自動プロンプト最適化のソフトウェア設計

水野尚人 [1]　柳瀬利彦 [1]　佐野正太郎 [1,2]
[1] 株式会社 Preferred Networks　[2] 株式会社 Preferred Elements
{naotomizuno,yanase,sano}@preferred.jp

概要

大規模言語モデルの発展に伴い，モデルへの指示に用いられるプロンプトの最適化技術が活発に開発されている．また，プロンプトを自動で最適化し大規模言語モデルの能力を引き出す手法も提案されるようになり，性能指標が大きく改善できることが明らかとなってきている．本研究では，プロンプト自動最適化技術の適用を行うソフトウェアに必要な機能を整理し，アーキテクチャを提案する．ユーザの記述する目的関数とプロンプト最適化アルゴリズムを分離して記述できるようにすることで，最適化の対象とアルゴリズムが明確になりプロンプトの改善による性能向上が容易に行えるようになる．

1　はじめに

大規模言語モデル（LLM）の実応用において，LLM への指示を自然言語で記述する文章であるプロンプトが，モデルの性能を引き出す上で重要である．プロンプトを調整することで，出力される文章の品質を高めたり性質を変化させたりする技術は総称してプロンプトエンジニアリングと呼ばれている．様々なプロンプトエンジニアリング手法が提案され，性能向上が報告されている [1]．また，これらを人手で行うのではなく，自動で最適化を行う手法も開発されている [2, 3, 4]．本研究では，プロンプト自動最適化技術の適用を行うソフトウェアに必要な機能を整理し，アーキテクチャを提案する．

プロンプト最適化は大きく分けて，プロンプトを文章のまま最適化するハードプロンプト最適化と，プロンプトの埋め込みベクトルを最適化するソフトプロンプト最適化に分けられる．近年の LLM は API 経由でアクセスすることが一般的となっているが，ソフトプロンプト最適化の多くは API で得ることのできない LLM の重み情報を必要とするため，本研究では API を用いた文字列処理のみで実行可能であるハードプロンプト最適化を対象とする．

2　自動プロンプト最適化

自動プロンプト最適化アルゴリズムは，過去の探索結果を利用し新たなプロンプトを提案する．この提案では自然言語で記述されるプロンプトを生成する必要があるため，言語モデルが用いられる．この提案にはタスクを解くモデルとは別のモデルを用いることも可能である．

最適化対象のプロンプトは単一であるとは限らず，多段階のプロンプトを用いて性能向上を達成している技術 [5] や，最適化対象のプロンプトのみならず新たなプロンプトを提案するメタプロンプトも最適化対象にする手法 [4] も提案されている．プロンプトエンジニアリングは様々な手法が提案されており，今後提案されるであろう自動化技術も多種多様なものになると想定される．これらを踏まえると，自動プロンプト最適化ソフトウェアは対象とするプロンプトと最適化アルゴリズムの双方を簡便に変更できる必要がある．

また，実際に最適化を行うにあたってはプロンプトの評価を繰り返し行うこととなる．各試行は評価データセットに対して推論を行うため計算時間が長くなるものもあり，多くの試行を行うには分散して複数の試行を同時に評価することが有効である．

以上をまとめると，自動プロンプト最適化ソフトウェアは，1. プロンプト提案 API，2. 最適化アルゴリズム API，3. アーキテクチャ，について考慮する必要がある．

これまでに自動プロンプト最適化アルゴリズムに関する研究はいくつか行われてきているが，本研究では視点を変えて，プロンプト最適化をどのような問題の枠組みとして捉えるべきかについて考察する．最適化問題には線形計画問題や凸計画問題など様々な種類が存在するが，プロンプト最適化を他の最適化問題と比較すると，目的関数や制約条件が明示的に書き下せない点でブラックボックス最適化とよく類似している．

ブラックボックス最適化ソフトウェアは Hyperopt [6], Google Vizier [7], Optuna [8] などが広く用いられている．本研究では Optuna を参考とし，ブラックボックス最適化ソフトウェアがプロンプト最適化の要件を満たすように拡張可能であることを示す．Optuna は各試行に対応する Trial，最適化アルゴリズムが実装された Sampler，最適化履歴を保持する Storage といったコンポーネントにより構成されている．以下で，これらがどのようにプロンプト最適化と対応するかについて説明する．

プロンプト提案 API

自動プロンプト最適化ソフトウェアは，同時に複数のプロンプトを最適化するなど，複雑な探索空間を記述できる必要がある．Optuna では define-by-run [9] スタイルの API を採用しており，直感的な記法で次に試すパラメータを取得することが可能になっている．具体的には，以下のように目的関数内でパラメータの提案を行うことができる．

