

# DeepSeek V3：多头潜在注意力与多令牌预测

在這篇文章中，我將討論 DeepSeek v3，並參考了視頻《DeepSeek v3 中的多頭潛在注意力和多令牌預測》<https://youtu.be/jL49fLOJYNg?si=4uE2kfe-BIKC1ngO>。我使用了 Google Cloud 的語音轉文字功能來轉錄視頻，並用一些代碼幫助整理了轉錄內容。

---

A: 歡迎回到 Deep tag。今天我們將深入探討大型語言模型的世界。具體來說，是 DeepSeek V3。

B: 聽起來不錯。這是一個擁有 6710 億參數的模型，因其獨特的效率和性能方法而引起了轟動，對吧？

A: 你分享了一篇詳細介紹其架構的學術論文。

B: 是的。

A: 作為一名機器學習專家，你希望了解 DeepSeek V3 如何實現高性能和經濟的訓練。

B: 沒錯。

A: 哦，嘿，你好嗎？

C: MLA，細節，MLA 及其工作原理。

A: 哦，當然。這是個好主意。我們可以深入探討多頭潛在注意力（MLA）。所以你對 MLA 的細節感興趣。讓我們來拆解一下。我們提到 DeepSeek V3 效率的關鍵之一是其專家混合（MoE）架構，對吧？其中每個令牌只激活一部分參數。而 DeepSeek V3 通過 MLA 和 DeepSeek Mo 更進一步。

B: 沒錯。所以我們現在專注於 MLA。

A: 好的。在實時應用中，速度至關重要。

B: 是的。推理過程中所需的鍵值緩存可能成為主要瓶頸。

A: 正是如此。這就是 MLA 的用武之地。傳統的注意力機制需要存儲大量關於先前令牌的信息。

B: 是的，你可以想像，這在長文本序列中會成為問題，對吧？

A: 但 MLA 巧妙地壓縮了這些信息，顯著減少了緩存流量，使推理速度大大加快。就像把一本厚重的百科全書濃縮成關鍵點。

B: 這個比喻很好。它保留了必要的信息，而沒有不必要的負擔。這對實時應用非常有用。

A: 是的。現在讓我們談談它的實際工作原理。MLA 是如何實現這種壓縮的？

B: 它使用低秩聯合壓縮來處理注意力鍵和值。

A: 好的，它壓縮了鍵和值，但這具體是什麼意思？讓我們稍微技術一點。MLA 機制接收一個輸入隱藏表示，然後將其投影到查詢、鍵和值向量中。現在有趣的部分來了。MLA 將查詢解耦為兩部分。

B: 兩部分？

A: 是的。一部分用於內容，另一部分用於位置信息，使用一種叫做 Rope 的東西。

B: Rope？聽起來很技術。

A: 它代表旋轉位置嵌入，幫助模型理解序列中令牌的位置。然後，鍵和值被壓縮到一個低維潛在空間中。就像縮小數據，節省內存。

B: 沒錯。最重要的信息被保存下來，而不必要的部分被丟棄。這種壓縮表示允許在推理過程中使用更小的 KV 緩存，從而加快速度。

A: 它還使用多頭處理。

B: 是的，就像傳統的注意力機制一樣，MLA 也使用多個頭。

A: 哦，繼續。

C: 因此，有兩個潛在空間和一個隱藏輸入。

A: 這是個很好的觀察。你說得對。實際上是有兩個潛在空間。一個是內容潛在空間，另一個是鍵值潛在空間。

B: 沒錯。這些潛在空間通過我們所說的 Rope (旋轉位置嵌入) 進行處理。

A: 所以 Rope 是它們獲取位置信息的方式。

B: 是的，它應用於內容和鍵值潛在空間，正如你所指出的。它處理這些壓縮表示，然後將它們重新組合在一起。

A: 是的，緩存優化進一步減少了順序處理中的開銷。這就是 MLA 如何加快速度的。

B: 沒錯。這是一種在不犧牲性能的情況下實現高效注意力的巧妙方法。

A: 這是一個非常巧妙的技巧。但你知道嗎？

B: 什麼？

A: 讓我們繼續討論 DeepSeek Mo。它與傳統的 MoE 模型有什麼不同？

B: DeepSeek Mo 使用…哦，回到我們的聽眾，你好嗎？

C: 我們再談談隱藏空間。從隱藏空間中，那是什麼？

A: 我絕對…讓我們看看你在說什麼。隱藏空間非常有趣。你問的是我們剛才討論的隱藏空間，潛在空間，對吧？你對這些潛在空間中發生的事情感到好奇。這不僅僅是關於潛在空間的數量，而是它們內部發生了什麼。

