ChatGPT

vLLM 參數對 GPU 記憶體使用量的關係與估算

大型語言模型（LLM）的推理服務主要消耗兩部分 GPU 記憶體：**模型權重**（Model Weights）與**注意力 KV 快取**（Key-Value Cache）。vLLM 利用 PagedAttention 技術來高效管理 KV 快取，使 GPU 記憶體的利用率接近最佳（浪費低於 4%）。以下我們針對每個參數說明其對 GPU 顯存配置和使用量的影響，並提供相關公式與估算方法。

模型權重大小與 KV 快取概要

在討論各參數前，先理解 **模型權重** 和 **KV 快取** 對記憶體的影響：

- **模型權重大小**：模型的參數會佔用固定的顯存。佔用量與參數數量及其資料型態有關，計算公式：`模型權重記憶體 = 參數個數 × 每個參數位元組數`。例如 FP16/半精度參數每個佔2位元組，7B模型約有70億參數，僅權重就約佔用14GB顯存；13B模型權重約26GB；34B模型（如CodeLlama-34B）權重可能高達~68GB（需多GPU或低精度才能載入）。

- **KV 快取大小**：KV快取存儲每個注意力層的Key與Value張量，用於加速後續token生成。**每個token**會在每一層產生一個Key和一個Value向量，合計大小可用公式表示：

$$\text{KV快取/每token} = 2 \times \text{num\_layers} \times \text{hidden\_size} \times \text{每元素位元組數},$$

等價於

$$2 \times \text{注意力頭數} \times \text{head\_dim} \times \text{num\_layers} \times \text{每元素位元組數}【35\dagger L293 - L301】。$$

例如LLaMA-7B模型（32層、hidden約4096、FP16）每個token約佔用0.5MB快取；LLaMA-13B單個序列長度達2048時，其KV快取高達約1.7GB [1]；更大的模型（如70B）每token佔用更大（約2.5MB/token）。**KV快取總大小**近似為上述每token大小乘以序列長度，再乘以同時並發的序列數。這意味著序列越長或並發序列越多，KV快取耗費的GPU記憶體線性增長。實務上，KV快取對顯存的消耗甚至可能超過模型權重本身。

- **Prefill階段與Decode階段**：在推理時，**Prefill階段**載入並處理輸入提示（prompt）的所有token，將其Key/Value填入快取；**Decode階段**則每次產生一個新token。Prefill通常一次處理大量token，因此**瞬時需要的運算暫存記憶體**較多，但隨後會釋放，只留下KV快取常駐於顯存。Decode階段逐token進行，運算中間佔用較少，但會逐步增加KV快取（每生成一個新token就增加一份其KV）。因此：
- **Prefill**結束時，GPU已消耗記憶體來存儲所有提示token的KV快取，同時也完成了相當的計算。
- **Decode**期間主要的額外記憶體消耗是緩慢增長的KV快取，以及少量即時運算緩衝；相對Prefill，Decode對瞬時記憶體的壓力較小，但GPU利用率也較低。

理解上述背景後，我們說明各參數如何影響模型權重和KV快取在GPU上的配置。

`--gpu-memory-utilization`（GPU記憶體利用率）

作用：此參數設定vLLM引擎可用的GPU記憶體比例（0～1）。預設為0.9，表示最多使用90%的顯存供模型推理之用。

對記憶體的影響：vLLM會在啟動時**預先分配**一塊GPU記憶體作為快取空間，其大小約為「總GPU記憶體 × 設定比例」，扣除模型權重和其他開銷後的剩餘部分。換言之：

- 可用於KV快取的顯存 ≈ `總GPU記憶體 × gpu_memory_utilization − (模型權重 + 非Torch預留 + 運算高峰暫存)`。

例如40GB GPU、利用率0.9、模型權重15GB，假設其他開銷1.4GB，則預留給KV快取約=40×0.9−(15+1.4)=19.6GB。

- **提高**此比例將讓vLLM預留更多顯存給KV快取，可支持更多序列或更長上下文（減少溢出到CPU的機率）。反之，**降低**比例（例如0.5）會限制可用顯存，只使用一半空間給引擎，KV快取空間減少，並發/長序列將更快用滿GPU而需等待或換出。

