ML 2026 Spring HW3
LLM Fast Inference

TA: 馮柏翰, 吳岳霖, 蘇炳揚
ntu-ml-2026-spring-ta@googlegroups.com

Deadline: 2026/**04/09(Thu)** 23:59:59 (UTC+8)

Outline

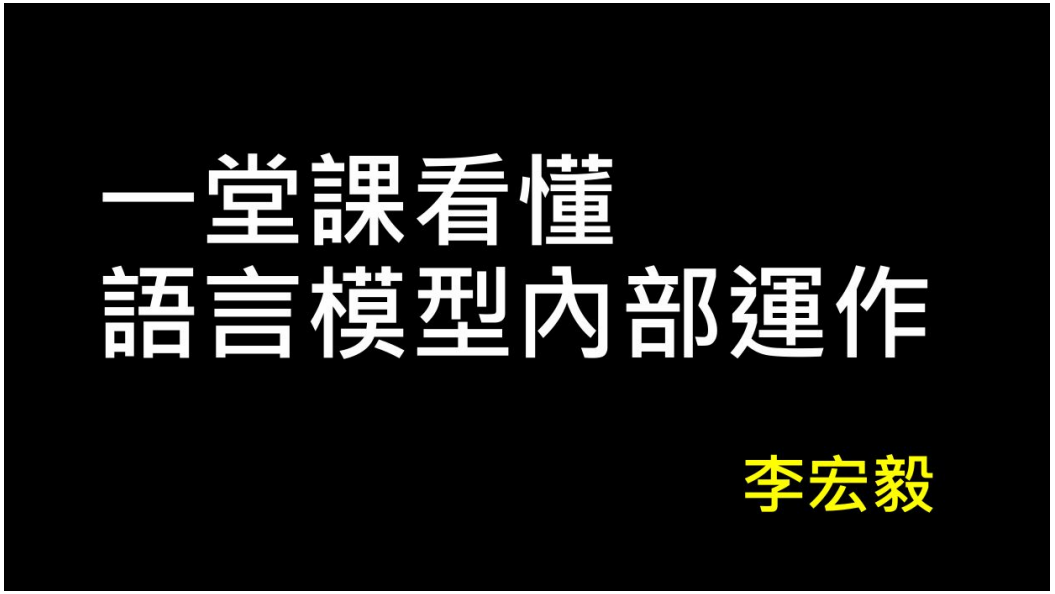
- Prerequisite
- Assignment Format
- Included Topics
  - Speculative Decoding
  - FlashAttention
  - VLLM
- Submission and Grading
- References
- HW3 Questions

Links

- ML 2026 Spring Course Website
- NTU Cool HW3 作業討論區
- Colab Sample Code

Prerequisite

Please watch Prof. Lee's following lecture video before working on this HW.



【生成式人工智慧與機器學習導論2025】第3講：解剖大型語言模型

Assignment Format

- There are **20 multiple-choice questions** with **0.5 point for each question**, so there are **10 points in total** for this HW.
  - Part 1: Paper Reading, 10 questions.
  - Part 2: Coding, 10 questions.
- **You only need to complete the quiz on NTU Cool and submit it.**

Assignment Format - Paper Reading (1/2)

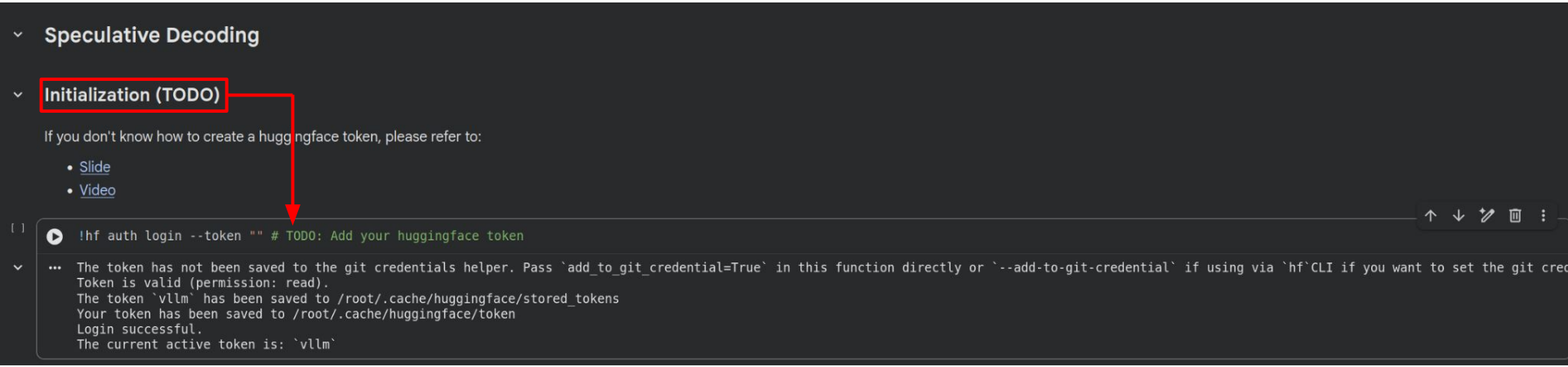
- Read the following papers to answer Q1~Q10:
  - Speculative Decoding
    - Fast Inference from Transformers via Speculative Decoding
    - Accelerating Large Language Model Decoding with Speculative Sampling
    - Inference with Reference: Lossless Acceleration of Large Language Models
    - SpecInfer: Accelerating Generative Large Language Model Serving with Tree-based Speculative Inference and Verification
  - FlashAttention
    - FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness
    - FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning
    - FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision

Assignment Format - Paper Reading (2/2)

- Some questions are more **general**, while others are more **specific**.
- It's recommended to first read through the paper at a **high level**, then look more closely at the sections related to the question.
- You can also use an **LLM** to find the parts of the paper related to the question, or to verify your own answer.
- More related papers (not used for the questions):
  - GitHub - hemingkx/SpeculativeDecodingPapers: 📰 Must-read papers and blogs on Speculative Decoding ⚡
  - FlashAttention-4: Algorithm and Kernel Pipelining Co-Design for Asymmetric Hardware Scaling

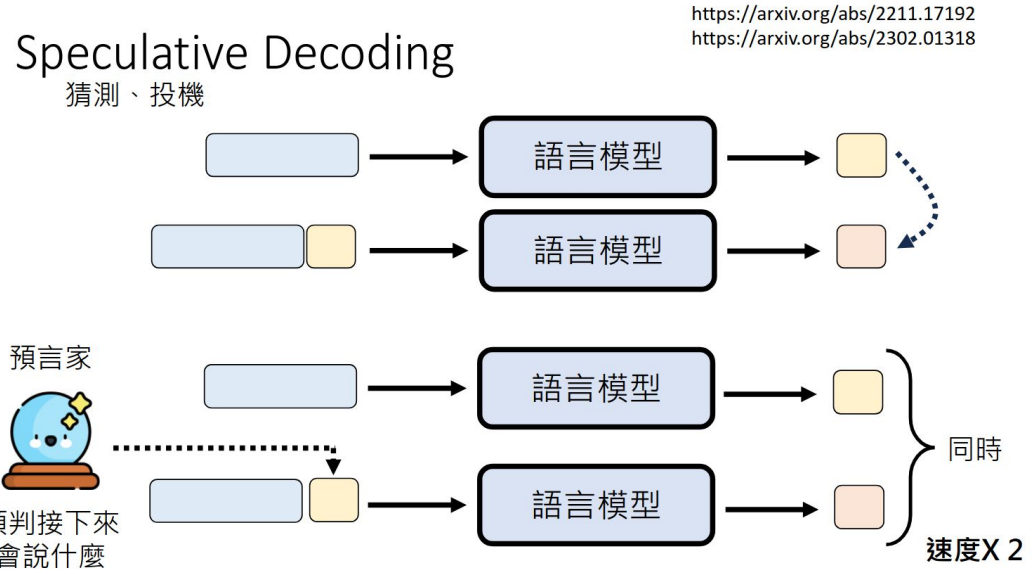
Assignment Format - Coding

- Fill the blanks in the sample code and analyze execution results to answer Q11~Q20.
- You can also finish coding questions before answering Q1~Q10.
- Only need to modify code blocks with "**# TODO: ...**" comment.

