修士論文

LLM の重みとバイアスに注目した
学習済みデータ特定手法

北村 拓斗

2025 年 1 月 30 日

岐阜大学大学院 自然科学技術研究科 知能理工学専攻 知能情報学領域
鈴木研究室

本論文は岐阜大学大学院自然科学技術研究科に
修士（工学）授与の要件として提出した修士論文である.

北村 拓斗

指導教員：
鈴木 優 准教授

LLM の重みとバイアスに注目した学習済みデータ特定手法*

北村 拓斗

内容梗概

メンバーシップ推論攻撃とは，与えられたデータがモデルの学習に使用されたデータかどうかを特定するタスクである．本研究では，与えられたテキストデータを大規模言語モデル (Large Language Model; LLM) で事前学習させたときに生じる LLM のパラメータの増減を観察することにより，事前学習に使用されたテキストデータかどうかを特定する精度の向上を目指す．LLM の事前学習では，与えられたテキストデータを出力できるように学習を行う．事前学習に使用されたテキストデータを学習する場合，LLM は既にテキストデータを出力できるため，事前学習時のパラメータの変化量は少ない．一方，事前学習に使用されていないテキストデータを学習する場合，LLM がテキストデータを出力できるように学習を行うため，事前学習時のパラメータの変化量が大きいという仮説を立てた．実験の結果，パラメータの変化量を利用したメンバーシップ推論攻撃の有効性が確認された．また，機械学習モデルとして Pythia を用いたときに提案手法の精度が既存手法である Min-K% Prob と SaMIA*zlib の精度を上回った．OPT と Llama-2 を用いたときには，提案手法の精度が既存手法である Min-K% Prob の精度を上回った．

キーワード

LLM，メンバーシップ推論攻撃，ニューラルネットワーク，機械学習，自然言語処理

*岐阜大学大学院 自然科学技術研究科 知能理工学専攻 知能情報学領域 修士論文, 学籍番号: 1234525031, 2025 年 1 月 30 日.

目次

図目次

表目次

第 1 章　はじめに

本研究では，大規模言語モデル (Large Language Model; LLM) の事前学習に使用されたテキストデータを特定するための手法を提案する．LLM の構築には，著作権が守られていない学習データを含んでいるという研究結果 [1, 2] が報告されている．このようなデータで学習されたモデルは，LLM が生成したデータの出典を明確に判断できず，利用者にとってリスクとなる可能性がある．また，モデルの性能評価を行うためのベンチマークデータがモデルの学習データを意図せず含んでいる場合，モデルは評価データを学習していることになり，モデルの公正な性能評価を行うことができない [3, 4]．そこで，事前学習に使用されたテキストデータを特定することは有用である．

メンバーシップ推論攻撃 (Membership Inference Attacks; MIA)[5] とは，与えられた画像データやテキストデータなどが攻撃対象のモデルを構築するために利用したデータセットに含まれているかどうかを特定する方法である．機械学習モデルを対象とした MIA の研究 [5, 6, 7] は数多く行われてきた．また，LLM が生成したした文章を観察して MIA を行う研究 [8, 9] もある．しかし，特に LLM に対して MIA を適用した場合，LLM が生成した文章は学習データをそのまま生成したものなのか，学習データには含まれないが条件付き確率に基づいて似た文章を偶然生成したものなのかを判断することは非常に難しいという問題がある．LLM は条件付き確率に基づいて文章を生成するため，必ずしも学習したテキストデータそのものを生成するわけではない．他方で，LLM が学習データをそのまま生成した事例 [1] も報告されている．そのため，LLM が出力した文章を観察し，学習データに含まれているかどうかを単純な一致や類似度の測定のみで特定することは困難である．例えば，対象の LLM が「吾輩は猫です」という文章を生成したとする．生成された文章は夏目漱石の作品 (吾輩は猫である) に登場する有名な文章の「吾輩は猫である」と類似している．この場合,「吾輩は猫である」という文章が学習データに含まれていてそのまま生成したのか,「吾輩は猫である」という文章は学習していないが学習データに基づいて偶然似た文章を生成したのかを明確に判断することはできない．

この問題の解決策として，LLM 内部のパラメータの動きを直接観察する手法を

提案する．パラメータとは，LLM を含むニューラルネットワークに存在している重みとバイアスから構成される行列である．ニューラルネットワークはパラメータを変化させることによって目的関数を最小化するように学習を行うため，パラメータには学習データの内容や特徴が反映されると考えた．生成された文章ではなくパラメータを直接観察することで，この問題を解決できると考えた．

ただ，パラメータを直接観察する方法には，正則化やドロップアウトなど過学習を抑制する機構によって，学習データ特有の特徴がモデルの内部や出力に反映されにくくなるという問題がある．先行研究 [6] でも言及されているように MIA が成功する理由の一つは，モデルが過学習している場合に学習済みデータと未学習データに対する挙動が異なることに起因している．実際に先行研究 [6] では，過学習したモデルに学習データと未学習データを入力したときの出力である予測ベクトルに観測可能な違いが確認されている．一方，過学習が抑制されたモデルは学習データと未学習データに対する挙動の違いが緩和され，MIA の精度は低くなることが知られている．

この過学習が抑制される問題の解決策として，本研究では正則化やドロップアウトのような過学習を抑制する機構があっても意図的に過学習を引き起こすために，事前学習時に一つのテキストデータだけを与えて繰り返し学習を行う．通常のニューラルネットワークの学習では，複数の異なるデータを用いて学習を行うことにより，汎化性能を向上させることを目的としている．我々の目的は，モデルを過学習させるであるため，意図的に一つの同じデータだけを繰り返し学習させることで，データ特有の特徴がパラメータに反映されるのではないかと考えた．

提案手法では，事前学習前後のパラメータ変化量を特徴量として抽出し，この特徴量から与えられたテキストデータが事前学習に使用されたかどうかを特定することを目指す．実験では，提案手法による MIA が可能であるかどうか，提案手法が既存手法である Min-$K\%$ Prob と SaMIA*zlib よりも優れているかを四つの LLM を用いて確かめた．実験の結果，機械学習モデルとして Pythia を用いたときに提案手法の精度が両方の既存手法の精度を上回った．OPT と Llama-2 では，提案手法の精度が既存手法である Min-$K\%$ Prob の精度を上回った．また，特徴量に使用するパラメータを選定することによって提案手法を用いた MIA の精度が向上することも分かった．

本論文における貢献は以下のとおりである．

- テキストデータを用いて LLM の事前学習を行ったとき，事前学習前と事前学習後におけるパラメータの変化量を特徴量としたメンバーシップ推論攻撃は可能であることを確認した．
- LLM に含まれる全てのパラメータを使用するのではなく，一部のパラメータのみを使用することでメンバーシップ推論攻撃の精度が向上することが分かった．

本論文の構成は以下の通りである．2 章では，基本的事項について述べる．3 章では，関連研究について述べる．4 章では，提案手法について述べる．5 章では，評価実験について述べる．6 章では，本論文のまとめと今後の課題について述べる．最後に Appendix では，実験環境と技術的な工夫について述べる．

第 2 章　基本的事項

2.1　機械学習

機械学習 (Machine Learning; ML) とは，コンピュータが画像やテキストなどのデータから学習を行い，予測や判断を行う技術である．一般に機械学習と同じ文脈で使用される言葉に人工知能 (Artificial Intelligence; AI) と深層学習 (Deep Learning; DL) があるが，これらの用語は全て異なるものを指している．人工知能とは，コンピュータに人間のような知能を持たせる技術全般であり，実現方法は機械学習に制限されていない．人工知能を実現するための技術の一つとして，機械学習がある．深層学習とは，人間の脳を模倣したニューラルネットワークを用いる技術を指し，ディープラーニングとも呼ばれる．機械学習に含まれる技術として，深層学習がある．

機械学習はさまざまな場面で使われつつある．例えば，多くの人々が使用している ChatGPT は機械学習を使用した代表的なサービスの一つである．ChatGPT では，ユーザの指示 (プロンプト) に従い，機械学習モデルがテキストや画像を用いたさまざまなタスクを処理することができる．ここで用いられているモデルは 2.1.1 節で述べる教師あり学習，教師なし学習，強化学習を全て組み合わせて構築されている．他には，自動運転にも用いられている．自動運転とは，車両に取り付けられたカメラやセンサーで車両の周囲を認識し，得られたデータをもとにして自動で運転，あるいは，運転時にドライバーのサポートをする技術である．自動運転技術では，歩行者や信号機，道路の白線や標識の認識を行って機械学習モデルが判断を行い，適切な速度や進行方向を決めている．自動運転技術では，チューリング*や日産自動車†が有名である．

十年ほど前では，Apple 社のスマートフォンに搭載された音声アシスタントの Siri，Netflix や YouTube の動画推薦システムなど，機械学習を活用したサービスやシステムは筆者の知る限りあまり多くはなかった．近年の機械学習に関する技術革新により，サービス数は飛躍的に増加している．

*https://tur.ing

†https://www2.nissan.co.jp/BRAND/DRIVING/

2.1.1 機械学習の種類

機械学習は主に三種類に分けることができる．

1. 教師あり学習
2. 教師なし学習
3. 強化学習

教師あり学習とは，入力されたデータと正解ラベルのセットを用いて，入力と正解の関係性をモデルが学習する方法である．教師あり学習をさらに細かくすると，分類 (Classification)，回帰 (Regression) に分けられる．入力された画像が犬であるか猫であるかを判定する画像分類，商品に対するレビューがポジティブであるかネガティブであるかを判定する感情分類，株価の変動を予測する回帰が教師あり学習に該当する．教師あり学習の利点は，正解データが存在するため精度の高いモデルを構築しやすいことにある．一方，欠点はラベル付けされたデータが大量に必要となるため，データの収集ができない場合にはモデルの学習を行うことが難しいことである．

教師なし学習とは，入力されたデータのみを用いて入力されたデータの構造や特徴をモデルが学習する方法である．教師なし学習をさらに細かくすると，クラスタリング (Clustering)，次元削減 (Dimensionality Reduction)，異常検知 (Anomaly Detection)，生成モデル (Generative Model) などに分けられる．YouTube の動画を似ているグループに自動で分けるクラスタリング，高次元データを低次元に圧縮する主成分分析 (Principal Component Analysis; PCA) や t-SNE(t-distributed Stochastic Neighbor Embedding) の次元削減，産業機械の故障検知のためのオートエンコーダによる異常検知が教師なし学習に該当する．また，BERT モデル [10] の事前学習に使用されるマスク言語モデル (Masked Language Model; MLM) は，教師なし学習の一種である自己教師あり学習に該当する．

強化学習とは, モデルが試行錯誤を行い, 試行錯誤の結果で得られる報酬を用いて学習する手法である．強化学習をさらに細かくすると，価値ベース (Value-Based)，方策ベース (Policy-Based)，深層強化学習 (Deep Reinforcement Learning) などに分けられる．囲碁や将棋などのゲーム AI，ロボットアームや自律型ドローンなどのロボット制御が強化学習に該当する．深層学習と強化学習を用いて囲碁のトッ

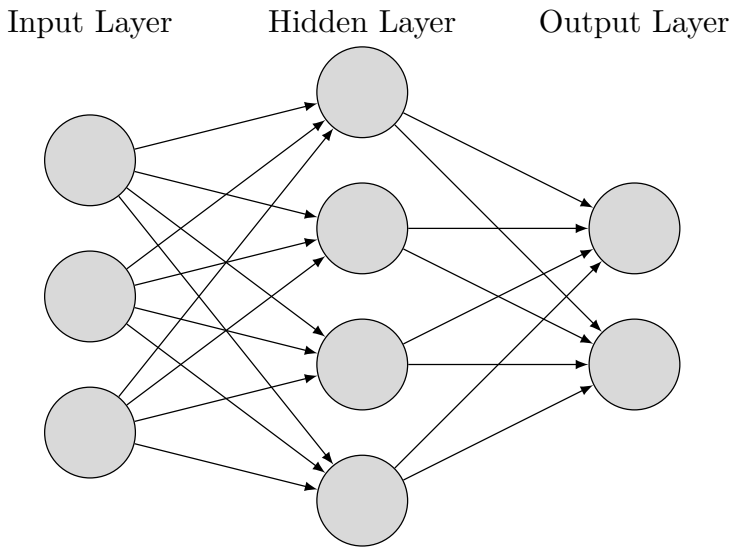

図 2.1: ニューラルネットワークの模式図

ププロに勝利した AI である AlphaGo[‡]は深層強化学習の代表的な成功例である．

2.1.2 ニューラルネットワーク

ニューラルネットワークとは，人間の脳内神経回路を模倣して設計された機械学習モデルである．従来の線形回帰やロジスティック回帰といった線形モデルでは，データに現れる非線形な関係をうまく捉えることができなかった．そのため，画像認識やテキスト分類などの現実的な問題に対して適用することが難しかった．一方，ニューラルネットワークは活性化関数と多層構造を組み合わせることによって非線形な関係を学習できるようになり，従来手法では難しい問題に対しても適用することができるため，画像認識や自然言語処理といった様々な分野で優れた性能を発揮している．

図 2.1 に示した三層のニューラルネットワークを用いて説明する．Input Layer は入力層，Hidden Layer は隠れ層あるいは中間層，Output Layer は出力層と呼ばれる．入力層は，入力されたデータを受け取る役割を持っている．中間層は，入力

[‡]https://deepmind.google/research/breakthroughs/alphago/

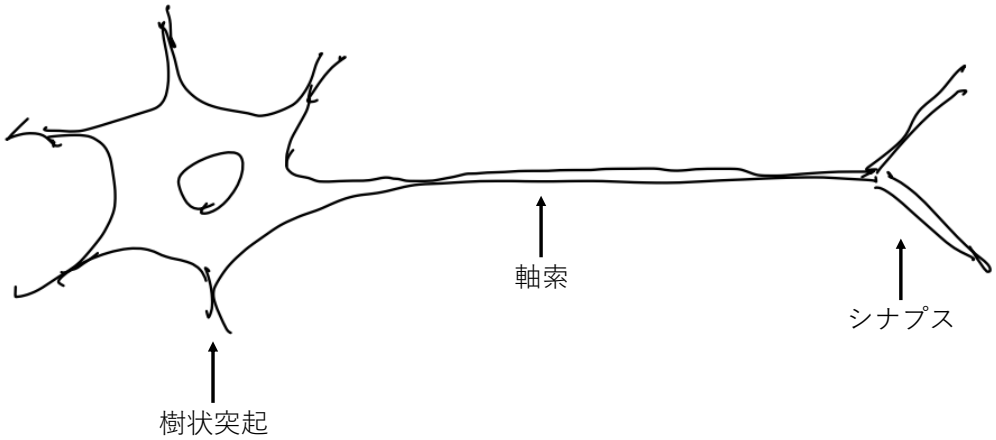

図 2.2: 人間の神経のイメージ図

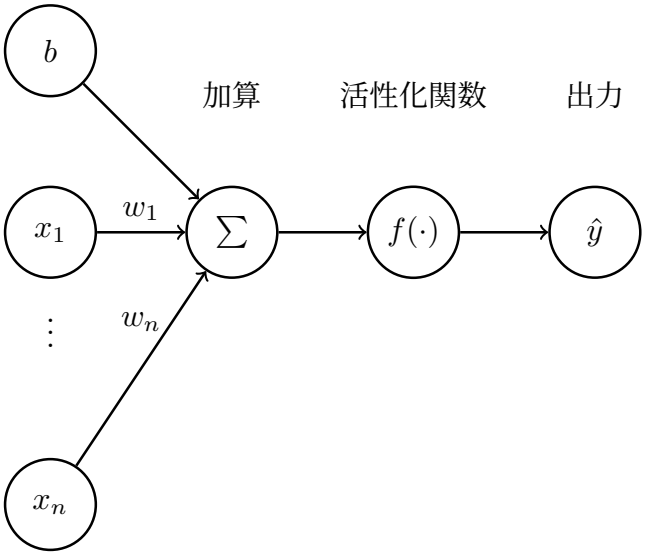

図 2.3: 人工ニューロンの模式図

層から得られた情報の処理を行い，特徴を抽出する役割を持っている．出力層は，中間層から得られた情報をもとに，モデルの予測を行う役割を持っている．これらの層はユニットと呼ばれるパーツから構成されている．ユニットとは図中の丸で示したものである．ニューラルネットワークは，ユニットや層が相互に接続することによって構成される．ユニットのことを人工ニューロンと呼ぶ場合がある．

人間の神経のイメージを図 2.2，人工ニューロンを図 2.3 に示す．人間の情報処理機構である神経を模したものを人工ニューロン，もしくはパーセプトロンと呼ぶ．人間が内部で情報を伝達する時，ニューロンにある樹状突起でシナプスでつながった他のニューロンから情報を受け取り，軸索が電気信号として他のニューロンへと伝達するというやり取りが行われる．樹状突起が入力データ $x_1, x_2, \ldots, x_n$，シナプスが重み w，軸索が活性化関数 $f(\cdot)$ に対応しており，人間の体内で行われる情報のやり取りを人工ニューロンによって疑似的に再現している．人工ニューロンでは，複数の入力データを受け取って非線形な変換処理を行っている．

変換処理には，順伝搬と逆伝搬の二つがある．順伝搬とは，ニューラルネットワークが受け取ったデータをもとに各層で計算を行い，最終的な出力を得るためのプロセスである．推論時に行われるのが順伝搬である．順伝搬を行う手順は以下のようである．