- 若模型權重本身體積接近GPU上限，此參數實際能給KV的空間就很有限。例如在24GB卡上跑13B模型（約6.8GB AWQ量化），gpu_memory_utilization=0.9理論上可用21.6GB，但模型+開銷可能用掉~10GB，只剩約12GB給KV。比例過低時甚至**模型都載入不完**，因此一般保持預設0.9以充分利用顯存。

注意：vLLM透過**記憶體剖析**來決定能在該比例下分配多少「GPU頁塊」給KV快取。若設定的比例太小，導致最大可存儲token數低於模型的 `max_model_len`，會在初始化時報錯要求調整。例如某8B模型config預設支持128k長上下文，但在24GB卡上僅能給KV快取分配約40k token容量，就會出現錯誤提示「模型max seq len大於KV快取可存token數」，建議**提高gpu_memory_utilization或降低max_model_len**。

總之，`gpu_memory_utilization`決定了**引擎總共能用多少顯存**，直接影響**KV快取的容量**。增加它能讓GPU裝下更多token的KV快取，但也應預留部分空間避免OOM（所以默認不到100%）。典型地0.9是平衡值，極端情況下可設1.0（追求極限性能但風險OOM）或更低（多卡情況下可能降低以避免單卡過滿）。

`--max-model-len`（模型最大上下文長度）

作用：設定模型的**最大序列長度**（上下文長度）上限。如果不設，vLLM會讀取模型配置的預設值。例如Llama2 7B/13B預設2048，許多新版模型可能支持更長上下文。

對記憶體的影響：這個參數決定**每個序列**可能佔用的**KV快取最大容量**。因為快取使用量與序列token數**線性相關**：

- **每個序列的KV快取上限** ≈ `每token快取大小 × max_model_len`。舉例來說，若模型每token快取~0.5MB（7B模型FP16情況），上下文上限2048，則單序列KV最多約0.5MB×2048≈1024MB（約1GB）。將max_model_len加倍到4096，則單序列KV可能達2GB。13B模型每token快取約0.8MB [1]，2048長度序列需要~1.7GB顯存；若允許長上下文如4096，那單序列快取最高可到3.4GB以上。

- **vLLM配置的GPU KV容量**必須>= `max_model_len × 每層每token大小 × 序列數`的一部分，否則在初始化時就會檢查出問題並報錯。上例中，在24GB GPU上跑128k上下文8B模型時，由於128k上下文對KV需求遠超可用顯存上限，vLLM直接在啟動時提示減小max_model_len或增加顯存配額。**因此，max_model_len應根據實際GPU快取容量進行設定**，避免設定一個GPU無法支持的過高值。

- **減小**max_model_len可以**節省顯存**：因為即使有fragmentation管理，vLLM仍會按此上限來規劃需要的GPU快取頁塊數。如果實際用不到那麼長的上下文，降低上限可以釋放部分快取空間，允許更多序列並發或減少CPU換出。反之**提高**max_model_len需要有足夠顯存支持，否則需提高gpu_memory_utilization或借助CPU swap。

簡而言之，`max_model_len`決定**單一序列KV快取的最大潛在佔用**。較大的上下文長度需要按比例更多的KV空間。不同模型預設的max_model_len各異，但**切忌隨意設過高**：要根據模型規格和可用顯存調整此參數，以免快取不足。

`--max-num-seq`（每次迭代最大序列數）

作用：設定vLLM在**單次推理迭代**中**最多處理的序列（請求）數**。預設值為256。這可視為批次中的最大序列數上限。

對記憶體的影響：
- **總KV快取用量**：在某一時刻，同時存在的**活動序列**越多，累計的KV快取佔用越大。理論上，同步並發的序列數乘以平均每序列已生成token數，就近似決定了已用的KV快取塊數。提高max-num-seq上限意味著允許更多序列平行生成，**如果所有這些序列都很長，總KV快取可能會吃光GPU空間**。當快取空間用滿時，新的請求將被掛起等待，或部分快取被換出到CPU（若允許swap）。因此max-num-seq雖不是直接分配顯存，但**實際可同時支持的序列數受限於預留的KV空間**。