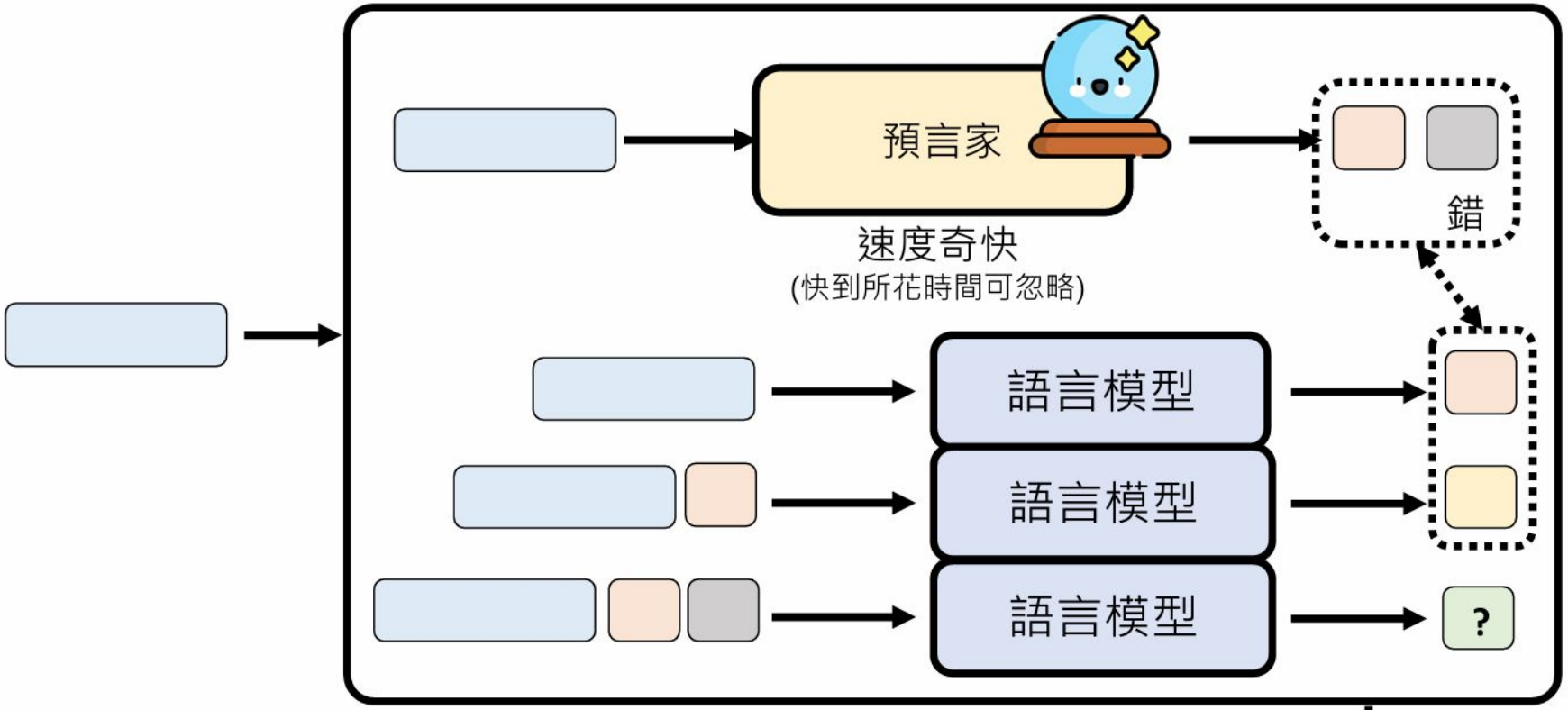


Included Topics - Speculative Decoding

A high-level algorithm using draft predictions verified by the main model.



【生成式AI導論 2024】第16講：可以加速所有語言模型生成速度的神奇外掛— Speculative Decoding



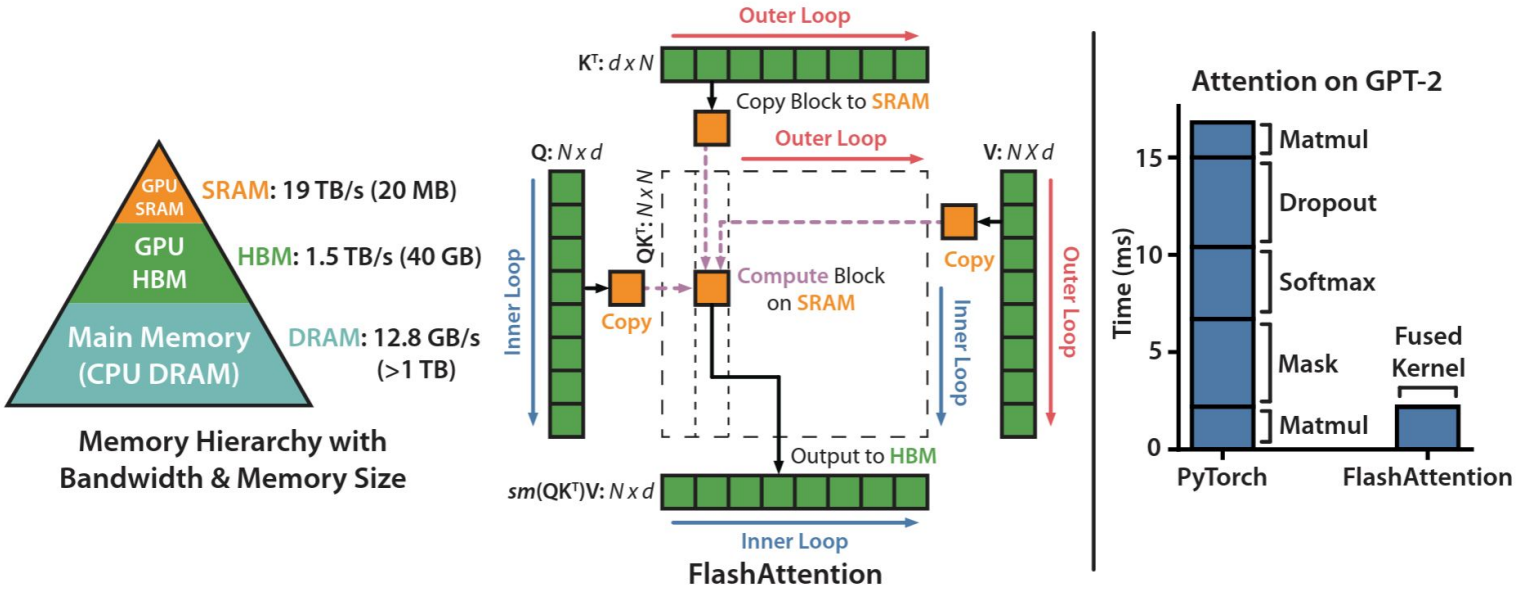
用原來只能輸出一個 Token 的時間，輸出二個 Token

還是有賺!

【生成式AI導論 2024】第16講：可以加速所有語言模型生成速度的神奇外掛— Speculative Decoding

Included Topics - FlashAttention

A low-level optimization that accelerates attention computation in LLMs.



FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Algorithm 1 FlashAttention

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M.

1: Set block sizes $B_c = \lceil \frac{M}{4d} \rceil$, $B_r = \min\left(\lceil \frac{M}{4d} \rceil, d\right)$.
2: Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}$, $\ell = (0)_N \in \mathbb{R}^N$, $m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
3: Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \ldots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
4: Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \ldots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \ldots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \ldots, m_{T_r}$ of size B_r each.
5: **for** $1 \le j \le T_c$ **do**
6: Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
7: **for** $1 \le i \le T_r$ **do**
8: Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
9: On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
10: On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
11: On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
12: Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}(\mathrm{diag}(\ell_i) e^{m_i - m_i^{\mathrm{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
13: Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.
14: **end for**
15: **end for**
16: Return $\mathbf{O}$.