1. 特徴量と重みの乗算を行う．
   入力されたデータ $x_1, x_2, \ldots, x_n$ に対して，対応する重み $w_1, w_2, \ldots, w_n$ を乗算する．具体的には，$x_1 \times w_1 + x_2 \times w_2 + \cdots + x_n \times w_n$ で表される．
2. 乗算結果とバイアス項 b の加算を行う．
   手順 1 での計算結果に，バイアス項の加算を行う．加算結果を z と表すと，$z = x_1 \times w_1 + x_2 \times w_2 + \cdots + x_n \times w_n + b$ で表される．
3. 活性化関数による非線形変換を行う．
   加算結果 z に対し，活性化関数 $f(\cdot)$ を通した後の結果を出力する．出力 y は，$y = f(z)$ で表される．

逆伝搬とは．順伝搬時の最終的な出力と正解との差を表す損失関数を最小化するために，ニューラルネットワークの重みとバイアスを最適化していくプロセスである．学習時に行われるのが逆伝搬である．逆伝搬を行う手順は以下のようである．

1. 出力層で誤差を計算する．
   モデルの予測値 $\hat{y}$ と正解値 y との誤差である損失関数 L を計算する．例えば，目的関数が平均二乗誤差である場合，$L = \frac{1}{2}\sum(y - \hat{y})^2$ で表される．
2. 重みとバイアスを更新する．
   勾配降下法を用いて，損失関数 L が最小になるように重み w とバイアス b

を更新する．以下の式で更新を行う．

$$w \leftarrow w - \alpha \cdot \frac{\partial L}{\partial w}, \quad b \leftarrow b - \alpha \cdot \frac{\partial L}{\partial b}$$

ここで α は学習率を表しており，重みやバイアスの更新度合いを制御する値である．

ニューラルネットワークの学習では，順伝搬と逆伝搬が交互に行われる．順伝搬によって入力データが入力層から出力層方向へと伝播してモデルの予測値が出力され，逆伝搬によって損失関数を基に計算された誤差が出力層から入力層方向へと伝播する．その後，パラメータの重みとバイアスが更新され，モデルの予測値と正解データとの誤差が小さくなる．順伝搬と逆伝搬を何度も繰り返すことでパラメータが最適化されていき，予測精度が高まっていくという仕組みである．学習率や損失関数の設計，ハイパーパラメータなどの選択は，ニューラルネットワークの予測精度や学習効率に大きな影響を与える．適切に選択されることで，ニューラルネットワークの学習を効率的に進めることができ，高い性能を示すことができる．

図 2.4 に損失関数 L と重みパラメータ w がどのように変化していくかを示す．$w = 0$ であるとき，この点を通る緑色で示した接線の傾きは右肩下がりである．この場合は，損失関数を最小化するように w が更新され，w は $w = 0$ よりも大きい値をとるようになる．一方，$w = 4$ であるとき，この点を通る緑色で示した接線の傾きは右肩上がりである．そのため，損失関数を最小化するように w が更新され，w は $w = 4$ よりも小さい値をとるようになる．

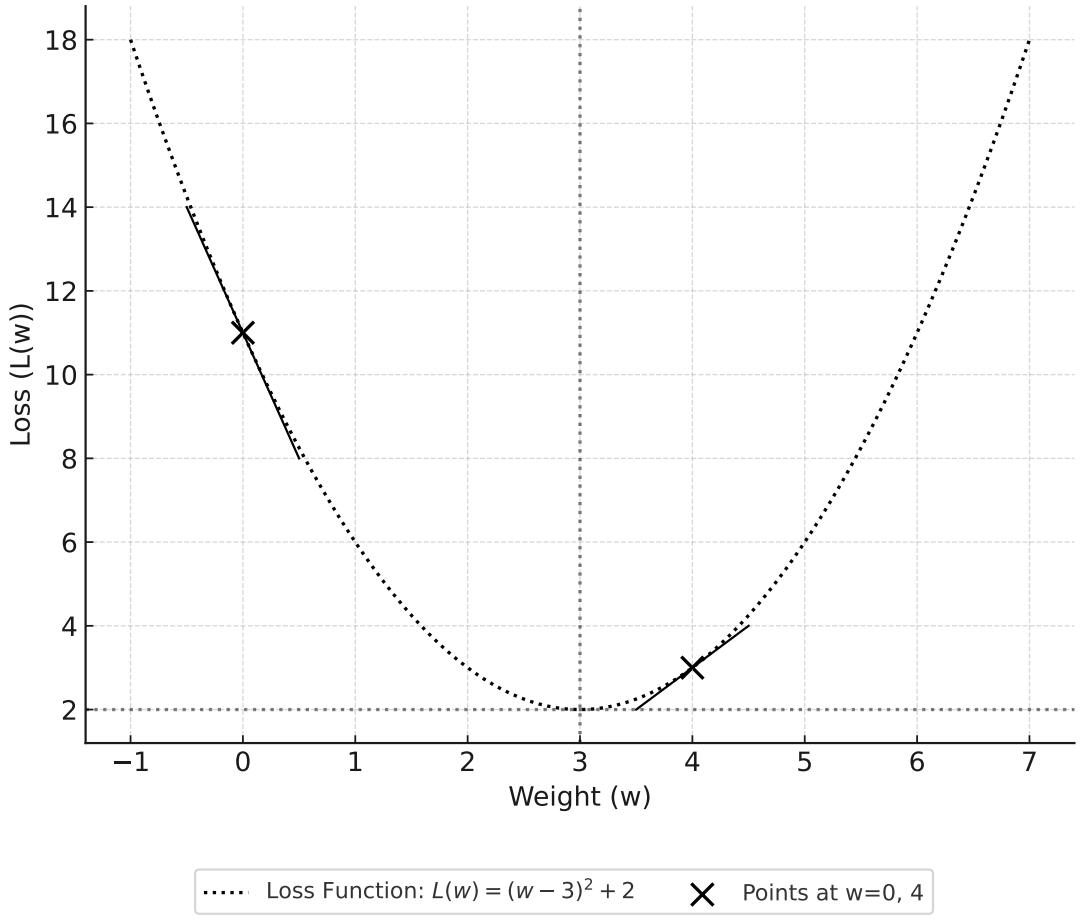

図 2.4: 損失関数 L と重み w の様子

2.1.3　ニューラルネットワークにおける数十年に及ぶ挑戦

近年のディープラーニングブームの勢いは凄まじいものである．しかし，ディープラーニングが成功するまでの道のりは険しいもので，数十年に及ぶ研究者たちの戦いがあった．

第一次 AI ブームは 1950 年ごろに起きた．ニューラルネットワークのアイデアは 1950 年ごろには知られており，マッカロックとピッツの形式ニューロン [11] やローゼンブラットのパーセプトロン [12] が提唱された．しかし，線形分離可能な問題しか解けず，非線形な問題に対応できなかったため，下火となった．

第二次 AI ブームは 1980 年ごろに起きた．コンピュータの高性能化が進み，エキスパートシステムと呼ばれる特化型 AI のようなものが提案された．しかし，知識獲得のボトルネックという問題や大量のデータを処理できない問題に直面し，再び下火となった．

第三次 AI ブームは 2000 年ごろに起き，現在も続いている．特に，2020 年以降の技術進歩が著しいため，2020 年以降を区別して第四次 AI ブームと呼ぶ場合もある．インターネットの普及と GPU の高性能化により，ビッグデータを効率的に処理できるようになった．また，2024 年のノーベル物理学賞を受賞した Geoffrey Hinton や OpenAI の研究者であった Ilya Sutskever らによって確立されたディープラーニングの基礎技術 [13, 14, 15] による功績が大きい．2025 年 1 月現在も様々なモデルが開発され続けており，2025 年 1 月の執筆時点で最新の LLM は DeepSeek [16] である．

2.2 大規模言語モデル（Large Language Model; LLM）

大規模言語モデル [17] とは，数 T（数兆）トークンもの大規模なデータセットを使用して訓練された大量のパラメータで構成されるニューラルネットワークである．一般に，パラメータ数が数十 B（数百億）以上の巨大なニューラル言語モデル [18] を指すが，明確な定義はなされていないため，BERT [10] のような 0.1B（一億）程度の比較的小さなモデルも LLM と呼ぶことがある．これまでに数百個以上ものLLM が開発されており，その代表例として ChatGPT [19] や Gemini [20]，PaLM [21]，Qwen [22]，DeepSeek [16] などが知られている．開発されてきた LLM の中には，文章の翻訳や添削，表作成など多種多様なタスクを行うことのできる LLM，コーディングに特化した LLM が存在する．

LLM にはいくつかの特性があることが知られている．一つ目は，スケーリング則である．スケーリング則とは，モデルを大規模化（スケール化）することによってモデルの性能が向上するという冪乗則である．LLM の飛躍的な性能向上は，2020 年に OpenAI 社によって発見されたスケーリング則 [23] の影響が大きい．以下の三つの冪乗則が知られている．

- モデルのパラメータ数を増やせば増やすほどモデルの性能は向上する．

- データセットのデータ数を増やせば増やすほどモデルの性能は向上する．
- モデルの学習に使用する計算資源を増やせば増やすほどモデルの性能は向上する．

上記のスケーリング則に従うと，小規模なモデルを開発するのではなく，大量の計算機資源を用いて巨大なモデルを大規模なデータで学習させるのが最適であると考えられてきた．その結果，大規模な LLM が開発され，性能が飛躍的に向上していった．2025 年 1 月現在でもこの流れは変わっていない．DeepSeek-R1 の論文 [16] でも，小規模モデルに対して学習を行うよりも，大規模モデルの知識を小規模モデルに継承させる蒸留によってモデルを小規模化した方が性能の面でも得られるメリットは大きいと主張されている．また，モデルのパラメータ数と学習に使用するデータ数との間に成立する Chinchilla 則 [24] も知られている．モデルのパラメータ数と学習データ数の比率は，1:20 が最適であると言われている．例えば，7B 程度のパラメータ数のモデルであった場合，パラメータ数の 20 倍である 140B のデータセットを用いて学習させるのが良いとされている．Chinchilla 則が学習時のコストのみを考慮していたのに対し，学習時と推論時のコストも考慮したスケーリング則 [25] も知られている．この論文では Chinchilla 則で最適とされたパラメータ数のモデルよりも，さらに小規模なモデルを大量のデータを用いて学習させている．Chinchilla 則はパラメータ数の 20 倍ほどのトークン数のデータ量を学習させるのが良いとされていたのに対し，推論時のコストも考慮するとパラメータあたりのトークン数を 10,000 倍にしてもモデルの性能が向上することが分かっている．さらに，OpenAI の o1 では，推論時にもスケーリング則が成り立つことが示唆されている[§]．推論時のスケジュール則とは，推論に時間をかければかけるほどモデルの性能は向上する，という法則である．我々人間においても，難しい問題であっても考える時間を増やすと解けるようになる，という経験が往々にしてある．o1 においても人間と同様の現象が見られることが確認された．難しい推論を行うベンチマークデータセットのうち，特に数学や物理学の問題において顕著な性能向上が確認された．

[§]https://openai.com/index/learning-to-reason-with-llms/

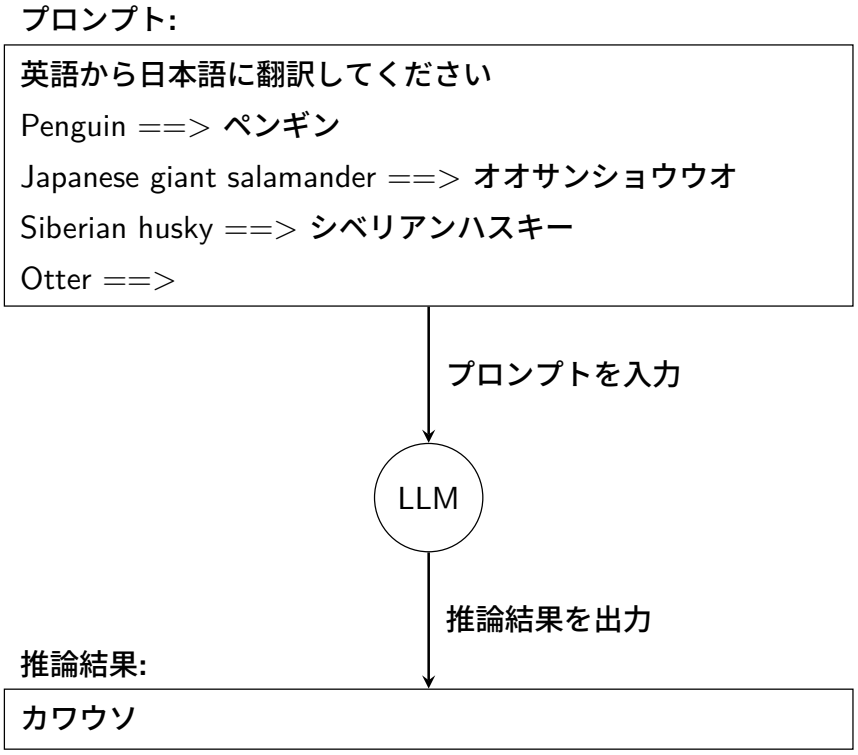

図 2.5: 文脈内学習（In Context Learning; ICL）の流れ

二つ目は，文脈内学習（In Context Learning; ICL）[26] である．図 2.5 が ICL の仕組みを示している．ICL とは，追加の学習を行わずに与えられた文脈から LLM がタスクを推測し，実行する能力を指している．例えば翻訳タスクの場合，プロンプトに「Penguin ==> ペンギン」といった少数の例を加えると，モデルはプロンプトからタスクの内容を推測し，性能が向上する．特に，例を一つだけ与える場合は one-shot 学習，例を複数与える場合は few-shot 学習，例を全く与えない場合は zero-shot 学習と呼ばれる．ICL の能力はプロンプトに追加する例の数を増やすほど向上し，LLM のモデルサイズが大きくなるほど向上する．また，ICL ではパラメータを更新する必要がないため，学習コストはかからず，タスクごとにモデルを作り変える必要もないというメリットもある．LLM 以前の手法では，指示と正解の回答ペアを用いてモデルの予測を修正するファインチューニングが主流であった．ファインチューニングはモデルのパラメータを更新する必要があるため，学習コストがかかる，或いは，解きたいタスクごとにモデルが変わるといったデメリッ

トがあった．

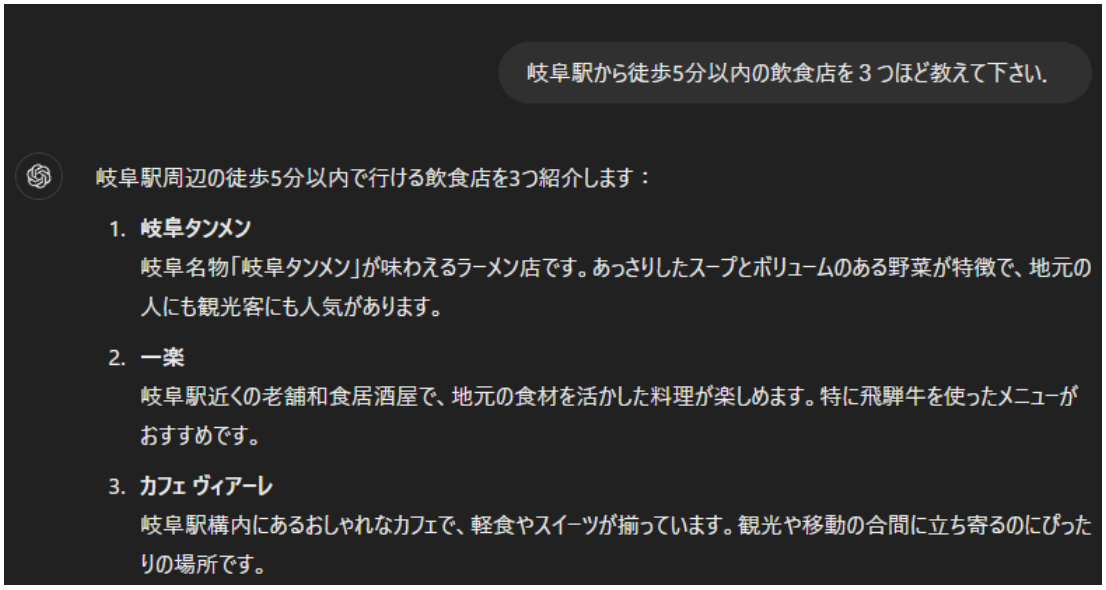

図 2.6: ハルシネーションの例

三つ目は，ハルシネーション（Hallucination）[27][¶]である．ハルシネーションとは，LLM が一見正しそうに見えるが誤りを含んだ出力を行う現象である．図 2.6[‖]の岐阜駅から徒歩 5 分以内の飲食店について ChatGPT の GPT-4o に尋ねた例を用いて説明する．図からも分かるように，ChatGPT は飲食店を三つほど教えて下さいという指示には従っており，どの飲食店も実在するため正しそうに見える．しかし，実際には岐阜タンメンという飲食店は岐阜県内には存在するが，岐阜駅周辺には存在しないため，誤りを含んだ回答となっている．LLM はこうした虚偽の回答を生成する場合もあるため，LLM を利用する際には生成された情報を鵜呑みにすることなく，本当に正しいかどうかを検証する必要がある．

[¶]https://confit.atlas.jp/guide/event/deim2024/static/tutorial?lang=ja#tu-d-2
[‖]https://chatgpt.com/share/678d17b1-9ae4-8001-8b7c-7e3219407438

2.2.1 アーキテクチャ (Transformer と Attention)