  - **批次處理**：vLLM的**連續批處理**機制允許將多個請求合併在一次前向傳播中處理，以提高吞吐。max-num-seq設定了一次批處理中最多能合併的請求數。**較大的值**（如256）可以在請求量大時充分利用GPU算力，但如果GPU記憶體不足，實際上vLLM可能自動降低每批處理的序列數（剩下的排隊等待下一批），以避免超出KV容量[2]。**較小的值**會限制每批處理的規模，可能導致GPU算力未充分利用，但也減少單批所需的臨時記憶體。

  - **極端情況**：如果max-num-seq設置遠超GPU可容納的並發數，實際效果是多數請求被延後。比如在有限KV空間下，或許只能同時執行幾十條序列，其餘雖然沒有超過max-num-seq但也無法同批，只能等待。因此**這個參數更多是控制批次大小上限，而非保證真的能同時跑那麼多序列**。實務中保持預設256即可，除非有特殊需求調小避免單批過大延遲。

總結：`max-num-seq`設定**單次計算中**能並行處理的請求數上限。它影響GPU在一個iteration內暫存計算的負載，但**對最終KV佔用的峰值主要取決於真正的並發需求和快取大小**。合理的並發極限應由可用KV空間計算得到，例如：若每序列KV平均佔用M GiB，GPU預留N GiB給快取，則大約支持N/M個序列同時活躍，超過則排隊。max-num-seq應不低於這個可支援並發數，以免限制吞吐；同時也沒必要遠高於實際可支援並發。

`--max-num-batched-tokens`（每次迭代最大累計 tokens 數）

作用：限制**單次迭代（一次前向計算）中**處理的**總token數**上限。這包括了所有批內序列的新輸入token數之和。此參數直接影響**Prefill階段**的**chunk**大小，以及**Decode階段**每批能合併多少序列的產生。

對記憶體和效能的影響：

  - **運算中間佔用**：單次計算中token總數越多，需要的**暫存運算記憶體**就越大。例如前向傳播時，注意力和FFN層的中間激活需為這些token分配空間。**較大的max_num_batched_tokens**允許批次涵蓋更多token，增大一次計算的佔用峰值。反之，**減小**此值會**降低單批運算的記憶體峰值**，因為拆分成更小的塊分次處理。vLLM在啟動profiling時，就是用max_num_batched_tokens來模擬最大批處理情況測量峰值激活記憶體。

  - **Prefill階段**：對於長提示的請求，vLLM可以將其拆成多個chunk來依序填充KV快取，稱為**Chunked Prefill**。max_num_batched_tokens 就是決定chunk大小的門檻。例如預設值常為512（在A100上測得較佳）。如果一個prompt有1000個token，vLLM會先處理512，再處理剩下的488，而不會一次性處

理1000。這樣做減少了**單次計算壓力**，也**允許在處理長prompt時順帶混合一些decode請求**（稱為**decode-maximal batching**）：當Prefill chunk還有空餘容量時，可以同時放入一些其他序列的decode token，一起批處理。總體上，**chunked prefill**與**小批解碼**讓GPU在Prefill和Decode階段的負載更平衡，提升吞吐並降低記憶體峰值。

- **Decode階段**：通常每個sequence在decode時一次只產生1個新token。如果只有decode請求，max_num_batched_tokens主要受**max-num-seq**限制（如256 seq就最多256 tokens）。但當同時有Prefill進行時，此參數確保**Prefill+Decode合計**不超過閾值。較低的max_num_batched_tokens可**優先滿足Decode**（小批）而將長Prompt拆段，提升首次輸出token延遲（TTFT）表現。

權衡：設置此值需要考慮**GPU型號**和**服務場景**： - **較大值**（趨近於max_model_len）意味著不切chunk，prefill一次完成，可能提升單一長請求的速度，但會佔用大量顯存暫存，減少並發能力。 - **較小值**會導致長請求拆分多次前向，稍微增加該請求的延遲，但能**讓Decode請求更及時地插入執行**，提升整體並發吞吐，并降低單次內存需求。

