KMI/UMIN Umělá inteligence

Deep Learning

9. Transformer

Jan Konečný

Katedra informatiky, PřF UP v Olomouci

15. listopadu 2026

Co nás dnes čeká

Proč paralelně

Tokenizace a embeddingy

Poziční kódování

Self-attention

Enkodér a dekodér

Cena a srovnání s RNN

Zahodit rekurenci

Mechanismus attention v modelech Seq2Seq původně doplňoval rekurentní síť. Vaswani a kol. ukázali, že sekvenční rekurenci lze mechanismem attention nahradit úplně.

Výsledná architektura byla navržena pro strojový překlad, ale stala se základem velkých jazykových modelů a rozšířila se také do zpracování obrazu a zvuku.

Dnes projdeme výpočet od tokenizace přes embeddingy a poziční kódování až k blokům enkodéru a dekodéru.

Skripta: kap. 13 *Transformer*

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin: Attention Is All You Need. NIPS (2017).

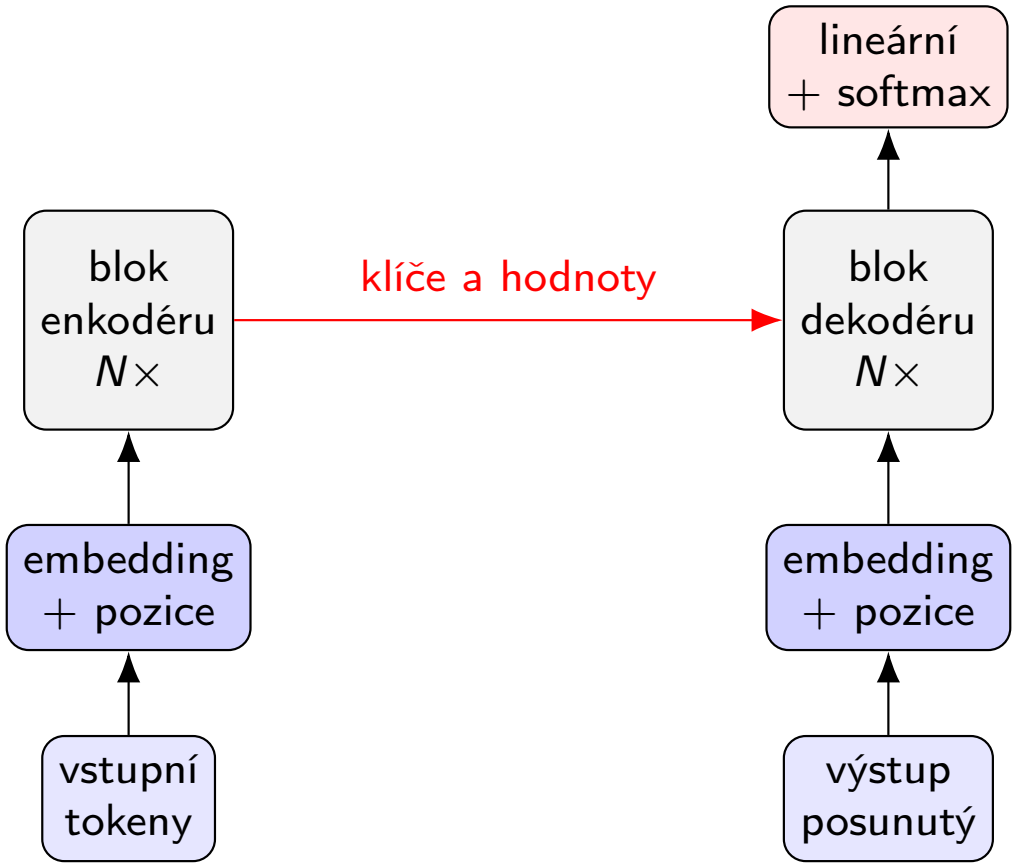
Dvě omezení rekurence

- **Nemožnost paralelizace.** Skrytý stav v čase t závisí na stavu v čase $t-1$, takže kroky nelze během trénování počítat současně. Moderní hardware je přitom stavěný na masivně paralelní výpočty.
- **Dlouhé závislosti.** Informace o vzdáleném tokenu se ke slovu dostává přes dlouhý řetěz mezikroků, po cestě se ředí a gradienty mizí. Attention tento problém zmírnila, ale rekurentní páteř zůstala.

Transformer zpracovává všechny tokeny *současně* a vztahy mezi nimi počítá jedinou sadou maticových operací. Tím získává:

- efektivní využití GPU,
- vzdálenost mezi libovolnými dvěma pozicemi $O(1)$ místo $O(n)$,
- výrazně rychlejší trénink.

Celkové schéma



Enkodér mapuje vstupní sekvenci na kontextualizované vektory, dekodér z nich generuje výstup. Oba se skládají z N identických bloků (v původním článku $N = 6$).

Tokenizace

Text se dělí na tokeny — elementární jednotky, s nimiž model pracuje.

- **slova**: slovník je obrovský a vždy se objeví slovo, které v něm chybí (překlepy, neologismy, cizí jména),
- **znaky**: problém neznámých slov mizí, ale sekvence jsou velmi dlouhé a tokeny nesou málo informace,
- **podslova**: praktický kompromis — běžná slova zůstanou jedním tokenem, vzácná se rozpadnou na části.

Nejznámější je Byte-Pair Encoding: začne se slovníkem znaků a opakovaně se přidává nejčastější dvojice sousedních tokenů, dokud slovník nedosáhne požadované velikosti.

Tokenizace ovlivňuje délku sekvence, cenu i schopnost pracovat s jazyky, kódem a čísly.

Stejný řetězec může mít v různých tokenizérech zcela jiný počet tokenů.

Demonstrace: BPE na českých tvarech

Korpus obsahuje tvary *les*, *lesa*, *lesu*, *lesy*; začínáme slovníkem znaků.

krok	nejčastější dvojice	stav
0	—	`l e s  l e s a  l e s u  l e s y`
1	`l e` (4×)	`le s  le s a  le s u  le s y`
2	`le s` (4×)	`les  les a  les u  les y`
3	žádná častější než 1×	konec

Algoritmus sám objevil kmen `les` a koncovky nechal jako samostatné tokeny. Nedělá přitom žádnou morfologickou analýzu — jen počítá četnosti sousedních dvojic.

U flektivních jazyků je to výhoda i omezení: kmen se sdílí, ale rozdělení nemusí odpovídat hranici morfémů, protože rozhoduje statistika korpusu.

Tokenizace není neutrální

Tentýž text stojí v různých jazycích různý počet tokenů, protože tokenizér byl naučen na korpusu s převahou angličtiny.

- **Čeština** se rozpadá na víc tokenů než angličtina téhož významu — delší sekvence, vyšší cena, kratší efektivní kontext.
- **Čísla** se dělí nekonzistentně; „1000“ a „1001“ mohou mít jinou tokenizaci, což ztěžuje aritmetiku.
- **Diakritika a emoji** se kódují po bajtech, takže jeden znak zabere několik tokenů.
- **Mezery** bývají součástí tokenu, takže „les“ na začátku věty a uprostřed jsou dva různé tokeny.

Důsledek pro praxi: délku kontextu i cenu je nutné měřit v tokenech daného modelu, ne ve znacích nebo slovech.

Embeddingy: sdílet, nebo ne

Embeddingová vrstva přiřazuje každému tokenu hustý vektor; na rozdíl od samostatně trénovaného Word2Vecu jsou tyto vektory parametry celého modelu.

Odlišné tabulky

Dávají smysl, když se vstupní a výstupní slovník liší (překlad). Cenou je vyšší počet parametrů.

Sdílené

Méně parametrů a konzistentní reprezentace tokenů na vstupu i výstupu. Modely rodiny GPT mají embeddingy sdílené.

Původní architektura používá $d_{\text{model}} = 512$.

Bez pozice nelze rozlišit slovosled

Attention zpracovává všechny dvojice stejnou operací, takže sama nerozlišuje pořadí tokenů. Bez poziční informace by model nerozlišil věty

„Tom bit a dog.“ versus „A dog bit Tom.“

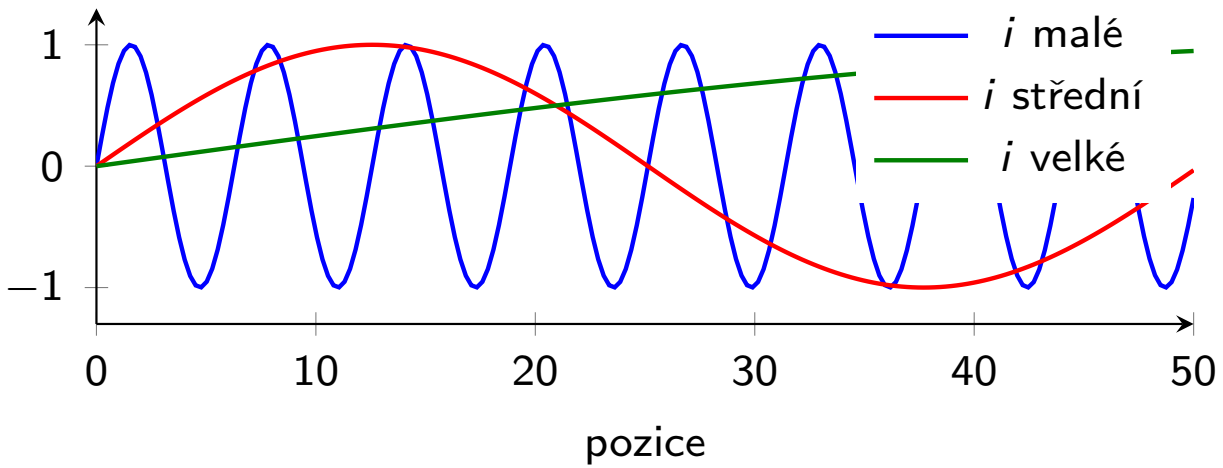
Řešením je poziční kódování: ke každému embeddingu se *přičte* vektor kódující pozici tokenu.

Proč se přičítá a nekonkatenuje

Konkatenace by prodloužila vektory i všechny navazující matice a přidala nový hyperparametr. Sčítání nic z toho nevyžaduje; embeddingová vrstva se navíc může naučit takové měřítko, aby obě složky zůstaly rozlišitelné.

Sinusové kódování

$$\mathrm{PE}_{pos,2i} = \sin\left(\frac{pos}{10000^{2i/d}}\right), \qquad \mathrm{PE}_{pos,2i+1} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$



První souřadnice kmitají rychle (rozlišují sousední pozice), poslední se mění velmi pomalu (rozlišují vzdálené oblasti). Vlnové délky tvoří geometrickou posloupnost od 2π do $10000 \cdot 2\pi$.

Relativní posun je lineární transformace

Ze součtových vzorců plyne

$$\begin{pmatrix} \sin(\omega_i(pos+k)) \\ \cos(\omega_i(pos+k)) \end{pmatrix} = \begin{pmatrix} \cos(\omega_i k) & \sin(\omega_i k) \\ -\sin(\omega_i k) & \cos(\omega_i k) \end{pmatrix} \begin{pmatrix} \sin(\omega_i\, pos) \\ \cos(\omega_i\, pos) \end{pmatrix}.$$

Matice vpravo je rotace o úhel $\omega_i k$ — závisí pouze na posunu k, nikoli na samotné pozici. Model tedy může vztah dvou pozic dekódovat z relativního posunu.

Autoři zkoušeli i *naučené* poziční embeddingy s téměř shodným výsledkem; sinusová verze má ale tu výhodu, že je definována pro libovolnou pozici, i delší, než jaké model viděl při trénování.

Novější poziční schémata

Sinusové kódování se přičítá ke vstupu. Novější modely místo toho zasahují přímo do výpočtu skóre:

- **RoPE** otáčí dvojice souřadnic Q a K o úhel úměrný pozici. Skalární součin pak závisí jen na *rozdílu* pozic, takže je relativní z konstrukce.
- **ALiBi** přičte ke skóre lineární penalizaci $-m\,|i-j|$ úměrnou vzdálenosti pozic. Nemá žádné parametry a model lze nasadit na delší kontext, než na jakém se učil.
- **Naučené absolutní embeddingy** (BERT, GPT-2) jsou nejjednodušší, ale mimo trénovanou délku nedefinované.

Rozdíl se projeví hlavně při extrapolaci: model naučený na 2k tokenech si s 8k poradí jen tehdy, když je poziční informace relativní.

J. Su a kol.: RoFormer: Enhanced Transformer with Rotary Position Embedding (2021). — O. Press, N. Smith, M. Lewis: Train Short, Test Long: Attention with Linear Biases. ICLR (2022).

Co má self-attention udělat

Vstupem je posloupnost vektorů (embedding + pozice). Úkolem je zvážit vztahy mezi všemi tokeny a vytvořit *kontextově závislé* reprezentace: z obecného vektoru tokenu „pes“ se má stát vektor tokenu „pes, který v této větě štěká na kočku“.

Výpočet ve čtyřech krocích:

1. tři naučené lineární projekce dají dotazy Q, klíče K a hodnoty V;
2. skóre: dotaz každého tokenu se porovná s klíči všech tokenů skalárním součinem;
3. softmax převede skóre na váhy;
4. hodnoty se zkombinují váženým součtem.

Na rozdíl od Seq2Seq, kde dekodér nahlížel do enkodéru, zde sekvence věnuje pozornost sama sobě — odtud předpona *self*.

Škálovaný skalární součin

Scaled dot-product attention

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

Prvek (i, j) matice $QK^\top$ je skóre relevance j-tého tokenu pro i-tý. Softmax vytvoří v každém řádku rozdělení vah a násobení maticí V spočítá příslušnou lineární kombinaci hodnot.

Proč se dělí $\sqrt{d_k}$

Pro velká d_k rostou skalární součiny co do velikosti (sčítá se d_k součinů), softmax se dostává do oblastí s extrémně malými gradienty — jedna hodnota se blíží 1 a ostatní 0 — a model se přestává učit. V základním Transformeru je $d_k = 64$.

Malý příklad

Příklad 1 (Attention nad dvěma hodnotami)

Jeden dotaz má po škálování skóre $(2, 1)$. Softmax vytvoří váhy přibližně (0,731, 0,269). Pro skalární hodnoty $(3, -1)$ je výstup

$$0{,}731 \cdot 3 + 0{,}269 \cdot (-1) \doteq 1{,}924.$$

Attention tedy nevybere jedinou pozici; vytvoří datově závislý vážený průměr. Změna dotazu změní skóre, váhy i výslednou reprezentaci.



váhy 0,731 a 0,269

Demonstrace: self-attention nad třemi tokeny

Pro názornost vezměme $d = 2$ a identické projekce, tedy $Q = K = V = X$:

$$X = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}, \qquad \frac{XX^\top}{\sqrt{2}} = \begin{pmatrix} 0{,}71 & 0 & 0{,}71 \\ 0 & 0{,}71 & 0{,}71 \\ 0{,}71 & 0{,}71 & 1{,}41 \end{pmatrix}.$$

token	váhy po softmaxu	výstup
1	(0,401; 0,198; 0,401)	(0,802; 0,599)
2	(0,198; 0,401; 0,401)	(0,599; 0,802)
3	(0,248; 0,248; 0,503)	(0,752; 0,752)

Třetí token je součtem prvních dvou, takže je oběma podobný a jeho váhy jsou symetrické. První token si ponechá nejvíc sám ze sebe, ale přibere i třetí, s nímž sdílí souřadnici.

Výstupem není výběr jednoho tokenu, ale nová reprezentace každé pozice poskládaná ze všech.

Multi-head attention

Token může mít víc než jednu funkci: zájmeno se váže k podstatnému jménu, sloveso ke svému podmětu a předmětu, slovo může být součástí ustáleného spojení. Jediný výpočet attention by musel všechny tyto vztahy namíchat do jediné sady vah.

Proto se počítá h paralelních attention (v základním modelu $h = 8$), každá ve vlastním podprostoru:

$$\mathrm{head}_i = \mathrm{Attention}(QW_i^Q,\, KW_i^K,\, VW_i^V),$$

$$\mathrm{MultiHead} = \left[\mathrm{head}_1; \ldots; \mathrm{head}_h\right] W^O.$$

Dimenze podprostorů se volí $d_k = d_{\mathrm{model}}/h$ (tedy $512/8 = 64$), takže celkový objem výpočtů zůstává srovnatelný s jednou hlavou plné dimenze.

Co jednotlivé hlavy dělají

Analýzy natrénovaných modelů nacházejí hlavy s rozpoznatelnou funkcí: pozičně zaměřené (dívají se na předchozí token), syntaktické (sledují závislost mezi slovesem a podmětem), nebo takové, které stahují pozornost na oddělovač a fakticky se „vypínají“.

Většina hlav není nezbytná

Značnou část hlav lze po natrénování odstranit s malým poklesem kvality; u některých vrstev stačí jediná. Redundance je tedy vlastností naučeného modelu, ne jen nadbytkem kapacity.

Z toho plyne dvojí ponaučení. Více hlav pomáhá při *učení* — dávají optimalizaci víc cest. Pro *nasazení* jich ale bývá potřeba méně, čehož se využívá při prořezávání a destilaci.

P. Michel, O. Levy, G. Neubig: Are Sixteen Heads Really Better than One? NeurIPS (2019). — E. Voita a kol.: Analyzing Multi-Head Self-Attention. ACL (2019).

Blok enkodéru

Dvě podvrstvy:

1. multi-head self-attention (vztahy mezi tokeny),
2. pozičně aplikovaná dopředná síť (zpracuje každý token nezávisle).

Každou doplňuje reziduální spojení a layer normalization.

Dopředná síť

$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$ s poměrem $1 : 4$ — matice 512×2048 dimenzi zvýší a 2048×512 ji sníží zpět. Váhy jsou stejné pro všechny pozice v rámci vrstvy, ale liší se mezi vrstvami.

Reziduální spojení znamená, že blok vstupní vektory nenahrazuje, ale *obohacuje* o informace získané z attention a FFN.

Kolik parametrů má blok

Pro $d = d_{\text{model}}$ a FFN s poměrem 1:4:

část	vzorec	pro $d = 512$
W^Q, W^K, W^V, W^O	$4d^2$	1 048 576
FFN ($d \to 4d \to d$)	$8d^2$	2 097 152
blok enkodéru	$12d^2$	3 145 728
blok dekodéru (+ cross)	$16d^2$	4 194 304

Šest bloků na každé straně dá $6 \cdot (12 + 16)d^2 \doteq 44$ M parametrů; se sdílenou embeddingovou tabulkou pro slovník 37 000 tokenů vyjde asi 63 M — což odpovídá 65 M uváděným v původním článku.

Všimněte si, že v tomto počtu nikde nevystupuje délka sekvence. Ta ovlivňuje paměť a čas, nikoli velikost modelu.

Proč layer normalization

BatchNorm používá statistiky mini-batche, což je u sekvencí proměnlivé délky nepohodlné a u malých dávek nespolehlivé.

LayerNorm normalizuje napříč souřadnicemi *jednoho příkladu*:

$$\widehat{x} = \frac{x - \mu_L}{\sqrt{\sigma_L^2 + \varepsilon}}, \qquad y = \gamma \widehat{x} + \beta.$$

Je tedy nezávislá na velikosti dávky a snižuje citlivost na inicializaci.

Původní model navíc používá dropout 0,1 na výstupech podvrstev i na součtech embeddingů s pozičním kódováním a label smoothing 0,1, který zhoršuje perplexitu, ale zlepšuje BLEU.

Kam patří normalizace

Post-LN (původní)

$x \leftarrow \mathrm{LN}(x + \mathrm{Sub}(x))$. Normalizace leží *na* reziduální cestě, takže se signál v každém bloku přeškáluje. Bez warmupu se hluboký model rozejde.

Pre-LN (dnes běžné)

$x \leftarrow x + \mathrm{Sub}(\mathrm{LN}(x))$. Reziduální cesta zůstává nedotčená, gradient jí projde beze změny měřítka. Trénink je stabilní i bez warmupu.

Jde o tutéž myšlenku jako u pre-activation reziduálních bloků ze sedmé přednášky: nechat zkratku volnou a všechny úpravy provádět jen ve větvi.

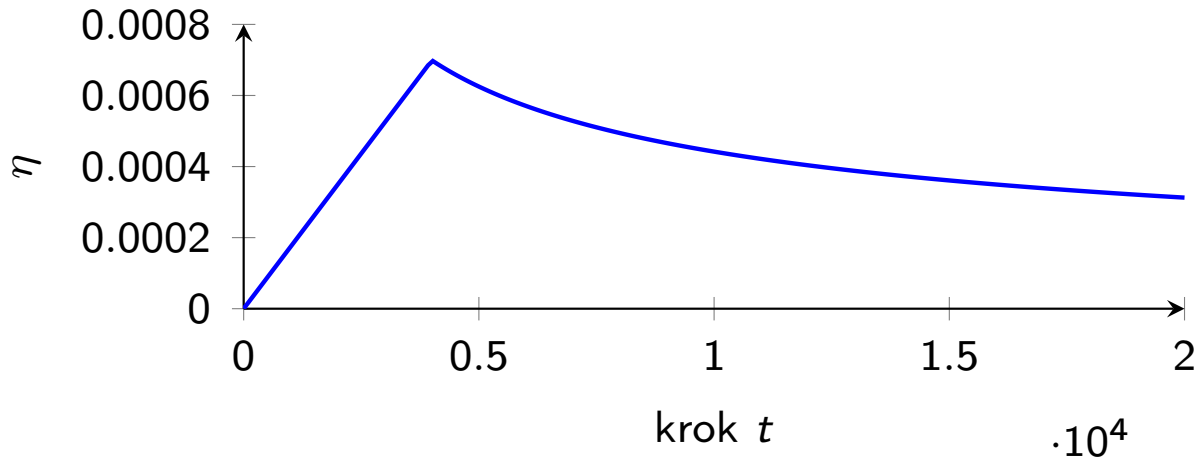
Cenou bývá o něco horší konečná kvalita při stejném rozpočtu; proto se u menších modelů někdy Post-LN stále používá.

R. Xiong a kol.: On Layer Normalization in the Transformer Architecture. ICML (2020).

Warmup a plán učicího kroku

Původní článek používá plán

$$\eta = d_{\text{model}}^{-0,5} \cdot \min\left(t^{-0,5},\ t \cdot t_{\text{warmup}}^{-1,5}\right), \qquad t_{\text{warmup}} = 4000.$$



Krok nejprve lineárně roste, po 4000 krocích dosáhne maxima $\doteq 7 \cdot 10^{-4}$ a pak klesá jako $t^{-1/2}$. Rostoucí fáze chrání model v době, kdy jsou váhy náhodné a gradienty velké — právě tam je Post-LN nejzranitelnější.

Blok dekodéru

Dekodér generuje výstup token po tokenu, autoregresivně:

$$P(\text{token}_t \mid \text{token}_1, \ldots, \text{token}_{t-1}).$$

Vstupem je jeho dosavadní výstup *posunutý doprava*.

Blok má oproti enkodéru jednu podvrstvu navíc:

1. **masked multi-head attention** — self-attention nad dosud vygenerovanými tokeny,
2. **cross-attention** — propojení s výstupem enkodéru,
3. **dopředná síť** — stejná jako v enkodéru.

U cross-attention pocházejí dotazy Q z dekodéru, zatímco klíče K a hodnoty V z enkodéru. Každý generovaný token se tak může „zeptat“ vstupní sekvence, které její části jsou pro něj relevantní.

Maskování

Při trénování je celá cílová sekvence známá, takže dekodér může predikovat všechny pozice paralelně. Nemaskovaná self-attention by ovšem zpřístupnila i token, který má daná pozice teprve předpovědět.

Maska se přičte ke skóre před softmaxem; pro sekvenci délky 4 má tvar

$$M = \begin{pmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

takže i-tý token vidí pouze tokeny $1, \ldots, i$.

Vedle kauzální masky se používá padding mask, která ignoruje výplňové pozice u různě dlouhých sekvencí v jedné dávce.

Generování a KV cache

Při generování přibývá v každém kroku jeden token. Naivně bychom celou sekvenci počítali znovu, tedy $O(n^2)$ práce na každý nový token.

Klíče a hodnoty už spočítaných pozic se však nemění — kauzální maska zaručuje, že nový token starší pozice neovlivní. Stačí je proto uložit (KV cache) a pro nový token spočítat jen jeho vlastní Q, K, V.

Příklad 2 (Co cache stojí)

Pro N bloků, n tokenů a dimenzi d zabere cache $2Nnd$ čísel. Pro $N = 32$, $d = 4096$ a $n = 2048$ je to $5{,}4 \cdot 10^8$ hodnot, tedy přes gigabajt i v šestnáctibitové přesnosti — a to na jedinou konverzaci.

Odtud plyne, proč je délka kontextu při nasazení dražší než při trénování a proč se hledají varianty se sdílenými klíči mezi hlavami.

Enkodér, nebo dekodér?

Pozdější modely si z Transformeru často berou jen polovinu:

- **BERT** používá pouze bloky enkodéru a hodí se pro porozumění textu (klasifikace, rozpoznávání entit, analýza sentimentu).
- **GPT** používá pouze bloky dekodéru bez cross-attention a je zaměřen na generování; autoregresivní predikce dalšího tokenu s kauzálním maskováním je princip, na němž stojí dnešní velké jazykové modely.
- **Encoder–decoder** zůstává přirozený pro překlad a sumarizaci.

Volba tedy není otázkou módy, ale toho, zda potřebujeme obousměrný kontext, nebo generovat.

Kvadratická cena

Matice skóre má rozměr $n \times n$ a každý prvek je skalárním součinem vektorů dimenze d, tedy

$$O(n^2 d).$$

Rekurentní vrstva potřebuje $O(nd^2)$ operací, ale nikdy nedrží matici $n \times n$ v paměti.

Příklad 3 (Cena zdvojnásobení kontextu)

Matice pro $n = 1\,024$ obsahuje přibližně 1,05 milionu prvků, pro $n = 2\,048$ už 4,19 milionu. Zdvojnásobení délky zečtyřnásobí tuto část paměti i výpočtu, i když počet parametrů modelu zůstane stejný.

Pro velmi dlouhé sekvence se proto používá redukce délky, řídká aproximace attention nebo hierarchické modely.

Jak se kvadratické ceny zbavit

- **Řídké vzory.** Longformer a BigBird nechají každý token vidět jen lokální okno plus několik globálních pozic. Složitost klesne na $O(n)$, ale vzor spojení je zvolen předem, ne naučen.
- **Nízkohodnostní aproximace.** Linformer promítne K a V na kratší délku, Performer nahradí softmax jádrovou aproximací. Výsledek je přibližný.
- **Přesné, ale úsporné provedení.** FlashAttention počítá totéž, jen nikdy nedrží celou matici skóre v paměti; postupuje po blocích a šetří přenosy. Výsledek je numericky týž.

Rozlišujte dvě různé věci

Aproximace mění model a její dopad je nutné změřit. Úsporné provedení mění jen implementaci a ověřuje se rovností výstupů.

I. Beltagy a kol.: Longformer (2020). — T. Dao a kol.: FlashAttention. NeurIPS (2022).

RNN versus Transformer

	RNN / LSTM	Transformer
paměť v délce	lineární	kvadratická
výpočet	sekvenční (n kroků)	plně paralelní
vzdálenost dvou pozic	$O(n)$ kroků	$O(1)$
dlouhé závislosti	obtížné	přímý přístup
vhodné pro	proudová data	dávkové trénování

Kvadratickou cenu Transformer vyvažuje právě tam, kde RNN historicky selhávaly. Paralelizovatelný trénink a přímé modelování vztahů mezi tokeny navíc umožnily dobré škálování — na této architektuře staví velké jazykové modely z jedenácté přednášky.

Trénovací recept původního modelu

optimalizátor	Adam, $\beta_1 = 0{,}9$, $\beta_2 = 0{,}98$, $\varepsilon = 10^{-9}$
plán kroku	warmup 4000, pak $t^{-1/2}$
regularizace	dropout 0,1, label smoothing 0,1
dávky	podle počtu tokenů, ne vět (asi 25 000 tokenů)
hardware	8 GPU; základní model 12 hodin, velký 3,5 dne

Nezvyklé $\beta_2 = 0{,}98$ (místo obvyklých 0,999) zkracuje paměť druhého momentu a pomáhá stabilitě při velkých gradientech na začátku učení.

Dávkování podle počtu tokenů je také podstatné: dávka se stejným počtem vět by měla velmi proměnlivou výpočetní náročnost i velikost gradientu.

A. Vaswani a kol.: Attention Is All You Need. NIPS (2017).

Co Transformer nemá zabudováno

Architektura je záměrně chudá na předpoklady, a to má obě strany:

- Bez pozičního kódování je *ekvivariantní vůči permutaci* — pořadí tokenů pro ni neexistuje.
- Nemá pojem lokality; že spolu sousední tokeny souvisejí, se musí naučit z dat.
- Nemá omezenou paměť ani stav, takže neumí zpracovat proud neomezené délky; kontext je vždy konečné okno.

Chybějící preference znamená, že model potřebuje víc dat, ale zároveň není omezen předpokladem, který nemusí platit. Stejná úvaha se vrátí v desáté přednášce u ViT proti CNN. Vysvětluje také, proč Transformer uspěl právě tam, kde jsou data k dispozici ve velkém.