Transformers: Advanced Topics

Lecture Notes Consolidated with Claude AI — CS6140 Machine Learning

---

*Optional. Assumes the **TRANSFORMER** note (blocks, the three attentions, GPT, pre-norm, sampling) and the **ATTENTION** note. Nothing here is needed for the required HW7 problems. It backs HW7's optional problems 8–11 and the optional questions about dev mode, and it is a bridge to a course on large language models.*

1 BERT in Detail: an Encoder Trained to Fill In Blanks

BERT (Devlin et al., 2019) is the encoder stack alone: no causal mask, so every token sees the whole text. It cannot be trained to predict the next token (it would see it), so it is trained to **fill in hidden tokens**: masked language modelling (MLM).

- **Input.** Token embedding + a *segment* embedding ("sentence A" or "B") + a learned position embedding. A special `[CLS]` token starts the input; `[SEP]` separates sentences.
- **MLM.** 15% of the tokens are chosen; of those, 80% are replaced by `[MASK]`, 10% by a random token, 10% left unchanged. The model predicts the original token at the chosen positions only. Why not always `[MASK]`? Because `[MASK]` never appears when the model is used later; the 10%/10% keep the model from relying on it.
- **Next-sentence prediction (NSP).** Is sentence B the real continuation of A (50%) or a random sentence? Later work (RoBERTa) found NSP unnecessary and dropped it.
- **Sizes.** Base: 12 blocks, $d = 768$, 12 heads, 110M parameters. Large: 24 blocks, $d = 1024$, 16 heads, 340M.
- **Fine-tuning** adds a small head per task: a classifier on the final `[CLS]` vector (is this review positive?), a per-token classifier (named entities), or two vectors s, e that score each position as the start or end of an answer span, $\text{softmax}_i(s \cdot h_i)$.

Worked example: one MLM training sentence

input: `[CLS] the cat [MASK] on the mat [SEP]` target at the blank: `sat`

The model uses *both* "the cat" and "on the mat" to fill the blank, which a left-to-right model cannot do. That is why BERT's vectors were the best features for understanding tasks around 2019: each token's vector is built from its whole context.

Key point: The price of seeing both directions: BERT cannot generate text left to right, and it learns from only 15% of its tokens per pass. GPT learns from every token. That is reason 2 of the basic note's "why decoder-only won".

2 T5 and BART: Encoder–Decoders Trained to Repair Text

T5 (Raffel et al., 2020) keeps both stacks and casts *every* task as text to text: "translate English to German: …" → German; "summarize: …" → summary; even classification outputs the label as a word. Its pretraining is **span corruption**: random spans of the input are replaced by sentinel tokens, and the decoder writes out the missing spans:

input: `Thank you <X> me to your party <Y> week.`
target: `<X> for inviting <Y> last <Z>`

BART (Lewis et al., 2020) is similar: corrupt the text (mask, delete, shuffle sentences), and train the decoder to write the original. Both are strong for translation and summarization, where the input and output are clearly separate texts.

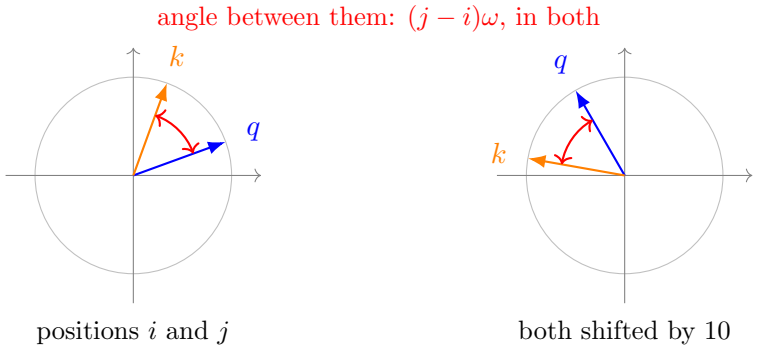
3 Positions Beyond Sin/Cos

The ATTENTION note's sinusoidal encoding is *added* once at the bottom. Three later ideas:

- **Learned absolute positions** (BERT, GPT-2, your GPT): one trainable vector per position. Simple, but there is nothing for positions beyond the training context.
- **Rotary position embedding, RoPE** (Su et al., 2021; LLaMA and most current models). Take the board's "position $i+\delta$ = a rotation of position i" (Lecture 8/8 p8) and apply it directly to the queries and keys: split q and k into pairs of coordinates and rotate the pair j of the token at position i by the angle $i\,\omega_j$. A rotation $R(\alpha)$ satisfies $R(\alpha)^\top R(\beta) = R(\beta-\alpha)$, so

  $$\big(R(i\omega)q\big)^\top \big(R(j\omega)k\big) = q^\top R\big((j-i)\omega\big)k,$$

  which depends only on the *offset* $j-i$. Relative position enters every attention score, in every layer, with no extra parameters.
- **ALiBi** (Press et al., 2022): no position vectors at all; subtract $m\cdot|i-j|$ from each score before the softmax (a different slope m per head), so attention prefers nearby tokens.

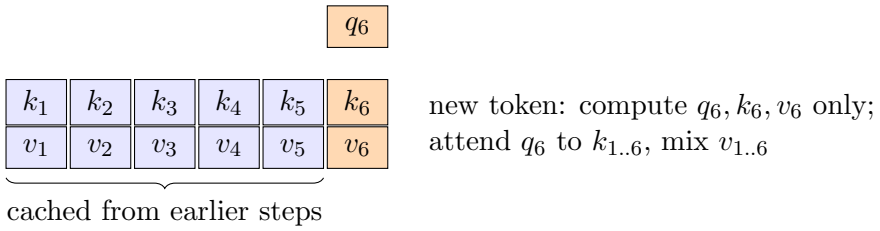


Shift both tokens by 10 positions: both vectors turn by the same extra angle, the angle *between* them is unchanged, and so is their dot product.

Worked example: RoPE sees only the offset With one random 2-D query and key and $\omega = 0.3$ (build script), the rotated score is 0.110237 for positions $(5,3)$, $(12,10)$ and $(40,38)$: three different absolute positions, one offset, one score.

4 What Generation Costs: KV Cache, MQA, GQA, FlashAttention

HW7's decoder recomputes every earlier position at every generated word. But with the causal mask, the keys and values of earlier positions never change once computed. The **KV cache** stores them per layer, so each new token computes only its own q, k, v and one row of attention against the cache: the work per new token drops from "the whole prefix" to "one position".



The cache is memory: $2 \cdot L \cdot T \cdot d$ numbers per sequence (keys and values, every layer). For GPT-2 small ($L = 12$, $d = 768$) at $T = 1{,}024$ that is 18.9M numbers, 36 MiB in 16-bit, for *one* conversation. Serving many users at long contexts, memory runs out before compute does. Two fixes share keys and values across heads:

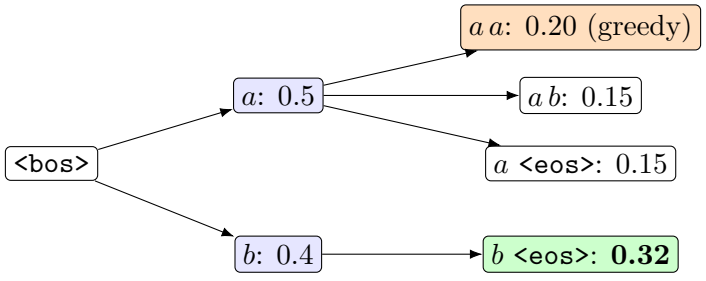
- **Multi-query attention, MQA** (Shazeer, 2019): all heads keep their own queries but share one key head and one value head; the cache shrinks h times (12× here).
- **Grouped-query attention, GQA** (Ainslie et al., 2023): heads share keys and values in groups (e.g. 8 query heads per key/value head), a middle ground used by LLaMA 2–3.

FlashAttention (Dao et al., 2022) computes *exactly* the same attention, but in tiles that fit in the GPU's fast memory, never storing the $T \times T$ weight matrix. It is an implementation change, not a model change, and it is what `F.scaled_dot_product_attention` uses when it can.

5 Beam Search (Optional Problem 10)

Greedy decoding commits to the best word at each step; the best *sentence* may start with a second-best word. **Beam search** keeps the k best partial sentences (the beam), scored by the sum of their log-probabilities. At each step, extend every partial sentence by every word, then keep the best k of all extensions. Finished sentences (ending in `<eos>`) are set aside.