N: sequence length
d: embedding dimension for each attention head
N >> d

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

Analyze Read/Write Memory

Standard Attention

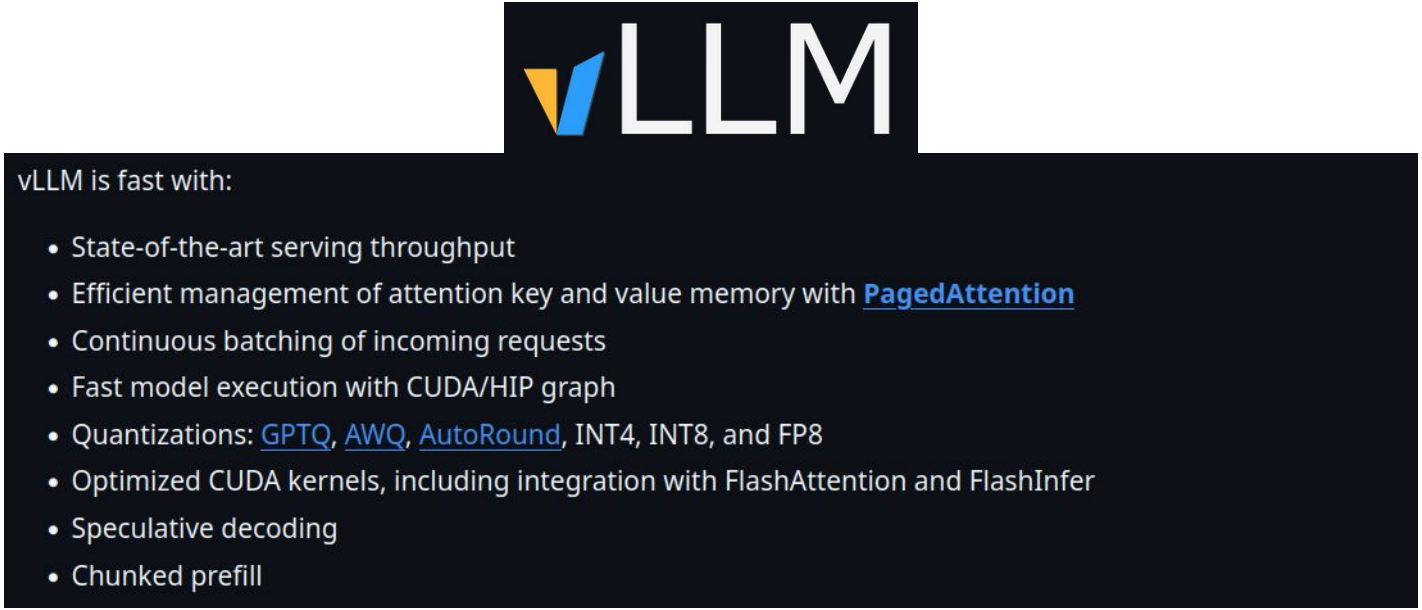
- Read
  - Q: N*d
  - K: N*d
  - S: N*N
  - P: N*N
  - V: N*d
- Write
  - S: N*N
  - P: N*N
  - O: N*d
- Total
  - N(4d+4N)

FlashAttention

- Read
  - K: N*d
  - V: N*d
  - Q: N*d
  - O: N*d
  - l: N*1
  - m: N*1
- Write
  - O: N*d
  - l: N*1
  - m: N*1
- Total
  - N(5d+4)

Included Topics - VLLM

A system that combines techniques to enable fast and efficient LLM inference.



A high-throughput and memory-efficient inference and serving engine for LLMs
Efficient Memory Management for Large Language Model Serving with PagedAttention

Submission & Deadline

- Submit your homework to **NTU Cool**, you don't need to submit your code.
- There is no submission limit for the NTU Cool quiz. Your highest score among all attempts will be taken as your final grade.
- 2026/**04/09** 23:59:59 (UTC+8)
- **No late submission is allowed**

Grading Release Date

- The grading of the homework will be released by 2026/**04/12** 23:59:59 (UTC+8)

Grading - Regulations

- You should NOT plagiarize.
- Do NOT share codes or prediction files with any living creatures.
- Your final grade x 0.9 and get a score 0 for that homework if you violate any of the above rules first time (within a semester).
- Your will get F for the final grade if you violate any of the above rules multiple (> 1) times (within a semester).
- Prof. Lee & TAs preserve the rights to change the rules & grades.

If You Have Any Questions

- NTU Cool **HW3** 作業討論區
  - 如果同學的問題不涉及作業答案或隱私，請**一律使用** NTU Cool 討論區
  - 助教們會優先回答 NTU Cool 討論區上的問題
- Email: ntu-ml-2026-spring-ta@googlegroups.com
  - Title should start with [ML 2026 Spring **HW3**]
  - Email with the wrong title will be moved to trash automatically
- TA Hours
  - Each Friday before / after class:
    - (Fri.) 13.20 ~ 14.10 / 17:30~18:00
    - Location: 博理 112

References

- GenAI-ML-2025 HW3
- 【生成式人工智慧與機器學習導論 2025】第3講：解剖大型語言模型 - YouTube
- 【生成式AI導論 2024】第16講：可以加速所有語言模型生成速度的神奇外掛 — Speculative Decoding
- A high-throughput and memory-efficient inference and serving engine for LLMs
- Efficient Memory Management for Large Language Model Serving with PagedAttention

HW3 Questions

- For those who are neither enrolled in nor auditing the course, we also provide all questions (without answers) here. The questions are identical to those on NTU Cool.
- The link for the document including answers and explanation will be released **here** after grading release date 2026/**04/12** 23:59:59 (UTC+8).

Chinese + English

(應選一項)參考論文 Fast Inference from Transformers via Speculative Decoding, 請排序以下 speculative decoding step 的步驟:

1. 使用 target model 與 approximation model 第 $n+1$ 個機率分佈, 計算出修正的分佈並從該分佈抽樣出 token "t"
2. 根據第 i 個 target model 生成的機率分佈決定是否接受 approximation model 生成的 token x_i, 從 $i=1$ 開始決定直到 $i=n+1$ 時 "x_{n+1}" 被拒絕, 最終找出接受的 token(s) "$[x_1, x_2, \ldots, x_n]$"
3. 回傳 token 序列 "prefix + $[x_1, \ldots, x_n, t]$"
4. 給定一字串 pefix 使用 approximation model 自回歸地生成 γ 個 tokens "$[x_1, x_2, \ldots, x_\gamma]$" 與對應的機率分佈
5. 用 target model 平行地根據輸入 "prefix", "prefix + $[x_1]$", …, "prefix + $[x_1, \ldots, x_\gamma]$" 生成 $\gamma+1$ 個下個 token 的機率分佈

A. 42513
B. 25143
C. 45213
D. 54213
E. 45123

Q1

(應選三項)參考論文 Fast Inference from Transformers via Speculative Decoding，下列哪些是使用 speculative decoding 加速大型語言模型推論時一定要承擔的後果？

A. 一定要找一個大型語言模型當作 approximation model
B. 同一組 target model 和 approximation model，在推論時會因為不同任務性質，導致 speculative decoding 加速效果不同
C. 使用 speculative decoding，跟一般 decoding 相比，電腦要花費更多的計算資源
D. 除了原本輸入的 tokens，一個 speculative decoding step 可能無法回傳任何一個新 token，但最多可以產生 $\gamma + 1$ 個新 tokens
E. 假設有無限算力，在 approximation model 生成的機率分佈跟 target model 差很多時，就算不斷把 γ 調大，一個 speculative decoding step 生成的 token 數也不一定會跟 γ 一樣大

Q2

（應選四項）參考論文 Accelerating Large Language Model Decoding with Speculative Sampling，關於選擇 draft model 加速 target model 推論時要注意的事，下列哪些敘述是正確的？（ draft model 等同前一篇論文的 approximation model）

A. 如果使用 random sampling (有設 temperature, top-p, top-k) 從模型的輸出分佈隨機選擇輸出的 token，即使 draft model 預測的 token 跟 target model 輸出的 token 很常相同，也有可能因為 draft model 的輸出分佈極度集中在幾個 token 導致 draft model 預測的 token 容易被拒絕（詳見 Algorithm 2）
B. 只要 draft model 推論速度快且 token 接受率夠高，並且 draft model 的詞彙表與 target model 相同，無論要加速哪個 target model 都不需要在意 draft model 的模型架構
C. 如果使用 greedy decoding 從模型的輸出分佈選擇機率最高的 token 當作輸出的 token，可以節省 Modified Rejection Sampling 計算 extra token 的時間，從實驗結果得知加速效果也比使用 random sampling 快一點
D. 在 draft model 一次預測多個 tokens 時，加速效果受限的原因除了有最後幾個預測 token 會被 target model 拒絕外，也有可能是 target model 在驗證 draft model 預測的 token 時用多筆輸入跑平行推論，因為算力不 夠導致無法像一次只用一筆輸入跑推論一樣快
E. 即使有模型輸出的機率分佈，仍無法使用非自回歸生成的模型作為 draft model，只能用自回歸生成的模型

Q3

（應選兩項）參考論文 Inference with Reference: Lossless Acceleration of Large Language Models，下列哪些大型語言模型加速的應用**不符合**論文中 Figure 1 的描述？