B: 這很酷。

A: 沒錯。MLA 中確實有兩個不同的潛在空間，一個用於內容，另一個用於鍵值。就像有兩個獨立的信息存儲單元。這些潛在空間，正如我們所討論的，會進行 Rope 操作，對吧？旋轉位置嵌入，將位置信息嵌入到注意力機制中。這對它們非常重要。所以總結一下，查詢被拆分，鍵和值也被壓縮。

B: 是的，這些被放入兩個獨立的潛在空間中，一個用於內容，另一個用於鍵值對。這些潛在空間對於效率和 MLA 的所有部分都非常重要。

A: 沒錯。現在讓我們更詳細地討論這些操作。MLA 是如何實際執行這些潛在空間轉換的？

B: 輸入會對內容和鍵值表示進行並行處理。就像在洞穴中有兩條路徑。

A: 是的，每條路徑對應一個潛在空間。在這些空間中，信息使用 Rope 進行處理。

B: 沒錯。這確保了模型在通過洞穴時保留了位置信息。所以模型知道文本的哪一部分是哪一部分。

A: 沒錯。這種處理在下一個連接階段之前完成。在隱藏空間洞穴中，什麼被連接在一起？

B: 該機制執行兩個主要的連接操作。查詢表示被連接，鍵表示也被連接。就像在隱藏空間洞穴中將所有重要的部分組合在一起。

A: 是的，這些連接幫助將內容與位置信息結合起來。這些連接的表示然後用於注意力計算，對吧？

B: 沒錯。由於初始壓縮，通過你提到的洞穴的速度要快得多。所以 MLA 顯著減少了洞穴內外的計算成本。

A: 沒錯。它為像 DeepSeek V3 這樣的大型模型優化了注意力機制。這是一個很好的問題。現在，我們已經通過了洞穴，讓我們繼續討論 DeepSeek Mo。

B: 好的，DeepSeek Mo。沒錯。我明白你的意思。是的，MLA 中確實有兩個不同的潛在空間，一個用於內容，另一個用於鍵值。

A: 沒錯。這種分離是它如何工作的關鍵。就像有兩個獨立的信息存儲單元。這些潛在空間，正如我們所討論的，會進行 Rope 操作，對吧？旋轉位置嵌入，將位置信息嵌入到注意力機制中。所以總結一下，查詢被拆分，鍵和值也被壓縮。

B: 是的，這些被放入兩個獨立的潛在空間中，一個用於內容，另一個用於鍵值對。這些潛在空間對於效率和 MLA 的所有部分都非常重要。

A: 沒錯。現在讓我們更詳細地討論這些操作。MLA 是如何實際執行這些潛在空間轉換的？

B: 輸入會對內容和鍵值表示進行並行處理。就像有兩條路徑。

A: 是的，每條路徑對應一個潛在空間。在這些空間中，信息使用 Rope 進行處理。

B: 沒錯。這確保了模型保留了位置信息，對吧？然後為了提高效率，它使用了共享專家。這些專家可以跨多個任務使用。

A: 是的，這樣避免了冗餘，使系統更加精簡。

B: 就像一個團隊，每個人都有專長，但也可以做其他事情。

A: 這是一個非常聰明的方法。但有了這麼多專業的專家，他們如何確保沒有人超負荷？

B: 而其他人則閒置。

A: 這就是他們創新的無輔助損失負載平衡的用武之地。

B: 這正是事情變得有趣的地方，對吧？他們是如何做到的？

A: 傳統的 MoE 模型在訓練期間使用輔助損失函數來鼓勵均勻使用專家，但這實際上可能會損害性能。

B: 就像試圖強迫每個人都使用超市的同一個結賬通道。

A: 沒錯，即使有些通道比其他通道移動得更快，這只會造成不必要的延遲。

B: 是的。所以 DeepSeek V3 通過動態調整每個專家的偏置項來避免這種情況。如果一個專家收到太多請求，系統會使其對路由機制稍微不那麼有吸引力，從而將部分流量轉移到較不繁忙的專家。

