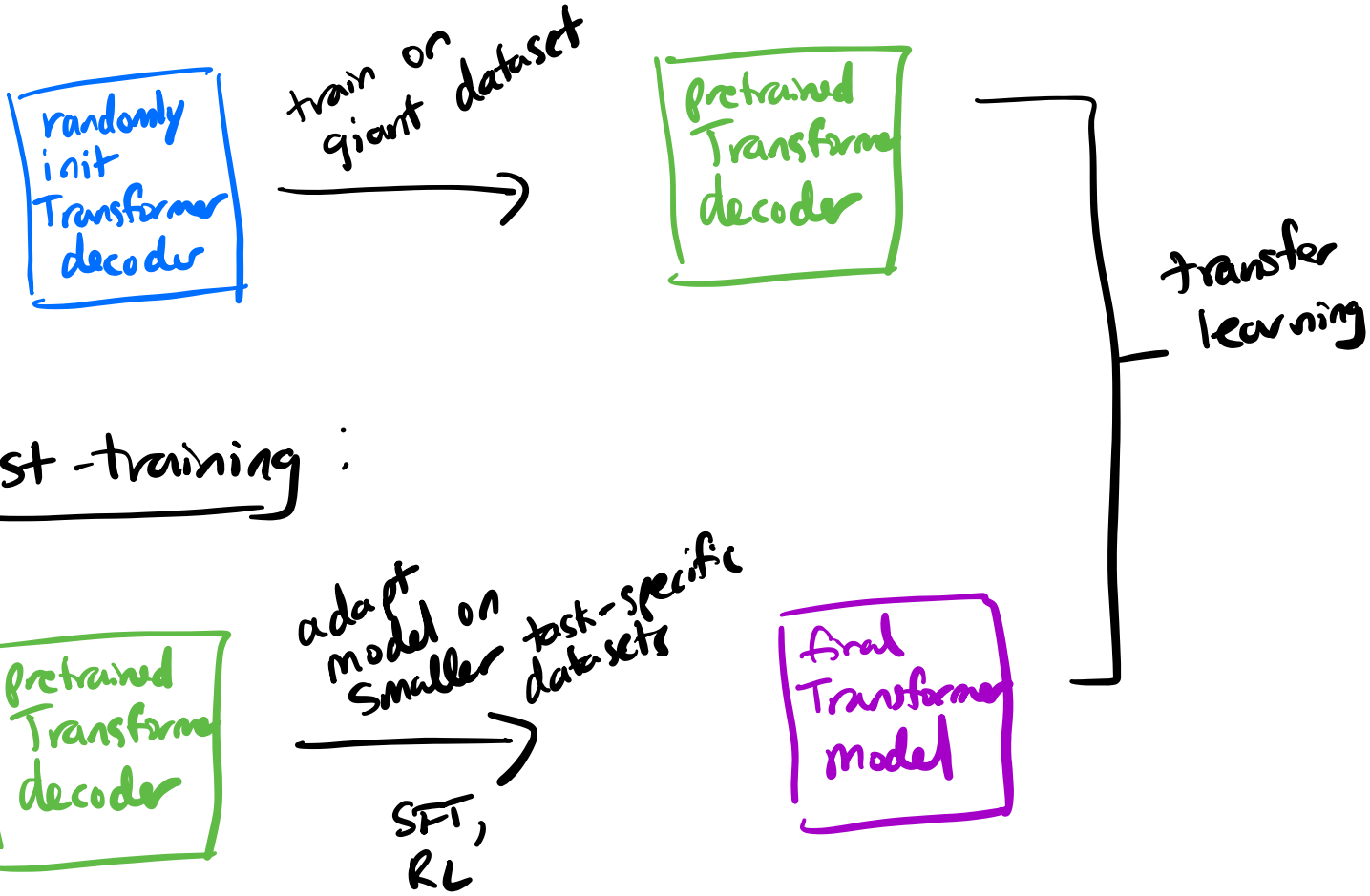
