第 1 章. ファウンデーション・モデルによる AI アプリケーション構築入門

2020 年以降の AI を一言で表すとすれば、それは*規模*だろう。ChatGPT、Google の Gemini、Midjourney のようなアプリケーションの背後にある AI モデルは、世界の電力のごく一部を消費するほどの規模であり、それらを訓練するためにパブリックに利用可能なインターネットデータを使い果たす危険性がある。

AI モデルのスケールアップは、2 つの大きな結果をもたらす。第一に、AI モデルはより強力になり、より多くのタスクに対応できるようになり、より多くのアプリケーションを可能にする。より多くの人々やチームが AI を活用して生産性を向上させ、経済価値を作成し、生活の質を向上させる。

第二に、大規模な言語モデル（LLM）のトレーニングには、データ、コンピューティングリソース、特殊化された人材が必要であり、そのような人材を確保できる組織は限られている。このため、*モデル・アズ・ア・サービス*が登場することになった。AI を活用してアプリケーションを構築したい人は誰でも、モデル構築に先行投資することなく、これらのモデルを利用できるようになった。

つまり、AI アプリケーションの需要が高まる一方で、AI アプリケーション構築へのエントリーの障壁は低下している。これにより、*AI エンジニアリング*（容易に入手可能なモデルの上にアプリケーションを構築するプロセス）は、最も急成長しているエンジニアリング分野のひとつとなった。

機械学習（ML）モデルの上にアプリケーションを構築することは新しいことではない。LLM が注目されるずっと以前から、AI は商品推奨、詐欺検出、解約予測など、多くのアプリケーションを動かしていた。AI アプリケーションを生産化する多くの原則は変わらないが、大規模で容易に利用可能な新世代のモデルは、新たな可能性と新たな課題をもたらしている。

この章ではまず、AI エンジニアリングの爆発的な普及の鍵となった基盤モデルについて概観する。続いて、AI が得意なこと、まだ得意でないことを示す、さまざまな AI の成功事例について説明する。AI の能力が日々拡大するにつれ、その将来の可能性を予測することはますます難しくなっている。しかし、既存の応用パターンは、今日の機会を発見し、将来 AI がどのように使われ続けるかを知る手がかりとなる。

この章の最後に、基礎モデルで何が変わったのか、何が変わらないのか、今日の AI エンジニアの役割は従来の ML エンジニアとどう違うのかなど、新しい AI スタックの概要を説明する。[1]

AI エンジニアリングの台頭

ファウンデーション・モデルは、大規模な言語モデルから生まれた。ChatGPT や GitHub の Copilot のようなアプリケーションは、突然現れたように見えるかもしれないが、1950 年代に最初の言語モデルが登場して以来、数十年にわたるテクノロジーの進歩の集大成なのだ。このセクションでは、言語モデルから AI エンジニアリングへの進化を可能にした重要なブレークスルーをたどる。

言語モデルから大規模言語モデルへ

言語モデルは以前から存在していたが、現在のような規模に成長できたのは、*自己監視*があったからだ。このセクションでは、言語モデルと自己監視の意味を簡単に説明する。すでに言語モデルや自己監視についてご存知の方は、このセクションは読み飛ばして構わない。

言語モデル

言語モデルは、1 つまたは複数の言語に関する統計的情報をエンコーディングする。直感的に言えば、この情報は、ある単語がある文脈で出現する可能性を教えてくれる。例えば、"My favorite color is __"という文脈が与えられた場合、英語をエンコーディングした言語モデルは、"car "よ

りも "blue "の方が頻度が高いと予測するはずだ。

言語の統計的性質は何世紀も前に発見されていた。1905 年の物語『踊る男たちの冒険』では、シャーロック・ホームズが英語の単純な統計情報を利用して、謎めいた棒グラフのシーケンスを解読した。英語で最もよく使われる文字が *E* であることから、ホームズは最もよく使われる棒グラフが *E* *を表し*ているに違いないと推理した。

その後、クロード・シャノンは、第二次世界大戦中に敵のメッセージを解読するために、より高度な統計学を用いた。英語をモデル化する方法に関する彼の研究は、1951 年の画期的な論文「Prediction and Entropy of Printed English」で発表された。この論文で紹介されたエントロピーを含む多くの概念は、今日でも言語モデリングに使われている。

その昔、言語モデルには 1 つの言語が含まれていた。しかし今日では、言語モデルは複数の言語を含むことができる。

言語モデルの基本単位は *トークン*である。トークンはモデルによって、文字であったり、単語であったり、単語の一部（-tion など）であったりする。[2]例えば、ChatGPT を支えるモデルである GPT-4 は、図 1-1 に示すように、"I can't wait to build AI applications "というフレーズを 9 つのトークンに分割する。この例では、"can't "という単語が "*can* "と "*t* "の2 つのトークンに分割されていることに注意してほしい。OpenAI の Web サイトでは、さまざまな OpenAI モデルがテキストをどのようにトークン化するかを見ることができる。