LLM は Transformer アーキテクチャ [28] をベースとして構築されることが一般的である．Transformer とは，Attention だけで構成されたニューラルネットワークの構成単位である．Transformer 以前のアーキテクチャである RNN や LSTM を用いた言語モデルもいくつか開発されていたが，これらのアーキテクチャには主に二つの構造的な問題があった．一つ目は，長い系列データを扱うのが苦手だという問題である．十数単語程度の文章であれば問題なく扱えるが，それ以上の単語数の文章になるとモデルの性能が著しく低下することが知られていた [29]．二つ目は，並列化して計算できないという問題である．RNN や LSTM は与えられたデータを逐次的に処理する機構である．そのため，並列化が行えずに学習に時間がかかるということが知られていた．そこで開発されたのが，従来の構成単位であった LSTM や RNN を用いず，Attention のみで構成された Trasnformer である．Trasnformer には，長い系列のデータであっても性能を低下させることなく扱うことができる，並列化して効率的に扱うことができるという二つの利点が存在する．

Attention とは，モデルに入力されたデータの特定の部分に注目してデータを処理する機構である．入力されたデータを全て同じ重要度で扱うのではなく，データごとに重要度を学習，計算し，どのデータを用いるかを決めている．重要度はクエリ（Query; Q）ベクトル，キー（Key; K）ベクトル，バリュー（Value; V）ベクトルと呼ばれる三種類のベクトルを用いて計算される．Attention は以下の式で算出される：

$$\text{Attention(Q, K, V)} = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.2.1}$$

ここで，$softmax$ は入力されたベクトルデータを最小値 0 から最大値 1 の確率分布に変換する Softmax 関数，$\sqrt{d_k}$ とはキーベクトルの次元数の平方根を表している．Attention では，クエリベクトルとキーベクトルの類似度を計算し，計算結果に基づいてバリューベクトルを取り出すという仕組みになっている．

図書館で本を探している状況を例にして Attention の計算の様子，及び，クエリベクトル，キーベクトル，バリューベクトルを説明する．まず，ある来館者が大規模言語モデルに関係する書籍を探しているとする．この時，クエリベクトルは大規

模言語モデルそのものを表している．つまり，探したい情報を表しているのがクエリベクトルである．次に，来館者は図書館に設置されている蔵書検索システムを用いる．図書館に収蔵されている各書籍には，情報学やミステリーといったジャンル，小説や新聞といったカテゴリ，発行年や出版社などのラベルが付与されている．この各書籍が持っているラベル情報を表しているのがキーベクトルである．最後に，蔵書検索システムが来館者の探し求めている書籍に近いと判断した書籍の情報を提示する．システムでは，来館者が探している書籍情報を入力とし，探している書籍と蔵書との関連性の高さや低さを表す類似度，すなわち，重要度を算出している．この類似度は，探したい情報を表すクエリベクトルと各書籍が持っているキーベクトルの内積を計算して求められる．ここで，提示された書籍の情報を表しているのがバリューベクトルである．Attention では，このような流れで重要度の高いデータを取り出すという操作を行っている．

Attention にはいくつかの種類がある．ここでは，代表的な Attention である Scaled Dot-Product Attention，Self-Attention，Multi-Head Attention について説明する．(2.2.1) 式で示しているのは，Scaled Dot-Product Attention である．クエリベクトルとキーベクトルの内積を計算した後に Softmax 関数を適用し，重要度を計算している．Self-Attention と Multi-Head Attention を構成しているのが Scaled Dot-Product Attention である．Self-Attention は入力されたデータ内の全要素に対し，一つの要素とそれ以外の全要素との重要度を計算する機構である．入力されたデータ，つまり，一つの文章に含まれる単語どうしの重要度を計算することで，文脈全体を考慮した処理を行っている．Multi-Head Attention は，入力されたデータに対して複数の Scaled Dot-Product Attention を計算する機構である．Multi-Head とあるように複数の Scaled Dot-Product Attention を並列に接続し，それぞれの Scaled Dot-Product Attention で異なる計算を行い，統合する処理を行っている．複数の Head を用いて計算を行うことにより，複数の異なる重要度で入力されたデータを扱うことができる．

Attention は Transformer における計算のほとんどを占めるため，効率的に行う必要がある．効率的な Attention 機構の代表例としては，Attention の計算を Sparse にした Sparse Transformer，Sparse な Attention を複数組み合わせた Big Bird，ハードの性質をうまく生かしてメモリへのアクセス回数を削減した

FlashAttention が知られている．

2.2.2 専門家の集合：新たなアーキテクチャ (Mixture of Experts; MoE)

2023 年 3 月以前は 2.2.1 節で述べたように，Transformer を複数重ね合わせた巨大な単一モデル構造を持つ LLM が主流であった．しかし，2023 年 3 月に GPT-4 が発表されて以降，MoE と呼ばれるモデル構造を持つ LLM が台頭してきた．GPT-4 は正式に MoE であると発表されたわけではないが，リーク情報より MoE モデルを採用していると言われている．

MoE [30] とは，エキスパートと呼ばれる専門家モデルの集合から入力に応じて適切な専門家モデルを選択し，計算効率を向上させる手法である．MoE の概念が初めて提唱された時期は，本論文執筆当時から 30 年ほど前の 1990 年代 [31] である．近年の LLM のスケール化に伴い，計算コストを抑えつつも LLM の性能向上が見込めることから注目されている．

MoE を構成するエキスパートは，それぞれが独立したニューラルネットワークの LLM である．これまでの巨大な単一モデル構造を持つ LLM では，一つのモデルが数学や法律，物理学といった異なる分野の問題を解いていた．一方，MoE では数学担当，法律担当，物理学担当のようにそれぞれの分野に特化したエキスパートを用い，与えられた問題に応じて最適なエキスパートを割り当てる仕組みとなっている．与えられた問題を担当するエキスパートの選択は，ルーターネットワークと呼ばれるニューラルネットワークによって行われる．ルーターネットワーク自身もニューラルネットワークであるため，学習を行うことによって最適なエキスパートを選択できるようになる．

MoE モデルにも 2.2 節で述べたスケーリング則が成立することが分かっている [32, 33]．Aidan ら [32] の研究では，エキスパートの数を 2 のべき乗数ごとに増やしていくと，MoE モデルの性能が向上していくことが示されている．また，MoE モデル全体のパラメータ数を増やしていくと，MoE モデルの性能が向上していくことも示されている．つまり，MoE モデルにおいても巨大な単一モデル構造を持つ LLM で見られたスケーリング則が成立することが確認された．

本格的に MoE の LLM が研究され始めたのは，2023 年 12 月に発表された

Mixtral AI 社の Mixtral-8x7B [34] であろう．ここ一年の間に研究開発が進み，軽量な MoE ながら GPT-4o の性能を上回る DeepSeek-V3 [35] が 2024 年 12 月に発表された．

2.2.3 大規模言語モデルの作り方

LLM は三段階のステップを経て作成される．

1. 事前学習（Pre-training）
2. 指示学習（Instruction Tuning）
3. アラインメント学習（Alignment Tuning）

ただし，この流れとは全く異なる構築方法を採用した LLM である DeepSeek-R1 [16] が 2025 年に発表された．DeepSeek-R1 の構築方法については，本節の最後に述べる．

まず，事前学習について説明する．事前学習では，与えられた単語列を LLM への入力として次に続く単語を予測するタスクを通して，あらゆる知識を獲得する．Next Token Prediction と呼ばれるタスクを行い，以下の式を最小化するように学習を行う：

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(s_{t+1}|s_1, s_2, \ldots, s_t) \tag{2.2.2}$$

ここで，T は文章 S に含まれる単語数，s_t は文章 S の t 番目の単語，$P(s_{t+1}|s_1, s_2, \ldots, s_t)$ は単語列 $s_1, s_2, \ldots, s_t$ が与えられた時にモデルが次に出現すると予測した単語 s_{t+1} の条件付き確率を表している．例えば，「日本一高い山は」という単語列を入力として LLM に与えた場合，LLM は次に来る s_{t+1} として「富士山」と予測できるように学習を行う．このようにして，LLM は単語どうしの意味的な関係性や文脈を理解し，自然な文章を生成できるようになる．事前学習に使用されるデータセットとしては，Wikipedia や C4[**] といった様々なドメインを含んだウェブ上のテキストを用いることが多い．また，Dolma [36][††] のようにプロ

[**] https://www.tensorflow.org/datasets/catalog/c4
[††] https://huggingface.co/datasets/allenai/dolma

グラムのコードを含んだデータセットが使用されることもある．

次に，指示学習について説明する．指示学習では，与えられた指示に対して回答を行うタスクを通して，与えられた指示に従った回答を行う能力を獲得する．Supervised Fine-Tuning（SFT）と呼ばれるタスクを行い，以下の式を最小化するように学習を行う：

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{V} A_{i,j} \log P(y_{i,j}|I_i) \tag{2.2.3}$$

$$A_{i,j} = \begin{cases} 1 & \text{LLM の出力した単語と正解データの単語が同じ場合} \\ 0 & \text{異なる場合.} \end{cases} \tag{2.2.4}$$

ここで，N は指示学習のデータセットに含まれるタスク指示文のデータ数，V は LLM が生成した文章の単語数，$y_{i,j}$ は i 番目のタスク指示文に対して LLM が出力した文章，$A_{i,j}$ は $y_{i,j}$ の文章に含まれる j 番目の単語と正解データの文章に含まれる j 番目の単語が一致していれば 1 を出力し，不一致であれば 0 を出力する関数，$P(y_{i,j}|I_i)$ はタスク指示文に対して LLM が $y_{i,j}$ を出力した時の条件付き確率を表している．LLM の条件付き確率 $P(y_{i,j}|I_i)$ と正解データ $A_{i,j}$ を比較し，指示に従った正しい出力を行えるように学習を行う．

最後に，アラインメント学習について説明する．アラインメント学習では，人々の価値観に合った出力をできるようになるタスクを通して，人々が望む回答を行う能力を獲得する．Reinforcement Learning from Human Feedback（RLHF）と呼ばれるタスクを行う．RLHF では以下の二つのステップを交互に繰り返して学習を行う．

step 1　人々の好みをラベル付けしたデータセットの作成と報酬モデルの学習
step 2　報酬モデルによる評価を用いた強化学習

まず，step 1 では人々の代わりに「LLM が生成した文章の良さ」を評価する報酬モデルを作成する．アラインメント学習を行う対象の SFT モデルを用意する．報酬モデルの作成にあたり，一つのプロンプトに対して LLM が生成した複数の文章をアノテータが比較し，文章の良さのランキング順に並べ替えてデータセットを作成する．その後，ランキングをモデルに学習させることで，入力された文章に対し

てスコアを出力する報酬モデルを作成する．次に step 2 では，SFT モデルが報酬モデルのスコアを最大化するように学習を行う．報酬モデルの最大化では，近接方策最適化（Proximal Policy Optimization; PPO）と呼ばれる強化学習技術を用いる．PPO は，Actor と呼ばれる行動を行うエージェント，Critic と呼ばれる Actor の行動を評価し，価値を予測するエージェントの二つを交互に学習し，選択した行動がどれほど優れているのかを Generalized Advantage Estimation(GAE) で評価してフィードバックする方法である．強化学習における方策（Policy）の変化を抑え（Proximal）ながら報酬の最適化（Optimization）を行うことで，極端な方策の更新を行わず，安定した学習を行うことができる．

人々が望む回答にはいくつかの基準がある．ここでは, 代表的な基準である HHH [37] について述べる．HHH とは，安全かつ信頼できる LLM が満たすべき性質である．Helpful, Honest, Harmless の頭文字をとって HHH と呼ばれている．Helpful とは，ユーザにとってどれほど役立つかを示す指標である．ユーザのプロンプトに対して簡潔な回答を行うこと，回答を行う際に足りない情報がある場合はユーザに聞き返して情報を引き出すこと，ユーザのレベルに応じて回答を行うことが Helpful に該当する．Honest とは，回答が真実に基づいていて，誤解を招かないような正直さを示す指標である．正確な情報を生成すること，生成した情報が正しいかどうかの確信度を述べること，モデル自身が自身の能力と知っている知識について明らかにすることが Honest に該当する．Harmless とは，ユーザや社会にとって安全である無害さを示す指標である．攻撃的でないこと，差別的でないこと，悪意のある行為への協力を求められた場合は断ることが Harmless に該当する．そのほかの基準については，こちらの論文 [38] を参照されたい．

ただし，RLHF を行う上でいくつかの問題が存在する．一つ目は，Reward Hacking と呼ばれる知識の忘却に関する問題である．報酬モデルを最適化するだけでは，高いスコアをもらえるような文章を生成する方策を学習してしまう恐れがある．これでは，報酬モデルに過学習した SFT モデルが出来上がるのみで，意図したモデルではない．そこで，KL Penalty [39] と呼ばれる罰則項を追加して学習を行うことで，生成する文章が元の SFT モデルの出力から変わりすぎないように制限をかけることが一般的である．二つ目の問題は，Alignment Tax と呼ばれる問題である．RLHF は事前学習，指示学習と複数の学習ステップを経た後に行われ，

異なる学習が行われるたびに LLM のパラメータは更新される．そのため，RLHF を行っている時に，事前学習時に獲得した知識を忘却してしまう場合がある．そこで，ChatGPT の開発では事前学習に使用したデータの一部を再度学習させることで，RLHF に影響を与えることなく知識の忘却を防いでいる[‡‡]．

LLM は事前学習，指示学習，アライメント学習の三段階のステップを経て構築されるのが一般的である．ところが，2025 年 1 月 22 日にこれまでとは異なる方法で構築された LLM が発表された．DeepSeek-R1 [16][§§]とは，中国の DeepSeek 社によって開発された LLM である．SFT を行わずに RL(強化学習) を適用してモデルを構築したこと，比較的小規模な計算資源 [40] でありながらも OpenAI の o1 に匹敵するモデルを構築したことから非常に注目されているモデルである．これまでの LLM 構築では得られなかった新たな知見に溢れているため，ここに紹介したい．

DeepSeek-R1-Zero では，事前学習済みモデルに対して指示学習を行わず，純粋に強化学習のみを適用して大規模な学習を行ったモデルである．強化学習では PPO や DPO ではなく，Group Relative Policy Optimization(GRPO) [41] が用いられており，強化学習を数千ステップ繰り返すと OpenAI の OpenAI-o1-0912 に匹敵する性能を示すことができた．GRPO とは，PPO のように Critic エージェントを用いた価値の予測を行わず，同一のプロンプトに対する複数の回答の平均値を価値としてフィードバックする強化学習の手法である．Actor エージェントと同規模のニューラルネットワークである Critic エージェントを用いないため，計算コストを抑えながら学習を行うことができる．

これまでの LLM では，Chain-of-Thought(CoT) と呼ばれる思考過程をプロンプトによって LLM に与えることで，段階的に推論を行うことができるようになり，難しい問題に対する回答精度を高めることができた．ところが，DeepSeek-R1-Zero では CoT プロンプティングのように明示的な指示を LLM に与えることをせず，強化学習によって LLM が自ら段階的に推論を行う能力を獲得することができたと報告されている．また，Self-Verification や Reflection のように LLM 自身が自分の推論過程を自己分析し，修正していく能力も獲得したとされている．さらに，推論途中にモデルが閃いた！と出力するようなアハモーメント (Aha Moment) と呼ば

[‡‡]https://openai.com/index/instruction-following/
[§§]https://github.com/deepseek-ai/DeepSeek-R1

れる現象も確認された．ただし，推論能力は獲得できたが，同じ文章を何度も生成する現象，複数の言語が混在して出力されてしまう現象が確認された．

そこで，DeepSeek-R1 では少量で高品質かつ長い CoT データを用いて事前学習済みモデルを訓練し，訓練後のモデルを初期値として DeepSeek-R1-Zero と同様の大規模な強化学習を行った．また，強化学習時の報酬関数に同一言語で生成できた場合に高いスコアを与える報酬関数を追加することで，DeepSeek-R1-Zero で発生していた複数の言語が混在して出力される現象を軽減した．

さらに，DeepSeek-R1 を用いて小規模な Qwen や Llama を蒸留する実験も行っている．実験の結果，強化学習を用いて小規模モデルを訓練しても推論能力を持ったモデルは作れないことが分かった．一方，高性能な DeepSeek-R1 を用いて作成した SFT データによる小規模モデルの蒸留では，小規模モデルに推論能力を付与することができたと報告されている．

2.2.4 分散並列学習と DeepSpeed

BERT のような比較的小規模な言語モデルは数千から数億パラメータ程度であり，一つのモデルを単一の GPU に転送して学習させることが可能であった．しかし，近年の大規模言語モデルは数百億パラメータ程度であり，一つのモデルを単一の GPU にアップロードして学習させることが難しくなっている．また，データセットのサイズも大規模化しているため，学習時間が膨大になってしまい，単一の GPU で学習させるのは難しいという問題もある．単一の GPU で学習できなくとも，複数の GPU を用いることができれば学習は可能である．そこで開発されたのが分散並列学習である．

分散並列学習とは，複数の GPU における並列化処理を用いた学習を行う技術である．単一のノードで複数の GPU を用いた学習をマルチ GPU 学習と呼び，複数のノードで複数の GPU を用いた学習をマルチノード学習と呼ぶ．並列する手法には以下の三つがある．

- データ並列
- パイプライン並列
- テンソル並列

データ並列とは，データセットを複数のサブセットに分割し，GPU ごとに異なるサブセットを用いて学習を行う並列化の手法である．データセットに含まれるデータ数を D_n，使用する GPU の数を G_n，学習にかかる時間を T とする．各 GPU に割り振られるデータ数は使用する GPU の数に反比例するため，$\frac{D_n}{G_n}$ と表わされる．また，学習にかかる時間も使用する GPU の数に反比例するため，$\frac{T}{G_n}$ と表わされる．つまり，理想的には学習時間を $\frac{1}{G_n}$ 倍に短縮することができる．しかし，並列化を行うのはデータのみであるため，各 GPU には同一のモデルのパラメータ，勾配情報，オプティマイザが配置されている．そのため，単一の GPU に乗せることができないモデルは学習を行うことができない．そこで考えられたのが，モデルそのものを並列化するパイプライン並列とテンソル並列である．

