VIP Cheatsheet: Transformer と大規模言語モデル

アフシン・アミディ　シェルビン・アミディ　共著
中井喜之　訳

2025 年 3 月 31 日

この VIP Cheatsheet は、『Super Study Guide: Transformer と大規模言語モデル』という書籍の概要です。この書籍では、250 ページにわたって約 600 点のイラストを用い、以下の概念を詳細に解説しています。詳細は `https://superstudy.guide` をご覧ください。

1 基礎

1.1 トークン

❒ **定義** – トークンは、単語、サブワード、文字などの分割できないテキストの単位です。事前に定義された語彙を構成します。

補足：未知トークン *[UNK]* は、未知のテキスト片を表します。パディングトークン *[PAD]* は、入力シーケンスの空の箇所を埋めて、長さを揃えるために使用されます。

❒ **トークナイザ** – トークナイザ T は、テキストを任意の粒度のトークンに分割します。

このテディベアはとっっっって もかわいい → T → この | テディベア | は | [UNK] | かわいい | [PAD] | ... | [PAD]

主なトークナイザの種類は以下のとおりです。

種類	長所	短所	図
単語	• 解釈が容易 • シーケンスが短い	• 語彙サイズが大きい • 単語の変化に対応できない	テディ | ベア
サブワード	• 語根を捉えられる • 直感的な埋め込み表現	• シーケンスが長くなる • トークン化が複雑になる	テ | ##ディ | ベア
文字 バイト	• 語彙外になる心配がない • 語彙サイズが小さい	• シーケンスが非常に長くなる • 低レベルすぎてパターンの解釈が困難	テ | デ | ィ | ベ | ア

補足：バイト対符号化 *(Byte-Pair Encoding, BPE)* とユニグラムは、広く使われているサブワード単位のトークナイザです。

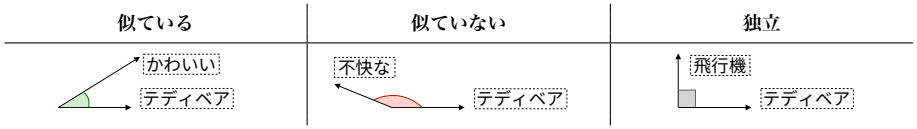
1.2 埋め込み表現

❒ **定義** – 埋め込みは、ある要素（トークン、文など）の数値表現です。ベクトル $x \in \mathbb{R}^n$ によって表されます。

❒ **類似度** – 2 つのトークン t_1 と t_2 の間のコサイン類似度 (similarity) は、以下のように計算されます。

$$\text{similarity}(t_1, t_2) = \frac{t_1 \cdot t_2}{||t_1||\ ||t_2||} = \cos(\theta) \in [-1, 1]$$

角度 θ は、2 つのトークン間の類似度を表します。

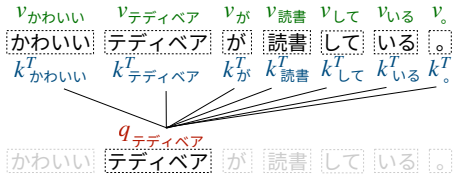


補足：近似最近傍探索 *(Approximate Nearest Neighbors, ANN)* と局所性鋭敏型ハッシュ *(Locality Sensitive Hashing, LSH)* は、大規模データベース上での類似度演算を効率的に近似する手法です。

2 Transformer

2.1 Attention

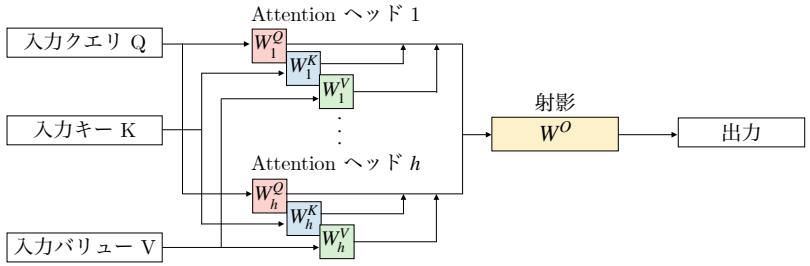
❒ **定式化** – あるクエリ q が与えられたとき、バリュー v と紐づくどのキー k に、クエリが「注意」(Attention) を払うべきかを求めます。



Attention は、クエリ q を含む行列 Q 、キー k を含む行列 K 、バリュー v を含む行列 V 、およびキーの次元 d_k を用いて、以下のように効率的に計算することができます。

$$\text{attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

❒ **MHA** – ***Multi-Head Attention*** (MHA) 層は、複数のヘッドで Attention 計算を行い、その結果を出力空間に射影します。