A. 在大模型輸出答案前，先用另一個小模型輸出答案將其視為參考文件，利用小模型與大模型輸出的文本重疊進行加速
B. 將過去的問題與答案存入快取，問模型新問題時，找出快取中類似的問題與答案，利用快取中舊的答案作為參考文件來引導推論
C. 根據使用者的問題，從外部資料庫中檢索出多份相關文件作為參考，並利用生成結果與這些文件之間的重疊性來加速
D. 在模型輸出答案時，同時監測模型前幾層 logit lens 的輸出，若與最後一層模型的輸出高度重疊則直接取前幾層的輸出，省下後幾層模型參數計算的時間
E. 在多輪對話中，模型被要求修正或解釋先前的輸出，此時可以將前幾輪生成的 內容視為參考文件，利用對話間的文本重疊進行加速

Q4

(應選一項)參考論文 SpecInfer: Accelerating Generative Large Language Model Serving with Tree-based Speculative Inference and Verification，下列關於 SpecInfer 架構中 Learning-based Speculator 與 Token Tree Verifier 的敘述哪些是正確的？

A. 在 Learning-based Speculator 使用 expansion-based token tree construction 建立 token tree 是考量到 small speculative model 的 top-k tokens 在 k 很小但大於 1 時，除了 $k = 1$ 的 token，其他 tokens 也很有可能為要加速的 LLM 所接受；尤其在 LLM 使用 greedy decoding 時驗證 small speculative model 的 top-k tokens，如果去除 $k = 1$ 的情況，獲得的驗證成功率會比 LLM 使用 stochastic decoding 還要高(詳見 Table 1)
B. 在 Learning-based Speculator 使用 Merge-based token tree construction，必須先準備多個 small speculative models 透過 boosting 的方式訓練，使它們的輸出分佈盡可能接近 LLM 的輸出；具體的 boosting 訓練方式是先讓 LLM 對多個 prompt 產生輸出拿來訓練第 1 個 small speculative model，訓練完後只要 LLM 跟 small speculative model 對同一個 prompt 輸出相同的 tokens，就把該資料標記，再來訓練下一個 small speculative model 時，就把標記資料的權重調低用相同的資料量訓練，希望多個 speculative models 彼此輸出越接近越好
C. 如果 Learning-based Speculator 產生四個 speculated token sequences: “machine learning model works”, “machine vision system adapts”, “machine vision system scales”, “machine learning model improves” 並將它們合成一個 token tree，該 token tree 的 expansion configuration 是 <2, 2, 1>
D. 在 Token Tree Verifier 用 LLM 驗證 speculated token sequences 時，因為多個 sequences 共享部份的 prefix，可以將共享 prefix 的 key 和 value 只存一份在 kv cache 中，避免每個 sequence 都要存一份相同的 key 和 value，同時節省計算與儲存 key 和 value 的時間空間
E. Token Tree Verifier 使用的 LLM 會透過 pipeline model parallelism 同時在多張 GPU 跑 LLM 同一層 transformer layer 的運算，以及透過 tensor model parallelism 把 LLM 多個 transformer layers 分配到多張 GPU 運算

Q5

（應選一項）參考論文 FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness，請排序以下 flash attention forward pass的步驟：

給定 $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in R^{N*d}$
1. 從 HBM 載入 $\mathbf{K}$, $\mathbf{V}$ 至 SRAM
2. 在晶片上計算 $\mathbf{S}, \tilde{m}, \tilde{\mathbf{P}}, \tilde{l}$
3. 將 $\mathbf{Q}$ 切分為維度 $B_r * d$ 的矩陣 $T_r = \text{ceil}(N / B_r)$ 塊，$\mathbf{K}$ 跟 $\mathbf{V}$ 切分為維度 $B_c * d$ 的矩陣 $T_c = \text{ceil}(N / B_c)$ 塊，$\mathbf{O}$ 切分為維度 $B_r * d$ 的矩陣 T_r 塊，l 跟 m 切分為維度 $B_r * 1$ 的矩陣 T_r 塊
4. 由 $\tilde{m}, \tilde{l}, \tilde{\mathbf{P}}$ 計算出 m^{new}, l^{new} 並更新 $\mathbf{O}$
5. 從 HBM 載入 $\mathbf{Q}$, $\mathbf{O}$, l, m 至 SRAM
6. 將 m^{new}, l^{new}, 更新後的 $\mathbf{O}$ 寫入 HBM

A.315642
B.126435
C.145632
D.364521
E.315246

Q6

（應選三項）參考論文 FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness，下列哪些是 FlashAttention 正確的敘述？

A. FlashAttention 在 backward pass 階段只有載入於 forward pass 階段寫入的 l, m 向量，不須其他 forward pass 的輸出
B. 雖然 FlashAttention 需要進行較多次的運算，但因為「把資料從 HBM 載入 SRAM」跟「把 SRAM 上計算好的資料寫入 HBM」的資料量總和較少，而比標準的 Attention 還要省時
C. 若 N 是序列長度、d 是 multi-head attention 每個 head 的 embedding dimension、M 是 SRAM 大小且 $d <= M <= Nd$，FlashAttention 已經達到要計算 Exact Attention 最快的時間複雜度
D. 若 N 是序列長度，FlashAttention 為了記錄 dropout mask 供 backward pass 計算，我們需要 $O(N^2)$ 的額外記憶體
E. FlashAttention 因為是在 SRAM 上計算矩陣 **S** 跟 **P**，不將它們存回 HBM，所以在 backward pass 計算 **Q**, **K**, **V** 的 gradients 時，要用額外儲存在 HBM 的 **l**, **m** 向量，重新計算出矩陣 **S** 跟 **P**

Q7

（應選三項）參考論文 FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning，下列哪些是 FlashAttention-2 基於初代 FlashAttention 多做的優化？

A. 減少非矩陣乘法在所有計算中的計算量佔比
B. 把 **Q**, **K**, **V** 矩陣拆開計算中間 值，最後再合併成最終的 attention layer output
C. 計算 attention 時除了對 attention head 和 batch 的資料筆數這兩個維度做平行化，額外對 sequence length 這個維度做平行化
D. 不只加速 forward pass，額外加速 backward pass 的計算
E. 在 GPU 的一個 thread block，把 **K** 和 **V** 兩個矩陣讓所有 wraps 都共享

Q8

（應選兩項）參考論文 FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision，下列哪些是正確的敘述？

A. Thread 是大量存在於 GPU 中負責管理運算的執行單位，而 Hopper GPU 管裡 thread 的階層由最頂層到最底層是：grids -> threadblock clusters -> threadblocks (cooperative thread arrays / CTA) -> warpgroups (8 contiguous warps) -> warps (32 threads) -> threads（詳見 section 2.2）
B. 有鑑於「指數運算」和「矩陣乘法」分別在 GPU 的「multi-function unit」和「Tensor Core」上執行，可以讓一個 warpgroup 在算 softmax 時由另一個 warpgroup 算矩陣乘法
C. 同一個 warpgroup 也可以讓不同 iterations 中 GEMM 和 softmax 的指令透過 pipelining 同利用更多 Tensor Core 達到部份平行化，但代價是把一個 iteration 的運算拆成多個步驟時，必須先花更多 register 儲存前一個步驟的運算結果才能跑下個步驟的運算
D. 在 FP8 精度計算 attention weights 時，FlashAttention-3 把 **Q** 與 **K** 矩陣相乘前，會先把 **Q** 與 **K** 各自乘上一個正交矩陣再做 FP8 量化，讓 **Q** 與 **K** 裡的極端數值因為與多數非極端數值加總藉此降低 quantization error，但 **Q** 與 **K** 和正交矩陣相乘有 $O(d^2)$ 時間複雜度（d 是一個 attention head 的 embedding dimension）
E. 只有 H100 / H20 等 Hopper 架構的 GPU 才能使用官方的 FlashAttention-3，像 A100 / V100 / RTX 4090 / RTX 3090 / T4 等 GPU 只能用官方的 FlashAttention-2（官方 FlashAttention：Dao-AILab/flash-attention: Fast and memory-efficient exact attention）

Q9

(應選三項)參考上課內容、作業頁投影片與讀過的所有論文，下列哪些是正確的敘述？

A. 某些加速技術不只可以更快完成 LLM 的訓練或推論，也可以讓 LLM 訓練或推論所需的記憶體減少，使其在需要長上下文的任務上表現得更好
B. 有時 LLM 的加速技術不一定只是讓計算本身變快，也有可能是能讓 LLM 更充分地運用電腦多出的運算資源，比如 FlashAttention-2 和 speculative decoding 都可以有效增加 GPU 的使用率
C. 加速 LLM 訓練或推論速度的各種技術往往要付出一些代價，像是 speculative decoding, quantization, attention approximation 會使 LLM 表現變差；flash attention, vllm 相較於 speculative decoding 需要在 GPU 底層運算做更複雜的優化
D. LLM 訓練與推論中包含多個計算步驟，找出多個步驟中最花時間的部份優化它是重要的，像是 FlashAttention-1~3 代都是加速 LLM 中 attention 的計算，而相較於一般 attention，FlashAttention-1 尤其減少了 attention 計算在 GPU 記憶體中讀寫「相同大小」資料所需要的時間
E. Speculative decoding 和 Flash Attention 是相容的 LLM 加速技術，同時使用兩種技術可以超過只使用一種技術的加速效果

