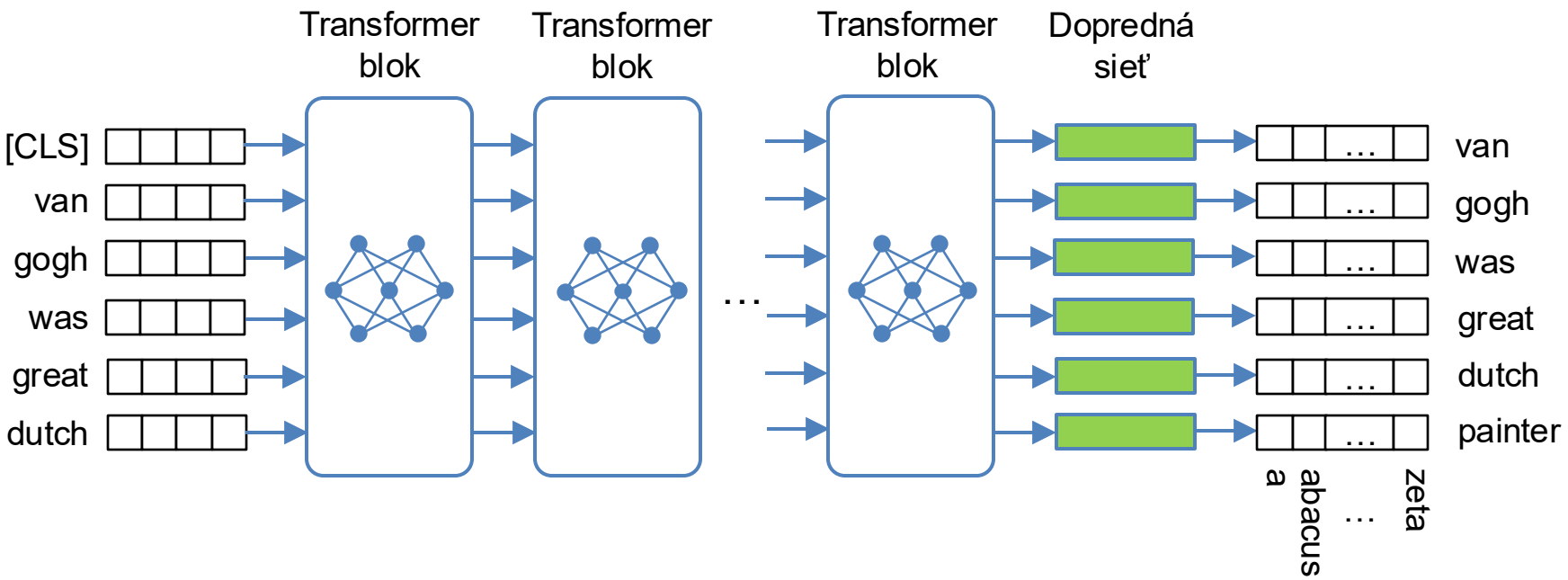
Klasifikácia/analýza sentimentu



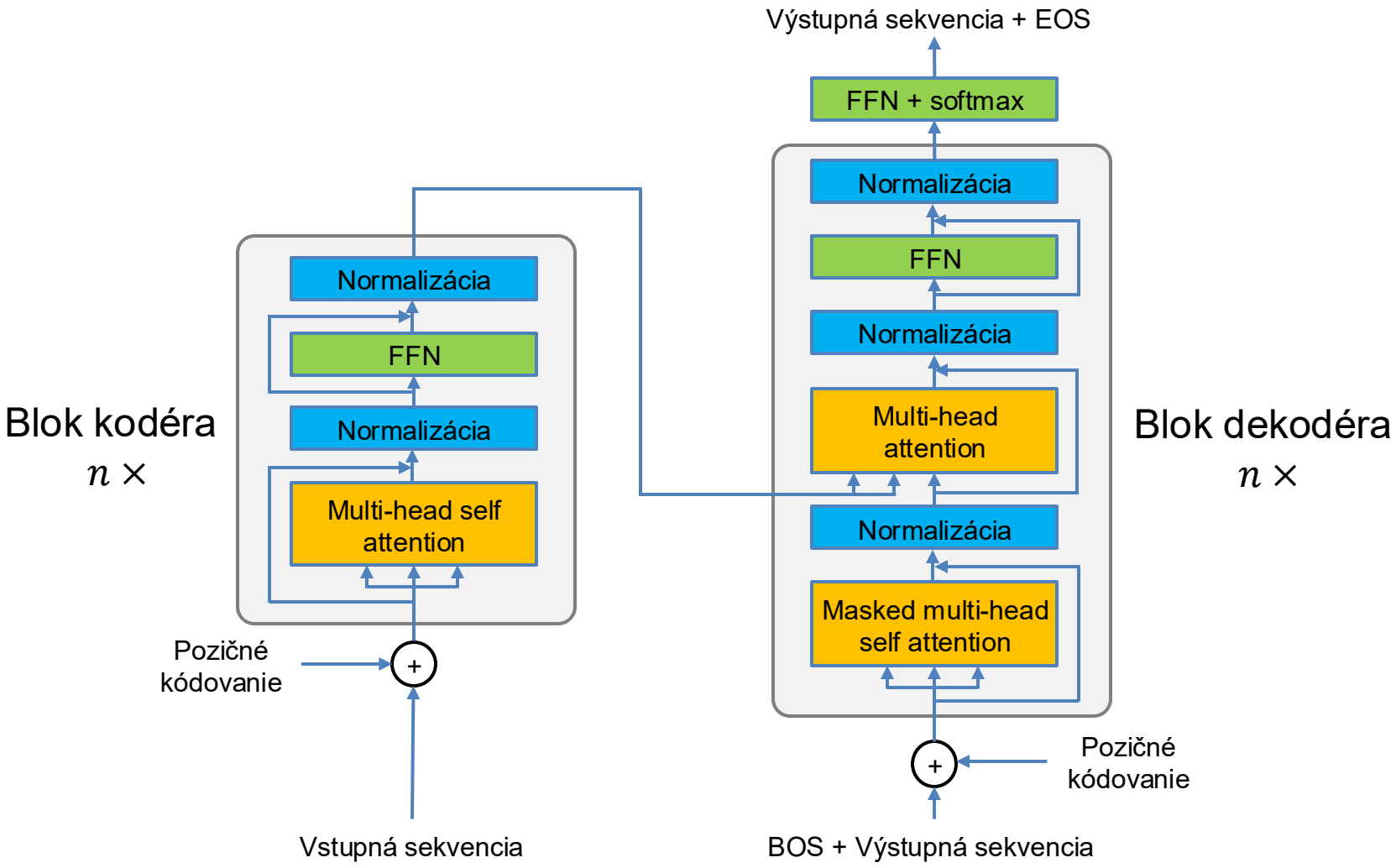
Extrahovanie entít



Jazykový model



Kodér-dekodér – transformer architektúra



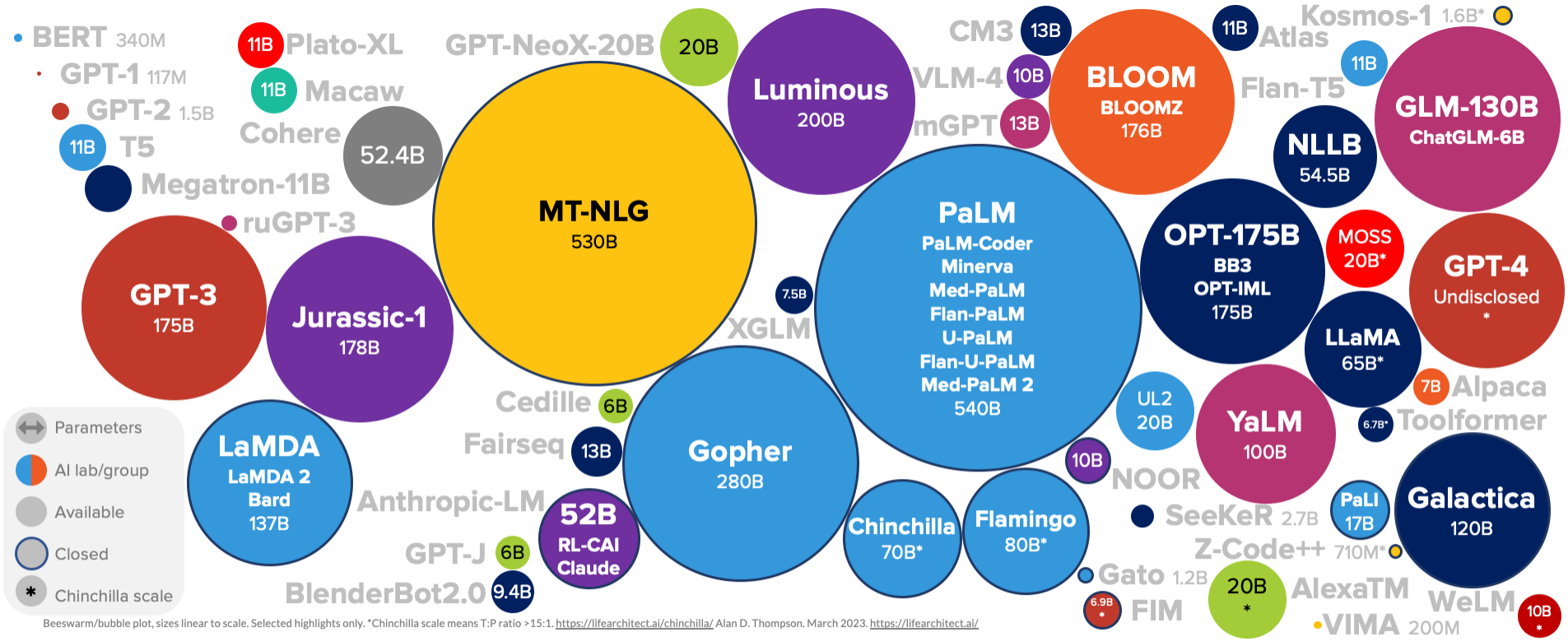
BERT

- Pôvodný model architektúry pre Transformer siete

- Slovník 30 000 tokenov, 1024 rozmerné vektory (*embeddings*)
- 24 blokov, 16 hláv (tzn. matice $\mathbf{\Phi}_{q,h}$, $\mathbf{\Phi}_{k,h}$ a $\mathbf{\Phi}_{v,h}$ majú rozmer 1024×64), skrytá vrstva doprednej siete má 4096 neurónov
- Celkový počet parametrov cca 340 mil.