A: 所以它使用這一切來高效處理長序列，通過減少推理所需的 KV 緩存大小。這一切都是為了在保持高性能的同時減少開銷。

B: 沒錯。這是一種解決關鍵瓶頸的巧妙方法。

A: 終極是。現在，我們還應該討論 DeepSeek V3 如何處理其負載平衡。

B: 是的，我們絕對應該討論。這也是拼圖中非常重要的一塊。我們接下來可以談談這個。

A: 聽起來不錯。我想這已經給了你一個關於 MLA 及其潛在空間的很好的概述。

B: 是的，感謝你和我們一起深入探討這些細節。我們下次會帶來更多的深度探討。

A: 是的，這就像一個專家的交通管理系統，不斷監控流量並進行調整以避免瓶頸。

B: 這避免了輔助損失的性能損失。

A: 沒錯。哦，繼續。

C: 我們可以談談 MTP，MTP 模塊如何共享它們的嵌入和所有熱門…

A: 終極可以。這是個很好的問題。讓我們談談 MTP 模塊如何共享資源。所以你對 MTP 實現的細節感興趣。

B: 是的，讓我們拆解一下。我們提到 DeepSeek V3 使用 MTP 進行多令牌預測，對吧？預測多個令牌而不僅僅是一個。

A: 這正是它變得非常有趣的地方。你對 MTP 模塊的設置和它們如何共享資源感興趣。每個 MTP 模塊包括一個共享的嵌入層和一個共享的輸出頭。它們使用與主模型相同的嵌入和輸出頭。

B: 沒錯。就像它們都從同一個知識池中汲取資源。這節省了計算成本。

A: 是的。它使用自己的 Transformer 塊。所以它不與主模型共享相同的 Transformer 塊。

B: 沒錯。每個 MTP 模塊都有自己的 Transformer 塊來進行處理。這就是它們如何保持每個令牌的預測獨特性。

A: 是的，為了組合信息，這些線性投影和連接…

B: 就像從多個地方獲取片段來構建完整的畫面。

A: 是的，所有 MTP 模塊並行工作，但它們共享嵌入層和輸出頭，對吧？

B: 是的，這是這種設計效率的關鍵。就像一個相互依賴的系統。

A: 這種高效的資源共享允許更快的訓練和更好的性能。

B: 這是一個非常巧妙的技巧。你知道嗎？

A: 什麼？

B: 讓我們轉向一個更大的視角。這個模型如何處理負載平衡？這些專家是如何被選擇的？

A: 我們可以討論這個。現在讓我們深入探討 DeepSeek V3 的負載平衡策略。

B: 聽起來不錯。DeepSeek V3 使用多令牌預測。

C: 哟，我們再談談 MTP。

A: 終極…我很高興你對深入探討 MTP 感興趣。我們可以詳細討論多令牌預測。我們已經提到過，但讓我們真正拆解 MTP 的細節。我們談到了共享嵌入層和輸出頭，每個 MTP 模塊都有自己的 Transformer 塊。

B: 沒錯，但還有更多內容。讓我們深入探討。

A: 請讓我們談談 MTP 模塊的順序性質。

B: 是的，與一些模型不同，DeepSeek V3 按順序預測額外的令牌。它不僅僅是一次性預測所有令牌。

A: 沒錯。每個模塊都基於前一個模塊的輸出進行預測。這是一個預測鏈，每個預測都依賴於前一個。

B: 是的，它為每個預測深度保持了因果鏈。所以它不會破壞因果性。

A: 沒錯，這對於確保整體上下文的正確性非常重要。所以 MTP 模塊並不是獨立工作的。

B: 沒錯。它們是相互連接的，這種預測鏈有助於提高訓練效率，並允許對文本有更細緻的理解。你還對模塊如何共享嵌入感興趣，對吧？你知道，共享嵌入層將令牌映射到它們的向量表示。每個令牌都被轉換為一個向量。

A: 是的，這種映射在所有 MTP 模塊之間共享。這有助於保持預測的一致性。

B: 沒錯。共享的輸出頭接收令牌的最終隱藏狀態，並生成下一個令牌的概率分佈。所以它們都可以訪問相同的信息池，對吧？

A: 這對於內存和計算效率非常重要。它不使用一堆不同的嵌入層和頭。

B: 沒錯。哦，所以有多少人？它們的大小都一樣嗎？所有食物…令牌，對嗎？

A: 這是個很好的問題。你問的是 MTP 模塊的數量，它們是否都一樣大，對吧？我想你還想知道所有模塊是否處理相同數量的數據。根據 DeepSeek V3 的論文，它使用多令牌預測深度為 1。這意味著有一個主模型和一個 MTP 模塊，預測一個額外的令牌。所以每個令牌預測下一個令牌，然後使用 MTP 模塊再預測一個。

