Transformer configs

causal

↳ "decoder" ⇒ lower triangular attn mask

↳ "encoder" ⇒ unmasked

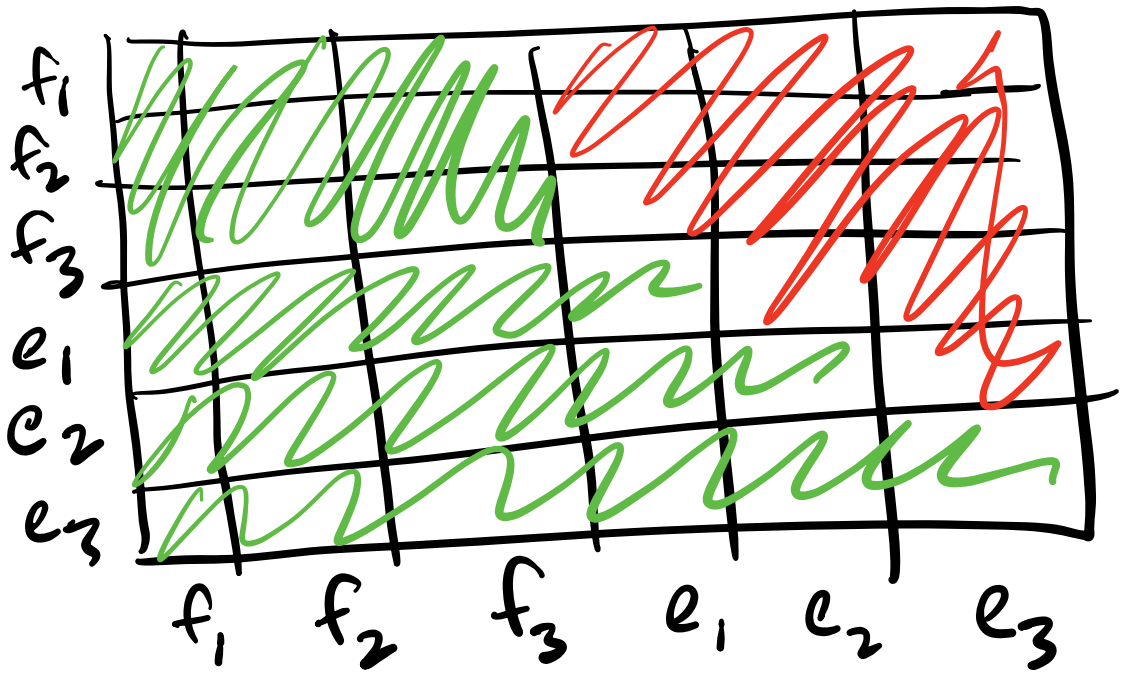
↳ encoder-decoder

↳ connected w/ "cross-attn"