Q10

（應選一項）參考論文 Accelerating Large Language Model Decoding with Speculative Sampling，從以下選項選出一項填入 **Manual Speculative Decoding** 實驗中 manual_speculative_step_causal 函式裡的空格 diff = ???（assistant model 的意思等同 draft model 和 approximation model）

```
q_full = F.softmax(a_prefix.logits[:, -1, :] / max(temperature, eps), dim=-1)
# TODO: calculate the difference of distribution of prob between p(x) and q(x)
diff =
denom = diff.sum(dim=-1, keepdim=True)
fallback = torch.argmax(p_i, dim=-1, keepdim=True)
```

A. torch.abs(p_i - q_full)
B. torch.clamp(p_i - q_full, min=0)
C. torch.abs(q_full - p_i)
D. torch.clamp(q_full - p_i, min=0)

Q11

（應選一項）參考論文 Accelerating Large Language Model Decoding with Speculative Sampling，關於 **Manual Speculative Decoding** 實驗中，對 manual_speculative_step_causal 函式裡其中一行程式 ratio = torch.clamp(p_x / (q_x + eps), max=1.0) 的描述哪項是正確的？（assistant model 的意思等同 draft model 和 approximation model）

```
# Compute accept ratio min(1, p(x)/q(x)) and sample Bernoulli accept.
ratio = torch.clamp(p_x / (q_x + eps), max=1.0)
accept = torch.rand_like(ratio) < ratio
```

A. 這是為了決定 target model 是否要接受 assistant model 預測的 token
B. 這是為了計算 assistant model 的損失函數，用於訓練它更接近 target model
C. 這是為了動態調整 assistant model 一次 speculative decoding step 預測的 token 數
D. 這是為了實作 top_p sampling，透過限制累積分佈機率來過濾掉低機率的 tokens

Q12

(應選兩項)在 **Manual Speculative Decoding** 實驗中，請分別使用 normal prompt 和 simple prompt 執行 speculative_generate_manual_seq2seq 函式，分析所獲得的執行時間以及接受率 alpha，並選出以下正確的敘述。

A. 用 simple prompt 執行時間比 normal prompt 短的主因是 simple prompt 的輸入 token 數更少
B. 如果 alpha 足夠高，提昇 gamma 所帶來的加速效果提昇，是比低的 alpha 值還要更高的
C. 用 simple prompt 執行時間比 normal prompt 短的主因是 target model 和 assistant model 用 simple prompt 生成時，兩者 token 的機率分佈比用 normal prompt 更相近
D. 因為 Speculative Decoding 本身就就是帶有隨機性的演算法，即使理論上確保 speculative 跟 autoregressive generation 都能有相同的輸出，在所有參數都固定的情況下，還是有可能因為隨機性導致一樣的程式重複執行，會有不一樣的 alpha 跟輸出的文字內容

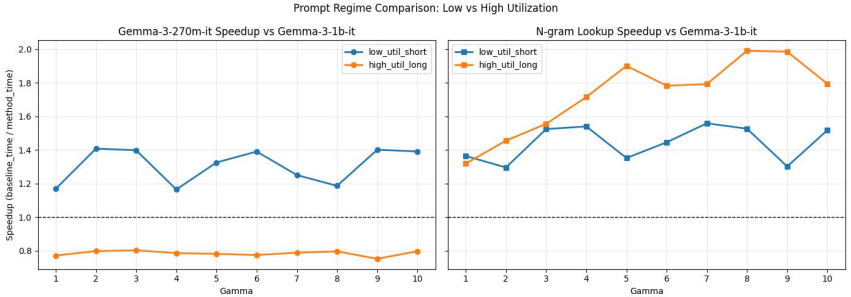
Q13

(應選一項)完成 **Benchmark with Two Prompt Regimes** 實驗後，根據最後 產生的圖片選出正確的敘述：

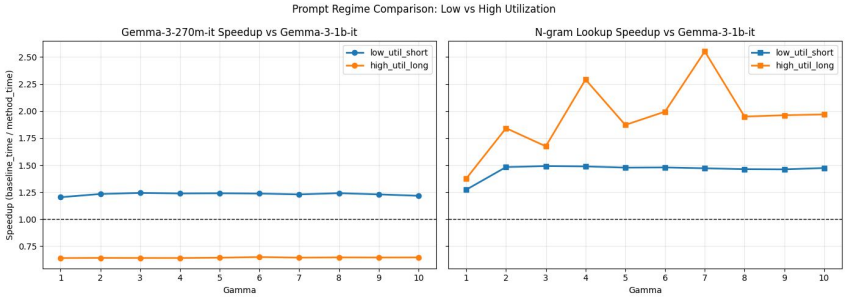
A. 無論使用何種 prompt 或 assistant model，加速效果都隨著 gamma 增加，呈現遞增或遞減的趨勢
B. 使用 high_util_prompt 時，speculative decoding 獲得的加速效果都會比使用 low_util_prompt 還要高 2 倍
C. 在 sample code 中只要使用 speculative decoding，至少就能有大於 1.5 倍的加速效果
D. 平均而言，N-gram 會比使用 gemma-3-270m-it 當 assistant model 有更快的加速效果

根據使用的 GPU 不同和部份隨機性，產生的圖片可能不同，以下提供一些不同 GPU 執行實驗後 產生的圖片供同學參考：

T4

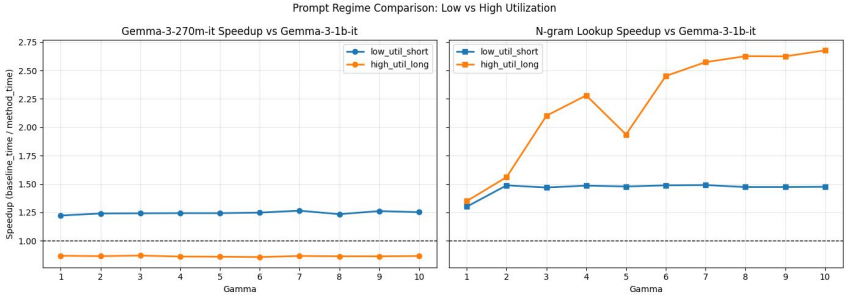


A100

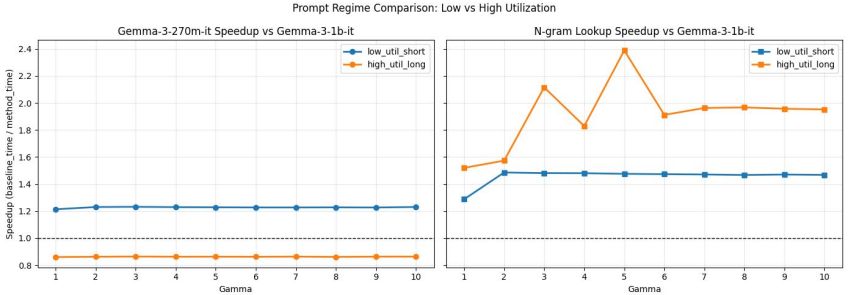


Q14

L4


Prompt Regime Comparison: Low vs High Utilization
Gemma-3-270m-it Speedup vs Gemma-3-1b-it
N-gram Lookup Speedup vs Gemma-3-1b-it
low_util_short
high_util_long
Speedup (baseline_time / method_time)
Gamma

G4 VM


Prompt Regime Comparison: Low vs High Utilization
Gemma-3-270m-it Speedup vs Gemma-3-1b-it
N-gram Lookup Speedup vs Gemma-3-1b-it
low_util_short
high_util_long
Speedup (baseline_time / method_time)
Gamma

Q14

（應選兩項）在 **Benchmark with Two Prompt Regimes** 實驗使用 speculative generation 後，如果用 T4 GPU 在某些參數設置下，加速效果比一般 autoregressive generation 更慢，有甚麼可能原因：