パイプライン並列とは，モデルを層ごとに分割し，GPU ごとに異なる層を配置して学習を行う並列化の手法である．ある LLM のパラメータサイズを P_n とする．各 GPU に割り振られる LLM のパラメータサイズは使用する GPU の数に応じて減少するため，各 GPU で使用するパラメータサイズは $\frac{P_n}{G_n}$ と表わされる．単一の GPU で使用するメモリ量を減らすことができるため，大規模なモデルであっても学習させることが可能になるメリットが存在する．多くの LLM は Transformer アーキテクチャを用いて構築されており，Transformer ブロックという層が繰り返し接続されている．そのため，Transformer を用いた LLM に対しパイプライン並列を適用した場合，各 GPU には同じ Transformer ブロックが配置されているため計算時間に大きな差はなく，ボトルネックにはなりにくい．一方，Transformer 以外の構造が含まれていた場合は計算時間に差が生じ，計算に時間のかかる層の処理が終わるまでの待ち時間が生じてしまう．そのため，パイプライン並列は繰り返し構造を含まないようなモデルにおいては有効でない場合もある．

テンソル並列とは，モデルの層をさらに細かいパーツに分割し，GPU に配置して学習を行う並列化の手法である．Transformer ブロックを細かいパーツに分割すると，Attention 機構や全結合層に分けられる．Attention 機構を GPU_A，全結合層を GPU_B のように異なる GPU に配置することで，パイプライン並列よりも使用する GPU メモリ量を削減することができる．テンソル並列もパイプライン

並列と同様に，GPU によって計算時間に差が出てしまう場合，学習時間の高速化が見込めない場合も存在する．

ただし，全ての並列化手法には共通するデメリットが存在する．異なる GPU に異なるモデルの情報を配置して計算する場合，GPU 間で情報をやり取りする必要性が生じる．情報をやり取りする量を表す通信量は，使用する GPU の数が増えるにつれて増大する．また，パイプライン並列やテンソル並列のようにモデルを細かなパーツに分割すればするほど，通信量は増大する．そのため，並列化手法は必ずしも学習時間の削減に寄与しない．一般には，NVLink[¶¶]などの技術を用いて GPU 間で高速な通信が行える場合にのみ計算時間の短縮が見込まれる．

分散並列学習を行うには公開されているライブラリを使用することが一般的である．有名なライブラリとしては，DeepSpeed[***]や Megatron-DeepSpeed[†††]が知られている．DeepSpeed は Microsoft 社によって開発された分散学習用のライブラリであり，DeepSpeed ZeRO [42] には三つのステージがある．

Stage 1　オプティマイザのみ並列化を行い，勾配とモデルのパラメータの並列化は行わない．

Stage 2　オプティマイザと勾配の並列化を行い，モデルのパラメータの並列化は行わない．

Stage 3　オプティマイザと勾配，及び，モデルのパラメータの並列化を行う．

ここで，AdamW オプティマイザと混合精度を用いた LLM の学習を例に，DeepSpeed ZeRO の適用によってメモリ使用量がどのように変化するかを説明する．パラメータ数が P_nB の LLM を G_n 個の GPU を用いて学習させる場合を考える．モデルのパラメータや勾配は FP16 で保持されるため，それぞれ $2P_n$ バイトのメモリを使用する．また，AdamW オプティマイザでは混合精度によって FP32 で保持されるため，モデルのパラメータと一次モーメント（勾配の移動平均），及び，二次モーメント（勾配の二乗の移動平均）はそれぞれ $4P_n$ バイトのメモリを使用する．そのため，オプティマイザでは $12P_n$ バイト，勾配では $2P_n$ バイト，モデルのパ

[¶¶] https://www.nvidia.com/ja-jp/design-visualization/nvlink-bridges/

[***] https://github.com/microsoft/DeepSpeed

[†††] https://github.com/microsoft/Megatron-DeepSpeed

ラメータでは $12P_n$ バイトのメモリを使用する．並列化処理を行わない場合に使用するメモリ量は，$2P_n + 2P_n + 12P_n$ より $16P_n$ バイトである．DeepSpeed ZeRO Stage 1 ではオプティマイザのみ並列化処理を行う．そのため，$2P_n + 2P_n + \frac{12P_n}{G_n}$ バイトのメモリを使用する．DeepSpeed ZeRO Stage 2 ではオプティマイザと勾配の並列化処理を行う．そのため，$2P_n + \frac{2P_n + 12P_n}{G_n}$ バイトのメモリを使用する．DeepSpeed ZeRO Stage 3 ではオプティマイザと勾配，及び，モデルのパラメータの並列化処理を行う．そのため，$\frac{2P_n + 2P_n + 12P_n}{G_n}$ バイトのメモリを使用する．例えば，32 個の GPU を用いて 7B の LLM の学習を行った場合，並列化適用前では 112GB，Stage 1 では 30.625GB，Stage 2 では 17.0625GB，Stage 3 では 3.5GB の GPU メモリを使用することになる．

本研究では，DeepSpeed ZeRO Stage 2 を用いて実験を行っている．今回の実験で使用できる GPU は，48GB の VRAM を持つ NVIDIA RTX A6000 が 6 枚，24GB の VRAM を持つ NVIDIA TITAN RTX が 6 枚，NVIDIA GeForce RTX 3090 が 1 枚，16GB の VRAM を持つ NVIDIA RTX A4000 が 6 枚であった．各サーバに設置されている CPU は RAM が 128GB から 256GB 程度であった．Stage 1 を用いた場合，使用する GPU メモリが 30GB 程度であるため，NVIDIA RTX A6000 のみしか使えない．この場合，最大限使用できる GPU のうち，32% の GPU しか活用することができない．また，Stage 3 を用いた場合，使用する GPU メモリが 4GB 程度であるため，全ての GPU を使用することができる．しかし，使用する GPU メモリの減少に伴って使用する CPU メモリが増加するため，256GB 程度の CPU では学習を行うことができない．Stage 2 を用いた場合，使用する GPU メモリが 17GB 程度であるため 63% もの GPU を活用することができ，128GB 程度の CPU であれば学習を行うことができる．そこで，DeepSpeed ZeRO Stage 2 を用いて実験を行った．補足説明については，A.1 節を参照されたい．

2.2.5 Docker と環境構築

実験の過程で，ライブラリが特定のバージョンでないと LLM の学習を行えないといった問題が頻繁に発生した．この場合，実験を行うホストマシンでライブラリのバージョンを変更することで問題を解消することができる．しかし，ホストマシンでライブラリのバージョンを変更してしまうと，他の利用者が使用しているプログラムが正常に実行できないといった別の問題が発生してしまう可能性がある．解決策の一つとして Docker を用いることが考えられる．

Docker とは，アプリケーションやライブラリなどをコンテナと呼ばれる隔離環境に集約し，動作させる仕組みである．似た概念として仮想マシンがある．仮想マシンは，プログラムやアプリケーションに加え，ハードウェアも仮想化することで環境を作る仕組みである．ハードウェアの仮想化を行うのが仮想マシンであるのに対し，OS レベルの仮想化を行うのが Docker である．

Docker と仮想マシンの違いについて住居の設計を例にして説明する．Docker は部屋単位で設計する方法である．例えば，ゲーム専用の部屋が欲しければゲーム部屋の設計書 (後述する Dockerfile のこと) を作成し，大きいキッチンが欲しければキッチンを大きくした台所の設計書を作成する．ただし，電気や水道などのインフラ設備 (ホスト OS のカーネルに相当) などは元からあるものを使うため，設計しなおす必要はない．一方，仮想マシンは住居単位で設計する方法である．所望の機能を持たせた部屋が欲しければ Docker と同様に設計書を作成するだけではなく，インフラ設備なども設計する必要がある．仮想マシンは住居単位で設計するため必要な資材が多く，建築時間 (起動時間) もかかる．しかし，Docker は部屋単位で設計するため必要な資材が少なく，建築時間も短くすることができる．

Docker のメリットには，「軽量性」，「移植性」，「再現性」の三つがある．軽量性は，ホスト OS のカーネルをそのまま使用するため，仮想マシンの時よりも省リソースで実行できる仕組みによって実現される．必要なアプリケーションやライブラリのみをコンテナに含むため，起動時間が短縮される．移植性は，アプリケーションとライブラリを Docker イメージにパッケージ化する仕組みで実現される．Windows と Mac のような異なる環境であっても，同じように動作させることが可能となる．再現性は，環境全体を Docker イメージとして保存するため，いつ誰がどこでコンテナを作成しても，全く同じコンテナが作成できる仕組みで実現される．

同じ Docker イメージや Dockerfile を用いて環境構築を行えば，全く同じ Docker コンテナを作成することができる．

Docker に関係する以下の用語について整理する．

Docker イメージ　Docker コンテナを作るための設計図である．アプリケーションの動作に必要なファイルや設定が書き込まれている．

Docker コンテナ　Docker イメージから作成される，アプリケーションが実際に動いている隔離環境である．

Dockerfile　Docker イメージを作成するための手順を示した命令書である．使用するイメージや追加するソフトウェア，実行するコマンドなどを記述することで，誰でもどこでも全く同じ Docker イメージを作成できる．

Docker レジストリ　Docker イメージを保管する倉庫である．Docker Hub[‡‡‡] と呼ばれるオープンなレジストリがあり，世界中のだれかが作成したイメージを利用することができる．

Docker compose　複数のコンテナをまとめて管理するためのツールである．YAML 形式の docker-compose.yaml に複数のコンテナを連携させるための設定やマウント設定を記載しておくことで，一つのコマンドで Docker コンテナを作成することができる．

また，Docker のライフサイクルについても整理する．イメージからコンテナを作成してコンテナ内で作業を行い，作業が終わり次第コンテナを削除する，という流れ (ライフサイクル) が基本的となる．

1. イメージの取得と作成　`docker pull` コマンドを用いて Docker Hub に公開されているイメージを取得する．或いは，自作した Dockerfile から `docker build` コマンドを用いてイメージを作成する．
2. コンテナの作成と実行　`docker run` コマンドでイメージからコンテナを作成し，実行する．
3. コンテナの使用　コンテナ内で作業したいときにのみコンテナを稼働させる．`docker stop` コマンドを用いて起動中のコンテナを停止し，`docker start`

[‡‡‡] `https://hub.docker.com/`

コマンドを用いて停止中のコンテナを起動する．

4. コンテナの整理整頓　不要となったコンテナは `docker rm` コマンド，不要となったイメージは `docker rmi` コマンドで削除する．また，ネットワーク設定や長期間使用していないイメージなどは `docker system prune` コマンドを用いると自動的に削除される．

2.3　機械学習モデルに対する攻撃

機械学習モデルに対するいくつかの攻撃手法は，AI セキュリティ 情報発信ポータル[§§§]やサーベイ論文 [43, 44] にて詳細にまとめられている．これらの手法の中にはメンバーシップ推論攻撃と抽出攻撃のように非常によく似ており区別が難しいものも存在するため，それぞれの手法について概説する．

2.3.1　データ汚染 (Data Poisoning)

データ汚染 (Data Poisoning) とは，攻撃者がモデルの学習時に悪意をもってモデルの推論を狂わせる，或いは，性能を低下させるような特定のデータを混入させる手法である．悪意のあるデータを学習させて攻撃対象のモデルを特定のクラスに誤って分類させるような方法，攻撃対象のモデルに特定の人物や組織に対して否定的な意見を生成させるような方法が知られている．似た概念として，本来はパンダである画像を猿であるとモデルに誤認識させるといった Adversarial Patches が知られている．Data Poisoning がモデルの学習時に悪意あるデータを混入させてモデルに誤った判断をさせるのに対し，Adversarial Patches はモデルの推論時にデータを混入させてモデルに誤った判断をさせる手法である．

[§§§] `https://www.mbsd.jp/aisec_portal/index.html`

2.3.2 メンバーシップ推論攻撃 (Membership Inference Attacks)

メンバーシップ推論攻撃 (Membership Inference Attacks) とは，攻撃者があるデータがモデルの学習に使用されたデータであるかどうかを特定する手法である．テキストデータの一部を攻撃対象のモデルにプロンプトとして与えて続きの文章を生成させる方法，攻撃対象のモデルの勾配や出力ベクトルをもとに学習に使用されたデータかどうかを判定する分類器を作成する方法が知られている．

LLM に対する MIA の先行研究での着眼点とは異なり，提案手法ではこれまで活用されてこなかったパラメータの変化量に注目している．LLM はパラメータ数が非常に多いため，パラメータそのものを扱って分析する研究はコストの面からも難しく，敬遠されがちである．本研究では敬遠されがちなパラメータに対して真正面から取り組み，分析を行った．また，本研究は LLM に対して攻撃を行うための手法開発を行っているわけではない．LLM は学習したデータや知識をどのように保存しているのかといった LLM の内部構造や知識保存の仕組みに関する観点，LLM が学習したデータを漏洩しないようにするためのプライバシー保護に関する観点，から研究を行っている．

2.3.3 抽出攻撃 (Extraction Attacks)

抽出攻撃 (Extraction Attacks) とは，攻撃者がモデルの重みや訓練に使用されたデータを盗み取る手法である．攻撃対象のモデルにランダムなベクトルデータを与えて学習を行い，モデルの勾配が変化する方向のデータを学習させる方法，攻撃対象のモデルにデータを与えた時の出力結果からモデルを複製する方法が知られている．Membership Inference Attacks はあるデータがモデルの学習に使用されたかどうかを特定するのに対し，Extraction Attacks はモデルの学習に使用されたデータを抜き取る手法である．

2.3.4 プロンプトインジェクション (Prompt Injection)

プロンプトインジェクション (Prompt Injection) とは，攻撃者がモデルに特定のプロンプトを与えることで，モデルに倫理的や法律的に問題のある有害な回答を

生成させる手法である．あらかじめ攻撃対象のモデルに設定されたシステムプロンプトを盗み出す Prompt Leaking という方法，悪意をもった攻撃者が困っている人になりきり，困っている人を助けようとする善意に付け込んで攻撃対象のモデルから有害な回答を引き出す方法が知られている．

2.3.5 ジェイルブレイク (Jailbreak)

ジェイルブレイク (Jailbreak) とは，モデルを構築した組織によって安全性や倫理的観点から制限されたガイドラインをかいくぐり，モデルに有害な回答を生成させる手法である．プロンプトと攻撃対象のモデルの回答結果のフィードバックから有害な回答を生成しやすいようにプロンプトを変更する方法，大量の攻撃パターンを自動的に生成して攻撃対象のモデルに与える方法が知られている．

2.4 評価指標

本研究では，提案手法の性能を測定するために AUROC（ROC-AUC）という評価指標を用いる．AUROC は真陽性率と偽陽性率を用いて算出される．AUROC，真陽性率，偽陽性率の説明のために，表 2.1 の混同行列を用いる．

混同行列とは，与えられたデータを二つのクラスに分ける分類モデルの予測結果を表形式で表した評価指標である．正解ラベルと書かれている行方向が実際のデータを表している．予測ラベルと書かれている列方向が分類モデルの予測結果を表し

表 2.1: 評価指標のための混同行列

		予測ラベル	
		陽性	陰性
正解ラベル	陽性	真陽性 （True Positive; TP）	偽陰性 （False Negative; FN）
	陰性	偽陽性 （False Positive; FP）	真陰性 （True Negative; TN）

ている．以下の四つの要素で構成されている．

真陽性　実際のデータが陽性であり，分類モデルの予測結果も陽性であった事例
偽陰性　実際のデータは陽性であり，分類モデルの予測結果が陰性であった事例
偽陽性　実際のデータが陰性であり，分類モデルの予測結果は陽性であった事例
真陰性　実際のデータが陰性であり，分類モデルの予測結果も陰性であった事例

2.4.1　真陽性率（True Positive Rate; TPR）

真陽性率とは，実際のデータが陽性であるもののうち，モデルが正しく陽性であると予測したものの割合を示している．真陽性率は，再現率や Recall とも呼ばれる．陽性であるものをどれだけ取りこぼすことなく陽性であると予測できたのかを評価したい場合，この評価指標が使用される．TPR の値は最小値 0.0 から最大値 1.0 の範囲を取り，1.0 に近付くほど予測精度の高いモデルであると評価できる．TPR は以下の式で算出される：

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2.4.1}$$

2.4.2　偽陽性率（False Positive Rate; FPR）

偽陽性率とは，実際のデータが陰性であるもののうち，モデルが誤って陽性であると予測したものの割合を示している．陰性であるものをどれだけ誤って陽性であると予測したのかを評価したい場合，この評価指標が使用される．FPR の値は最小値 0.0 から最大値 1.0 の範囲を取り，0.0 に近付くほど予測誤りの少ないモデルであると評価できる．FPR は以下の式で算出される：