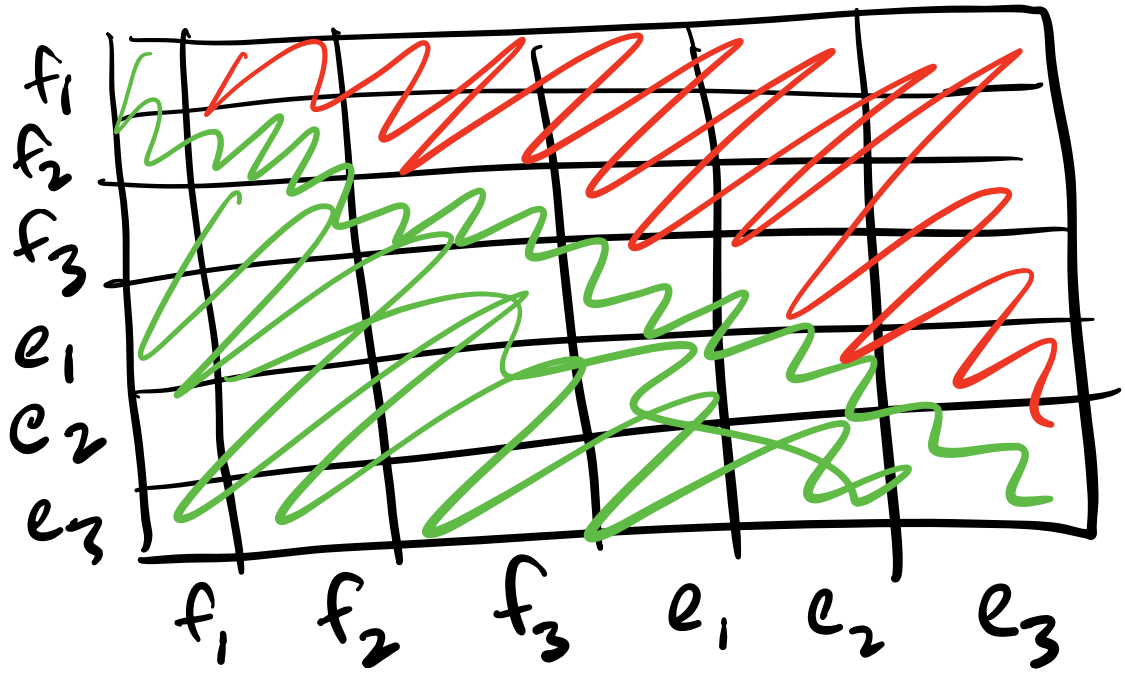
Prefix LM mask

↳ unify encoder-decoder models into one Transformer

↳ consider input/output as one seq



causal mask:



prefix LM mask

↳ bidirectional attn for input

↳ causal attn for output

↳ loss mask: response only

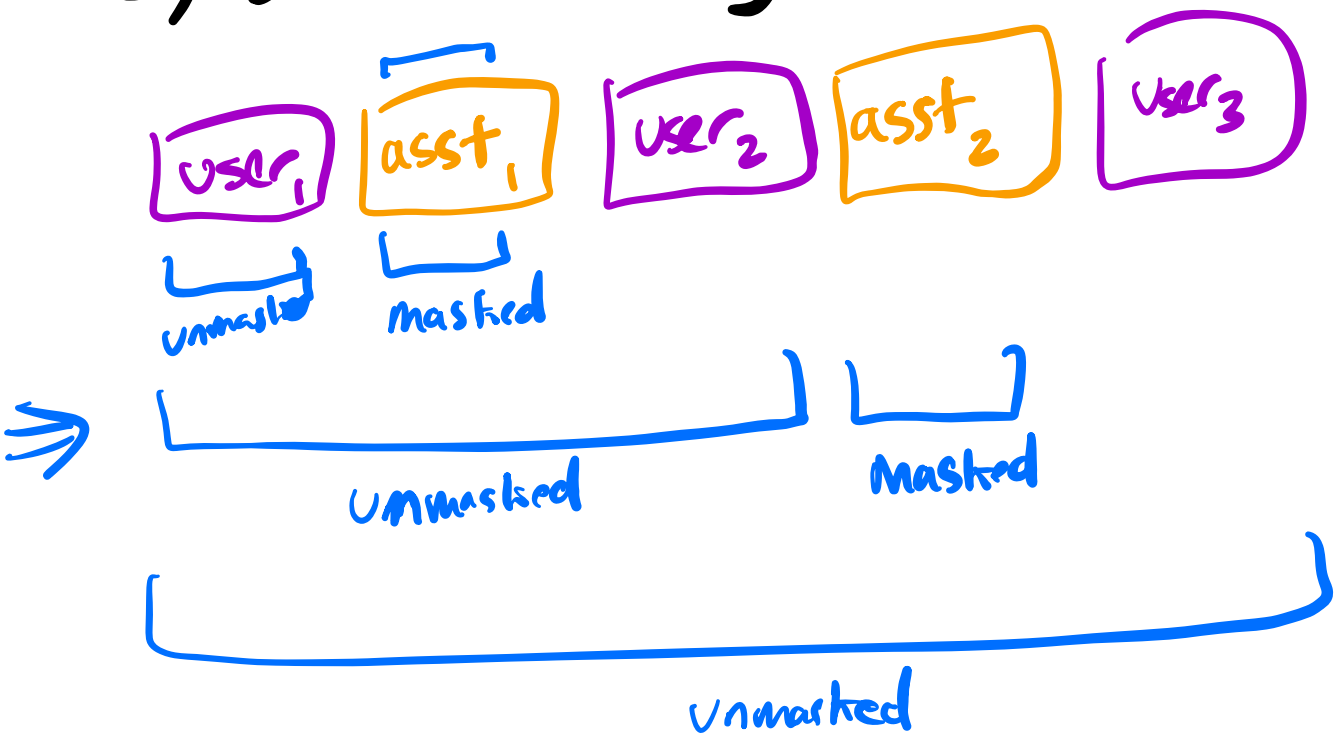
why don't modern LLMs use prefix LM?

