KV Cache、Speculative Decoding 與 Early Exit

討論筆記 • 2025

背景：Autoregressive 生成與 KV Cache

Autoregressive 語言模型每次生成一個新 token，下一個 token 的機率分布取決於所有歷史 token。

因果遮罩（Causal Mask）： 為維持 autoregressive 的因果性，token `i` 只能 attend 到位置 `1` 到 `i`，不能看到未來 token。此約束以下三角遮罩實作——attention 矩陣中上三角的位置被 mask 為 `-∞`，softmax 後權重歸零。

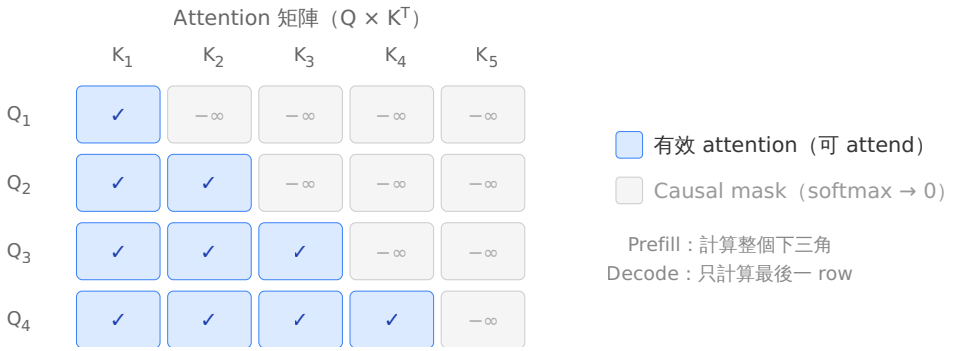


圖 1：Causal Mask——下三角為有效 attention，上三角遮罩為 −∞

KV Cache 是計算的產物，而非查表： KV Cache 中的 `K`、`V` 並非從 input token embedding 直接查得，而是每一層 attention 計算後的輸出。具體而言，第 `l` 層的：

$$K_i^{(l)} = W_K^{(l)} \cdot h_i^{(l-1)}, \quad V_i^{(l)} = W_V^{(l)} \cdot h_i^{(l-1)}$$

其中 $h_i^{(l-1)}$ 是第 `l-1` 層對 token `i` 的 hidden state output，而非原始 embedding。因此，若沒有逐層計算各 token 的 hidden state，就無法得到 KV Cache。

由此產生兩個截然不同的計算階段：

	Prefill	Decode
輸入	整段 prompt（m tokens）	單一新 token
Attention 形狀	m × m 下三角矩陣	1 × n 單一 row
目的	建立初始各層 KV Cache	逐步生成新 token
效能瓶頸	Compute-bound	Memory-bound

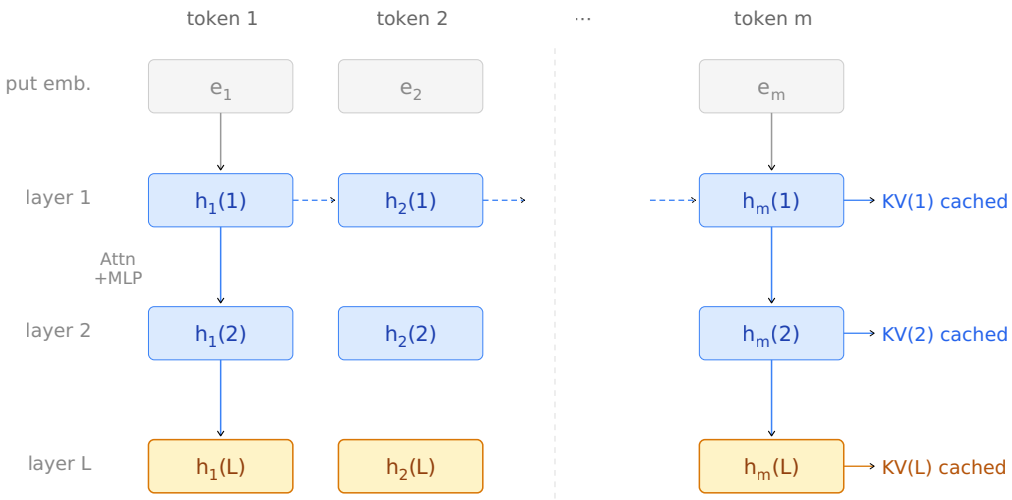
§ 1 Prefill 為何需要計算完整下三角？

「我有 m 個 profiling tokens，生成第 m+1 個 token 時，是否只要算第 m 個 token 的 attention row 就好？」

常見誤解：既然已知全部 m 個 token 的 embedding，只需計算最後一個 token 的 attention row，就能產生第 m+1 個 token。

此誤解的根源在於混淆了「input embedding」與「各層 hidden state」。KV Cache 需要的 `K`、`V` 是各層 hidden state 的線性投影，而非原始 embedding 的查表。