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}} \tag{2.4.2}$$

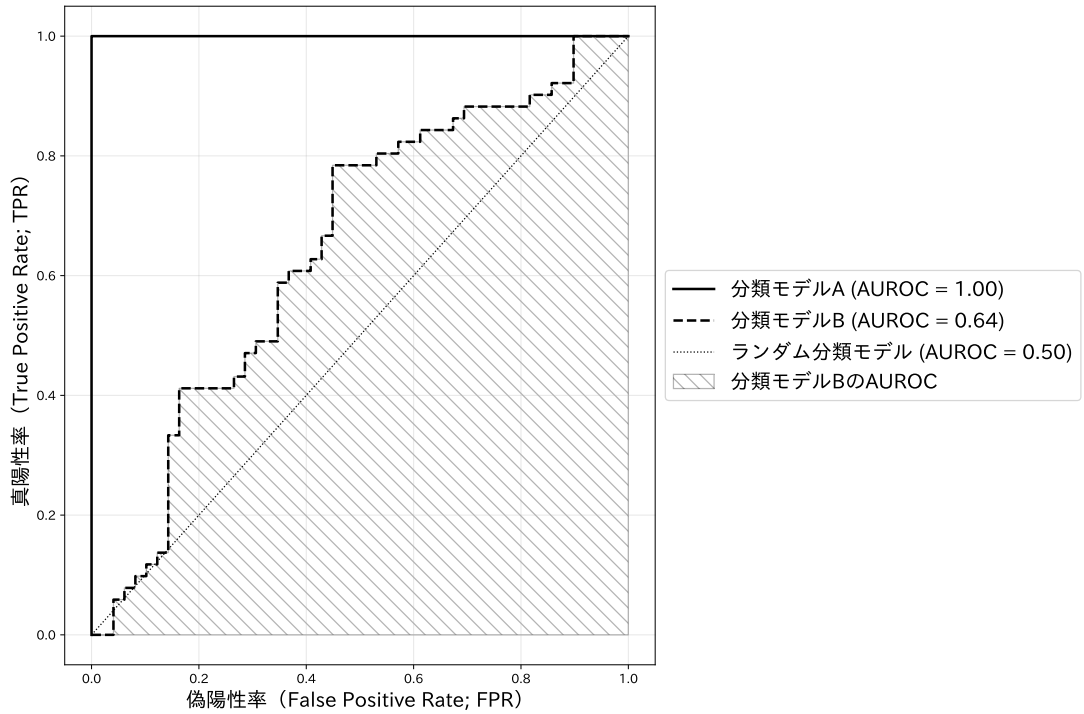

図 2.7: ROC 曲線

2.4.3 ROC 曲線

ROC（Receiver Operating Characteristic）曲線とは，分類モデルの性能をグラフ化した評価指標である．閾値 τ を最小値 0.0 から最大値 1.0 まで動かした時の真陽性率と偽陽性率を用い，異なる閾値での TPR と FPR をプロットして作成するグラフである．

ROC 曲線の例を図 2.7 に示した．分類モデル A は完璧に予測を行うことのできるモデル，分類モデル B は誤って予測することのあるモデルである．分類モデル A の ROC 曲線を実線，分類モデル B の ROC 曲線を破線で示している．性能の低いモデルであるほど，点線で示されたランダムな予測を行う分類モデルの ROC 曲線に近付いていく．

2.4.4 AUROC

AUROC とは, 評価したいモデルの ROC 曲線, TPR=0.0 の直線, FPR=1.0 の直線からなる三つの曲線で囲まれた下部領域面積である．論文によっては AUROC, AUC（Area Under the ROC Curve），ROC-AUC と呼ばれるが，全て同じものを示している．本論文では AUROC と呼んでいる．AUROC は最小値 0.0 から最大値 1.0 の範囲を取り，1.0 に近付くほど分類性能の良いモデルであると評価できる．

図 2.7 の斜線部は，誤って予測することのある分類モデル B における AUROC を示している．性能の低いモデルであるほど，点線で示されたランダムな予測を行う分類モデルの AUROC である 0.5 に近付いていく．

実際の AUROC の計算では，有限個のデータセットでの予測結果に基づくため，図 2.7 に示したような離散的な AUROC を算出している．そのため，いくつかの閾値 τ における TPR と FPR をサンプリングし，AUROC を算出することになる．AUROC は台形近似を用いた以下の式で算出される：

$$\mathrm{AUROC} = \sum_{i=1}^{n-1} (\mathrm{TPR}(\tau_i) + \mathrm{TPR}(\tau_{i+1})) \cdot \frac{\mathrm{FPR}(\tau_{i+1}) - \mathrm{FPR}(\tau_i)}{2} \tag{2.4.3}$$

ここで，$\mathrm{TPR}(\tau)$ は閾値 τ における真陽性率，$\mathrm{FPR}(\tau)$ は閾値 τ における偽陽性率，τ_i は i 番目の閾値，n は使用される閾値の総数を表している．

第 3 章　関連研究

MIA に関する先行研究について述べる．Hu ら [5] は MIA の研究について包括的に調査した．攻撃の状況設定には，攻撃者が攻撃対象のモデルの予測結果や予測ベクトルといった出力情報にのみアクセスできるブラックボックス設定，攻撃対象のモデルに関する勾配やパラメータといった内部情報にもアクセスできるホワイトボックス設定がある．また攻撃手法は，学習済みデータかどうかを特定する分類器を構築する分類器ベースの手法，学習済みデータかどうかを特定するための特徴量を用いるメトリックベースの手法の二つに大別される．

Shokri ら [6] は 2017 年に MIA の概念を初めて提唱し，機械学習モデルに対する MIA 手法を開発した．この手法は，攻撃対象のモデルと似た振る舞いをするシャドウモデルを複数構築し，学習済みデータや未学習データをシャドウモデルに入力したときの出力ベクトルであるロジットを使用して訓練された攻撃モデルを用いて特定を行う．シャドウモデルは攻撃対象のモデルから得られる合成データや統計情報を用いて作成される．攻撃モデルは入力されたロジットが学習済みデータか未学習データのどちらから得られたのかを分類する．実験の結果，攻撃対象のモデルが過学習している場合に MIA の精度が向上することが分かった．また，攻撃対象のモデルに正則化を行って過学習を抑制すると，MIA の精度が著しく悪化することが分かった．

Nasr ら [7] は，勾配や活性化値といった機械学習モデルの内部情報で訓練された分類器ベースの手法を開発した．この手法は，攻撃対象のモデルに画像データを入力したときの勾配や活性化値，損失値を使用して訓練された攻撃モデルを用いて特定を行う．攻撃モデルでは，勾配を畳み込みニューラルネットワーク層，活性化値や損失値を全結合層を用いて処理を行い，学習済みかどうかの確率を出力する．攻撃対象のモデルが教師あり学習モデルの場合は確率の値をそのまま用い，教師なし学習モデルの場合はスカラ値を用いてクラスタリングを行う．実験の結果，既存手法と比較して MIA の精度が 20% 程度向上した．

Shi ら [8] は Min-K% Prob を開発した．この手法は，テキストデータを LLM に入力したときの各トークンごとの対数尤度を用い，確率の低い上位 K% に含まれるトークンの平均対数尤度をもとに特定を行う．平均対数尤度が閾値以上の場合，

入力されたテキストデータは学習に使用された可能性が高い．LLM で対数尤度を算出したときに，学習済みではないテキストデータには負の対数尤度の高いトークンが多く含まれているという仮説に基づいている．パラメータ数の異なる LLaMA モデル (7B, 13B, 30B, 65B) を対象とする実験の結果，パラメータ数の多いモデルほど MIA の精度が向上すると分かった．

Kaneko ら [9] は SaMIA(Sampling-based Membership Inference Attacks) を開発した．また，SaMIA を改良した SaMIA*zlib も開発している．この手法は，LLM にテキストデータの前半部分を入力し，前半部分に続くテキストデータの候補の生成を複数回行い，生成されたテキストデータの候補とテキストデータの後半部分との一致度をもとに特定を行う．一致度が閾値以上である場合，入力されたテキストデータは学習に使用された可能性が高い．LLM の事前学習時に，LLM に入力されたテキストデータに続くようなテキストデータの生成タスクを学習することに着目している．実験の結果，特にテキストデータの単語長が長い場合，SaMIA は有効であると分かった．

DeAlcala ら [45] は画像認識モデルを対象として，MIA の精度に影響を与える四つの要因について分析を行った．学習過程の中で一度しか学習されないデータよりも，何度も学習されたデータに対する MIA の精度の方が高いことが分かった．学習時のバッチサイズの大きさは MIA の精度に影響を与えないことが分かった．過学習を緩和する機構であるドロップアウト層を削除したモデルは，ドロップアウト層ありのモデルと比較して MIA の精度が高いことが分かった．同じアーキテクチャのモデルであっても，学習に使用した損失関数の種類によって MIA の精度が変化するということが分かった．

本研究では，テキストデータを LLM に入力して事前学習を行い，事前学習前後のパラメータの変化量に基づいて MIA を行う．パラメータを用いるため，パラメータにもアクセスできるモデルを対象としたホワイトボックス設定を想定している．また，提案手法は分類器を構築して MIA を行う手法ではなく，学習済みデータかどうかを特定するための特徴量を用いるメトリックベースの手法に分類される．既存手法とは異なり，本研究ではパラメータの変化量に着目する点が新しい．

第 4 章　提案手法

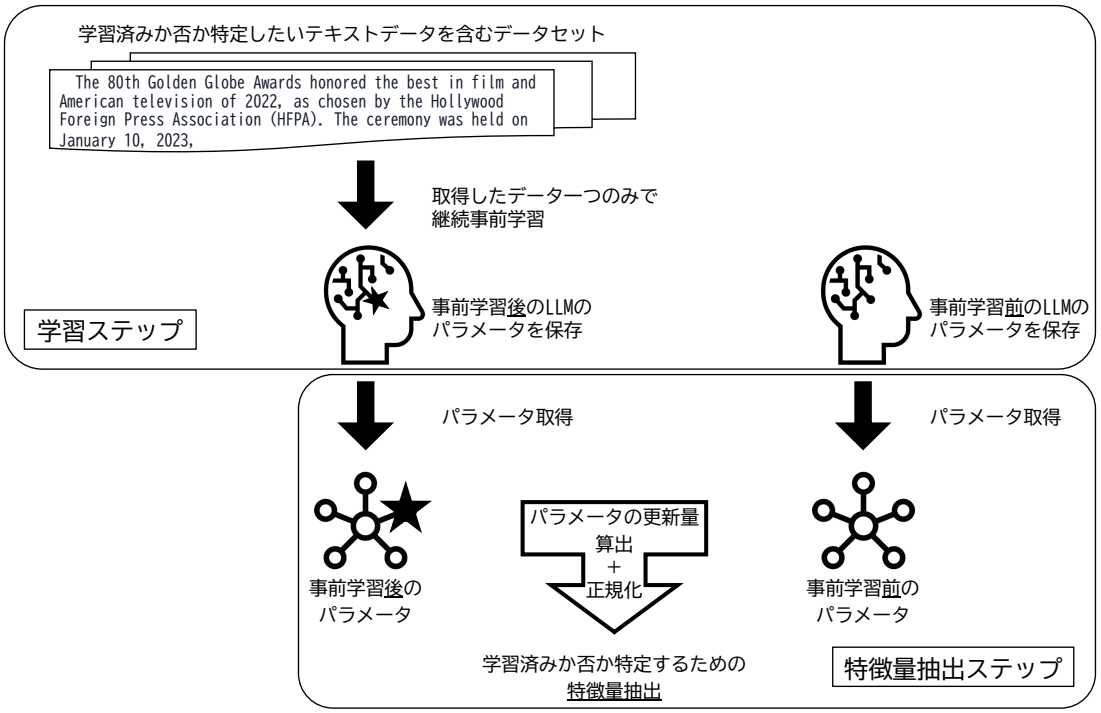

図 4.1: 提案手法における学習から特徴量抽出までの流れ

提案手法における特徴量抽出の手順を図 4.1 に示す．提案手法は，学習ステップと特徴量抽出ステップから構成される．

1. 学習ステップ (4.1 節)
   各テキストデータを一つずつ実験対象の LLM に入力し，継続事前学習を行う．
2. 特徴量抽出ステップ (4.2 節)
   事前学習前後のパラメータの変化量を正規化し，特徴量として抽出する．

4.1　学習ステップ

テキストデータ x_j を J 個含むデータセット $\mathcal{D} = \{x_j \mid j = 1, 2, \ldots, J\}$ が与えられていると仮定する．$\mathcal{D}$ から x_j を一つだけ取り出し，実験対象の LLM に対して継続事前学習を行う．j を 1 から J まで一つずつ変更し，4.1 節の操作を繰り返す．

継続事前学習とは，事前学習済みの LLM のパラメータを初期化せず追加で事前学習を行う手法であり，事前学習前のパラメータのうち学習に使用した x_j に関連するパラメータが勾配更新によって変化する．継続事前学習では，事前学習時に行う一般的なタスクである Next Token Prediction を用いる．トークン (Token) とは LLM がテキストデータを扱うときに使用するテキストデータの最小単位であり，単語や単語の部分文字列を指す．Next Token Prediction では，LLM は入力されたトークンに続くトークンを生成するように学習される．x_j の一部のトークンを LLM に入力し，入力されたトークンに続くトークンを正解ラベルとして与える．

4.2　特徴量抽出ステップ

特徴量抽出ステップは，以下の三つから構成される．

i. パラメータ行列の取得ステップ (4.2.1 節)
   事前学習前の LLM のパラメータ，および，4.1 節で作成した事前学習後の LLM のパラメータを取得する．
ii. パラメータ行列の変化量の算出ステップ (4.2.2 節)
   i. で取得したパラメータを用いて，事前学習前後のパラメータの変化量を求める．
iii. パラメータ行列の変化量の正規化ステップ (4.2.3 節)
   ii. で算出したパラメータの変化量に対し，正規化処理を行う．

実験対象の LLM が K 層で構成される場合，4.2.1 節から 4.2.3 節までの操作を K 回繰り返す．

4.2.1　パラメータ行列の取得

モデルのパラメータを保存したディレクトリから，l_k 層における事前学習前後のパラメータ行列を取得する．l_k 層における事前学習前のパラメータ行列 $\theta_{\text{before}}^{(l_k)}$ は次のように表される：

$$\theta_{\text{before}}^{(l_k)} = \left\{ \mathbf{W}_{\text{before}}^{(l_k)}, \mathbf{b}_{\text{before}}^{(l_k)} \right\} \tag{4.2.1}$$

$\mathbf{W}_{\text{before}}^{(l_k)}$ は事前学習前の LLM におけるパラメータの重み行列，$\mathbf{b}_{\text{before}}^{(l_k)}$ は事前学習前の LLM におけるパラメータのバイアス行列を表している．重み行列のサイズは $m^{(l_k)}$行 $n^{(l_k)}$列，バイアス行列のサイズは $m^{(l_k)}$行 1 列 である．ここで，$m^{(l_k)}$ は l_k 層における出力のユニット数，$n^{(l_k)}$ は l_k 層における入力のユニット数を表す．l_k 層における x_j の事前学習後のパラメータ行列 $\theta_{\text{after}}^{(l_k)}(x_j)$ は次のように表される：

$$\theta_{\text{after}}^{(l_k)}(x_j) = \left\{ \mathbf{W}_{\text{after}}^{(l_k)}(x_j), \mathbf{b}_{\text{after}}^{(l_k)}(x_j) \right\} \tag{4.2.2}$$

$\mathbf{W}_{\text{after}}^{(l_k)}$ は事前学習後の LLM におけるパラメータの重み行列，$\mathbf{b}_{\text{after}}^{(l_k)}$ は事前学習後の LLM におけるパラメータのバイアス行列を表している．$\theta_{\text{before}}^{(l_k)}$ と $\theta_{\text{after}}^{(l_k)}(x_j)$ のサイズはいずれも $m^{(l_k)}$行 $(n^{(l_k)}+1)$ 列 である．

4.2.2　パラメータ行列の変化量の算出

4.2.1 節で取得した $\theta_{\text{before}}^{(l_k)}$ と $\theta_{\text{after}}^{(l_k)}(x_j)$ を用い，事前学習前後のパラメータ行列の変化量を算出する．まず，計算を簡単にするために事前学習前後のパラメータ行列をベクトルに一次元化する．l_k 層における事前学習前のパラメータ行列を一次元化した列ベクトル $\mathbf{v}_{\text{before}}^{(l_k)}$，および，$l_k$ 層における x_j の事前学習後のパラメータ行列を一次元化した列ベクトル $\mathbf{v}_{\text{after}}^{(l_k)}(x_j)$ は次のように表される：

$$\mathbf{v}_{\text{before}}^{(l_k)} = \text{vec}(\theta_{\text{before}}^{(l_k)}), \quad \mathbf{v}_{\text{after}}^{(l_k)}(x_j) = \text{vec}(\theta_{\text{after}}^{(l_k)}(x_j)) \tag{4.2.3}$$

$\text{vec}(\cdot)$ は行列を列単位で順に連結し，一列の列ベクトルに変換する演算子である．$m^{(l_k)}$行 $(n^{(l_k)}+1)$ 列 である $\theta_{\text{before}}^{(l_k)}$ と $\theta_{\text{after}}^{(l_k)}(x_j)$ に対して変換を行った $\mathbf{v}_{\text{before}}^{(l_k)}$，および，$\mathbf{v}_{\text{after}}^{(l_k)}(x_j)$ のサイズはどちらも $m^{(l_k)} \times (n^{(l_k)}+1)$ 行 1 列 である．

次に，事前学習前のパラメータベクトル $\theta_{\text{before}}^{(l_k)}$ と事前学習後のパラメータベクトル $\theta_{\text{after}}^{(l_k)}(x_j)$ との変化量をユークリッド距離で計算する．x_j の l_k 層における変化量 $\delta^{(l_k)}(x_j)$ は以下の式で計算する：

$$\delta^{(l_k)}(x_j) = \left\| \mathbf{v}_{\text{after}}^{(l_k)}(x_j) - \mathbf{v}_{\text{before}}^{(l_k)} \right\|_2 \tag{4.2.4}$$

$\|\cdot\|_2$ はベクトル要素の二乗和の平方根を表している．l_k 層における事前学習前後のパラメータ行列の変化量は，スカラ値の集合として次のように表される：

$$\Delta^{(l_k)} = \left\{ \delta^{(l_k)}(x_j) \mid j = 1, 2, \ldots, J \right\} \tag{4.2.5}$$