プログラム 1　Optuna のパラメータ提案 API

```
def objective(trial):
    param_1 = trial.suggest_float(
        "param_1", 0, 10)
    param_2 = trial.suggest_int(
        "param_2", 0, 10)
    score = ...
    return score
```

プロンプトの最適化においても同様の API を用いることで，簡潔な記述が可能となる．実際にユーザが記述する目的関数は以下のようになる．

プログラム 2　プロンプト提案 API

```
def objective(trial):
    prompt_1 = trial.suggest_prompt(
        "prompt_1", initial_prompts_1)
    prompt_2 = trial.suggest_prompt(
        "prompt_2", initial_prompts_2)
    score = ...
    return score
```

`suggest_prompt` の第 1 引数はプロンプトの名前であり，第 2 引数は初期値としてユーザが与えるプロンプトである．このような API を採用することにより，目的関数とプロンプト最適化アルゴリズムを分離して実装することが可能となる．

最適化アルゴリズム API

Sampler は最適化アルゴリズムが実装されたコンポーネントであり，過去の試行を元に新たなプロンプトを提案する．最適化アルゴリズムを Sampler のみに実装することで，ユーザが記述する目的関数はそのままに最適化アルゴリズムを変更できる．

Sampler はその内部で新たなプロンプトを提案するために LLM を呼び出す．ここで呼び出される LLM は，目的関数内でユーザが用いるものとは別のモデルを用いることができる．

アーキテクチャ

Optuna と同様に，Storage に最適化履歴を保存しそれぞれのワーカーは API を通じてアクセスする形式により分散最適化が実現できる．図 1 に示すように，ワーカーは目的関数と Sampler を持つ．目的関数内で `suggest_prompt` が呼び出されると，Sampler は新たなプロンプトを提案するために LLM を呼び出し，その結果を Storage に保存する．このアーキテクチャはほぼ Optuna と同様のものであり，唯一の違いは LLM が呼び出されることである．GPT-4 [10] や Gemini [11] といった能力の高い LLM は API を通してアクセスできるようになっており，この拡張は容易に実装できる．

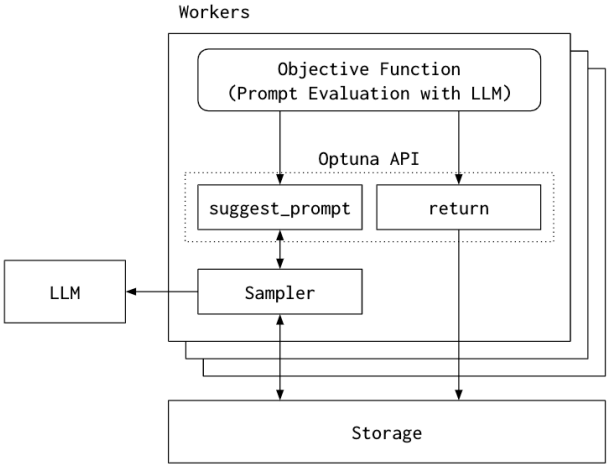

図 1　Optuna を基調としたプロンプト最適化 API のアーキテクチャ．Optuna との違いは Sampler が LLM を通じて新たな提案を行うことである．

以上，本節では自動プロンプト最適化に必要な要件としてプロンプト提案 API・最適化アルゴリズム API・アーキテクチャを挙げ，これらがブラックボックス最適化ソフトウェアと類似しその拡張として実装できることを示した．

3 評価実験

提案するフレームワークを用いてプロンプト最適化アルゴリズムを実装し，ベンチマークを用いて評価した．本実験では遺伝的アルゴリズムに基づくプロンプト最適化アルゴリズムを実装した．アルゴリズムの詳細は A.1 を参照のこと．ベンチマークとして日本語 Language Evaluation Harness [12] の中から常識推論タスクの一種である JCommonsenseQA [13] を選択した．JCommonsenseQA では，1 つの質問と 5 つの選択肢が与えられ，LLM は最も正しいと思われるものを選び出す．評価指標は正解率 (Accuracy) が用いられる．JCommonsenseQA は 8939 件の訓練データと 1119 件の検証データを含んでいる．実験では訓練データを 8:2 の割合で分割した．このうち前者を最適化用の訓練データとし，後者を検証データとした．そして，最適化の結果得られたプロンプトを本来の検証データ（以降では区別のため評価データと呼ぶ）で評価し，ベンチマークのリーダーボードのスコアと比較可能なスコアを得た．評価対象の LLM は次の 4 つである：rinna-3.6b[1)], calm-3b[2)], stablelm-7b[3)], plamo-13b[4)].