一般建議使用官方推薦值（如512）或根據硬體試驗調整。總之，`max-num-batched-tokens`透過限制**每批token總量**來**控制運算時的記憶體峰值**，進而影響KV快取與運算資源的分配。合理設置能在**不OOM**的前提下最大化GPU利用率。

`--block-size`（KV快取區塊大小）

作用：設定KV快取記憶體的**分頁大小**（每個區塊包含的token數）。可選值一般為8、16、32，預設為16。PagedAttention將每個序列的KV快取劃分為固定長度的連續區塊，透過**非連續內存存儲**來靈活管理 [3] 。

對記憶體的影響：

- **碎片浪費**：區塊化管理使每個序列**只有最後一個區塊可能未填滿**，造成少量浪費。**較小的block-size**意味著每塊容納的token更少，因此最後一塊的浪費上限更低。例如block-size=16時，每序列最多浪費15個token的空間；block-size=8時最多浪費7個token空間。實驗顯示，默認16已使浪費低於4%；若block-size進一步降為8，理論碎片更少，但**管理開銷**（塊表映射與查找）稍增，**效能可能略有影響**。因此16通常是折衷值。

- **快取彈性**：無論block-size大小，PagedAttention都允許**按需分配**物理塊。新token生成時再分配新塊，不必預先為每條序列分配完整max_model_len大小的連續空間 [4] 。這極大提高了記憶體利用率，能動態騰出未使用的塊給其他序列。block-size只是決定了**分配單元**的粒度，小粒度使未用容量更細碎但更少浪費，大粒度則相反。

- **共享和複用**：PagedAttention還支援**前綴快取共享**（Prefix Sharing），對於從相同提示生成多個輸出的情況，可以讓多個序列邏輯上引用相同的物理塊。block-size不直接改變此機制，但**較小塊**可能有利於更靈活地共享部分前綴（因為前綴長度不一定是大塊的整數倍）。總體而言，block-size對**多序列共享**的影響較小，主要影響碎片率。

建議：大多數情況使用預設值16即可，因為它已經提供了接近最佳的記憶體利用率。只有在極端記憶體緊張並且序列長度分布很特殊的情況下，才考慮調整此值。例如序列長度普遍很短時，減小block-size可能進一步降低浪費，但收益不大。需要注意的是，block-size變小**可能稍微降低GPU計算效率**，因為每次注意力計算需要處理的塊更多（儘管PagedAttention Kernel已優化多塊查找 [5] ）。綜上，`block-size`透過調整**KV快取塊的粒度**影響**記憶體碎片和分配靈活性**，一般無需修改。

`--swap-space`（CPU換出空間）

作用：為每張GPU指定可用的**CPU記憶體交換空間**大小（GiB）。預設為4 GiB。這相當於vLLM允許在GPU顯存不足時，最多使用多少CPU記憶體來暫存溢出的KV快取區塊。

對記憶體的影響：

- **KV快取溢出處理**：當累計的KV快取需求超過了`gpu_memory_utilization`分配的GPU快取空間時，PagedAttention會啟用**虛擬內存機制**：將部分不常用的KV快取「頁塊」移動到CPU RAM（swap-space）儲存。這類似於作業系統的分頁交換，騰出GPU空間給新的token生成。當這些被換出的塊再次需要參與注意力計算時，再從CPU調回GPU。

- **swap-space大小**決定最多能換出多少KV數據到CPU。假設預設4GB，那最多可在CPU存放約4GB的KV快取（例如13B模型約可存放~2個長序列的快取）。如果將該值設更大，如8或16GB，那當並發序列非常多或上下文極長時，GPU滿了可以繼續將更多KV搬移到CPU而不中斷請求。**代價**是訪問這些換出的KV時會有**較大延遲**，因CPU和PCIe頻寬遠低於GPU顯存。同時，太多的swap也可能導致頻繁的記憶體換入換出，降低吞吐。

- **禁用swap**：將swap-space設為0則**不使用CPU交換**。這意味著一旦GPU快取滿了，新請求就只能等待（TTFT大幅增加），或者vLLM可能報錯無法繼續生成。禁用swap可以避免CPU/GPU之間的資料搬移開銷，但要求你**嚴格限制並發數或上下文長度**以適應固定的GPU快取。一般只有在低延遲極為敏感且能保證不溢出的情況才禁用。