Veľké jazykové modely

M – 1 000 000, B – 1 000 000 000



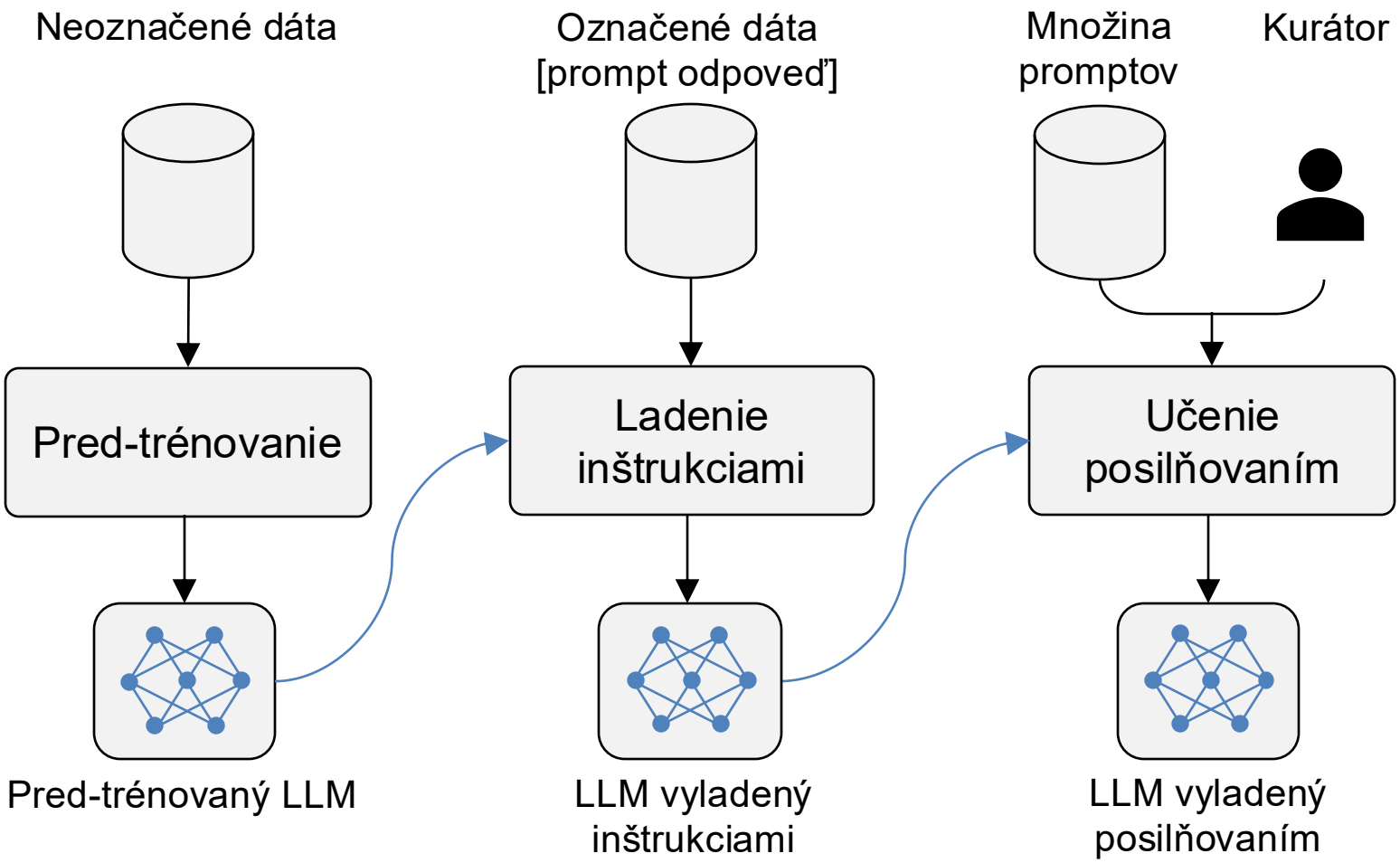
LifeArchitect.ai/models

Trénovanie veľkých jazykových modelov

- Trénovanie modelu LLaMA – 65 miliárd (≈ 65 000 000 000) parametrov vyžaduje 21 dní, 2048 A100 grafických kariet (každá má 80 GB pamäte)
- Odhadované náklady pre GPT3 boli $4.6M

- Open source modely
  - Falcon
  - OpenLLaMA / LLaMA 2
  - Alpaca
  - MPT
  - FastChat-T5
  - Bloom