Worked example: greedy misses, beam of 2 finds Step 1: $P(a) = 0.5$, $P(b) = 0.4$. Step 2 after a: a 0.4, b 0.3, `<eos>` 0.3; after b: `<eos>` 0.8. Greedy picks a then a: probability $0.5 \cdot 0.4 = 0.20$. A beam of 2 keeps a and b, and finds "b `<eos>`" with $0.4 \cdot 0.8 = 0.32$.



Two practical details. (1) Sums of log-probabilities favour short outputs, so beams are compared with a **length penalty**, e.g. GNMT's $\log P / \mathrm{lp}(Y)$ with $\mathrm{lp}(Y) = \big((5 + |Y|)/6\big)^{\alpha}$, $\alpha \approx 0.6$. (2) Each beam needs its *own* decoder state: for a GRU, its hidden state; for a Transformer, its own KV cache, copied when a beam splits. Chat models usually *sample* (basic note) instead: beam search finds high-probability text, which for open-ended writing is bland and repetitive (Holtzman et al., 2020; top-p sampling comes from that paper).

6 Where the Compute Goes, and Why Dev Mode Looks the Way It Does

Count the multiply-adds of one block on one sequence of T tokens:

part	cost	grows with
Q, K, V and output projections	$4\,T d^2$	T and d^2
feed-forward network	$2\,T d\, d_{ff} = 8\,T d^2$	T and d^2
attention scores $QK^\top$ and the weighted sum AV	$2\,T^2 d$	T^2 and d

A stack of L blocks costs L times one block: **linear in depth, quadratic in width, and quadratic in length only through attention.** With the build script's numbers:

model (per block, one sequence)	projections	FFN	attention	attention share
HW7 P5 GPT, full ($T = 64$, $d = 256$)	16.8M	33.6M	2.1M	4%
HW7 P5 GPT, dev ($T = 64$, $d = 64$, $d_{ff} = 128$)	1.0M	1.0M	0.5M	20%
GPT-2 small ($T = 1{,}024$, $d = 768$)	2.4G	4.8G	1.6G	18%
the same at $T = 32{,}768$	77G	155G	1,649G	88%

HW7's dev-mode question. Dev mode cuts the width from 256 to 64 and the depth from 4 to 2: since most of the cost goes like d^2, the per-block cost drops to 5%, and with half the blocks the model to about 2.5%. It keeps the 64-token windows, because the windows, the causal mask and the positions are part of what is being *tested*. The reverse task's dev sequences are shorter (30 symbols): that halves the recurrent steps and quarters the attention.

Parameters and scaling. Per block, parameters are $\approx 12d^2$ ($4d^2$ attention, $8d^2$ FFN); at $d = 256$ that rule gives 786,432 against the exact 789,760. Training runs are planned with *scaling laws*: the test loss of language models falls as a smooth power law in parameters, data and compute (Kaplan et al., 2020), which is why "make it bigger" kept working.

7 Training Stability and the Modern Block

Post-norm stacks need a **learning-rate warm-up** (start near 0 and increase over the first thousands of steps) to train at depth; Xiong et al. (2020) show that post-norm's gradients near the output are large at initialization, and that pre-norm removes the need for warm-up. We saw a small version while building HW7: a 4-block post-norm GPT at learning rate 10^{-3}, without warm-up, stayed at the unigram model (perplexity about 300) for the whole run, while the pre-norm GPT and Hugging Face's GPT-2 trained normally at the same rate; at $5 \cdot 10^{-4}$ (the HW's setting) all three train. One setting, so treat it as an illustration, not a law.