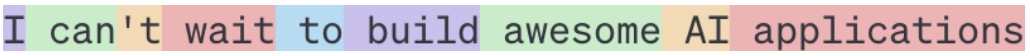

図 1-1. GPT-4 がどのようにフレーズをトークン化するかの例。

原文をトークンに分割するプロセスを *トークン化*と呼ぶ。GPT-4 では、平均的なトークンの長さは単語の約 3/4 である。つまり、100 個のトークンは約 75 個の単語ということになる。

モデルが扱うことのできるすべてのトークンのセットが、モデルの *語彙*

である。アルファベットの数文字を使って多くの単語を作るのと同じように、少ない数のトークンを使って大きな数の異なる単語を作ることができる。Mixtral 8x7B モデルの語彙数は 32,000 である。GPT-4 の語彙サイズは 100,256 である。トークン化のメソッドと語彙サイズはモデル開発者が決める。

注

なぜ言語モデルは*単語*や *文字ではなく* *トークン*を単位とするのか？主な理由は 3 つある：

1. 文字に比べて、トークンは単語を意味のある構成要素に分割することができる。例えば、"cooking "は "cook "と "ing "に分割することができる。
2. 一意な単語よりも一意なトークンの方が少ないため、モデルの語彙サイズが小さくなり、モデルがより効率的になる（第 2 章で説明）。
3. トークンは、モデルが未知の単語を処理するのにも役立つ。例えば、"chatgpting "のような造語は、"chatgpt "と "ing "に分割され、モデルの構造理解に役立つ。トークンは、個々の文字よりも多くの意味を保持しながら、単語よりも少ない単位でバランスがとれている。

言語モデルには大きく分けて、*マスク言語*モデルと*自己回帰言語モデル*の2 種類がある。これらは、トークンを予測するためにどのような情報を利用できるかによって異なる：

マスク言語モデル

マスクされた言語モデルは、欠落した*トークンの前後の文脈を使用し*て、シーケンス内の任意の場所で欠落したトークンを予測するように訓練される。要するに、マスクされた言語モデルは、空白を埋めることができるように訓練される。例えば、"My favorite __ is blue"（私の好きな__は青です）という文脈があれば、マスキングされた言語モデルは、空白が "color"（色）である可能性が高いと予測する。マスクされた言語モデルのよく知られた例は、トランスフォーマーからの双方向エンコーディング表現（BERT）である（Devlin et al.

本稿執筆時点では、マスクされた言語モデルは、センチメント分

析やテキスト分類などの非生成タスクに一般的に使用されている。また、コードのデバッグのように、全体的なコンテキストの理解を必要とするタスクにも有用である。このようなタスクでは、モデルはエラーを特定するために前後のコードを理解する必要がある。

自己回帰言語モデル

自己回帰言語モデルは、*直前のトークンのみを使用して*、シーケンス内の次のトークンを予測するように学習される。私の好きな色は○○です」の次に何が来るかを予測する。[3]自己回帰モデルは、次々とトークンを生成し続けることができる。今日、自己回帰言語モデルはテキスト生成に適したモデルであり、そのためマスク言語モデルよりも人気がある。[4]