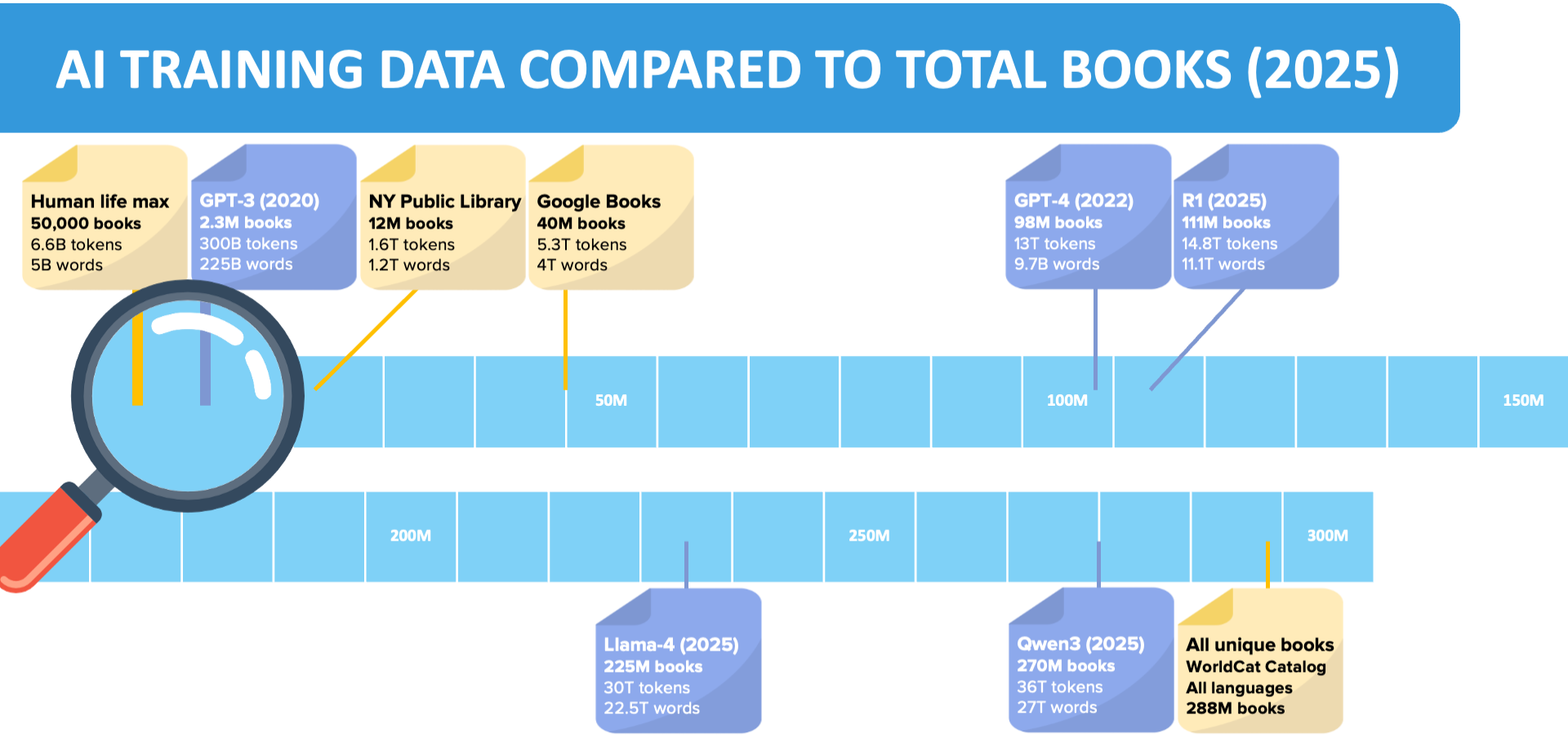
Trénovanie veľkých jazykových modelov



Pred-trénovanie

- Samo-kontrolované učenie
  - Predikcia nasledujúceho tokenu
  - Cieľom je hlavne zachytiť jazykové vlastnosti
- Rozsiahle textové dáta
  - Web, Wikipédia, knihy, ...
- Rozsah dát
  - PALM – 780 miliárd tokenov
  - RedPajama – 1.2 biliónov tokenov, 2.67 TB dát

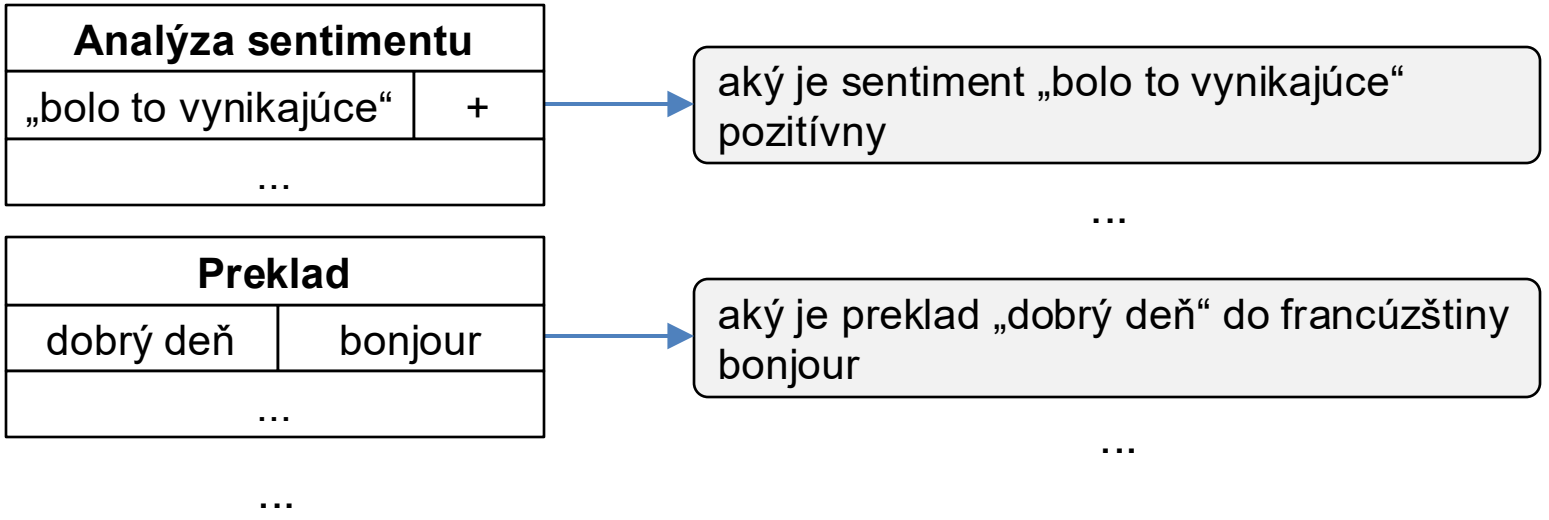
Veľkosť trénovacích dát



LifeArchitect.ai/models

Ladenie inštrukciami (1)

1. Využitie existujúcich označených dátových množín
   - Rôzne typy úloh (klasifikácia, preklad, zodpovedanie otázok, ...)
   - Použitím šablón sa prepíšu do tvaru [prompt - očakávaný výstup]
   - Učenie prebieha jednotne, predikovaním nasledujúceho tokenu



Ladenie inštrukciami (2)

2. Ručným zadaním požadovaného výstupu pre predchádzajúce prompty
   - InstructGPT/OpenAI – 13 000 promptov, 40 anotátorov

3. Využitie LLM na generovanie trénovacích dát
   - Anotátor zadá iba jeden prompt pre danú úlohu ako príklad, pred-trénovaný LLM vygeneruje množinu promptov a odpovedí, ktoré sa použijú na ladenie

Ladenie inštrukciami – príklady modelov

- Alpaca – 52K vygenerovaných inštrukcií na LLaMA modely, podobný výkon ako GPT-3, oveľa menší model (7B oproti 176B), ľahko reprodukovateľný (600$)
- Flan-PaLM – 1.8K inštrukcií, lepší než PaLM (v priemere o 9.4%), ale vyžaduje na trénovanie iba 0.2% času