A. T4 GPU 已經在推論過程中「小的 assistant model 預測 gamma 個 tokens」的步驟被完全佔滿使用率，達到計算速度的瓶頸（可從終 端機使用 “sudo apt install nvtop” + “nvtop” 觀察 ）
B. Target model 驗證多個 tokens 時，速度仍然比自回歸生成一個 token 慢，無法完全平行化
C. 使用的 assistant model 其參數量與 target model 太過接近，無法忽略 assistant model 自回歸生成 gamma 個 tokens 的時間
D. 因為 speculative generation 演算法比 autoregressive generation 複雜，像是「隨機決定是否接受 assistant model 生成的 tokens」和「使用 target model 與 assistant model 修正最後一個回傳 token 的機率分佈」這兩個步驟，會導致 CPU 花費比 GPU 更多的時間計算

Q15

（應選一項）若 sequence length (N) 為 1024，執行 **FlashAttention** 實驗時，從 HBM 讀取的浮點數數量最接近下列哪一項？

A. 800,000
B. 1,000,000
C. 900,000
D. 700,000

Q16

（應選一項）若 sequence length (N) 為 2048，FlashAttention 與 Standard Attention 相比，在 T4 GPU 的理論加速的倍率最接近下列哪一項？

A. 1.835511
B. 1.899839
C. 1.934696
D. 1.952866

Q17

(1) 請截圖 **KV Cache Multi-turn Test** 實驗的結果（0.25 分）
(2)（應選一項）kv cache 第 234 輪問題的速度提升，是因爲哪部分的時間被節省了？（ 0.25 分）

A. 節省了將模型權重 (Model Weights) 從 CPU 載入到 GPU VRAM 的傳輸時間
B. 節省了生成階段 (Decode Phase) 中，預測每一個新 Token 所需的計算時間
C. 節省了 GPU 記憶體頻寬，因為 KV Cache 提升了 VRAM 的物理讀寫速度
D. 節省了共同前文 (Shared Prefix) 在 Prefill 階段中，通過 Attention 與 FFN 層進行矩陣乘法運算的時間

Q18

(1) 請截圖 **KV Cache Invalidation** 實驗的結果（0.25 分）
(2)（應選一項）下列哪些是在 prefix 加入一個空格之後原先計算的 kv cache 失效的原因（0.25 分）

A. 空格字元在 tokenizer 中會被自動忽略，因此 prefix 實際上沒有改變，kv cache 失效只是系統隨機重新計算所致
B. 空格改變了 prefix 字串的雜湊值 (Hash Value)，導致系統的 Semantic Cache 匹配失敗，判定為全新請求
C. 因為前文特徵與位置的變動，後續 Token 在計算注意力機制 (Attention) 時，產生的 Key 與 Value 矩陣會與原快取截然不同
D. 在 PagedAttention 機制中，額外的 Token 會造成記憶體區塊 (Block) 內部碎片化溢位，觸發系統強制清空並重新計算該段 Cache

Q19

(1) 請從 **vLLM CPU RAM/Speed Trade-off** 實驗截圖兩個不同的 cpu offload config（0.25 分）
(2)（應選一項）爲什麼把 model weight 搬到 cpu throughput 會變慢（0.25 分）

A. 因為每次產生一個新的 Token，前向傳播（Forward Pass）都需要重新從 CPU 搬運一次被卸載的權重
B. 因為權重在搬運過程中需要即時進行 FP16 到 FP32 的格式轉換
C. 因為 CPU 無法處理多用戶並發的請求，導致 Request 必須排隊
D. 因為權重搬運會覆寫掉原本存在 GPU 裡的 KV Cache

Q20

English

(Choose one) With reference to the paper *Fast Inference from Transformers via Speculative Decoding*, order the following steps of a speculative decoding step:

1. Using the $(n+1)$-th probability distributions from the target model and the approximation model, compute the corrected distribution and sample a token "t" from that distribution
2. Based on the i-th probability distribution generated by the target model, decide whether to accept the token x_i generated by the approximation model. Starting from $i = 1$, continue this process until "x_{n+1}" is rejected at $i = n + 1$, and finally determine the accepted token(s) "$[x_1, x_2, \ldots, x_n]$"
3. Return the token sequence "$\text{prefix} + [x_1, \ldots, x_n, t]$"
4. Given a string prefix, use the approximation model to autoregressively generate γ tokens "$[x_1, x_2, \ldots, x_\gamma]$" and their corresponding probability distributions
5. Use the target model, in parallel, to generate $\gamma + 1$ next-token probability distributions based on the inputs "prefix", "$\text{prefix} + [x_1]$", …, "$\text{prefix} + [x_1, \ldots, x_\gamma]$"

A. 42513
B. 25143
C. 45213
D. 54213
E. 45123

Q1

(Choose three) With reference to the paper *Fast Inference from Transformers via Speculative Decoding*, which of the following are unavoidable consequences when using speculative decoding to accelerate inference for large language models?

A. It is necessary to use a large language model as the approximation model
B. Even with the same target model and approximation model, the speedup from speculative decoding can differ across tasks because of differences in task characteristics
C. Compared with standard decoding, speculative decoding requires the computer to spend more computational resources
D. Besides the original input tokens, a single speculative decoding step may fail to return any new token, but it can generate up to $\gamma + 1$ new tokens
E. Assuming unlimited compute resources, if the probability distribution generated by the approximation model differs greatly from that of the target model, then even if γ is continuously increased, the number of tokens generated in one speculative decoding step does not necessarily approach γ

Q2

(Choose four) With reference to the paper *Accelerating Large Language Model Decoding with Speculative Sampling*, which of the following statements are correct regarding what should be considered when choosing a draft model to accelerate inference for a target model? (The draft model is equivalent to the approximation model in the previous paper)

A. If random sampling is used to select the output token from the model's output distribution (with temperature, top-p, or top-k applied), then even if the token predicted by the draft model is often the same as the token output by the target model, the token predicted by the draft model may still be likely to be rejected because the draft model's output distribution is extremely concentrated on only a few tokens (see Algorithm 2 for details)
B. As long as the draft model has sufficiently fast inference speed and a sufficiently high token acceptance rate, and the draft model uses the same vocabulary as the target model, then regardless of which target model is being accelerated, there is no need to care about the architecture of the draft model
C. If greedy decoding is used to choose the highest-probability token from the model's output distribution as the output token, the time required by Modified Rejection Sampling to compute the extra token can be saved, and experimental results show that the speedup is also slightly better than when random sampling is used
D. When the draft model predicts multiple tokens at once, the speedup may be limited not only because the last few predicted tokens may be rejected by the target model, but also because the target model verifies the tokens predicted by the draft model by running parallel inference on multiple inputs; if the available compute is insufficient, this may not be as fast as running inference on only a single input.
E. Even if the model's output probability distributions are available, non-autoregressive generation models still cannot be used as the draft model; only autoregressive generation models can be used

Q3

(Choose two) With reference to the paper *Inference with Reference: Lossless Acceleration of Large Language Models*, which of the following acceleration applications for large language models do **not** match the description in Figure 1?

A. Before the large model outputs an answer, another smaller model is first used to generate an answer, which is treated as a reference document; the overlap between the smaller model's output text and the larger model's output is then exploited for acceleration
B. Past questions and answers are stored in a cache. When a new question is asked to the model, similar past questions and answers are retrieved from the cache, and the old answers in the cache are used as reference documents to guide inference
C. Based on the user's question, multiple relevant documents are retrieved from an external database as references, and the overlap between the generated result and these documents is exploited for acceleration
D. While the model is generating an answer, the outputs of the logit lens from several earlier layers are monitored at the same time. If they highly overlap with the final-layer output, the outputs from the earlier layers are directly used, thereby saving the computation time of the later model layers
E. In multi-turn conversations, when the model is asked to revise or explain its previous outputs, the content generated in earlier turns can be treated as reference documents, and the textual overlap across conversation turns can be exploited for acceleration

Q4

(Choose one) With reference to the paper *SpecInfer: Accelerating Generative Large Language Model Serving with Tree-based Speculative Inference and Verification*, which of the following statements about the **Learning-based Speculator** and the **Token Tree Verifier** in the SpecInfer architecture is correct?