4.2.3 パラメータ行列の変化量の正規化

4.2.2 節で算出した $\Delta^{(l_k)}$ の全要素に対して，最小値 0 以上最大値 1 以下の範囲となるように正規化 (Min-Max Normalization) する．正規化された変化量 $\delta_{\text{norm}}^{(l_k)}(x_j)$ は，$\delta^{(l_k)}(x_j)$ と $\Delta^{(l_k)}$ を用いて次のように表される：

$$\delta_{\text{norm}}^{(l_k)}(x_j) = \left\{ \frac{\delta^{(l_k)}(x_j) - \min(\Delta^{(l_k)})}{\max(\Delta^{(l_k)}) - \min(\Delta^{(l_k)})} \right\} \tag{4.2.6}$$

正規化処理を適用した l_k 層における事前学習前後のパラメータ行列の変化量 $\Delta_{\text{norm}}^{(l_k)}$ は，次のように表される：

$$\Delta_{\text{norm}}^{(l_k)} = \left\{ \delta_{\text{norm}}^{(l_k)}(x_j) \mid j = 1, 2, \ldots, J \right\} \tag{4.2.7}$$

ここでは，l_k 層におけるパラメータ行列の変化量の計算方法を説明した．ただし，事前学習に使用されたテキストデータかどうかの特定には，LLM を構成する層 $(l_1, l_2, \ldots, l_L)$ の一部，あるいは，全てを用いる．どの層のパラメータ行列を特徴量として用いるかは 5.2.5 節で議論する．

第 5 章　評価実験

本研究では 2 種類の実験を行った．実験 1 の目的は，与えられたテキストデータが事前学習に使用されたテキストデータかどうかを特定するために，パラメータの変化量を用いることが可能かどうかを検証することである．実験 2 の目的は，ベンチマークデータセットを用いて，どのような条件下で提案手法が既存手法よりも優れているのかを調査することである．

5.1　実験 1 (自作データセットでの検証)

ベンチマークデータセットで実験を行う前に，提案手法の有効性を確認するための予備実験を行った．まず，収集したデータを用いて MIA を行うことが可能なデータセットを構築した．その後，自作したデータセットを用いて実験を行った．

5.1.1　データセット

実験 1 では継続事前学習データセットと myMIA データセットの二つを作成した．二つのデータセットはそれぞれ異なる目的を持つ．

継続事前学習データセットは，学習したテキストデータの分かっていない LLM に対して継続事前学習を行い，学習したテキストデータと学習していないテキストデータが明らかな LLM を構築するために使用するデータセットである．このデータセットは，2023 年 8 月以降に新規に作成された Wikipedia 日本語記事を収集することで構築した．収集した記事は 2023 年 8 月以前には存在していないため，2023 年 8 月以前に構築された LLM の事前学習データには使用されていないことが保証されている．収集過程では，記事の末尾にある参考文献，リダイレクトページや曖昧さ回避ページといった不要なテキストデータ，記事の編集途中で未完成となっているテキストデータを除去した．また，十分な文章量を確保しつつ LLM に入力可能な単語長を超えないために，100 単語以上かつ 300 単語以下の記事のみを収集対象とした．2023 年 8 月以降に追加された記事のうち合計で 30,000 件の記事を収集し，これらの記事をランダムに 10,000 件ずつ訓練データ，検証データ，テ

ストデータに分割して継続事前学習データセットと定義する．このデータセットを用いて 5.1.2 節で述べる LLM に対して継続事前学習を行い，得られた LLM を myLLM と定義する．

一方，myMIA データセットは作成した myLLM に対して MIA を行うためのデータセットである．MIA の精度を検証するためのデータセットが満たすべき要件は，LLM の学習に使用されたテキストデータと使用されていないテキストデータが分かっていることである．継続事前学習データセットに含まれる訓練データは myLLM が学習したテキストデータ，テストデータは myLLM が学習していないテキストデータとして扱うことができる．そのため，継続事前学習データセットの訓練データとテストデータを利用することで，myLLM に対して MIA の精度を検証することが可能である．訓練データとテストデータをそれぞれ 10,000 件ずつ含んだデータセットを myMIA と定義する．

5.1.2 実験条件

実験 1 では，elyza/ELYZA-japanese-Llama-2-7b*という LLM を使用する．学習済みテキストデータや未学習テキストデータの明らかな LLM を構築するために，5.1.1 節で作成した継続事前学習データセットを用い，ELYZA-japanese-Llama-2-7b に対して継続事前学習を行い，myLLM を作成する．継続事前学習では，DeepSpeed†の ZeRO Stage-2 を使用した並列分散学習を行う．学習率は 3.0×10^{-5} とし，グローバルバッチサイズは Gradient Accumulation(勾配累積) を適用したため 256 とし，1epoch の学習を行う．myLLM は事前学習に使用したテキストデータと使用していないテキストデータが明らかになっているため, myMIA を使用した MIA の実験を行うことができる．

提案手法の継続事前学習ではバッチサイズを 1 に設定して学習を行う．オプティマイザには AdamW を使用し，$\beta_1 = 0.9$，$\beta_2 = 0.95$，$\epsilon = 1.0 \times 10^{-5}$，$\lambda = 0.1$ と設定する．学習率には 3.0×10^{-5} を設定し，スケジューラには cosine スケジューラを用いる．損失の最小値が 50epoch 連続で更新されなかった場合に学習を停止

*https://huggingface.co/elyza/ELYZA-japanese-Llama-2-7b
†https://github.com/microsoft/DeepSpeed

する Early Stopping を適用している．連続する epoch 間で訓練データにおける損失が 0.0001 未満の差となった場合を，損失の最小値が更新されなかったとしている．Llama 2 モデルの論文に準拠した学習率 3.0×10^{-4} を用いて実験を行っていたところ，提案手法の継続事前学習時の訓練データにおける損失が nan になり，学習できない問題が発生したため，学習率を 3.0×10^{-5} と小さくすることにより解決した．

5.1.3 実験手順

実験は以下の 3 ステップで行う．

1. 4.1 節に従い，myMIA データセットからテキストデータを一つのみ取り出し，5.1.2 節で述べた myLLM に対して継続事前学習を行う．
2. 4.2 節に従い，事前学習前と事前学習後における LLM のパラメータ行列をそれぞれ取得する．その後，学習前後のパラメータ行列の差からパラメータの変化量を求める．さらに，変化量を最小値 0 から最大値 1 の範囲に正規化し，特徴量として抽出する．
3. 抽出した特徴量を用いて AUROC を算出する．また，学習済みテキストデータと未学習テキストデータにおけるパラメータの変化量の平均値をそれぞれ算出する．

5.1.4 実験結果と考察

実験 1 では myMIA に含まれるテキストデータのうち，事前学習に使用したデータと使用していないデータを 40 件ずつランダムサンプリングして実験を行った．ここではパラメータの選定を行わず，全ての層におけるパラメータを特徴量として用いている．

実験結果を図 5.1 に示す．提案手法の AUROC は 0.72 という結果であった．ランダムに出力する分類モデルを上回る性能を有していることが分かる．つまり，パラメータの変化量を特徴量とした提案手法による MIA は有効であることが確認さ

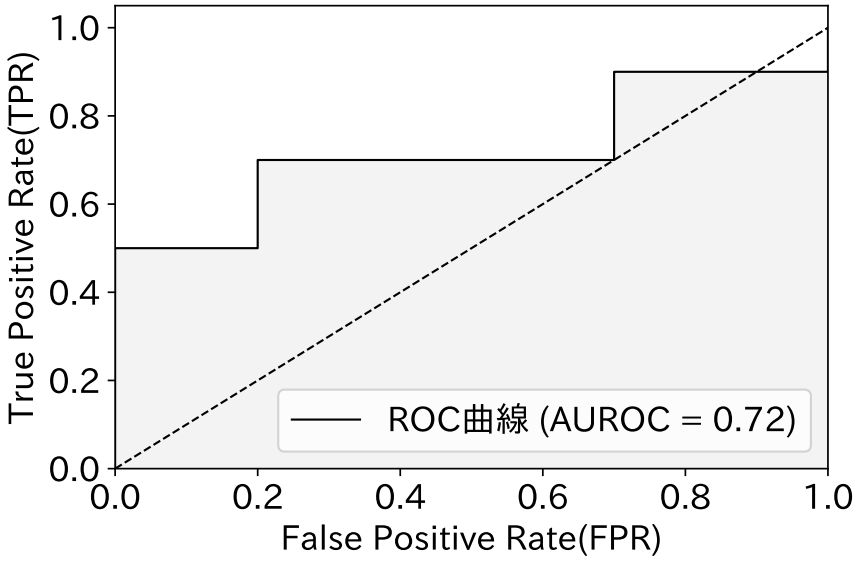

図 5.1: 実験 1 での AUROC

れた．ただし，実験 1 の結果のみでは提案手法の適用範囲は限定的である．実験 1 で使用した myMIA データセットはテキストデータの言い換え処理を行っていないため，学習に使用したデータと全く同じデータを含んでいる．そのため，MIA が比較的容易であった可能性や全く同じデータを含んでいたために MIA が成功した可能性がある．

また，学習済みテキストデータと未学習テキストデータを再度学習させたときにおけるパラメータの変化量の平均値を測定した．ここでのパラメータの変化量は，全てのパラメータを使用した場合のパラメータの変化量である．実験の結果，学習済みテキストデータを学習させたときのパラメータの変化量は 0.1667 であるのに対し，未学習テキストデータを学習させたときのパラメータの変化量は 0.1904 であった．この結果から，学習済みテキストデータを学習したときのパラメータの変化量は，未学習テキストデータを学習したときのパラメータの変化量よりも小さいことが分かる．つまり，未学習テキストデータを学習するときにはモデルのパラメータを大きく更新する必要があるため，パラメータの変化量が大きくなる傾向にあると推測される．次に，学習済みテキストデータと未学習テキストデータのパラメータ変化量に有意な差が見られるかどうかを検定した．有意水準 $\alpha = 0.10$ で二

標本対応なし t 検定を行うと，p 値は 0.0538 を示した．検定結果より，統計的に有意な差が生じていることが確認された．本研究の仮説は，学習したことのあるテキストデータを学習したときのパラメータの変化量は小さく，学習したことのないテキストデータを学習したときのパラメータの変化量は大きいというものである．実験結果と検定結果より，仮説は正しいと考えられる．つまり，パラメータの変化量を特徴量として用いることは，学習済みデータかどうかを特定するために有効である可能性が高いことを示している．

5.2 実験 2 (wikiMIA での検証)

wikiMIA を使用した提案手法の精度評価を行う．また，提案手法との比較をするために先行研究での精度評価も行い，提案手法の結果について分析する．

5.2.1 データセット

実験 2 では，Shi ら [8] によって開発され MIA のベンチマークデータセットである wikiMIA[‡]を使用した．wikiMIA は Wikipedia から収集したテキストデータで構成されており，単語長によって四つのサブセットに分けられている．本研究では単語長が 32 単語のサブセットのみを用いて実験を行っている．

本データセットは，事前学習に使用したテキストデータを 2017 年以前に発生したイベントに関する Wikipedia ダンプデータ，事前学習に使用していないテキストデータを 2023 年以降に発生したイベントに関する Wikipedia ダンプデータ，と定義して収集している．LLM はインターネット上のあらゆるデータを学習していると考えられており，その中には Wikipedia ダンプデータも含まれる．2017 年から 2023 年の間にリリースされた LLM であれば，2017 年以前に発生したイベントを含む Wikipedia ダンプデータを事前学習に使用していると考えられる．何故なら，LLM の事前学習に使用されるデータセットには Wikipedia ダンプデータが多く含まれており，Llama-2 や OPT といったモデルは 2017 年以降にリリースされているためリリース時点で入手できる Wikipedia ダンプが事前学習に使用される

[‡]https://huggingface.co/datasets/swj0419/WikiMIA

データセットに含まれていると考えられる．そのため，2017 年以前に作成された Wikipedia のイベント記事は，Llama-2 や OPT といったモデルの事前学習データセットに含まれている可能性が高い．一方，LLM がリリースされた 2023 年以降に発生したイベントを含む Wikipedia ダンプデータは，学習時点では発生していない未来のイベントであるため，事前学習に使用することはできない．この特徴を利用して設計されたデータセットであるため，2017 年から 2023 年の間に作成された学習済みモデルに対してのみ本データセットを使用することができる．

wikiMIA では，テキストデータに対して ChatGPT を使用した言い換え処理を施しているため，事前学習に使用されたテキストデータと完全に一致したテキストデータが含まれるわけではない．一方，5.1.1 節の自作した myMIA では，テキストデータに対する言い換え処理を行っていないため，事前学習に使用されたテキストデータと完全に一致したテキストデータが含まれている．つまり，wikiMIA と myMIA では，データセットの性質が異なる．

5.2.2 実験条件

実験で使用するモデルやハイパーパラメータについて述べる．実験 2 では，GPT-J-6B[46][§]，OPT-6.7B[47][¶]，Pythia-6.9B[48][‖]，Llama-2-7B[49][**]の四つの LLM を対象に実験を行う．これらのモデルでは，いずれも Hugging Face に公開されている事前学習済みモデルのパラメータを使用する．

提案手法の継続事前学習ではバッチサイズを 1 に設定し，DeepSpeed を使用した並列分散学習を行う．使用するオプティマイザや学習率などは 5.1.2 節の実験条件と同様である．Early Stopping を行う場合，実験 1 とは異なり損失の最小値が 20epoch 連続で更新されなかった場合に学習を停止する．

[§] https://huggingface.co/EleutherAI/gpt-j-6b
[¶] https://huggingface.co/facebook/opt-6.7b
[‖] https://huggingface.co/EleutherAI/pythia-6.9b
[**] https://huggingface.co/meta-llama/Llama-2-7b

5.2.3 実験手順

5.2.2 節で述べた実験対象の LLM に対して提案手法を適用し，パラメータの変化量を特徴量として抽出し，AUROC を算出する．全てのパラメータを使用する場合を 5.2.4 節，一部のパラメータのみを使用する場合を 5.2.5 節で述べる．また，提案手法における AUROC と既存手法における AUROC の比較を行う．

5.2.4 全てのパラメータを使用した場合の実験結果と考察

AUROC の値を表 5.1 に示す．停止条件の列は学習を止める場合の基準を示しており，epoch は学習を行った回数，ES は早期停止させたことを表す．また，平均値の列は停止条件ごとに AUROC の平均をとった値を示している．表 5.1 の実験結果より，全ての停止条件において AUROC は 0.50 を上回っていることが分かる．つまり，wikiMIA においても提案手法による MIA は有効であることが確認された．

また，ES 以外の停止条件における平均の列を見ると，学習を行った回数を増やすと AUROC の精度も上昇することが確認できる．ES の行以外の平均値と学習を行った epoch 数とのピアソンの積率相関係数を算出した．検定を行った結果，学習回数と AUROC との相関係数は 0.954，p 値は 0.046 であり，これらの二変数の間には強い正の相関があることが統計的に示された．つまり，提案手法におい

表 5.1: 全てのパラメータを使用した場合における
提案手法の AUROC

停止条件	攻撃対象の LLM				平均値
	GPT-J	OPT	Pythia	Llama-2	
5epoch	0.59	0.51	0.52	0.55	0.542
10epoch	0.61	0.56	0.51	0.50	0.545
20epoch	0.66	0.51	0.64	0.53	0.585
30epoch	0.64	0.51	0.59	0.62	0.592
ES	0.57	0.54	0.68	0.54	0.584

て学習を行った回数を増やせば増やすほど，MIA の精度は向上することが分かった．今回の実験結果の平均値を見ると，5epoch，10epoch の停止条件の場合よりも，20epoch，30epoch，ES の停止条件の場合が良いと考えられる．

Shokri ら [6] の先行研究では，過学習したモデルであるほど学習データと学習していないデータとの特有の違いが漏洩し，MIA の精度は高くなることが分かっている．DeAlcala ら [45] の先行研究では，一度しか学習されないデータよりも，何度も学習されたデータに対する MIA の精度の方が高いことが分かっている．学習を行った回数を増やすと AUROC の精度が向上するという我々の結果は，これらの先行研究の知見とも一致する結果となった．

5.2.5　一部のパラメータを使用した場合の実験結果と考察

一部のパラメータを使用した場合の提案手法の AUROC と既存手法との比較結果を表 5.2 に示す．使用パラメータの列は，全てのパラメータのうちどのパラメータを使用したのかを表している．まず，パラメータを役割ごとに分割して使用した

表 5.2: 一部のパラメータを使用した場合における
提案手法の AUROC と既存手法の AUROC

手法		攻撃対象の LLM				平均値
		GPT-J	OPT	Pythia	Llama-2	
Attention 層	20epoch	0.63	0.53	0.64	0.57	0.592
	30epoch	0.62	0.52	0.57	0.61	0.579
	ES	0.56	0.55	0.67	0.55	0.582
MLP 層	20epoch	0.68	0.51	0.64	0.51	0.582
	30epoch	0.66	0.50	0.57	0.62	0.586
	ES	0.59	0.52	0.69	0.53	0.583
LayerNorm 層	20epoch	0.62	0.54	0.54	0.56	0.563
	30epoch	0.61	0.50	0.52	0.62	0.564
	ES	0.58	0.51	0.63	0.53	0.562
変化量の大きい top-K%	20epoch	0.65	0.70	0.65	0.61	0.652
	30epoch	0.69	**0.71**	0.61	**0.64**	0.660
	ES	0.63	0.57	**0.76**	0.56	0.634
Min-K%		0.71	0.67	0.71	0.58	0.668
SaMIA*zlib		0.75	0.81	0.75	0.66	0.743