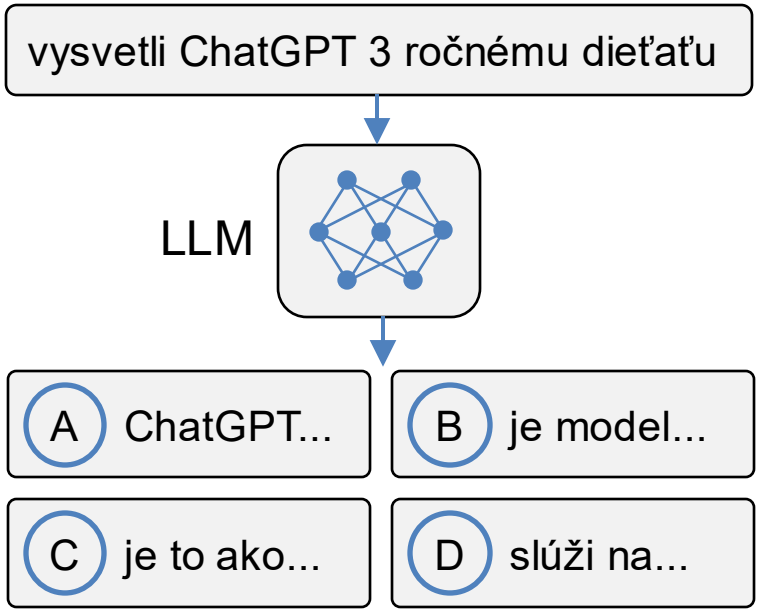
Učenie posilňovaním

- Snažíme sa zarovnať model (*alignment*) tak aby produkoval správny výstup
- Anotátor môže ohodnotiť celkový výstup, ktorý sa však generuje po tokenoch – ako propagovať chybu výstupu a určiť gradient pre jednotlivé tokeny?

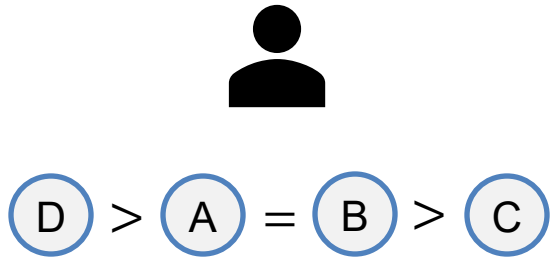
- Metódy učenia posilňovaním (*reinforcement learning*)
  - Algoritmus vygeneruje postupnosť akcií (pre LLM postupnosť tokenov)
  - Určí sa ohodnotenie celej postupnosti
  - Metóda učenia posilňovaním určí ako sa má zmeniť prediktívna funkcia

Učenie ohodnotenia (1)