核心前提：KV Cache 是 per-layer 的。
第 `l` 層的 $K_i^{(l)}$、$V_i^{(l)}$ 依賴第 `l` 層的輸入 $h_i^{(l-1)}$，而 $h_i^{(l-1)}$ 本身是第 `l-1` 層對**所有前序 token** 跑完 Attention + MLP 後的輸出。



每層需要所有 token 的 $h_i(l-1)$，才能計算該層的 KV Cache

圖 2：Hidden state 逐層傳遞——KV Cache 是每層 Attn+MLP 計算後的產物

為何不能只算最後一個 row？

假設只計算 token m 的 attention row：
・第 1 層的 $h_m^{(1)}$ 需要 $h_1^{(0)}$ … $h_m^{(0)}$（input embedding）作為 K、V 來源 → 可行
・第 2 層的 $h_m^{(2)}$ 需要 $h_1^{(1)}$ … $h_m^{(1)}$ 作為 K、V → 但這些只有在第 1 層算完所有 token 後才存在
・以此類推，每層都必須先算完所有 token 的 hidden state，才能進入下一層。

結論： KV Cache 並非 input embedding 的查表，而是各層 Attention + MLP 計算的產物。Prefill 必須讓資訊逐層流過所有 token，計算完整的 m × m 下三角 attention 矩陣，才能建立完整的 per-layer KV Cache，供後續 decode 使用。

---

§ 2 Speculative Decoding 的驗證機制

「小模型算出的 draft tokens，大模型如何對答案？」

Speculative Decoding 使用輕量的 **Draft Model (DM)** 快速生成 γ 個 draft tokens， 再由 **Target Model (TM)** 一次 prefill 並行驗證全部 draft tokens。

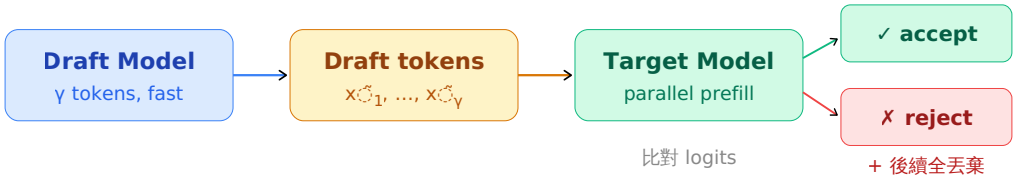


圖 3：Speculative Decoding 驗證流程

> **驗證細節：** TM 將 draft tokens 一次 prefill，取得每個位置的 logits，與 DM 的輸出比對。 一旦某個 draft token 被拒絕，**其後所有 draft tokens 全部丟棄**，並由 TM 的 logits 重新取樣一個正確 token。

Challenge — Draft model 選擇困難： DM 必須在「輸出品質夠接近 TM（acceptance rate 高）」與「速度夠快（降低 latency）」之間取得平衡。 DM 太弱則 draft tokens 頻繁被拒、加速效果消失；DM 太強則本身就慢，失去使用 speculative decoding 的意義。

---

§ 3 Layer-level Early Exit

「Early exit 後，KV Cache 如何處理？」

Layer-level Early Exit 讓模型在中間層提前離開——若某層的輸出已達到足夠信心（confidence threshold）， 則跳過後續所有層，直接輸出預測結果，節省計算。

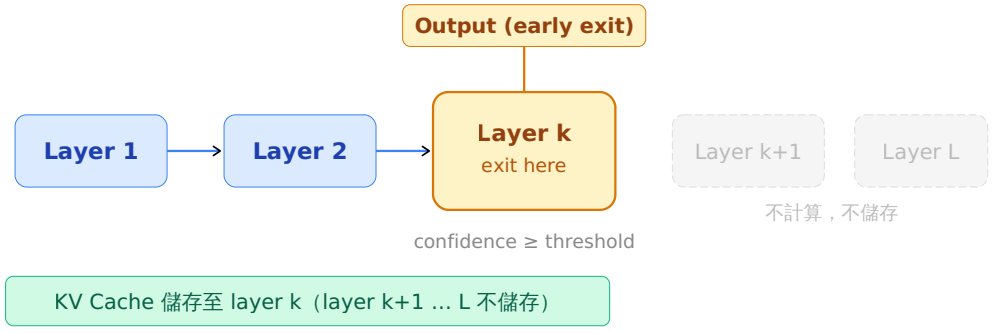


圖 4：Layer-level Early Exit 與 KV Cache 儲存範圍

> **KV Cache 處理策略：** Early exit 後，僅儲存第 1 層到第 `k` 層的 KV Cache。 若後續需要使用完整 KV（例如任務複雜度提升），再以 **parallel prefill** 補算剩餘層的 KV，而非逐步重算。

Challenge — Threshold 設定困難： Confidence threshold 需要在「提早退出節省計算」與「輸出品質下降」之間取得平衡。 Threshold 過低則品質受損；過高則幾乎不觸發 early exit，

效益消失。 此外，不同任務、不同 token 位置的最佳 threshold 可能差異顯著，靜態設定難以兼顧。