A. The reason why the Learning-based Speculator uses **expansion-based token tree construction** to build a token tree is that, when k is small but greater than 1, besides the token corresponding to $k = 1$, the other top-k tokens from the small speculative model are also likely to be accepted by the LLM being accelerated. In particular, when the LLM uses greedy decoding to verify the small speculative model's top-k tokens, if the $k = 1$ case is excluded, the resulting verification success rate is higher than when the LLM uses stochastic decoding (see Table 1 for details)
B. When the Learning-based Speculator uses **merge-based token tree construction**, it is necessary to first prepare multiple small speculative models and train them with a boosting-based method so that their output distributions are as close as possible to that of the LLM. Specifically, the boosting procedure is as follows: first, let the LLM generate outputs for multiple prompts and use them to train the first small speculative model; after training, as long as the LLM and the small speculative model generate the same tokens for the same prompt, that data is marked. Then, when training the next small speculative model, the marked data is assigned lower weight while keeping the same amount of training data, with the goal of making the outputs of multiple speculative models as similar to each other as possible
C. Suppose the Learning-based Speculator generates four speculated token sequences: "machine learning model works", "machine vision system adapts", "machine vision system scales", "machine learning model improves" , and merges them into a token tree. Then the expansion configuration of that token tree is <2, 2, 1>
D. When the Token Tree Verifier uses the LLM to verify speculated token sequences, since multiple sequences share part of the prefix, the keys and values of the shared prefix can be stored only once in the KV cache, instead of storing an identical copy of the keys and values for each sequence, thereby saving both the computation time and the memory space needed for computing and storing keys and values
E. The LLM used by the Token Tree Verifier uses **pipeline model parallelism** to run the computation of the same Transformer layer of the LLM simultaneously across multiple GPUs, and uses **tensor model parallelism** to distribute multiple Transformer layers of the LLM across multiple GPUs for computation

Q5

(Choose one) With reference to the paper *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*, order the following steps of the FlashAttention forward pass:

Given $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in R^{N*d}$

1. Load $\mathbf{K}, \mathbf{V}$ from HBM to SRAM
2. Compute $\mathbf{S}, \tilde{m}, \tilde{\mathbf{P}}, \tilde{l}$ on chip
3. Split $\mathbf{Q}$ into $T_r = \text{ceil}(N / B_r)$ blocks of size $B_r * d$; split $\mathbf{K}$ and $\mathbf{V}$ into $T_c = \text{ceil}(N / B_c)$ blocks of size $B_c * d$; split $\mathbf{O}$ into T_r blocks of size $B_r * d$; and split l and m into T_r blocks of size $B_r * 1$
4. Use $\tilde{m}, \tilde{l}, \tilde{\mathbf{P}}$ to compute m^{new}, l^{new}, and update $\mathbf{O}$
5. Load $\mathbf{Q}, \mathbf{O}, l, m$ from HBM to SRAM
6. Write m^{new}, l^{new}, and the updated $\mathbf{O}$ back to HBM

A.315642
B.126435
C.145632
D.364521
E.315246

Q6

(Choose three) With reference to the paper *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*, which of the following statements about FlashAttention are correct?

A. In the backward pass, FlashAttention only needs to load the l and m vectors written during the forward pass, and does not require any other outputs from the forward pass
B. Although FlashAttention requires more computation, it is still faster than standard attention because the total amount of data moved for “loading data from HBM to SRAM” and “writing computed data from SRAM back to HBM” is smaller
C. Let N be the sequence length, d be the embedding dimension of each head in multi-head attention, and M be the SRAM size, with $d <= M <= Nd$. FlashAttention already achieves the fastest possible time complexity for computing exact attention.
D. Let N be the sequence length, , then in order to record the dropout mask for use in the backward pass, FlashAttention requires an additional $O(N^2)$ memory
E. Because FlashAttention computes the matrices $\mathbf{S}$ and $\mathbf{P}$ in SRAM and does not write them back to HBM, when computing the gradients of $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ in the backward pass, it needs to use the $\mathbf{l}$ and $\mathbf{m}$ vectors additionally stored in HBM to recompute the matrices $\mathbf{S}$ and $\mathbf{P}$

Q7

(Choose three) With reference to the paper *FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning*, which of the following are optimizations that FlashAttention-2 additionally introduces on top of the first-generation FlashAttention?

A. Reducing the proportion of non-matrix-multiplication operations in the overall computation
B. Splitting the $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ matrices apart to compute intermediate values separately, and then combining them into the final attention layer output
C. In addition to parallelizing attention computation over the two dimensions of attention heads and batch size, further parallelizing over the sequence length dimension
D. Not only accelerating the forward pass, but also additionally accelerating the backward pass
E. Within a GPU thread block, making the $\mathbf{K}$ and $\mathbf{V}$ matrices shared by all warps

Q8

(Choose two) With reference to the paper *FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision*, which of the following statements are correct?

A. A thread is an execution unit that exists in large numbers on a GPU and is responsible for managing computation. On Hopper GPUs, the hierarchy for organizing threads from top to bottom is: grids -> threadblock clusters -> threadblocks (cooperative thread arrays / CTAs) -> warpgroups (8 contiguous warps) -> warps (32 threads) -> threads (see Section 2.2 for details)
B. Since the **exponential operation** and **matrix multiplication** are executed on the GPU's **multi-function unit** and **Tensor Cores**, respectively, one warpgroup can perform matrix multiplication while another warpgroup performs softmax
C. Even within the same warpgroup, the GEMM and softmax instructions from different iterations can also be partially parallelized through pipelining to make better use of Tensor Cores. However, the trade-off is that when the computation of one iteration is split into multiple steps, more registers are needed to store the intermediate results of the previous step before the computation of the next step can proceed
D. When computing attention weights in FP8 precision, FlashAttention-3 multiplies $\mathbf{Q}$ and $\mathbf{K}$ by an orthogonal matrix before quantizing them to FP8. This allows extreme values in $\mathbf{Q}$ and $\mathbf{K}$ to be reduced in quantization error by being summed together with many non-extreme values. However, multiplying $\mathbf{Q}$ and $\mathbf{K}$ by the orthogonal matrix has time complexity $O(d^2)$, where d is the embedding dimension of one attention head
E. Only Hopper-architecture GPUs such as the H100 / H20 can use the official FlashAttention-3, while GPUs such as the A100 / V100 / RTX 4090 / RTX 3090 / T4 can only use the official FlashAttention-2 (official FlashAttention: [Dao-AILab/flash-attention: Fast and memory-efficient exact attention]())

Q9

(Choose three) With reference to the course content, the homework slides, and all the papers you have read, which of the following statements are correct?

A. Some acceleration techniques can not only make LLM training or inference faster, but can also reduce the memory required for LLM training or inference, allowing the model to perform better on tasks that require long contexts
B. Sometimes, LLM acceleration techniques do not necessarily only make the computation itself faster; they may also allow LLMs to make better use of additional available computing resources. For example, both FlashAttention-2 and speculative decoding can effectively increase GPU utilization
C. Various techniques for accelerating LLM training or inference often come with certain trade-offs. For example, speculative decoding, quantization, and attention approximation may degrade LLM performance; compared with speculative decoding, flash attention and vLLM require more complex low-level GPU optimizations
D. LLM training and inference consist of multiple computation steps, so it is important to identify and optimize the most time-consuming parts among them. For example, FlashAttention-1 through FlashAttention-3 all accelerate attention computation in LLMs, and compared with standard attention, FlashAttention-1 in particular reduces the time required to read and write the **same amount** of data in GPU memory during attention computation
E. Speculative decoding and FlashAttention are compatible LLM acceleration techniques. Using both techniques together can achieve greater speedup than using only one of them

Q10

(Choose one) With reference to the paper *Accelerating Large Language Model Decoding with Speculative Sampling*, choose **one** of the following options to fill in the blank diff = ??? in the manual_speculative_step_causal function used in the **Manual Speculative Decoding** experiment. (Here, “assistant model” is equivalent to “draft model” and “approximation model”)

```
q_full = F.softmax(a_prefix.logits[:, -1, :] / max(temperature, eps), dim=-1)
# TODO: calculate the difference of distribution of prob between p(x) and q(x)
diff =
denom = diff.sum(dim=-1, keepdim=True)
fallback = torch.argmax(p_i, dim=-1, keepdim=True)
```

A. torch.abs(p_i - q_full)
B. torch.clamp(p_i - q_full, min=0)
C. torch.abs(q_full - p_i)
D. torch.clamp(q_full - p_i, min=0)

Q11

(Choose one) With reference to the paper *Accelerating Large Language Model Decoding with Speculative Sampling*, which of the following statements correctly describes the purpose of the line of code `ratio = torch.clamp(p_x / (q_x + eps), max=1.0)` in the `manual_speculative_step_causal` function used in the **Manual Speculative Decoding** experiment? (Here, “assistant model” is equivalent to “draft model” and “approximation model”)

```
# Compute accept ratio min(1, p(x)/q(x)) and sample Bernoulli accept.
ratio = torch.clamp(p_x / (q_x + eps), max=1.0)
accept = torch.rand_like(ratio) < ratio
```

A. This is used to decide whether the target model should accept the token predicted by the assistant model
B. This is used to compute the loss function of the assistant model, in order to train it to better match the target model
C. This is used to dynamically adjust the number of tokens predicted by the assistant model in one speculative decoding step
D. This is used to implement top-p sampling by filtering out low-probability tokens through constraining the cumulative distribution probability

Q12

(Choose two) In the **Manual Speculative Decoding** experiment, run the speculative_generate_manual_seq2seq function with both a **normal prompt** and a **simple prompt**, analyze the resulting execution time and acceptance rate alpha, and choose the correct statements below.