- **與gpu_memory_utilization關係**：gpu_memory_utilization決定了**GPU上**可用快取空間；swap-space則是在**超出**GPU部分額外提供的**緩衝**。理想情況下，大多數熱點序列的快取仍留在GPU中，只有長時間不被關注的序列塊被換出。swap-space並不是用多少就全占滿，而是按需使用，最多不超過設定值。增加swap-space不會影響GPU端已經分配的空間，但會允許**更高的峰值快取需求被滿足（透過CPU）**，代價是可能降低部分請求的速度。

小結：`swap-space`提供了**擴展的KV快取容量**，保障在**高並發或超長上下文**時不至於因GPU記憶體不足而失敗。它相當於在GPU記憶體之上再疊加的一層「虛擬快取」。實務上，如果你的應用需要支持非常多的同時對話或極長的輸入，建議調高swap-space（如8～16GB甚至更多，取決於主機RAM），以免超限。但同時應監控延遲，確保換出不影響關鍵請求的響應時間。

模型大小差異對記憶體的影響

不同規模的模型在權重大小和KV快取佔用方面差異顯著：

- **7B 模型**（如LLaMA2-7B）：約32層、隱層4096。FP16權重約佔14～15GB顯存。每個token的KV快取約0.5MB；2048上下文則約1GB快取。這通常可在24GB GPU上同時容納數個長序列（例如19.6GB快取空間可支援約20k tokens的總容量）。

- **13B 模型**（LLaMA2-13B）：約40層、隱層5120。FP16權重約26～27GB。每token KV快取~0.8MB，單序列2048時佔用約1.7GB顯存 [1]。若GPU是A100 40GB，可容納權重+幾個序列的快取；在24GB卡上通常需量化或啟用CPU swap才能支持長上下文或較多並發。**KV快取內存消耗**已接近甚至超過模型權重（1.7GB vs 權重26GB，10個滿長度序列就等同一個模型大小）。

- **34B 級模型**（如CodeLlama-34B、Bloomz-30B等）：通常60層左右、更大隱層（6k+）。FP16權重可能超過60GB，**無法單卡完整載入**，需多GPU平行或使用4/8-bit量化。KV快取每token可能~1.3MB甚至更多，2048長度序列快取約可達2.5～3GB。這意味著即使權重透過量化裝入單卡，KV快取也很容易吃滿24GB顯存。vLLM在服務此類模型時，**更依賴swap或多GPU張分擔**。例如一個34B模型在24GB GPU上量化後權重約16GB，那剩下的8GB顯存最多只裝下2～3個滿長度序列的KV，再多就需要換出。

- **更大模型**（如70B、176B）：70B模型每token快取約2.5MB（80層、隱層8192），2048長度快取高達5GB+；176B模型甚至達4MB/token，基本無法在單卡完全保留長序列快取。這類模型幾乎必須多卡部署，並充分利用vLLM的分塊管理和swap機制，把部分KV儲存在其它GPU或CPU。

總結：模型參數數量增加時，**權重顯存**線性增加，而**每token的KV快取**也隨著層數和隱藏維度增加而增大。因此大模型對顯存的壓力**雙重增加**：不但載入權重需要多GPU或壓縮，服務過程中每一個額外token帶來的快取開銷也遠高於小模型。vLLM透過提高gpu_memory_utilization、使用更小的塊、以及啟用swap等手段來應對，但使用者在部署不同大小模型時仍需**權衡**：例如，小模型可以不啟用swap就支持較多並發，大模型則幾乎必須依賴swap或降低同時序列數。也可考慮**降低KV快取精度**（如FP8快取）減半KV所需空間，以支援更多token（這是vLLM提供的選項）。