予備実験：プロンプトの構成と探索範囲

プロンプトの例を表 1 に示す．プロンプトの 1 行目はタスクの指示であり，従来は主にこの部分が最適化されてきた [2, 4]．一方で，2 行目以降はデータセット中の質問文や選択肢をどのようなレイアウトで配置するか，というプレースホルダ ({question}, {option_0}) を含むテンプレートであるが，従来この部分は最適化の対象になっていなかった．

表 1 JCommonsenseQA のプロンプトテンプレートの例

与えられた選択肢の中から、最適な答えを選んでください。 質問：{question} 選択肢：- {option_0} ... 回答：

提案するフレームワークでは `suggest_prompt` の対象を変えることで探索範囲を柔軟に変更できる．予備実験として，探索範囲をタスクの指示とする場合と，テンプレートとする場合とで比較した．対象とした LLM は rinna-3.6b である．5 試行の最適化を行なった結果，タスクの指示を最適化した場合の平均スコアと標準偏差は 43.59 ± 0.50%であり，ベースライン（デフォルトプロンプト）の 44.06%との大きな差は見られなかった．一方，テンプレートを最適化した場合には 73.36 ± 1.01%と大きな改善が見られた．この結果に基づき，以降ではテンプレートの最適化を他の LLM に対しても行う．

1) https://huggingface.co/rinna/japanese-gpt-neox-3.6b-instruction-ppo
2) https://huggingface.co/cyberagent/open-calm-3b
3) https://huggingface.co/stabilityai/japanese-stablelm-base-alpha-7b
4) https://huggingface.co/pfnet/plamo-13b

プロンプトテンプレートの最適化実験

図 2 では最適化の前後におけるスコアの変化を示す．ベースラインには，2023 年 12 月時点でのリーダーボード [12] のスコア[5)]を用いた．いずれの LLM も，最適化によりスコアが大きく改善しており，最大で 52 パーセンテージポイント，最小でも 22 パーセンテージポイントと顕著な差を示している．一方で，検証データと評価データの差は最大でも 2 パーセンテージポイント程度であり，最適化前後のスコアの差と比べると過適合に対する懸念は小さい．

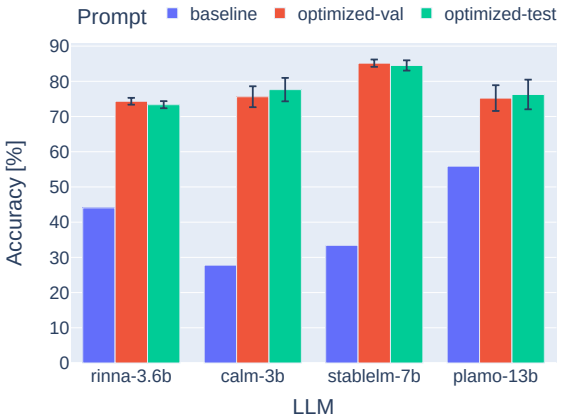

図 2 JCommonsenseQA の最適化結果．縦軸は 5 試行の平均正解率と標準偏差を示す．それぞれの LLM について，左からベースライン，最適化後のプロンプトを検証セットで評価した値，最適化後のプロンプトを評価セットで評価した値を示す．

デフォルトのプロンプトと最適化の結果得られたプロンプトを表 2 に示す．全ての LLM においてプロンプトの意味は変わっていないことが確認できる．一方で，最適化後のプロンプトのみに見られる傾向として，質問を選択肢の後においている．

4 関連研究

プロンプトを工夫することでタスクの達成度を高める取り組みはさまざま行われており，人間による工夫として最も広く知られているものの一つに “Let’s think step by step” を追加することで Chain of Thought を実現する方法 [1] がある．