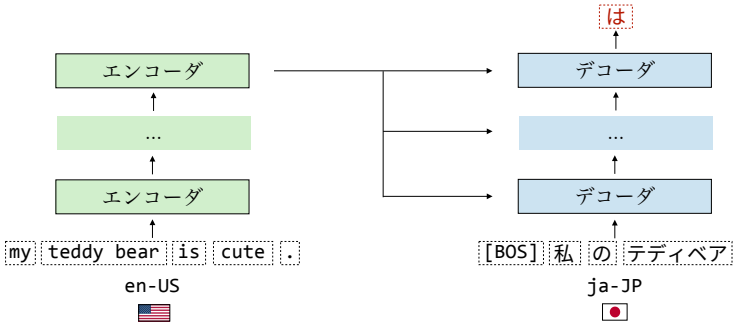


この層は、h 個の Attention ヘッドを持ち、行列 W^Q, W^K, W^V による射影で、入力からクエリ Q、キー K、バリュー V を得ます。行列 W^O を用いて出力を射影します。

補足：*Grouped-Query Attention (GQA)* と *Multi-Query Attention (MQA)* は、*MHA* の変種です。*Attention* ヘッド間でキーとバリューを共有することで計算コストを削減します。

2.2 構造

❒ **概要** – Transformer は、Self-Attention 機構に基づく画期的なモデルで、エンコーダとデコーダから構成されます。エンコーダは意味を考慮した埋め込み表現を入力から計算し、デコーダはそれを用いてシーケンス内の次のトークンを予測します。

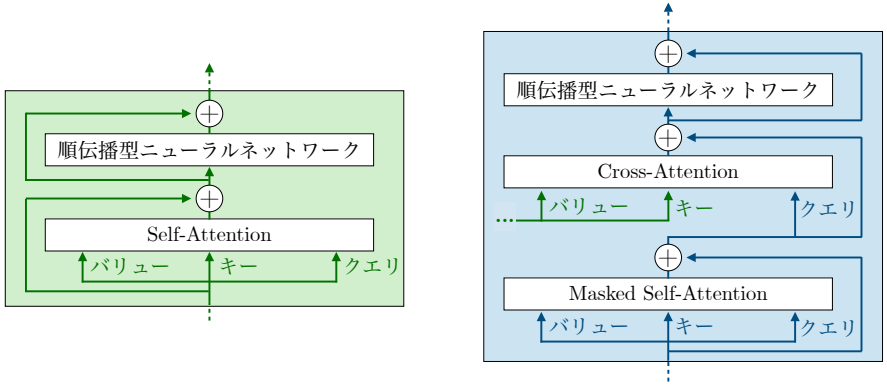


補足：*Transformer* は当初、翻訳タスクのためのモデルとして提案されましたが、現在では他の多くの分野に広く応用されています。

❒ **構成要素** – エンコーダとデコーダは、Transformer の 2 つの主要な構成要素であり、それぞれ以下のような役割があります。

エンコーダ
エンコードされた埋め込み表現は、入力の意味を保持している

デコーダ
デコードされた埋め込み表現は、入力とこれまでに予測された出力の両方の意味を保持している

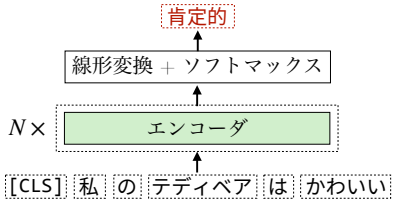


❒ **位置埋め込み表現** – 位置埋め込み表現は、トークンの文中における位置情報を含み、トークン埋め込み表現と同じ次元です。これらは任意に定義することも、データから学習することもできます。

補足：回転位置埋め込み表現 *(Rotary Position Embeddings, RoPE)* は、広く使われている効率的な手法です。クエリとキーのベクトルを回転させることで、相対的な位置関係を埋め込みます。

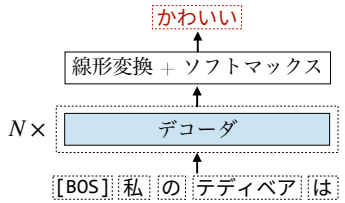
2.3 バリエーション

❒ **エンコーダのみ** – ***B**idirectional **E**ncoder **R**epresentations from **T**ransformers* (BERT) は、Transformer ベースのモデルであり、複数のエンコーダの積み重ねで構成されています。テキストを入力として受け取り、意味を考慮した埋め込み表現を出力し、それらは後続の分類タスクなどで使用されます。



[CLS] トークンは、文の意味を捉えるためにシーケンスの先頭に追加されます。そのエンコードされた埋め込み表現は、感情抽出などの後続のタスクでよく使用されます。

❒ **デコーダのみ** – ***G**enerative **P**re-trained **T**ransformer* (GPT) は、Transformer ベースの自己回帰モデルであり、複数のデコーダの積み重ねで構成されています。BERT とその派生モデルとは異なり、GPT はすべての問題を Text-To-Text の問題として扱います。



現在最先端の大規模言語モデル (LLM) の大部分は、GPT 系モデル、LLaMA、Mistral、Gemma、DeepSeek といった、デコーダのみの構造を採用しています。

補足：*T5* のようなエンコーダ・デコーダモデルも自己回帰型であり、デコーダのみのモデルと多くの共通する特性があります。

2.4 最適化

❒ **Attention の近似** – Attention 演算は $\mathcal{O}(n^2)$ の計算量を持ち、シーケンス長 n が増加するにつれてコストが高くなる可能性があります。この演算を近似する方法は主に 2 つあります。

- スパース化：Self-Attention がシーケンス全体ではなく、より関連性の高いトークン間でのみ発生するようにします。

- 低ランク化：Attention の計算式を低ランク行列の積として簡略化し、計算負荷を軽減します。

❒ **Flash Attention** – Flash Attention は、GPU ハードウェアをうまく活用した Attention 演算の厳密な最適化手法です。高速な ***S**tatic **R**andom-**A**ccess **M**emory* (SRAM) を行列演算に使用し、結果を低速な ***H**igh **B**andwidth **M**emory* (HBM) に書き込みます。

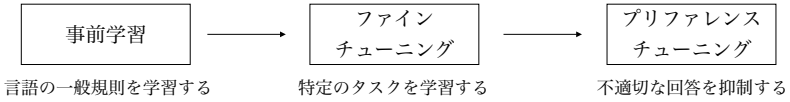
補足：実際にこの手法により、メモリ使用量が削減され、演算速度が向上します。

3 大規模言語モデル

3.1 概要

❐ **定義** – 大規模言語モデル (***L**arge **L**anguage **M**odel*, LLM) は、強力な自然言語処理能力を持つ Transformer ベースのモデルです。「大規模」とは、通常、数十億のパラメータがあることを意味します。

❐ **ライフサイクル** – LLM は 3 つのステップでトレーニングされます：事前学習、ファインチューニング、プリファレンスチューニング

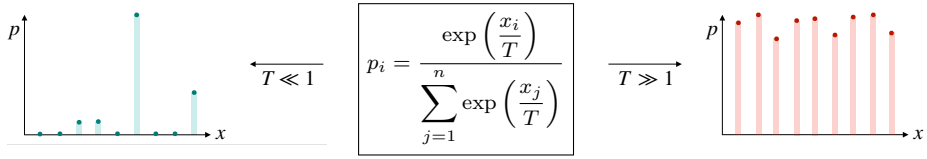


ファインチューニングとプリファレンスチューニングは、モデルを特定のタスクに適応させるための、トレーニング後の調整手法です。

3.2 プロンプト

❐ **コンテキスト長** – モデルのコンテキスト長とは、入力として与えることのできるトークンの最大数です。一般的に、数万から数百万トークンの範囲です。

❐ **デコード時のサンプリング** – トークンの予測は、ハイパーパラメータである温度 T によって制御される、予測確率分布 p_i からサンプリングされます。



補足：温度が高いほど創造的な出力が得られ、温度が低いほど決定論的な出力が得られます。

❐ **思考の連鎖** – 思考の連鎖 (***C**hain-**o**f-**T**hought*, CoT) は、モデルが複雑な問題を一連の中間ステップに分解して推論する手法で、正確な最終回答を生成するのに役立ちます。思考の木 (***T**ree **o**f **T**houghts*, ToT) は、CoT の発展形です。

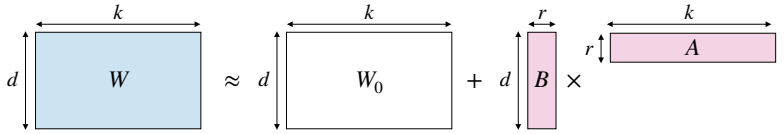
補足：自己整合性 *(Self-Consistency)* は、*CoT* による推論経路全体からの回答を集約する手法です。

3.3 ファインチューニング

❐ **SFT** – 教師ありファインチューニング (***S**upervised **F**ine**T**uning*, SFT) は、モデルの挙動を特定の最終タスクに適応させるための、トレーニング後の調整手法です。タスクと整合性の取れた高品質な入出力ペアを使用します。