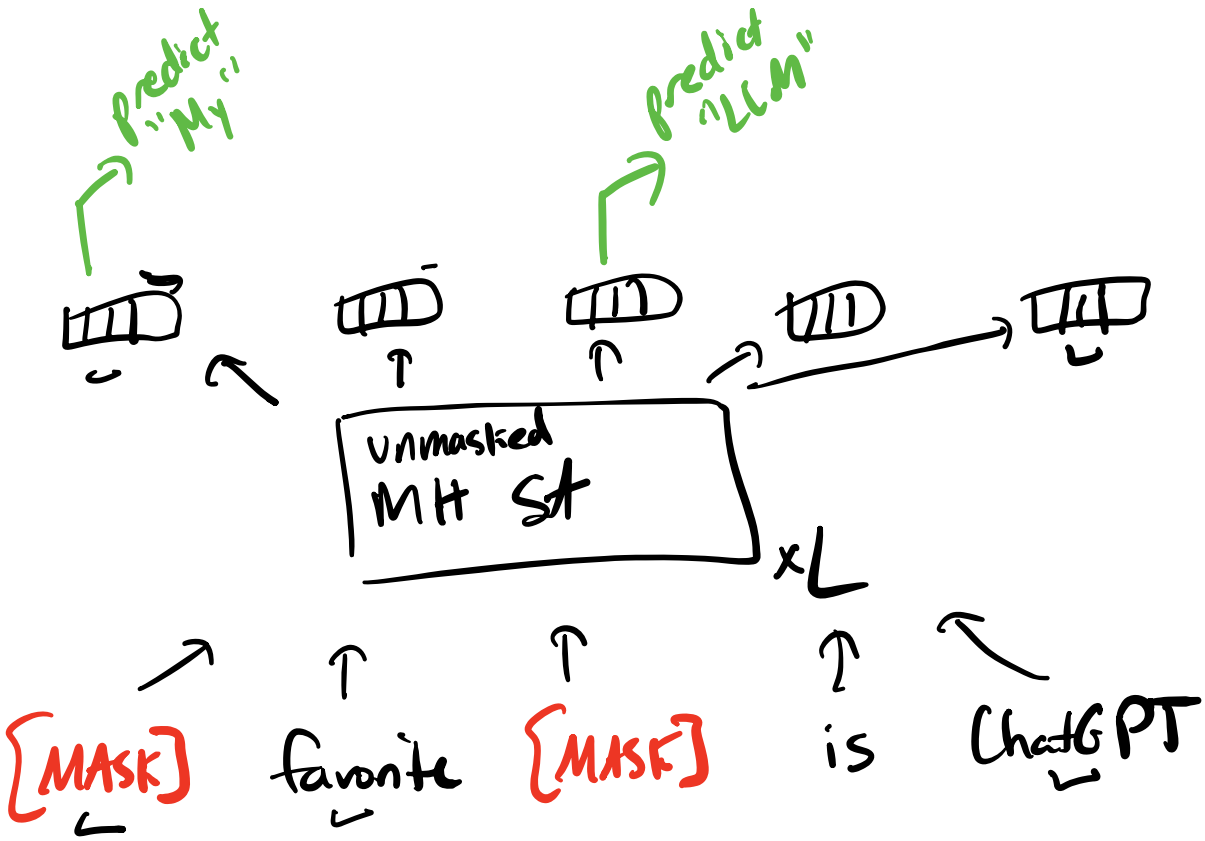
Pretraining:



Post-training:

BERT

- ↳ encoder Transformer
- ↳ pretraining objective: masked LM



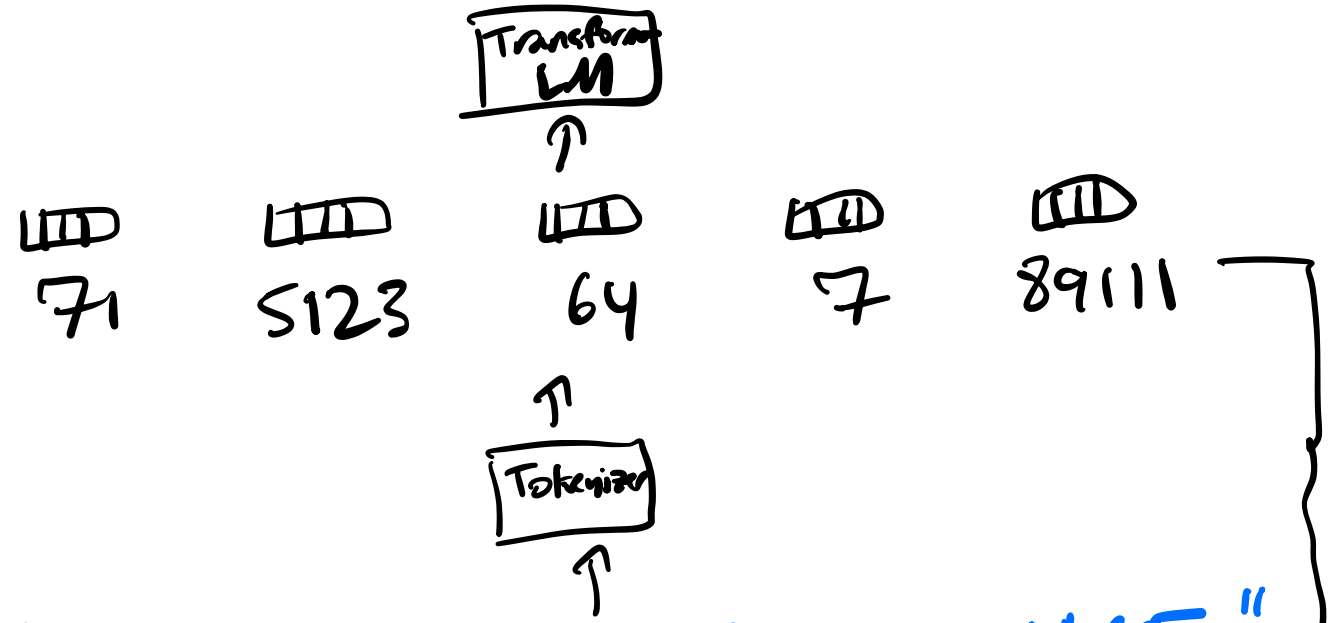
↳ masking rate 15-30% of tokens

hyperparam

↳ not good for generation

Tokenization:

↳ mapping raw strings into sequences of discrete tokens drawn from a vocab of types



↳ easy: split on whitespace / punctuation

Mr. O'Neill thinks LLMs aren't useful.

O'Neill → O' + Neill

aren't → aren' t / are+n't

↳ requires lot of specialized rules to deal w/ edge cases

↳ Spacy

↳ doesn't work on Thai, Chinese

↳ Unknown words

Unknown words:

↳ at test-time, I see a new word

My favorite LLM is GPT6

↳ not in vocab

My favorite LLM is <UNK> → special token

↳ information loss

↳ need to train w/ <UNK> to get it to understand how to deal w/ it

alternative tokenizers

↳ open
opens
opened
opening
openings

} all different types in vocab

↳ open
+s
+ed
+ing

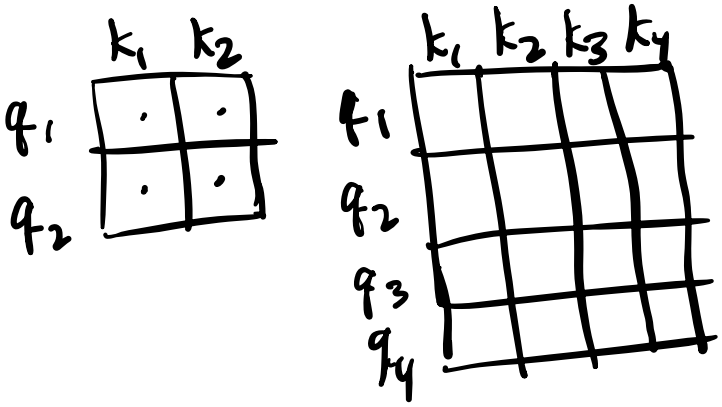
} stem, suffixes ⇒ morphological analyzer

↳

↳ o+pte+n+i+n+g } character-level splitting ⇒ small vocab, very small # of UNKs

↳ very long sequences

↳ complexity of self-attn is quadratic in seq. length



Subword tokenization:

↳ balance char / word tokenization

↳ byte pair encoding (BPE)

words	count
hug	10
pug	5
pun	12
bun	4
hugs	5

Step 1:
init vocab of just chars

$V_0 = \{h, u, g, p, n, b, s\}$

Step 2: tokenize w/ current vocab

Words	
h+u+g	10
p+u+g	5
p+u+n	12
b+u+n	4
h+u+g+s	5

Step 3: count each token pair in dataset

pairs	count
h+u	15
u+g	20
p+u	17
⋮	

Step 4: choose most common pair and add it to vocab, retokenize

$V_1 = \{h, u, g, p, n, b, s, ug\}$

words
h+ug
p+ug
p+u+n
b+u+n
h+ug+s

repeat process $\vec{S}$ times

↳ 32k, 64k

$V_2 = \{h, u, g, p, n, b, s, ug, pu\}$

$V_3 = \{h, u, g, p, n, b, s, ug, pu, hug\}$

⋮

↳ bytes vs. characters

↳ 160K unicode chars

↳ why do we need a static tokenization?

↳ more recent: dynamic tokenization