図 1-2 は、この 2 種類の言語モデルを示している。

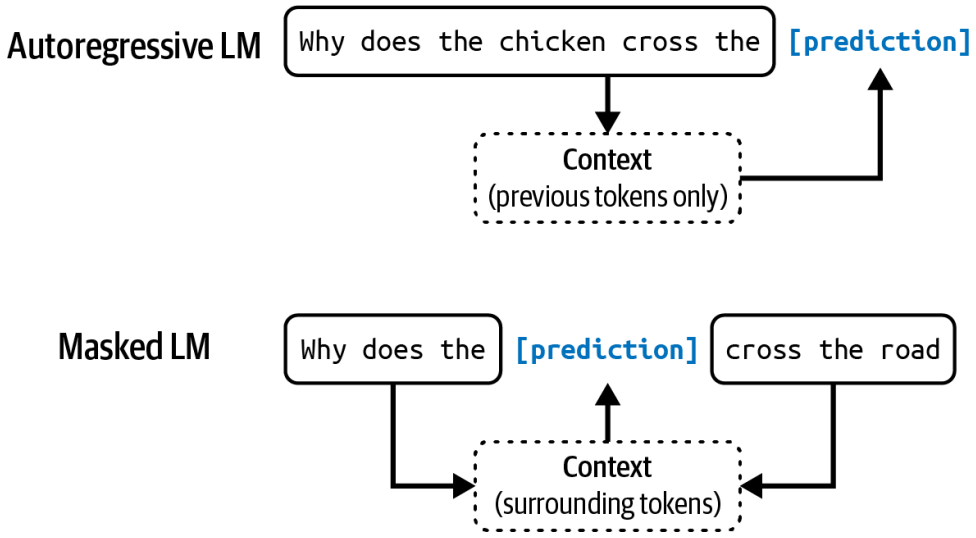

図 1-2. 自己回帰言語モデルとマスク言語モデル。

注

本書では、明示的な記述がない限り、*言語モデル*は自己回帰モデルを指す。

言語モデルの出力は自由である。言語モデルは、固定された有限の語彙を使って、無限に可能な出力を構築することができる。オープンエンド

な出力を生成できるモデルは*生成的*と呼ばれ、それゆえ*生成的AI*と呼ばれる。

言語モデルは*補完*マシンと考えることができる：テキスト（プロンプト）が与えられると、そのテキストを補完しようとする。これがその例だ：

```
Prompt (from user): "To be or not to be"
Completion (from language model): ", that is the question."
```

重要なのは、補完は確率に基づいた予測であり、正しいことが保証されているわけではないということだ。言語モデルのこのような確率的な性質が、言語モデルをエキサイティングなものにも、もどかしいものにもしているのだ。この点については、第 2 章で詳しく説明する。

単純なことのように聞こえるが、補完は信じられないほど強力だ。翻訳、要約、コーディング、数学の問題を解くなど、多くのタスクは完了タスクとしてフレーム化できる。例えば、次のような促しがある：「というプロンプトが与えられたとする：「Comment ça va "と、ある言語から別の言語へ効果的に翻訳することができる。

別の例として、プロンプトが与えられたとする：

```
Question: Is this email likely spam? Here's the email: <email content>
Answer:
```

言語モデルは、それを完成させることができるかもしれない：「スパムの可能性が高い」と補完することができるかもしれない。

補完は強力だが、補完は会話をすることとは違う。例えば、あなたが補完マシンに質問をした場合、補完マシンは質問に答える代わりに別の質問を追加することで、あなたが言ったことを補完することができる。"Post-Training "では、ユーザからのリクエストに対してモデルを適切に応答させる方法について説明する。

図 1-4. Super-NaturalInstructions ベンチマークのタスクの範囲（Wang et al.）

あなたが望むものをモデルに生成させるために使えるテクニックは複数ある。例えば、望ましい製品説明の例を用いて、詳細な指示を作ることができる。このアプローチは*プロンプト・エンジニアリング*である。より良い説明を生成するために、モデルを顧客レビューのデータベースに接続することもできる。データベースを使用して説明を補足することは、*検索補完型生成*（RAG）と呼ばれる。また、高品質の商品説明のデータセットでモデルを*微調整*し、さらに訓練することもできる。

プロンプト・エンジニアリング、RAG、ファインチューニングは、あなたのニーズにモデルを適合させるために使用できる、非常に一般的な 3 つの AI エンジニアリングテクニックである。本書の残りの部分では、これらすべてについて詳しく説明する。

既存の強力なモデルをあなたのタスクに適応させることは、一般的に、あなたのタスクのためにゼロからモデルを構築するよりもずっと簡単である。例えば、10 例と週末 1 日であるのと、100 万例と 6 ヶ月であるのとは違う。ファウンデーション・モデルは、AI アプリケーションの開発をより安価にし、市場投入までの時間を短縮する。モデルを適応させるためにどれだけのデータが必要かは、使用するテクニックによって異なる。本書では、各テクニックについて説明する際に、この問題にも触れる。しかし、タスクに特化したモデルにはまだ多くの利点がある。例えば、かなり小さくなり、より速く、より安く使えるようになるかもしれない。

独自のモデルを構築するか、既存のモデルを活用するかは、チーム自身が答えを出さなければならない典型的なバイ・オア・ビルドの問題である。本書を通しての議論は、その決断の助けとなるだろう。

基礎モデルから AI エンジニアリングまで

AI エンジニアリング 、基礎モデルの上にアプリケーションを構築するプロセスを指す。ML エンジニアリングや MLOps（ML 演算子の略）として知られるプロセスだ。なぜ今 AI エンジニアリングなのか？

従来の ML エンジニアリングが ML モデルの開発を伴うとすれば、AI エンジニアリングは既存のモデルを活用する。強力な基盤モデルが入手可能で、利用しやすいということは、3 つの要因につながり、これらが相まって、AI エンジニアリングが学問分野として急成長するための理想的な条件付きとなっている：AI 製品において AI がどのような役割を果たすかは、アプリケーションの開発とその要件に影響する。アップル社には、AI を製品に使用するさまざまな方法を説明した素晴らしい文書がある。今回の議論に関連する 3 つのポイントを紹介しよう：