1. Pre zvolené prompty sa nasampluje viacero odpovedí

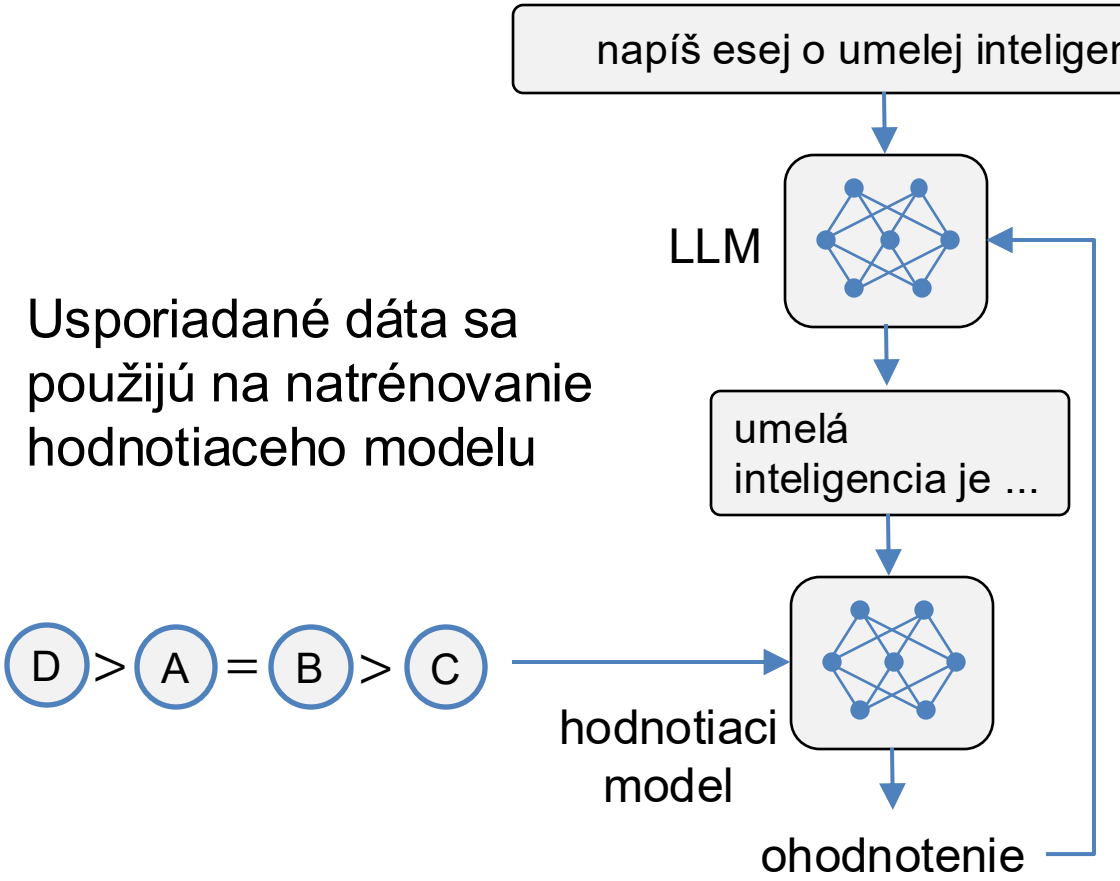


2. Anotátor určí ich poradie od najlepšej po najhoršiu



Učenie ohodnotenia (2)

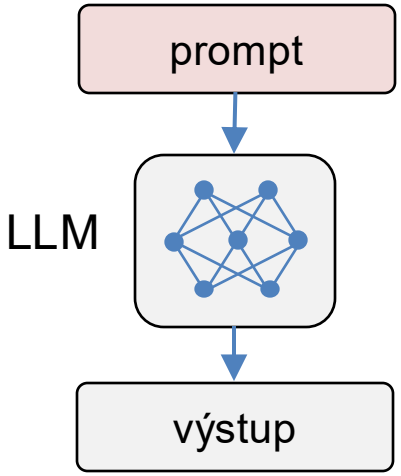
3. Usporiadané dáta sa použijú na natrénovanie hodnotiaceho modelu



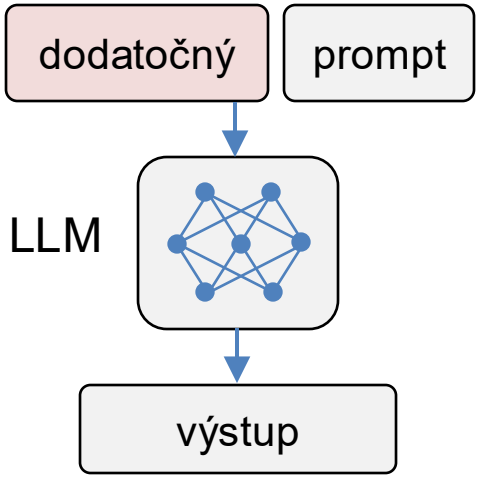
4. Hodnotiaci model vypočíta hodnotenie pre vygenerovaný výstup