場合について説明する．今回はパラメータを Attention 層，MLP 層，LayerNorm 層に分割して実験を行った．Attention 層は入力されたデータの各部分を異なる重要度で扱う機構，MLP 層はニューラルネットワークのユニットどうしが互いに接続されている機構，LayerNorm 層はニューラルネットワークの出力を正規化する機構である．Transformer モデルは，おおまかに約 3 割が Attention 層，約 6 割が MLP 層，約 1 割が LayerNorm 層から構成されている．先行研究 [50] によると，Transformer モデルの Feed-Forward 層 (二層の MLP 層に相当) はニューラルネットワークのデータ保存機構として働くことが分かっている．そのため，知識を持っていると考えられる MLP 層におけるパラメータの変化量を観察することで，提案手法の精度が向上するのではないかと考えた．Attention 層と LayerNorm 層についても同様に分析を行った．モデルによっては MLP のように命名されていない場合もあったため，モデルに関する論文やモデル構造を確認しながら分割を行った．

実験結果を表 5.2 の Attention 層，MLP 層，LayerNorm 層の行に示す．Attention 層のパラメータのみを使用した場合の AUROC の平均値は 0.584，MLP 層のパラメータのみを使用した場合の AUROC の平均値は 0.584，LayerNorm 層のパラメータのみを使用した場合の AUROC の平均値は 0.563 である．これらの結果より，LayerNorm 層のパラメータを使用する場合よりも，Attention 層のパラメータや MLP 層のパラメータを使用するのが良いという結果が得られた．先行研究より MLP 層のパラメータは学習したデータの記憶領域として働くことが知られているため，MLP 層のパラメータを使用することが提案手法の精度向上に寄与する可能性が高いと考えられる．しかし表 5.1 の平均値の結果と比較すると，全てのパラメータを使用した場合の平均値とほとんど差がないことが確認できる．つまり，パラメータを役割ごとに分割して使用することによって，提案手法の精度は向上しないことが分かった．特定の層のパラメータは記憶機構として働くが，あくまでパラメータの値そのものが記憶を表現しているにすぎない．そのため，パラメータの変化量と記憶機構とは直接的な関係がないため，結果として精度向上には寄与しなかったと考えられる．

次に，変化量の大きいパラメータのみを使用した場合について説明する．本研究の仮説は，学習したことのあるテキストデータを学習したときのパラメータの変化量は小さく，学習したことのないテキストデータを学習したときのパラメータの変

化量は大きいというものであった．5.1.4 節の検定結果より，この仮説は正しいと考えられる．そこで，全てのパラメータではなく，パラメータの変化量が大きいパラメータのみを使用することによって，提案手法の精度が向上するのではないかと考えた．何故なら，変化量の小さいパラメータは，学習済みデータと未学習データとの差異が現れなくなってしまうためである．

実験結果を表 5.2 の変化量の大きい top-K% の行に示す．K を 1 から 50 まで変化させたときに最も精度が高くなった結果のみを表に示している．分析の結果，K は 1% から 5% のときに最も高い精度を示した．20epoch の場合における AUROC の平均値は 0.652, 30epoch の場合は 0.660, ES の場合は 0.634 となった．表 5.1 の平均値の結果と比較すると，全てのパラメータを使用した場合は最も良い AUROC が 0.592 であったのに対し，変化量の大きい top-K% のパラメータを使用した場合は最も良い AUROC が 0.660 であった．AUROC としては 6.8 %も精度が向上したことになる．つまり，変化量の大きいパラメータのみを特徴量として用いることで，提案手法の精度を改善することができた．変化量の大きいパラメータは他のパラメータと比較して，MIA の特定精度向上に重要な情報を含んでいると考えられる．

最後に，既存手法である Min-K% Prob[8]，SaMIA*zlib[9] との比較を行った．既存手法にはいくつかのハイパーパラメータが存在するが，論文中で最適とされているハイパーパラメータを使用した場合の結果を示している．提案手法の AUROC がいずれかの既存手法よりも高い精度であった場合，太字で強調表示をしている．既存手法と比較した結果，提案手法の AUROC が 0.76 となり，Min-K% Prob の 0.71，SaMIA*zlib の 0.75 よりも高い AUROC を記録した．また，OPT と Llama-2 に関しては，Min-K% Prob の AUROC を上回ることができた．ただし，GPT-J ではいずれの手法も上回ることができなかった．これらの結果から，提案手法は特定のモデルにおいて既存手法を上回る性能を示すことができた．しかし，実験対象モデルに対する提案手法の AUROC の平均値は 0.660 であり，Min-K% Prob の AUROC の平均値である 0.668 とは互角の性能を示すことができたが，SaMIA*zlib の AUROC の平均値である 0.743 には及ばない結果となった．提案手法の平均的な精度を向上させるためには，学習を行う epoch 数をさらに増やすこと，パラメータの変化量の算出方法を変えることなどの手法改善が必要であると考

えられる.

第 6 章　おわりに

本研究では，LLM に対するホワイトボックス設定のメンバーシップ推論攻撃における新たな手法の開発を目的とした．LLM を含むニューラルネットワークは目的関数を最適化するように，パラメータを変化させながら学習を行う．そのため，パラメータには学習データの内容や特徴が反映される．また，学習したことのあるデータと学習したことのないデータでは，目的関数を最適化するために必要な学習量が異なるため，パラメータの変化量に違いが生じると考えた．そこで，事前学習前後のパラメータの変化量を特徴量として用いる新たな手法を提案した．

事前学習前後のパラメータの変化量を使用したメンバーシップ推論攻撃が可能かどうかを確かめることによって，提案手法の有効性を検証した．実験 1 では，myLLM と myMIA データセットを用いて実験を行った．実験の結果，学習したことのあるデータと学習したことのないデータを学習したときのパラメータの変化量に違いが生じることが確認された．この実験より，設定した仮説が正しいことが裏付けられた．実験 2 では，GPT-J-6B，OPT-6.7B，Pythia-6.9B，Llama-2-7B の四つの LLM を対象に実験を行った結果，提案手法を使用したメンバーシップ推論攻撃は可能であることを確認した．また，全てのパラメータを特徴量として使用するのではなく，変化量の大きいパラメータのみを選定して特徴量として使用すると，精度が向上することが分かった．既存手法である Min-K% Prob や SaMIA*zlib との比較について，Pythia では既存手法をどちらも上回る精度を示し，OPT と Llama-2 では Min-K% Prob を上回る精度を示した．特定の LLM において既存手法を上回る精度を示すことはできたが，平均的な精度に関して提案手法は既存手法を大きく上回ることはできなかった．

今後の展望として，二つの改善案を考えている．一つ目は，パラメータの変化量をコサイン類似度やマハラノビス距離などのユークリッド距離以外の距離関数で測定することを考えている．コサイン類似度はベクトル相互の向きの違いを測る距離関数である．マハラノビス距離はユークリッド距離に対してデータの分布に基づいた正規化を行った距離関数である．今回の実験で使用したユークリッド距離ではパラメータの変化量を特徴量として取り出せなかった可能性が考えられるため，他の距離関数を利用して実験を行い，MIA の精度向上を目指したいと考えている．

二つ目は，モデルから得られたパラメータ行列をそのまま比較するのではなく，次元削減を行うなどしてから比較することを考えている．パラメータの変化量のような高次元データを直接扱うとデータにスパース性が生じ，分析精度が低下してしまう可能性 [51] がある．可能性としては，Bellman によって提唱された次元の呪いと呼ばれる問題 [52] に起因していることが考えられる．次元の呪いによって，変化量を特徴量として扱えなかった可能性が考えられる．高次元データを低次元データとして扱うことで高次元データの問題を回避できる可能性もあるため [51]，PCA などの次元削減手法も試す必要があると考えている．

謝辞

謝辞を書き始めるにあたり，これまでの研究室生活の振り返りみたいなものを書こうと思います．だらだらと書くので，読みたい人がいたら読んでみてください．

約三年間の鈴木研究室での活動は充実したものだったと思います．論文投稿に関しては，四つの会議 (IJCNLP-AACL2023, PAKDD2024, DaWaK2024, iiWAS2024) にチャレンジし，四度目の投稿にてようやく論文が Accept されました．PAKDD2024 と DaWaK2024 は会議のスコープが違うといった査読結果であったため考慮しないとすると，一勝一敗といったところでしょうか．B4 の当時の自分は修士学生として過ごしたほとんどの時間 (2023 年 4 月から 2024 年 12 月までの 20 ヶ月間... 長すぎるだろ！) を，論文との格闘に費やすことになるとは思いもしなかったことでしょう．もしタイムマシンで戻れるのならば，「なんでお前 (自分) はこの実験をやってないんだ！早く実験しろ！！」と殴りに行きたいです．とはいえ，最低でも一本は主著で査読論文を通すという自分の中での目標を達成できたので，わりと満足しています．論文投稿に関して後悔はあまりありませんが，強いていうならスロバキアやドイツに行くことが出来ず，シュニッツェルやドイツビールを味わえなかったことくらいでしょうか．

学会発表に関しては，国内での発表を四回 (DBWS2022, DEIM2023, DBWS2023, DEIM2025)，国外での発表を一回 (iiWAS2024)，チュートリアル発表を一回 (DEIM2024) 行うことが出来ました．特に，チュートリアル発表は一番の思い出です．美味しいものを食べたいというしょうもない理由と，研究室内で流行っていた謎文化 (やったっていい！) に毒されてチュートリアル発表の実施を決めたのですが，その場のノリで決めてよいものじゃないですね．9 時から 17 時まで日立製作所中央研究所のインターンシップにて研究を行い，18 時から 21 時まで実世界データ演習のミーティングやコーディング，資料作成に取り組み，22 時から 24 時過ぎまでチュートリアル発表のための論文読みと資料作成を行い... イカれたスケジュールで活動に取り組んでいました．大変な思いはしましたが，苦労しただけに得られたメリットはとても大きかったです．活動に興味を持って下さった方がいたり，様々な場で好意的な意見をいただくことができました．僕のわがままに付き合ってくれた三島先輩，桑原君，高橋君 (?)，ありがとうございました．

共同研究に関しては，約二年ほどプログラム作成や資料作成を行い，様々な出来事を経験しました．初めての共同研究ということで，期待と緊張で溢れていました．週一回のミーティング参加やプログラムのデバッグ，プログラムの作成など，色んなタスクをこなしました．アカデミック用の TwitterAPI が使えなくなったこと，LLM の進化が想像以上に凄まじかったこと，三島先輩の卒業などによってメンバーが減ったこと，様々な課題にも直面しました．大学でのゼミ報告と企業の定例報告との違い，研究者ではない企業の方向けに説明することの難しさ，共同研究そのものの難しさ，多くのことも学びました．共同研究の結果・成果としてはイマイチだったのかもしれませんが，個人的に得られた経験は多々あったと思います．思い出は色々ありますが，ソニー本社にあるソニースクエアにてソニー製品を体験できたこと，27 時くらいまで大学に残ってクラウドソーシングシステムのデバッグを行っていたこと，東京駅にあるつけ麺屋さんが美味しかったこと，この先も忘れないでしょう．つけ麺屋さんの URL*をつけておきます．つけ麺だけに．

知識面に関しては，ニューラルネットワークや LLM に関係する知識を増やすことができたと思います．情報学や AI 分野を選択したのは，高校時代にデータサイエンスをかじり，「AI ってなんかすごそう」と魅力を感じたことがきっかけですが，当時の私は AI についての知識は全くありませんでした．AI について何も知らなかった高校生の頃と比べると，かなり成長できたと思います．論文を読んで「あぁこういうことをやってんだな」と理解したり，「この問題にはあの技術が使えそうだな」と即座に反応したりできるようになりました．三菱電機の情報技術総合研究所のインターンシップでは，企業研究者の方に「北村君は機械学習に関する知識が十二分にある．」と評価していただきました．これまで頑張って研究を続けてきてよかった，研究や勉強をもっと頑張りたい，と思える契機となりました．また，就職報告も兼ねて高校時代の部活動の恩師に連絡をしたところ，恩師から「AI を使った研究をやりたいから高校生に画像認識について教えてくれないか？」というお話もいただきました．2025 年 1 月現在も，高校に教えに行ったり，メールでやり取りしたり，画像認識のプログラムを書いたり，定期的に仕事をしています．こうした機会に恵まれたのも，けっこう真面目に？取り組んできたおかげかなと思ってます．

*https://rokurinsha.com/

さらに，東京大学の松尾・岩澤研究室が主催する LLM 講座[†]では，四か月ほど座学とコーディングに取り組みました．真面目に取り組んだおかげか，2,000 人以上いる 29 歳以下の学生受講者の中で，優秀賞第四位という成績も収めることが出来ました．日本で四番目に LLM に詳しい学生，と言っても過言ではないのではないでしょうか．そして，松尾研究室で AI に関する基礎研究を一緒にやらないか，というお誘いもいただきました．(来年以降も学生であるという条件であり，修士で留年するか D 進しないと条件を満たせなかったため，渋々辞退しました...) とにかく，LLM に関しては誰よりも詳しくなれたと自負しています．自信をもってそう言えるほど一つのことに取り組んだ経験は，これから先の人生においても大きな財産になると確信しています．

だらだらと書いていたら 2 ページを超えていました．筆が止まらないって良い時と悪い時がありますね．そろそろ本題に入りたいと思います．

卒業研究を進めるにあたり，指導教員の鈴木優准教授には多くのことでご指導ご鞭撻を賜りました．研究に関する悩みが生じていたときには，面談などでアドバイスをして下さいました．LLM の研究を始めたとき，「研究設備が足りていないからやりたい研究をできないのは良くないなぁ (意訳)」ということで，GPU などの計算機資源を確保していただきました．今の計算機資源がなければ修士論文の研究を行うことはできませんでした．本当にありがとうございました．蛇足ですが，来年以降は怪我に見舞われないことを心の底から願っております．お大事になさってください．

NVIDIA 様にも大変お世話になりました．NVIDIA 様の GPU がなければ，あらゆる実験を行うことができなかったでしょう．特に，配属されてから 3 年ほど酷使していた NVIDIA TITAN RTX 君，お気に入りの NVIDIA RTX A6000 君，計算するのが一番得意だった NVIDIA GeForce RTX 3090 君，君たちには感謝してもしきれません．壊れずに計算し続けてくれてありがとう．

研究室の皆様にも様々な面で支えていただきました．秘書の井尾さん．研究室活動を円滑に行えるよう，様々な事務手続きを行ってくださいました．iiWAS の航空券の返金手続きでは，お手数をおかけしました．今よりわずかばかりでも良いので仕事量が減ることを祈念しております．

[†]https://weblab.t.u-tokyo.ac.jp/lecture/course-list/large-language-model/

OB の古田先輩，沢田先輩，三島先輩，あと小林先輩．偉大な先輩方の背中を追いかけて駆け抜けた三年間でしたが，先輩方の凄さを再認識し，ぽっかりと空いてしまった穴の大きさを感じる日々でした．研究室での姿や社会人として頑張っている姿を目にするたび，その背中に少しでも近づけるように努力し続けていきたい，と気を引き締めています．これから先は仕事関係で出会うことがあるかもしれませんが，その時はよろしくお願いいたします．

優秀な同期のエルゲン君，太田さん，桑原君．お互いの研究にアドバイスをしあい，研究をより良いものにアップデートし続けていけたと思います．また，同期が頑張っているから自分もがんばろう！と思えた機会が何度もありました．本当に，良い同期に巡り合えたと思います．皆さんの就職先は関東か東海でバラバラですが，定期的に会えたらよいですね．これからもお互いに頑張っていきましょう．いつかまた逢う日まで．

M1 の川上君，高橋君，何故か鈴木研所属ではなくなってしまった城所君，林君．面白い人たちが多くて，いつも笑わせてもらってました．研究に苦労することも多かったと思いますが，諦めることなく一歩ずつでも歩んでいく姿勢は見習わなければなと思っています．願わくば，楽しく研究を行える日が来ますように，草葉の陰から祈っております．あと，コーヒーおじさんの淹れるコーヒーが美味しかったです．コーヒー飲めなくなるのは寂しいです．

B4 の尾関さん，田中君，中村君，藤田君．配属されてからいろんな出来事があり，この先大丈夫かなと心配になることもあったと思います．ですが，君たちなら大丈夫だと信じてます．人からいろいろ言われることもあると思いますが，やるべきことに取り組み，頑張り続けられる人たちであることを僕は知っているつもりです．M1 になると就職活動が始まり，就職活動と研究の両立に苦労すると思います．大変だとは思いますが，これからも応援しています．あと，誤字には気を付けてください．

B3 の江崎君，小出さん，もう一人の藤田君，三浦君．関わった期間は短かったけれど，面白い学生が入ってくれたなぁと嬉しく思ってます．本格的に研究が始まって大変になっていくと思いますが，研究を「楽しむ」気持ちを忘れずに．しんどいときは無理せず，休みましょう！僕が遊びに行ったときは，色々話を聞かせてください．特に江崎君．いつか僕もバーに連れて行ってください．

研究室関係の人たちの中では，特に三島先輩には公私ともにお世話になりました．ニューラルネットワークに関する造詣が深く，研究談議に花を咲かせることも多かったと思います．お互いの研究について意見交換しあうだけではなく，他人 (小林先輩) の研究について工学部駐車場で 25 時くらいまで議論したりもしました．プライベートでは，関市近くにある美味しいラーメン屋さん[‡]や愛知にある海鮮の有名な定食屋さん[§]など，色々なお店に連れて行ってくださいました．また，しゅうちゃんと一緒に Overwatch や APEX で遊んだりもしました．本当に楽しかったです．就職してからも東京あたりでご飯に行きましょう．本当にありがとうございました．

中学・高校の友人である A 君にもお世話になりました．研究で悩んでいるときに相談に乗ってもらったり，麻雀や PC ゲームをしたり，愚痴を聞いてもらったり，東京のいろんな場所に連れて行ってくれたり，とにかくリフレッシュできました．日本酒の美味しい居酒屋にもまた行きたいです．阪大法学部に進学した君とは違う分野で働くと思っていたのに，まさか同じ IT 分野で働くことになるとは思いもしなかったです．世の中どうなるか分かりませんね．富士通とうちの日立とは競合他社になるけれど，お互いに頑張っていきましょう．これからも迷惑かけることがあると思うけど，よろしくお願いします．

最後に，今日までの二十数余年ものあいだ僕を支え，叱咤激励し，応援してくれた家族，ありがとうございました．途中で逃げ出したくなることもあったけれど，それでも何とかここまでやってこれたのは，見守ってくれた家族のおかげです．四月からは企業研究者として働くことになりますが，なんとかしがみついて頑張っていきたいです．これからも応援していてください．

最後になりますが，本研究を無事に終えることができたのは，これまで携わってきた全ての方々の援助があってこそだと思います．改めて，本当に本当に，本当にありがとうございました．この場を借りて御礼申し上げます．

[‡]http://www.dcreate.co.jp/appare/seki/

[§]http://www.maruha-net.co.jp/shop/honten

参考文献

[1] Antonia Karamolegkou, Jiaang Li, Li Zhou, and Anders Søgaard. Copyright violations and large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7403–7412, Singapore, December 2023. Association for Computational Linguistics.