A. The main reason why execution with the simple prompt is faster than with the normal prompt is that the simple prompt has fewer input tokens
B. If alpha is sufficiently high, then the speedup improvement brought by increasing gamma is greater than when alpha is low
C. The main reason why execution with the simple prompt is faster than with the normal prompt is that, when generating from the simple prompt, the probability distributions of the target model and the assistant model are closer to each other than when generating from the normal prompt
D. Because Speculative Decoding itself is a stochastic algorithm, even though it is theoretically guaranteed that speculative generation and autoregressive generation can produce the same output distribution, it is still possible that, with all parameters fixed, repeated runs of the same code may produce different alpha values and different generated text due to randomness

Q13

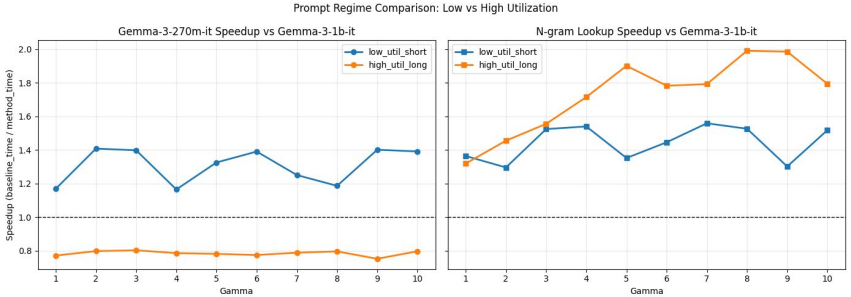
(Choose one) After completing the **Benchmark with Two Prompt Regimes** experiment, select the correct statement based on the final generated figure:

A. Regardless of which prompt or assistant model is used, the speedup always shows either a monotonically increasing or monotonically decreasing trend as gamma increases
B. When using high_util_prompt, the speedup obtained by speculative decoding is always twice as high as when using low_util_prompt
C. In the sample code, as long as speculative decoding is used, the speedup is guaranteed to be greater than 1.5×
D. On average, N-gram achieves faster speedup than using gemma-3-270m-it as the assistant model

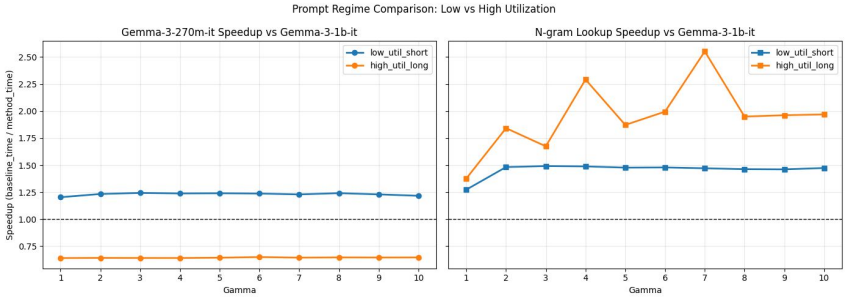
Q14

Because the generated figure may differ depending on the GPU used and some randomness, below are several figures produced by running the experiment on different GPUs for students' reference:

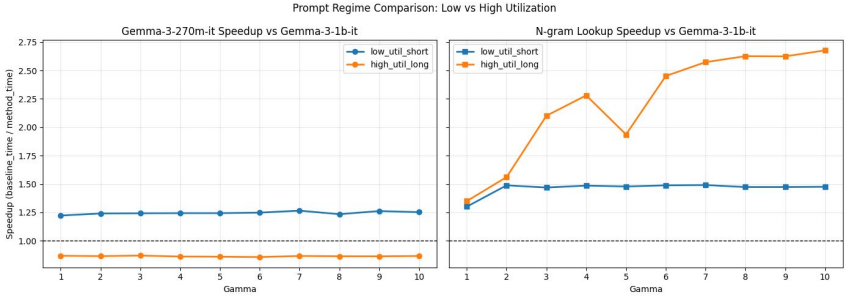
T4



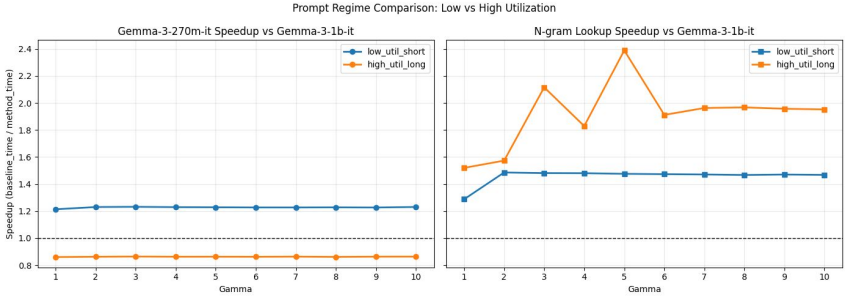
A100



L4



G4 VM



Q14

(Choose two) After using speculative generation in the **Benchmark with Two Prompt Regimes** experiment, if the speedup is slower than standard autoregressive generation under some parameter settings on a T4 GPU, what are some possible reasons?

A. During inference on the T4 GPU, the step where the small assistant model predicts gamma tokens has already fully saturated GPU utilization and reached the computational bottleneck (This can be observed in the terminal using “sudo apt install nvtop” + “nvtop”)
B. When the target model verifies multiple tokens, it may still be slower than autoregressively generating one token at a time, meaning the computation cannot be fully parallelized.
C. The assistant model being used has a number of parameters too close to that of the target model, so the time for the assistant model to autoregressively generate gamma tokens cannot be ignored.
D. Because the speculative generation algorithm is more complex than autoregressive generation, steps such as “randomly deciding whether to accept the tokens generated by the assistant model” and “using the target model and the assistant model to correct the probability distribution of the final returned token” may cause the CPU to spend more time computing than the GPU

Q15

(Choose one) If the sequence length (N) is 1024, when running the **FlashAttention** experiment, which of the following is closest to the number of floating-point values read from HBM?

A. 800,000
B. 1,000,000
C. 900,000
D. 700,000

Q16

(Choose one) If the sequence length (N) is 2048, compared with **Standard Attention**, which of the following is closest to the theoretical speedup of **FlashAttention** on a T4 GPU?

A. 1.835511
B. 1.899839
C. 1.934696
D. 1.952866

Q17

(1) Please take a screenshot of the results of the **KV Cache Multi-turn Test** experiment (0.25 pt)
(2) **(Choose one)** In the 234th-round question, which part of the time savings explains the speedup brought by the KV cache? (0.25 pt)

A. It saves the transfer time for loading the model weights from the CPU into the GPU VRAM
B. It saves the computation time required to predict each new token during the decode phase
C. It saves GPU memory bandwidth because KV Cache increases the physical read/write speed of VRAM
D. It saves the time spent in the prefill phase on matrix multiplications through the Attention and FFN layers for the shared prefix

Q18

(1) Please take a screenshot of the results of the **KV Cache Invalidation** experiment (0.25 pt)
(2) **(Choose one)** Which of the following explains why the previously computed KV cache becomes invalid after adding one space to the prefix? (0.25 pt)

A. The space character is automatically ignored by the tokenizer, so the prefix does not actually change; the KV cache invalidation is simply caused by the system randomly recomputing it
B. The space changes the hash value of the prefix string, causing the system's semantic cache matching to fail and making it treat the request as a completely new one
C. Because the features and positions of the preceding context change, the Key and Value matrices produced by subsequent tokens during attention computation become completely different from those in the original cache
D. Under the PagedAttention mechanism, the additional token causes an internal fragmentation overflow in a memory block, which triggers the system to forcibly clear and recompute that part of the cache

Q19

(1) Please take screenshots of two different CPU offload configurations from the **vLLM CPU RAM/Speed Trade-off** experiment (0.25 pt)
(2) **(Choose one)** Why does throughput become slower when model weights are moved to the CPU? (0.25 pt)

A. Because every time a new token is generated, the forward pass needs to transfer the offloaded weights again from the CPU
B. Because the weights need to be converted from FP16 to FP32 on the fly during the transfer process
C. Because the CPU cannot handle concurrent multi-user requests, causing requests to be queued
D. Because transferring the weights overwrites the KV cache already stored on the GPU

Q20