補足：*SFT* のデータが指示に関するものである場合、このステップは「指示チューニング」と呼ばれます。

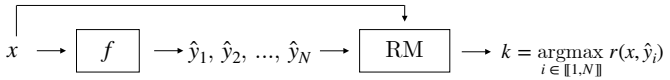
❐ **PEFT** – パラメータ効率の良いファインチューニング (***P**arameter-**E**fficient **F**ine**T**uning*, PEFT) は、SFT を効率的に行うための手法の総称です。例えば、低ランク適応 (***L**ow-**R**ank **A**daptation*, LoRA) は初期値 W_0 を固定し、代わりに低ランク行列 A, B を学習することで、学習可能な重み W を近似します。



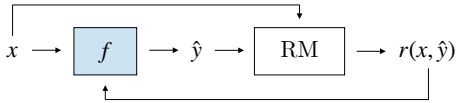
補足：その他 *PEFT* 手法には、プレフィックスチューニングやアダプタ層の挿入などがあります。

3.4 プリファレンスチューニング

❐ **報酬モデル** – 報酬モデル (***R**eward **M**odel*, RM) は、入力 x が与えられたときに、出力 $\hat{y}$ が望ましい行動とどの程度一致するかを予測するモデルです。***B**est-**o**f-**N*** (BoN) サンプリングは棄却サンプリングとも呼ばれ、報酬モデルを使用して、N 個の生成された応答の中から最適なものを選択する手法です。



❐ **強化学習** – 強化学習 (***R**einforcement **L**earning*, RL) は、RM を活用し、生成された出力に対する報酬に基づいてモデル f を更新する手法です。RM が人間の選好に基づいている場合は、人間のフィードバックによる強化学習 (***R**einforcement **L**earning from **H**uman **F**eedback*, RLHF) と呼ばれます。

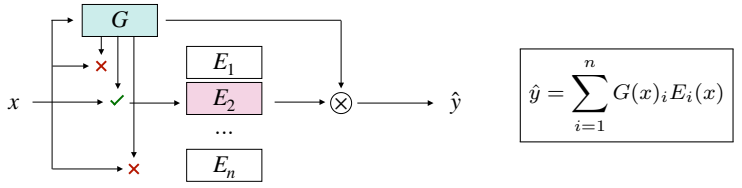


近接方策最適化 (***P**roximal **P**olicy **O**ptimization*, PPO) は、広く使われている RL アルゴリズムで、報酬ハッキングを防ぐためにモデルを元のモデルに近づけながら、報酬の最大化を促します。

補足：教師ありの手法も存在します。例えば *Direct Preference Optimization (DPO)* は、*RM* と *RL* を *1* つの教師ありステップに統合します。

3.5 最適化

❐ **混合エキスパート** – 混合エキスパート (***M**ixture **o**f **E**xperts*, MoE) は、推論時にニューロンの一部のみを活性化させるモデルで、ゲート G とエキスパート $E_1, \ldots, E_n$ から構成されます。



MoE ベースの LLM は、その順伝播型ニューラルネットワーク内でこのゲート機構を使用します。

補足：*MoE* ベースの *LLM* のトレーニングは非常に難しいことが知られています。*LLaMA* の論文で、著者らは推論時の効率性にもかかわらず、この構造を使用しないことにしたと述べています。

❐ **蒸留** – 蒸留は、(小さな) 学生モデル S を (大きな) 教師モデル T の予測出力でトレーニングする手法です。以下の KL ダイバージェンス損失を使用してトレーニングされます。

$$\boxed{\mathrm{KL}(\hat{y}_T||\hat{y}_S) = \sum_i \hat{y}_T^{(i)} \log\left(\frac{\hat{y}_T^{(i)}}{\hat{y}_S^{(i)}}\right)}$$

補足：トレーニングラベルは、クラスに属する確率を表すため、「ソフト」ラベルと見なされます。

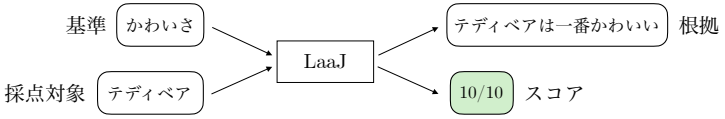
❒ **量子化** – モデル量子化は、得られるモデルの性能への影響を抑えながら、モデルの重みの精度を下げる技術の総称です。結果として、モデルのメモリ使用量が減少し、推論が高速化されます。