Použitie veľkých jazykových modelov

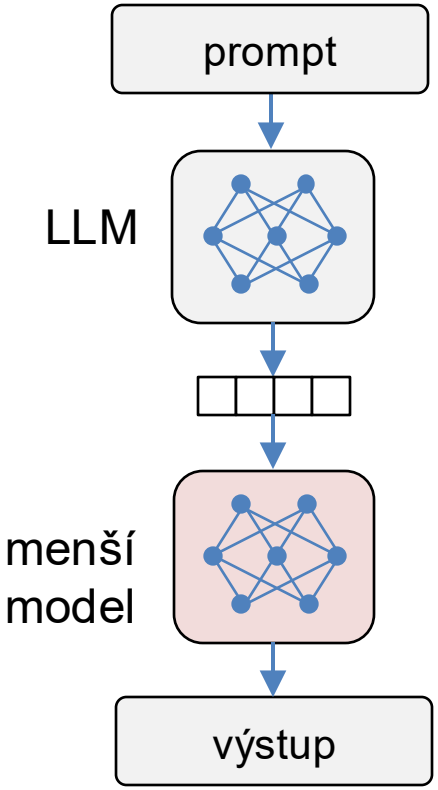
1 Vytváranie promptov



2 Ladenie promptov



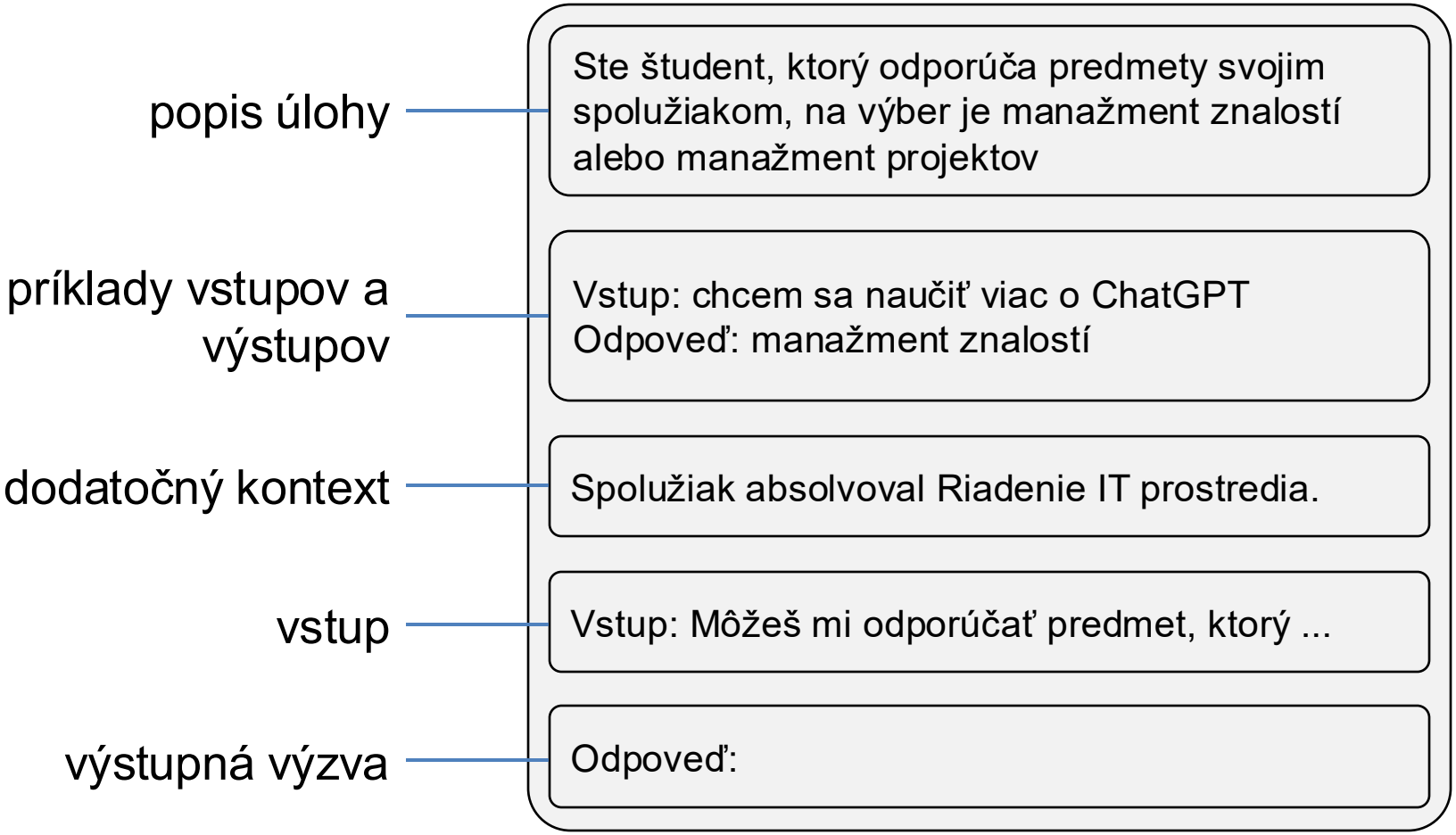
3 Využitie reprezentácie



Vytváranie/ladenie promptov

- *Few-shot* – uvedieme pár príkladov aj s odpoveďami a cieľovú otázku
- Generovaná znalosť – najprv sa opýtame všeobecnú otázku z danej domény a vygenerovanú odpoveď použijeme ako kontext pre špecifickú otázku
- Samplovanie – vygenerujeme viac odpovedí a použijeme majoritné hlasovanie pre získanie finálnej odpovede
- *Chain-of-thoughts* – v prompte uvedieme viac krokov ako zodpovedať otázku

Príklad promptu



Otvorené otázky a výskum jazykových modelov

- Kvalitné trénovacie dáta a menšie modely
- Dostupnejšie metódy pre prispôsobenie modelov
- Dlhší kontext

- Dôveryhodnosť
- Ochrana súkromia a bezpečnosť
- Ochrana autorských práv