[2] Matthieu Meeus, Igor Shilov, Manuel Faysse, and Yves-Alexandre de Montjoye. Copyright traps for large language models. In *Forty-first International Conference on Machine Learning*, 2024.

[3] Oscar Sainz, Jon Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10776–10787, Singapore, December 2023. Association for Computational Linguistics.

[4] Changmao Li and Jeffrey Flanigan. Task contamination: Language models may not be few-shot anymore. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38, pp. 18471–18480, 2024.

[5] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S Yu, and Xuyun Zhang. Membership inference attacks on machine learning: A survey. *ACM Computing Surveys (CSUR)*, Vol. 54, No. 11s, pp. 1–37, 2022.

[6] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pp. 3–18. IEEE, 2017.

[7] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*, pp. 739–753. IEEE, 2019.

[8] Weijia Shi, Anirudh Ajith, Mengzhou Xia, Yangsibo Huang, Daogao Liu, Terra Blevins, Danqi Chen, and Luke Zettlemoyer. Detecting pretraining data from large language models. In *NeurIPS 2023 Workshop on Regulatable ML*, 2023.

[9] Masahiro Kaneko, Youmi Ma, Yuki Wata, and Naoaki Okazaki. Sampling-based pseudo-likelihood for membership inference attacks. *arXiv preprint arXiv:2404.11262*, 2024.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[11] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Vol. 5, pp. 115–133, 1943.

[12] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, Vol. 65, No. 6, p. 386, 1958.

[13] GE Hinton. A fast learning algorithm for deep belief nets. *Neural Computation*, 2006.

[14] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, Vol. 313, No. 5786, pp. 504–507, 2006.

[15] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, Vol. 15, No. 1, pp. 1929–1958, 2014.

[16] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song,

Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao-

hui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[17] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[18] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, Vol. 13. MIT Press, 2000.

[19] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[20] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[21] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

[22] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[23] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and

Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[24] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[25] Nikhil Sardana, Jacob Portes, Sasha Doubov, and Jonathan Frankle. Beyond chinchilla-optimal: Accounting for inference in language model scaling laws. In *Forty-first International Conference on Machine Learning*, 2024.

[26] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, Vol. 33, pp. 1877–1901. Curran Associates, Inc., 2020.

[27] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, Vol. 55, No. 12, pp. 1–38, 2023.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[29] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[30] Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts. *arXiv preprint arXiv:2407.06204*, 2024.

[31] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, Vol. 3, No. 1, pp. 79–87, 1991.

[32] Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, et al. Unified scaling laws for routed language models. In *International conference on machine learning*, pp. 4057–4086. PMLR, 2022.

[33] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

[34] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[35] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[36] Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Pete Walsh, Luke Zettlemoyer, Noah A. Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld,

Jesse Dodge, and Kyle Lo. Dolma: An Open Corpus of Three Trillion Tokens for Language Model Pretraining Research. *arXiv preprint*, 2024.

[37] Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, et al. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*, 2021.

[38] Yuxia Wang, Haonan Li, Xudong Han, Preslav Nakov, and Timothy Baldwin. Do-not-answer: A dataset for evaluating safeguards in llms. *arXiv preprint arXiv:2308.13387*, 2023.

[39] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, Vol. 33, pp. 3008–3021, 2020.

[40] Wei An, Xiao Bi, Guanting Chen, Shanhuang Chen, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Wenjun Gao, Kang Guan, et al. Fire-flyer ai-hpc: A cost-effective software-hardware co-design for deep learning. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–23. IEEE, 2024.

[41] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[42] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

[43] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, p. 100211, 2024.

[44] Arijit Ghosh Chowdhury, Md Mofijul Islam, Vaibhav Kumar, Faysal Hos-

sain Shezan, Vinija Jain, and Aman Chadha. Breaking down the defenses: A comparative survey of attacks on large language models. *arXiv preprint arXiv:2403.04786*, 2024.

[45] Daniel Dealcala, Gonzalo Mancera, Aythami Morales, Julian Fierrez, Ruben Tolosana, and Javier Ortega-Garcia. A comprehensive analysis of factors impacting membership inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 3585–3593, June 2024.

[46] Ben Wang and Aran Komatsuzaki. Gpt-j-6b: A 6 billion parameter autoregressive language model, 2021.

[47] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

[48] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, Usvsn Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. Pythia: A suite for analyzing large language models across training and scaling. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, Vol. 202 of *Proceedings of Machine Learning Research*, pp. 2397–2430. PMLR, 23–29 Jul 2023.

[49] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[50] Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021*

Conference on Empirical Methods in Natural Language Processing, pp. 5484–5495, 2021.

[51] Srikanth Thudumu, Philip Branch, Jiong Jin, and Jugdutt Jack Singh. A comprehensive survey of anomaly detection techniques for high dimensional big data. *J. Big Data*, Vol. 7, No. 1, p. 42, 2020.

[52] Richard Bellman. Dynamic programming. *science*, Vol. 153, No. 3731, pp. 34–37, 1966.

発表リスト

[1] 北村拓斗，鈴木優 『LLM 生成テキストの検出』，東海関西データベースワークショップ，2023
[2] 北村拓斗，三島惇也，桑原悠希，高橋巧実，鈴木優 『LLM の嘘¶』，第 16 回データ工学と情報マネジメントに関するフォーラム，2023
[3] Takuto Kitamura，Yu Suzuki 『Finding Adequate Additional Layer of Auxiliary Task in BERT-Based Multi-Task Learning』，The 26th International Conference on Information Integration and Web Intelligence (iiWAS2024)，2024

¶https://confit.atlas.jp/guide/event/deim2024/static/tutorial?lang=ja#tu-d-2

受賞歴

[1] 北村拓斗,『最終課題コンペティション [優秀賞 U-29 部門] 第 4 位』, 大規模言語モデル DeepLearning 応用講座 2024|Fall[‖](東京大学松尾・岩澤研究室), 2024

[‖] https://weblab.t.u-tokyo.ac.jp/lecture/course-list/large-language-model/

付録 A　Appendix

Appendix では，提案手法を実装する際に苦労した点，工夫した点について述べる．A.1 節では LLM の事前学習を行う際の環境構築，A.2 節では提案手法を実装する際の技術的工夫について述べる．

A.1　実験環境

本研究では LLM の事前学習を行った．研究を行ってきた中で最も大変だったことは，事前学習を行うための環境構築である．参考例として，llm-jp*の事前学習を行うための環境構築方法†を参照されたい．非常に多くの手順が必要となる．今回の事前学習では，Transformers‡や DeepSpeed§，FlashAttention¶や Datasets‖といった多くの Python パッケージを使用する．これらのパッケージはモデル構築や分散並列学習，Attention 機構の計算処理の高速化やデータ処理を担うため，LLM の事前学習を効率的に行うために必要不可欠である．しかし，パッケージのバージョンによってはコンフリクトを起こしてしまい，いずれかのパッケージを使用できなくなる場合もある．また，複数の依存関係が複雑に絡む実験環境で適切なバージョン管理を行えなかった場合，突然動作しなくなるリスクや予期せぬエラーに見舞われてしまうリスクも高まる．そこで，Docker を導入した．

Docker は仮想化技術を用いることにより，ホストマシンに影響を与えることなく，ホストマシンの影響を受けることもなく環境構築を行うことが可能となる．ホストマシンの環境が変わったとしても Docker 内の環境には影響が出ないため，パッケージの依存関係やバージョンの違いによる問題を防ぎ，実験環境を安定的に保つことができる．また，実験を行った実験環境を他のマシンや環境に簡単に移植することもでき，全く同じ環境を何度も構築することができる．

*https://huggingface.co/llm-jp
†https://github.com/hiroshi-matsuda-rit/NLP2024-tutorial-3
‡https://pypi.org/project/transformers/
§https://pypi.org/project/deepspeed/
¶https://pypi.org/project/flash-attn/
‖https://pypi.org/project/datasets/

構築した環境で実験を行う中で多くの問題に直面した．特に，ソフト側の設定がハード側に影響を及ぼし，問題として表面化するという事例が多々あった．そのため，ハードかソフトのどちらに原因があるのかを切り分けるのが難しく，解決に至るまでにかなりの時間を要し，サーバ室に籠って原因の特定にあたっていた．遭遇したいくつかの問題のうち，ここでは二つの問題について触れる．

一つ目の問題は，ホストマシン上では GPU が認識されているのに，docker 内では GPU が認識されないという事象である．直接的な原因は，cgroup の管理が systemctl と Docker との間で競合していたことにあった．この競合により NVIDIA に関連するツールが正常に動作せず，GPU が認識されなかったと考えている．問題の解決には，NVIDIA Container Toolkit の設定を見直し，cgroup を明示的に指定する必要があった．問題の詳細と解決策については，Zenn の記事**や GitHub の Issues††を参考にされたい．

二つ目の問題は，LLM の事前学習を開始するとサーバがクラッシュし，異常停止してしまうという事象である．直接的な原因は，CPU のメモリ不足にあった．当初は電力不足や実験する LLM のモデルサイズの大きさ，複数のコンテナを同時に起動していることが原因と考えられた．そのため，`sudo nvidia-smi -pl` コマンドで GPU が使用できる電力を制限する，LLM のパラメータ数を減らして学習を行う，一つのコンテナだけを使用して実験するなどの検証を行った．あらゆる可能性を排除していった結果，CPU のメモリの過剰使用に原因があることを突き止めた．この問題は，2.2.4 節で述べた DeepSpeed の設定に関連している．LLM の事前学習では LLM を単一の GPU に載せられないため，LLM のパラメータやオプティマイザの一部を CPU に載せる必要があり，この役割を担うのが DeepSpeed である．最初に使用していた DeepSpeed Stage3 はパラメータやオプティマイザのほとんどを CPU に載せる設定であるが，CPU のメモリ (RAM) 256GB では容量が不足してしまう．システムのメモリが不足した場合に自動的に動作する OOM killer(Out-Of-Memory Killer) が発動し，システムにとって重要なプロセスを殺してしまった結果，サーバの異常停止に至ったと考えられる．問題の解決には，DeepSpeed Stage3 を DeepSpeed Stage2 に変更し，使用する CPU メモリを減ら

**https://zenn.dev/bilzard/scraps/287fc28447698c

††https://github.com/NVIDIA/nvidia-docker/issues/1730

す必要があった.

A.2　技術的工夫

本研究では LLM のパラメータをそのまま用いて実験を行った．そのため，パラメータの保存やパラメータの値に関する問題が絶えなかった．本節では，パラメータに関連して生じた問題に対し，筆者が行った対処法について述べる．

A.2.1　パラメータの保存方法

4.1 節では，提案手法の学習フェーズについて説明している．学習フェーズでは，LLM の事前学習を行うだけではなく，LLM のパラメータを保存する処理も行っている．パラメータの保存では，事前学習前の LLM のパラメータ，事前学習後の LLM のパラメータをそれぞれ保存している．

今回の実験対象とした LLM は 7B 程度のサイズのモデルである．そして，事前学習時には全てのレイヤのパラメータを更新するため，7B モデルのパラメータすべてを保存する必要がある．ここで，7B モデルのパラメータすべてをそのまま保存した際の保存サイズを計算してみる．パラメータ数の 7B は 70 億であるため，7×10^9 と表わせる．また，データ型が FP32 であった場合は，32 ビット (4 バイト) である．4 バイトのデータが 70 億個あるため，$4 \times 7 \times 10^9$ より，28GB である．さらに，Transformer の Trainer を用いてモデルの保存を行った場合はモデルのパラメータだけではなく，オプティマイザも保存される．オプティマイザはモデルの保存容量の数倍であるため，モデルとオプティマイザの容量を合計すると，一つの 7B モデルで 120GB ほどの保存容量となる．一つの実験では 100 から 1,000 個ほどのモデルを保存するため，最低でも 12TB の保存容量が必要となってしまう．

当初は保存に必要なデータ容量を考えずに実験を行ってしまったため，研究室のサーバの保存容量数十 TB をすぐに使い切ってしまい，他のメンバーのデータ保存ができなくなるというインシデントを引き起こした．そこで，軽量な保存方法を模索した．結論として，試行錯誤した範囲では NumPy のバイナリファイル形式 (.npy ファイル) としてパラメータを保存するのが最も良かった．

本手法では，モデルのパラメータだけを扱い，学習後のモデルを使用して推論することはない．そのため，推論に必要なデータも保存する Hugging Face Transformers 形式（pytorch_model.bin ファイル）や TensorFlow 形式（.ckpt ファイル）を使用する必要はない．パラメータの情報だけ保存できれば良いため，csv 形式での保存を試した．しかし，csv 形式での保存は保存に時間がかかるだけではなく，保存容量もあまり削減することができなかった．そこで，バイナリファイルとして保存することで，保存容量の削減に成功した．バイナリファイルとして保存することで，一つのモデルあたり 120GB 必要であった保存容量を 12GB まで削減することができた．

A.2.2 モデルの保存に要する時間

LLM の学習を終えた際には，モデルを保存する必要がある．また，Transformer に用意されているデフォルトの Early Stopping を使用すると，毎 epoch ごとにモデルが保存される．この際，保存に要する時間が全体の学習時間のボトルネックとなる場合がある．指定した epoch 数の学習後にモデルを保存する場合，モデルの保存回数は一回で済むため，モデルの保存時間はボトルネックとならない．しかし，Early Stopping を行う場合，毎 epoch ごとにモデルが保存されるため，モデルの保存がボトルネックとなる．

そこで，Early Stopping を行う場合のモデル保存方法を以下のように変更した．

- 毎 epoch ごとにモデルの保存を行わず，検証データで最も誤差が小さかった epoch 数の時のモデルを変数として保持する．
- 学習完了後，最も誤差が小さかったモデルを一度だけ保存する．

Hugging Face のデフォルトの仕様では，各 epoch の終了時に検証データの誤差を比較し，最小値を更新するたびにモデルをファイルに上書き保存していた．この部分を修正し，モデルをファイルではなくメモリ上の変数に保存することで，保存時間を削減した．これらの変更により，モデル保存に要する時間を大幅に短縮し，学習全体の効率を向上させた．モデルの保存にかかる時間を t，モデルを保存する回数を n_s とする．変更前のプログラムでは $t \times n_s = tn_s$ もの時間がかかっていた

のに対し，変更後のプログラムでは t しか時間がかからないため，n_s 倍の時間短縮を実現することができた．

A.2.3 nan 対策

FP16 で LLM の事前学習を行っていた際，Loss が nan になる現象が多発した．Loss が nan になると学習が停止してしまい，提案手法の実験を行うことが出来なくなってしまう．そこで，学習率を下げることで Loss が nan になる現象を無くすことができた．

Loss が nan になる原因としては，以下の二点が考えられる．

1. 学習率が高すぎたこと
2. 浮動小数点の精度不足

実際に，多くの LLM 開発記事[‡‡][§§][¶¶]では，FP16 ではなく BF16 を使うように推奨している．しかし，今回の実験環境では BF16 を使用することができなかった．理由としては，大半の GPU が BF16 を扱うことのできない旧式の GPU であったためである．GPU の性能表[***]を見ると，GPU と扱うことのできる計算精度との対応関係が示されている．実験環境では，NVIDIA RTX A6000 や NVIDIA TITAN RTX を用いていたが，NVIDIA TITAN RTX は GPU アーキテクチャが Turing と数世代前のもので，BF16 による計算ができない．そのため，FP16 で実験を行うこととなった．

上記の開発記事では，BF16 での学習に変更する以外に，学習率を下げることで対処できた事例も記されていた．そこで，通常の LLM の事前学習で使用される学習率よりも十倍程度小さい学習率に設定して実験を行った．結果として，学習率を下げることで Loss が nan になる問題を解決した．

[‡‡] https://zenn.dev/matsuolab/articles/92647e0664ee76
[§§] https://discuss.huggingface.co/t/training-loss-0-0-validation-loss-nan/27950
[¶¶] https://qiita.com/takeuchiseijin/items/909c48b57127a37fbd12
[***] https://www.hpc.co.jp/product/wp-content/uploads/sites/3/2022/07/GPU-list_A3.pdf