補足：*QLoRA* は、*LoRA* を量子化した手法で、広く使われています。

4 応用

4.1 LLM-as-a-Judge

❒ **定義** – ***LLM-as-a-Judge*** (LaaJ) は、与えられた出力を、指定された基準で採点するために LLM を使用する方法です。特に、スコアの根拠を生成することもできるため、解釈可能性を向上させます。



Recall-Oriented Understudy for Gisting Evaluation (ROUGE) のような LLM 以前の指標とは異なり、LaaJ は参照テキストを必要としないため、あらゆる種類のタスクを評価できる利点があります。特に、LaaJ は大規模で強力なモデル（GPT-4 など）を使用した場合、人間の評価と強い相関関係を示します。これは、優れた性能を発揮するために推論能力が求められるためです。

補足：*LaaJ* は迅速な評価を繰り返す際に有用ですが、*LaaJ* の出力と人間の評価の間の整合性を監視して、乖離がないことを確認することが重要です。

❒ **一般的なバイアス** – LaaJ モデルは、以下のバイアスを示すことがあります。

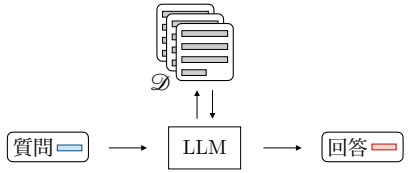
	位置バイアス	冗長バイアス	自己強化バイアス
問題	ペアの比較で最初の位置にあるものを好む	より冗長なコンテンツを好む	自身が生成した出力を好む
解決策	位置をランダム化して指標の平均を取る	出力長にペナルティを加える	異なるベースモデルで構築された LaaJ を使用する

これらの問題への対策として、カスタム LaaJ のファインチューニングもありますが、多大な労力が必要です。

補足：上記のバイアス一覧は、すべてを網羅しているわけではありません。

4.2 RAG

❒ **定義** – 検索拡張生成 (***Retrieval-Augmented Generation***, RAG) は、LLM が与えられた質問に答える際に、外部の関連知識にアクセスできるようにする手法です。LLM の事前学習知識の締切日以降の情報を取り込みたい場合に特に有用です。



知識基盤 $\mathscr{D}$ と質問が与えられたとき、検索器は最も関連性の高い文書を取得し、出力を生成する前にプロンプトを関連情報で拡張します。

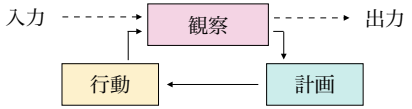
補足：検索段階では通常、エンコーダのみのモデルによる埋め込み表現が使用されます。

❒ **ハイパーパラメータ** – 知識基盤 $\mathscr{D}$ は、文書をサイズ n_c の断片に分割し、サイズ $\mathbb{R}^d$ のベクトルに埋め込むことで初期化されます。

4.3 エージェント

❒ **定義** – エージェントは、ユーザに代わって自律的に目標を追求し、タスクを完了するシステムです。そのために、異なる LLM を連鎖的に呼び出す場合があります。

❒ **ReAct** – ***Reason*** + ***Act*** (ReAct) は、複雑なタスクを完了するために複数の LLM を連鎖的に呼び出すことを可能にするフレームワークです。



このフレームワークは、以下のステップで構成されています。

- 観察：以前の行動結果をまとめ、現在分かっている状況を明示的に記述します。
- 計画：達成すべきタスクと呼び出すツールを詳細に記述します。
- 行動：API を介してタスクを実行するか、知識基盤で関連情報を探します。

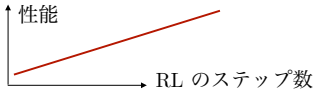
補足：エージェントシステムの評価は困難ですが、各ステップの入出力からコンポーネントごとに評価することや、連鎖的な呼び出しの結果からシステム全体を評価することができます。

4.4 推論モデル

❒ **定義** – 推論モデルは、数学、コーディング、論理的思考といった複雑なタスクを解決するために、思考の連鎖 (CoT) に基づく推論過程を用いるモデルです。推論モデルの例として、OpenAI の o シリーズ、DeepSeek-R1、Google の Gemini Flash Thinking などがあります。

補足：*DeepSeek-R1* は、`<think>` タグの間に推論過程を明示的に出力します。

❒ **スケーリング** – 推論能力を向上させるため、以下のような 2 種類のスケーリング手法があります。

	説明	図
トレーニング時のスケーリング	モデルが回答の前に CoT に基づく推論過程を生成する方法を学習できるよう、RL をより長く実行する	
テスト時のスケーリング	「待て」などの時間的制約を課すキーワードを用い、モデルが回答する前に、より長い時間思考するようにする