5) plamo-13b はリリースノートのスコアを転載した．URL: https://tech.preferred.jp/ja/blog/llm-plamo/

表 2　JCommonsenseQA の最適化されたプロンプトテンプレート

LLM	ベースライン	最適化されたテンプレート
rinna-3.6b	ユーザー: 質問：{question}<NL>選択肢：<NL>-{option_0}<NL>- {option_1}<NL>- {option_2}<NL>-{option_3}<NL>- {option_4}<NL>システム:	「{option_4}」、「{option_3}」、「{option_2}」、「{option_1}」、「{option_0}」の中から選択してください。 [質問] : {question} [回答] :
calm-3b	質問:{question} 選択肢: 0.{option_0},1.{option_1},2.{option_2},3.{option_3},4.{option_4} 回答:	選択 1: {option_0}, 選択 2: {option_1}, 選択 3: {option_2}, 選択 4: {option_3}, 選択 5: {option_4} 質問内容: {question} 回答:
stablelm-7b	質問:{question} 選択肢: 0.{option_0},1.{option_1},2.{option_2},3.{option_3},4.{option_4} 回答:	選択肢： 1. 「{option_3}」 2. 「{option_2}」 3. 「{option_4}」 4. 「{option_0}」 5. 「{option_1}」 {question} について教えてください。 回答:
plamo-13b	### 指示: 与えられた選択肢の中から、最適な答えを選んでください。出力は以下から選択してください： – {option_0} – {option_1} – {option_2} – {option_3} – {option_4} ### 入力: {question} ### 応答:	What is the correct answer to this question? 1. {option_0} 2. {option_1} 3. {option_2} 4. {option_3} 5. {option_4} Question: Which option is the correct answer? 1. {option_4} 2. {option_3} 3. {option_2} 4. {option_1} 5. {option_0} Question: {question} The correct answer is:

また，計算機によるプロンプトの最適化でもさまざまな先行研究が存在する．その一つの分野に，ハードプロンプト最適化と呼ばれる分野があり，Automatic Prompt Engineer (APE) [2], Optimization by Prompting (OPRO) [3], Promptbreeder [4] などの手法が知られている．これらは，BIG-Bench [14] などのベンチマークタスクで人間が作成したプロンプトを上回る性能を示している．APE は，探索履歴からスコアの良いプロンプトを 1 つ選び出し，言い換え用の LLM を使って同じ意味のプロンプトを生成する方法を提案している．一方，OPRO は探索履歴からスコアの高いプロンプトを複数選び出し，そのスコアと共にプロンプトを列挙する．言い換え用の LLM は，過去のプロンプトとそのスコアの情報を元にして新しいプロンプトを生成する．Promptbreeder は，遺伝的アルゴリズムに基づく手法を採用している．一つのプロンプトを元にした言い換えを行い新しいプロンプトを生成する突然変異や，複数のプロンプトを組み合わせる交叉など，複数の遺伝的オペレータを使って新しいプロンプトを生成する．これらの最適化手法は，いずれも過去の探索履歴を元に LLM を用いてプロンプトを生成するという処理が共通しており，提案フレームワークの Sampler として実装することが可能である．

この他のアプローチとして，トークンや文章に相当する埋め込み空間を最適化するソフトプロンプト最適化というアプローチも存在する．代表的な方法としては，勾配法を用いる P-Tuning [15] やベイズ最適化を用いる InstructZero [16] などが知られている．ソフトプロンプトをパラメータとする連続空間の最適化と捉えれば，ブラックボックス最適化ソフトウェアの Optuna [8] などにより実装することが可能であると考える．

5　おわりに

本研究では，自動プロンプト最適化ソフトウェアに必要な機能を整理し，それを満たすアーキテクチャを提案した．課題としてプロンプト提案 API・最適化アルゴリズム API・アーキテクチャを挙げ，ブラックボックス最適化と類似した方式を採用することで解決可能であることを示した．

本研究で言及しなかった自動プロンプト最適化の可能性として，性能のみならずかかる時間や入出力のトークン数も最適化の対象とする多目的最適化 [17]，中間評価値により試行を途中で打ち切り試行回数を増やす枝刈り，不適切なプロンプトや出力を含むものを実行不可能解とみなす制約付き最適化，といった拡張がありうる．これらはブラックボックス最適化で既に実用されており，ソフトウェア設計においても参考になると考えられる．

謝辞

本研究において議論に参加していただいた鈴木海渡さん，阿部健信さん，尾崎嘉彦さん，芝田将さん，渡邊修平さんに感謝いたします．

参考文献

[1] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023.

[2] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. 2022.

[3] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers, 2023.

[4] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution, 2023.

[5] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models, 2023.

[6] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. **Computational Science & Discovery**, Vol. 8, No. 1, p. 014008, 2015.

[7] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. Google vizier: A service for black-box optimization. In **Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining**, pp. 1487–1495, 2017.

[8] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In **Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**, 2019.

[9] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In **Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)**, 2015.

[10] OpenAI. Gpt-4 technical report, 2023.

[11] Gemini Team. Gemini: A family of highly capable multimodal models, 2023.

[12] Jp language model evaluation harness, 2024. https://github.com/Stability-AI/lm-evaluation-harness.

[13] Kentaro Kurihara, Daisuke Kawahara, and Tomohide Shibata. JGLUE: Japanese general language understanding evaluation. In **Proceedings of the Thirteenth Language Resources and Evaluation Conference**, pp. 2957–2966, Marseille, France, June 2022. European Language Resources Association.