↳ during pretraining, we always use the causal mask

↳ distribution shift w/ switching from causal to prefix LM mask

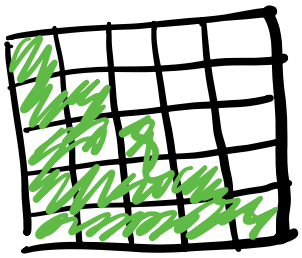
after pretraining >> gains from bidirectional attn

↳ breaks caching

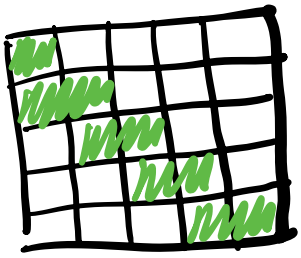


---

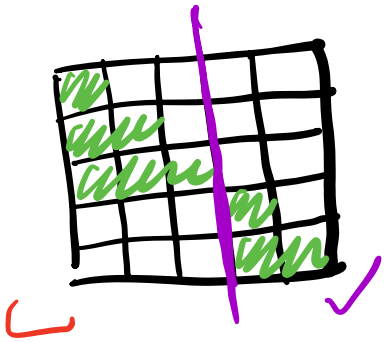
attention masks



⇒ causal mask, standard for pretraining/finetuning

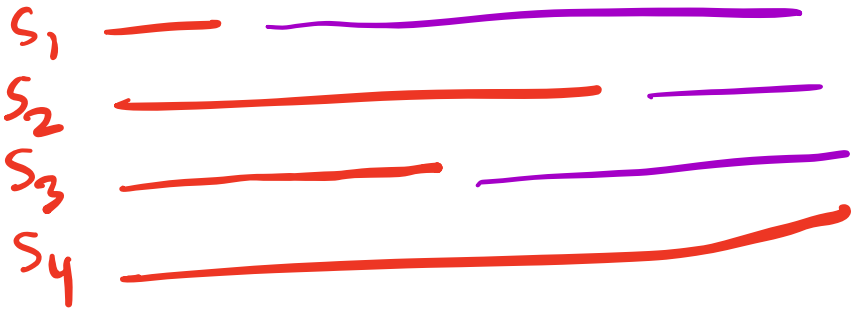


⇒ sliding window attn/ local attn



⇒ block mask / document packing mask

used when multiple documents concat into one seq, during pretraining

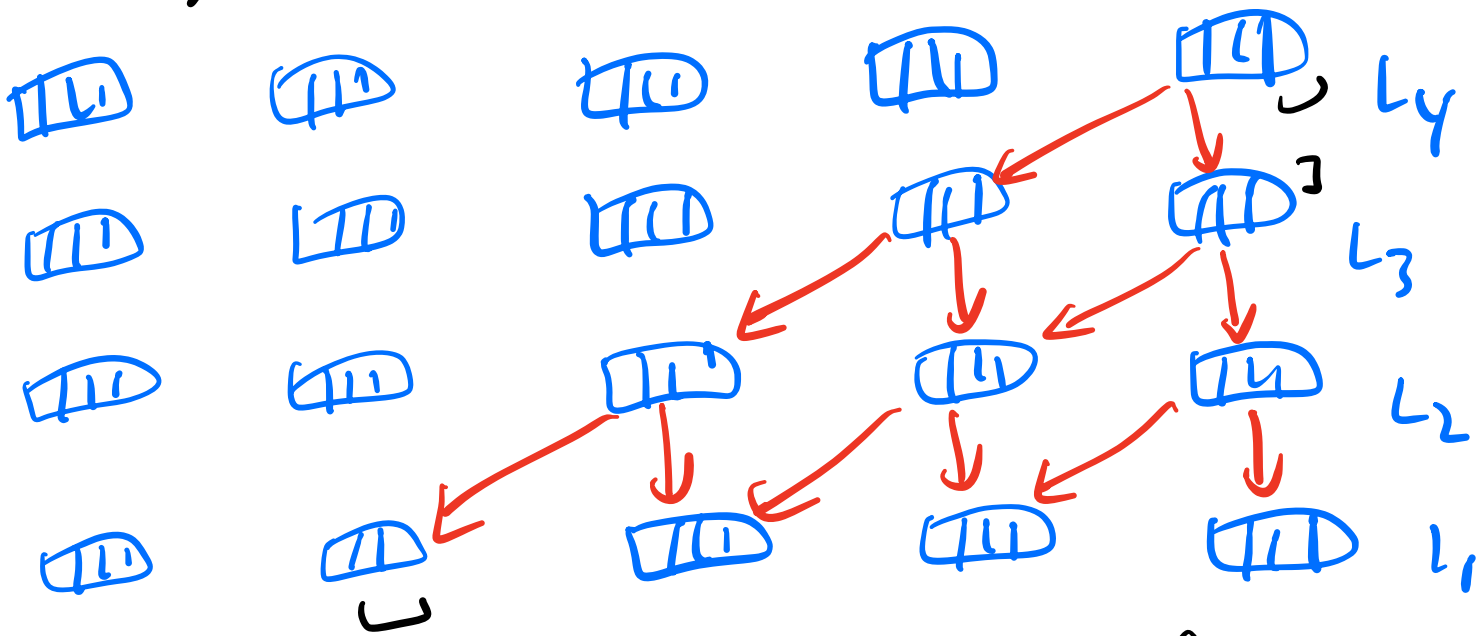


---

local attn:

↳ window size grows w/ depth

↳ each token sees only the prev. W tokens, W=2



↳ each layer can see $W-1$ more tokens than the prev layer

↳ after L layers, window size is $L \cdot W$

↳ effective window size does <u>not</u> grow linerly w/ block mask, but resets at document boundaries

↳ no overlap btwn blocks

---

tokenization

input: "My favorite LLM is Claude"

↳ My , favorite, LLM, is, Claude

172 63001 6791 5 2351

what if "LLM" is not in vocab