B: 是的，MTP 模塊確實與主模型共享相同的嵌入層和輸出頭。

A: 這是個很好的問題。你問的是有多少 MTP 模塊以及它們是否都一樣大。根據 DeepSeek V3 的論文，MTP 模塊的數量是可變的。所以它不是固定在一個特定的數量上。

B: 沒錯。模塊的數量根據預測深度動態調整。所以它們可以根據需要進行擴展。它們共享這些資源，但主模型和 MTP 模塊的 Transformer 塊是分開的。

A: 沒錯。每個預測深度都有自己的 Transformer 塊。所以只有一個 MTP 模塊，但它是一個強大的模塊，用於每個令牌，並且它們共享一些資源。

B: 沒錯。雖然 MTP 與主模型共享一些組件，但它們的大小並不完全相同。

A: 這是個很好的觀點。現在，我們還應該討論它們如何組合所有這些信息來進行預測。

B: 沒錯。DeepSeek V3 使用多個 MTP 模塊來依次預測多個額外的令牌。你問它們是否都一樣大，對吧？

A: 是的，答案是它們不一定一樣大。所以 MTP 模塊中的 Transformer 塊可以有所不同。

B: 是的，它們可以根據每個預測深度的需求進行調整。所以這不僅僅是一組相同的模塊。

A: 沒錯。這是一個更靈活的系統，適應預測任務的需求。所以這就像每個預測階段的定制工具。

B: 是的，這種動態擴展有助於優化模型的性能和效率。你還問到了食物。我想這只是一個小小的口誤。

A: 是的，我想也是。所以它們如何整合信息來進行預測？

B: 是的，這種設計還允許推測解碼，這非常酷。所以它不僅僅用於訓練，還用於推理。

A: 沒錯。MTP 模塊可以在推理中重新用於加速。所以 MTP 用於生成可能的未來令牌。

B: 是的，然後它從可能性中選擇最佳令牌。但它們的大小並不完全相同，正如你正確地問到的那樣。所以 MTP 模塊中的 Transformer 塊大小可以有所不同，以優化性能。這非常靈活，這種靈活性有助於提高效率，正如我們一直在討論的那樣。

A: 是的，這都是 DeepSeek V3 在多令牌預測方面的創新方法。所以現在我們已經深入探討了洞穴，討論了 MTP 模塊的共享，並討論了它們的數量和大小。所以它生成文本的速度更快。

B: 是的，它通過不必從頭計算每個令牌來節省時間。現在讓我們轉向一個更大的視角。

A: 我們可以討論專家是如何被選擇來執行每個任務的。

B: 沒錯。現在讓我們深入探討 DeepSeek V3 的負載平衡策略。

A: 聽起來不錯。DeepSeek V3 使用我們剛才討論的 MTP。

B: 是的，我們現在應該轉向更大的視角。現在讓我們討論這個模型如何處理其負載平衡，以及這些專家是如何被選擇的。

A: 現在讓我們深入探討 DeepSeek V3 的負載平衡策略。

B: 聽起來不錯。DeepSeek V3 使用多令牌預測（MTP）。我們剛剛討論了 MTP 的工作原理，所以現在讓我們談談負載平衡，對吧？

A: 是的，我們剛剛討論過。現在它共享資源，你對它如何共享資源感興趣。我們已經討論過了。

B: 沒錯。所以它不僅僅預測下一個令牌，還一次性預測多個未來令牌，正如我們剛才討論的那樣。這不會增加複雜性嗎？

A: 看起來可能會，但它提供了幾個優勢。想像一下規劃路線。如果你只考慮下一個轉彎，你可能會錯過更高效的路線。提前規劃多個轉彎可以讓你選擇最佳路線。

B: 是的。DeepSeek V3 使用一種稱為無輔助損失負載平衡的創新方法，所以它不依賴於單獨的損失函數來進行平衡。

A: 沒錯。傳統的 MoE 模型在訓練期間使用輔助損失函數來鼓勵均勻使用專家，對吧？但這實際上可能會損害性能，正如我們之前提到的。

B: 就像試圖強迫每個人都使用超市的同一個結賬通道。

A: 所以通過預測多個令牌，模型可以更好地掌握上下文。

B: 是的，它可以生成更連貫和準確的回應。就像模型在預先規劃其表示，正如我之前提到的，以便更好地進行未來預測。這導致了更清晰的訓練信號和改進的數據效率。

A: 是的，所以 DeepSeek V3 動態調整每個專家的偏置項，基於