A 2024 block (LLaMA-style) still has the 2017 skeleton with small changes:

	Vaswani 2017 / your P4	GPT-2 / HF in P5	LLaMA-style
norm placement	post-norm	pre-norm + final LN	pre-norm + final norm
norm	LayerNorm	LayerNorm	RMSNorm (no mean subtraction, no β)
FFN activation	ReLU	GELU	SwiGLU (a gated FFN)
positions	sin/cos, added	learned, added	RoPE, inside attention
attention heads	multi-head	multi-head	grouped-query (GQA)
biases	yes	yes	none

8 From Bahdanau to the Transformer in Five Steps (Optional Problem 11)

Last year's HW7 walked from the attention GRU to the Transformer by replacing one part at a time. Each step is a working translator, so each step's BLEU tells you what that one change bought:

1. **Multi-head attention in the GRU decoder**: the decoder's query is split into heads, each attending to the encoder states. (Reshape $(B, T, d) \to (B, h, T, d/h)$; the ATTENTION note.)
2. **Self-attention on top of the GRU encoder**: Add & Norm blocks refine the encoder states, so each Spanish vector sees the whole sentence, not just its left context.
3. **Masked self-attention + cross-attention in a GRU decoder**: the decoder rereads its own history before consulting the source.
4. **Transformer decoder on the GRU encoder**: no recurrence on the English side.
5. **The full Transformer**: no recurrence anywhere (HW7 P4).

The question to ask at each step: what can a position see, in how many steps, and what must still be computed one word at a time?

9 A Decoder-Only Model Translates (Optional Problem 8)

Translation as language modelling: one sequence `es <sep> en <eos>` over one *shared* vocabulary (24,368 Spanish + English words + `<sep>`), the HW7 GPT with the causal mask, and the loss only on the English part. Careful with the shift: the label at each position is the *next* token, so the position holding `<sep>` is the one that predicts the first English word, and `<eos>` is a label too:

input	`tengo`	`hambre`	`<sep>`	`i'm`	`hungry`
label	(ignored)	(ignored)	`i'm`	`hungry`	`<eos>`

So the labels are ignored where they are Spanish (the first two), not where the input is Spanish. At test time, the prompt is `es <sep>` and the model continues. One practical point: the output layer over 24k words, at every position, is most of the memory, so compute it only at the positions that have a label (the English half). In the reference run: **43.9 BLEU** (9.4M parameters) vs 45.4 for your encoder–decoder and 44.4 for `nn.Transformer`: about equal at this budget. The prompt is the task description; this is how one GPT does every task.

10 Your Models and Hugging Face's (Optional Problem 9)

your HW7 model	Hugging Face class	differences
P4 encoder blocks (no causal mask)	`BertModel`	segment embeddings, learned positions, GELU, MLM head
P5 GPT	`GPT2LMHeadModel`	pre-norm, GELU, final LN, no output bias
P4 encoder–decoder	`T5ForConditionalGeneration`	pre-norm, RMSNorm-style norm, relative position biases

One trap when copying weights into GPT-2: its projections are `Conv1D` layers that store the weight as $(d_{\text{in}}, d_{\text{out}})$, the *transpose* of `nn.Linear`'s $(d_{\text{out}}, d_{\text{in}})$, and its Q, K, V projections are packed in one matrix `c_attn` of width $3d$.

Further reading

- Vaswani et al. (2017), *Attention Is All You Need* (arXiv:1706.03762).
- Devlin et al. (2019), *BERT* (arXiv:1810.04805). Liu et al. (2019), *RoBERTa* (arXiv:1907.11692).
- Radford et al. (2019), *Language Models are Unsupervised Multitask Learners* (GPT-2).
- Raffel et al. (2020), *T5: Exploring the Limits of Transfer Learning* (arXiv:1910.10683). Lewis et al. (2020), *BART* (arXiv:1910.13461).
- Su et al. (2021), *RoFormer: Rotary Position Embedding* (arXiv:2104.09864). Press et al. (2022), *ALiBi* (arXiv:2108.12409).
- Shazeer (2019), *Fast Transformer Decoding* (MQA, arXiv:1911.02150). Ainslie et al. (2023), *GQA* (arXiv:2305.13245).
- Dao et al. (2022), *FlashAttention* (arXiv:2205.14135).
- Xiong et al. (2020), *On Layer Normalization in the Transformer Architecture* (arXiv:2002.04745).
- Kaplan et al. (2020), *Scaling Laws for Neural Language Models* (arXiv:2001.08361).
- Holtzman et al. (2020), *The Curious Case of Neural Text Degeneration* (arXiv:1904.09751).
- d2l book §10.8 (beam search), §11.9 (pretraining), §15.8–15.10 (BERT).