↳ My favorite <UNK> is Claude

What do we want in a tokenizer?

↳ single method for all languages, shared vocab across languages

↳ minimize UNK tokens

↳ minimize seq length after tokenization

↳ algo #1: split on whitespace + punctuation

↳ Mr. O'Neill thinks LLMs aren't evil

↑ ↑

O' + Neill aren' t

↳ Mr. O'Neill is waiting for GPT8.

↑

<UNK>

↳ algo #2: split on characters

↳ small vocab

↳ no <UNK>

↳ very long sequences!

↳ want a middleground between algos #1 + #2

↳ idea: represent frequent words with their own word types, rare words get broken up into smaller pieces

byte pair encoding

example dataset:

hug ×10
pug ×5
pun ×12
bun ×4
hugs ×5

Step 1: init. vocab w/ characters

V_0: {h, u, g, p, n, b, s}

Step 2: tokenize our dataset w/ V_0

h u g ×10
p u g ×5
p u n ×12
b u n ×4
h u g s ×5

Step 3: count up freq. of adjacent pairs

h u x15
u g x20
p u x17
u n x16
b u x4
g s x5

Step 4: merge the most freq. pair into new type + add it to V

V: {h, u, g, p, n, b, s, ug}

repeat steps 2-4 for a fixed # of iterations

h+ug x10
p+ug x5
p u n x12
b u n x4
h+ug+s x5

h+ug : 15