內存估算規則與公式匯總

綜合以上，可用以下方式粗略估算vLLM部署時的GPU記憶體需求：

1. **模型權重顯存**：

$$\text{權重內存} = \text{參數總數} \times \text{每參數位元組數}$$

例如7B FP16約=7×10^9×2B≈14GB；若使用4-bit量化則約0.5×FP16大小（7GB左右）。

2. **每序列KV快取**（假設使用FP16快取）：

$$\text{每序列KV內存} = 2 \times \text{num\_layers} \times \text{hidden\_size} \times 2\ \text{bytes} \times L$$

（L為該序列當前token數）。可改寫為`2 * 注意力頭數 * head_dim * 層數 * 2B * L`。例如13B模型40層隱層5120，L=2048時約1.7GB [1]。

3. **總KV快取內存**：將每序列KV加總。對於同批並發的N個序列且各長度相近，可近似

$$N \times \text{每序列KV（取平均長度）}。$$

例：每序列需要200MB快取的情況下，10個並發序列約需2GB快取。

4. **GPU快取容量**（由gpu_memory_utilization決定）：

$$\text{可用KV空間} = \text{GPU總內存} \times \text{utilization} - (\text{模型權重} + \text{其他預留})$$

。如上例40GB GPU、util=0.9、權重15GB、預留1.4GB，則KV空間≈19.6GB。

5. **並發上限估計**：在KV空間固定的情況下，最大並發數約為：

$$\text{max\_concurrent\_seqs} \approx \frac{\text{可用KV空間}}{\text{單序列平均KV佔用}}$$

。超過此數後，新的請求將等待（或需啟用swap）。例如可用KV=19.6GB，每序列佔用200MB，則可同時支撐約98個此類序列。

6. **調整參數對內存的影響**：

7. 增加 `gpu_memory_utilization` 提高可用KV空間。
8. 降低 `max_model_len` 減少單序列KV最大需求。
9. 降低 `max_num_seqs` 限制並發數，實際內存用量上限降低（但也降低吞吐）。
10. 降低 `max_num_batched_tokens` 減少單次計算峰值內存，讓profiling結果更多內存留給KV。
11. 減小 `block-size` 微幅降低碎片浪費（提升幾個百分點內存利用率）。
12. 增加 `swap-space` 在超出GPU時提供額外容量，理論上可以支持的token總量 = GPU KV容量 + swap容量。

透過以上公式和規則，部署者可以大致估算不同模型和配置下所需的GPU記憶體。例如，若要在單卡24GB上跑13B模型且支援8並發長對話，每個對話上下文假定平均1000　token，則： - 權重（FP16）≈26GB（超過24GB，需8bit量化降至~13GB）。 - 每對話KV快取≈1000×0.8MB=800MB，8個約6.4GB。 - 模型+KV總需求≈13GB+6.4GB=19.4GB，符合在24GB卡上（util=0.9時約21.6GB可用）運行，餘量留給暫存和碎片。 - 若並發提高到12個類似對話，KV需求~9.6GB，模型+KV=22.6GB，已接近上限，建議提高swap-space以避免溢出。

上述估算僅為粗略指引。實際中還需考慮**暫存高峰**（尤其大模型大batch時）、**碎片頭尾**、以及**模型配置差異**（如不同模型層數、head維度等）。不過，透過理解這些參數與GPU記憶體的關係，我們可以更有效地調整vLLM部署，使模型權重、KV快取的分配達到最佳平衡，在**不OOM**的前提下充分利用每一分GPU顯存資源。上述分析與估算公式均基於vLLM官方文件與論文中的原理 [3] 以及開發人員提供的經驗數據。希望這些說明有助於您優化vLLM的部署配置。

參考資料：

- vLLM 官方文件與部落格對參數的定義與說明 [6]
- 《Efficient Memory Management for LLM Serving with PagedAttention》 論文與部落格 [3]
- vLLM 作者在GitHub Issue中的說明與使用者經驗
- 相關部落格文章對KV快取大小和內存估算的分析
- 以上引用片段標註於文中。

---

[1] [3] [4] [5] [6] vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention | vLLM Blog
https://blog.vllm.ai/2023/06/20/vllm.html

[2] Error gpu memory utilization with awq model when tp>1. · Issue #1472 · vllm-project/vllm · GitHub
https://github.com/vllm-project/vllm/issues/1472