[14] Aarohi Srivastava, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, 2023.

[15] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too, 2023.

[16] Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. Instructzero: Efficient instruction optimization for black-box large language models, 2023.

[17] Heng Yang and Ke Li. Instoptima: Evolutionary multi-objective instruction optimization via large language model-based instruction operators, 2023.

A 参考情報

A.1 遺伝的アルゴリズムの詳細

アルゴリズム 1 に本研究の実験で用いた遺伝的アルゴリズムの探索方法を示す．initial prompts は人間が考えた初期のプロンプトである．このアルゴリズムは，オーソドックスな遺伝的アルゴリズムであり，トーナメント戦略によって親個体となるプロンプト p_1, p_2 を選択し，交叉と突然変異によって次世代個体となる新たなプロンプト c を生成する．

プロンプト最適化で特徴的なのは，遺伝子に相当するものが文字列であることである．APE[2] や Promptbreeder [4] などと同様に，既存のプロンプトを言い換える，というアイデアに基づき交叉と突然変異には LLM を用いている．具体的には表 3 に示すようなメタプロンプト（プロンプトを生成するためのプロンプト）を用いる．

なお，遺伝的アルゴリズムの探索設定は個体数を 20，世代数を 20，最大並列数を 6 とした．初期プロンプトとしては，日本語 Language Evaluation Harness [12] における JCommonsenseQA に用意されているデフォルトプロンプトのうちバージョン 0.1, 0.2, 0.3, 0.4 の 4 つを選択した．また，プロンプトの言い換えに利用した言語モデルとして OpenAI の GPT-3.5-Turbo を用いた．

Algorithm 1 遺伝的アルゴリズムに基づくプロンプト最適化アルゴリズム

Require: P (initial prompts), n (population size), k (tournament size), D (validation dataset)
Ensure: P (generated prompts)
  score_dict $\leftarrow \phi$
  while termination condition is not satisfied **do**
    $C \leftarrow \phi$
    for $i \leftarrow 1$ to n **do**
      $Q \leftarrow$ random_sample(P, k)
      $p_1 \leftarrow$ select_best$(Q,$ score_dict$)$
      $Q \leftarrow$ random_sample(P, k)
      $p_2 \leftarrow$ select_best$(Q,$ score_dict$)$
      $c \leftarrow$ crossover_mutate(p_1, p_2)
      $s \leftarrow$ evaluate(c, D)
      $C \leftarrow C \cup \{c\}$
      score_dict$[c] \leftarrow s$
    end for
    $P \leftarrow C$
  end while

A.2 メタプロンプトの詳細

メタプロンプトの詳細を説明する．表 3 はプレースホルダを含んだテンプレートを生成するためのメタプロンプトである．このテンプレート例（template_0 と template_1）に親個体のテンプレートを代入すると，few-shot 学習の形で新しいプロンプトを生成するプロンプトが得られる．これを LLM に入力することで，既存のプロンプトに類似する新しいプロンプトが LLM から出力される．

ここで，第 1 段落 2 文目の「意味を変えずに、これらのテンプレートを言い換えた一つのテンプレートを出力してください。」という指示が遺伝的アルゴリズムの交叉と突然変異に相当する．複数のテンプレートを例示しているところが交叉に，テンプレートの言い換えを指示している部分が突然変異に対応する．

さらに第 2 段落では，プレースホルダを生成するテンプレートを残すように LLM へ指示している．プレースホルダはテンプレートの例に含まれているが，プレースホルダのシンボル名を言い換えて良いかは LLM にとっては非自明であるため，明示的に禁止した．

一方で第 3 段落では，プロンプトのどこに何を書くか，というレイアウトの変更を許可している．本実験で示唆されたように，質問を選択肢の前に書くか，後に書くかはスコアに大きく影響する．そのため，プロンプトの意味が変わらない範囲での変更を明示的に許可している．

表 3 本実験で用いたメタプロンプト（プロンプト生成のためのプロンプト）

以下に大規模言語モデルを評価するためのプロンプトのテンプレートを例示します。意味は変えずに、これらのテンプレートを言い換えた一つのテンプレートを出力してください。 ‘{}’ で囲まれた文字列は Python の f-string 形式のテンプレートのシンボルです。シンボルは絶対に消したり名前を変えたりせず、全く同じものを出力に含めるようにしてください。 意味さえ変わらなければ、文の構造やシンボルの順番は入れ替えても構いません。#などの記号を変えるのも問題ありません。1 個のテンプレートを出力してください。テンプレート以外には何も出力しないでください。 テンプレート例: {template_0} テンプレート例: {template_1} テンプレート: