DEEP LEARNING JP
[DL Papers]

# LLMによるアルゴリズム生成に関わる論文

Automated Design of Agentic Systems
Discovering Preference Optimization Algorithms with and for Large Language Models

松尾研共同研究チーム　岡田　領

# Outline

- I. AIによるアルゴリズム生成について紹介
- II. 紹介する論文
  - III. Automated Design of Agentic Systems
  - IV. Discovering Preference Optimization Algorithms with and for Large Language Models

# AIによるアルゴリズム生成

## ３つの方向性

1. メタ学習アーキテクチャ
   - Neural Architecture Search
   - ニューラルネットワークの設計を自動化
2. 学習アルゴリズムのメタ化
   - MAML, Meta-RL
   - 優れたサンプル効率，汎化，複数タスクの継続学習のための「学習する学習」
3. 効果的な学習環境と学習データの生成
   - オープンエンドな方法で学習環境を生成

# 紹介する論文

LLMを使ったアルゴリズム生成に関わる論文として２本紹介

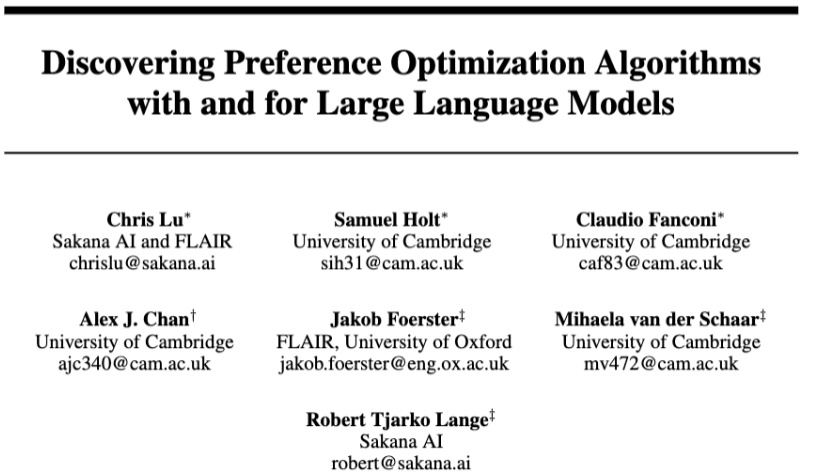
Automated Design of Agentic Systems, arXiv 2024/8

2024-8-19

**Automated Design of Agentic Systems**

Shengran Hu[1,2], Cong Lu[1,2] and Jeff Clune[1,2,3]
[1]University of British Columbia, [2]Vector Institute, [3]Canada CIFAR AI Chair

Discovering Preference Optimization Algorithms with and for Large Language Models, arXiv 2024/9



**Discovering Preference Optimization Algorithms with and for Large Language Models**

**Chris Lu***
Sakana AI and FLAIR
chrislu@sakana.ai

**Samuel Holt***
University of Cambridge
sih31@cam.ac.uk

**Claudio Fanconi***
University of Cambridge
caf83@cam.ac.uk

**Alex J. Chan**†
University of Cambridge
ajc340@cam.ac.uk

**Jakob Foerster**‡
FLAIR, University of Oxford
jakob.foerster@eng.ox.ac.uk

**Mihaela van der Schaar**‡
University of Cambridge
mv472@cam.ac.uk

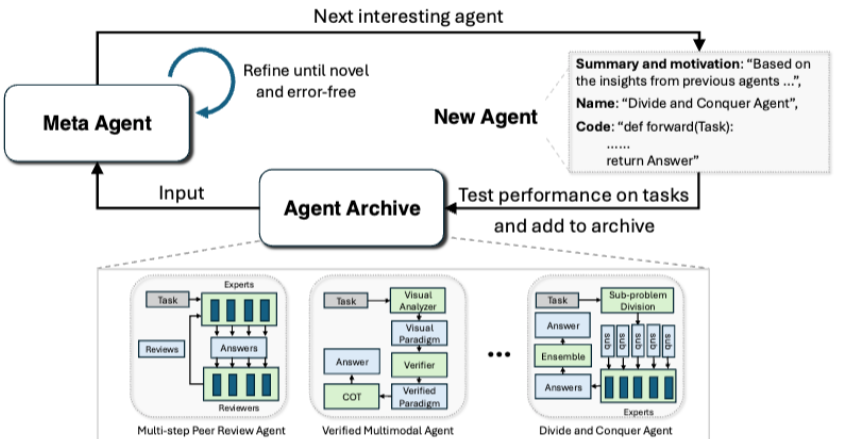
**Robert Tjarko Lange**‡
Sakana AI
robert@sakana.ai

- LLMを扱って生成する論文（前述の1と2の両方に該当）
- 生成する観点は異なる（左: エージェントシステム全体と右; 損失関数）

# Automated Design of Agentic Systems

背景・概要

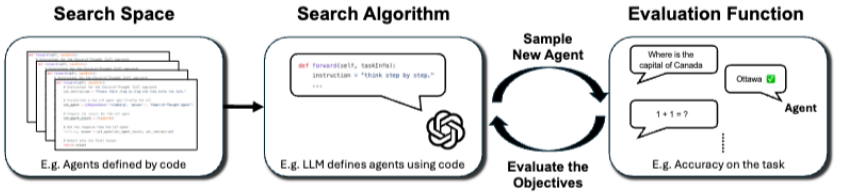
- LLMをエージェントシステムの構成要素として活用した汎用エージェントの開発が進んでいる
- 多くの場合手動設計であり専門知識と多大な労力が必要とされる.
- 本論文ではエージェントシステムの設計を自動化する新しい研究分野として、ADAS（Automated Design of Agentic Systems）を提唱
- エージェントシステム全体をコードで定義し、メタエージェントがより優れたエージェントを自動的にプログラミングするという、アプローチを提案



Examples of Discovered Agents

# Automated Design of Agentic Systems

手法の大枠




- メタエージェントによって新しいエージェントシステムのコードを生成
- コード空間内を探索し、プロンプト、ツール使用、制御フローなど、エージェントシステムのあらゆる要素を自動的に設計
- Meta Agent Search: 新規性や有用性を評価基準として、興味深い新しいエージェントを反復的に生成. 生成されたエージェントはアーカイブに保存され、メタエージェントは過去の発見に基づいて、より優れたエージェントを生成していく
  - 新規性と正しさに関するself-reflectionを２回, エラーが起きた場合最大３回更新する
  - 検証データを用いて評価（成功率やF1）

# Automated Design of Agentic Systems

メタエージェントのプロンプト

**System prompt for the meta agent.**

You are a helpful assistant. Make sure to return in a WELL-FORMED JSON object.

We use the following prompt for the meta agent to design the new agent based on the archive of previously discovered agents.

**Main prompt for the meta agent.**

You are an expert machine learning researcher testing various agentic systems. Your objective is to design building blocks such as prompts and control flows within these systems to solve complex tasks. Your aim is to design an optimal agent performing well on [Brief Description of the Domain].

[Framework Code]

[Output Instructions and Examples]

[Discovered Agent Archive] (initialized with baselines, updated at every iteration)

# Your task
You are deeply familiar with prompting techniques and the agent works from the literature. Your goal is to maximize the specified performance metrics by proposing interestingly new agents.
Observe the discovered agents carefully and think about what insights, lessons, or stepping stones can be learned from them.
Be creative when thinking about the next interesting agent to try. You are encouraged to draw inspiration from related agent papers or academic papers from other research areas.
Use the knowledge from the archive and inspiration from academic literature to propose the next interesting agentic system design.
THINK OUTSIDE THE BOX.

- ハイライトは変数
  - ドメインの説明: タスクの説明.
  - コード（framework）: クエリやプロンプトのフォーマットを記載した100行以内のコード（探索の効率化のため）
  - 出力指示と例: 出力のフォーマット指定
  - Agentの例: 用意しておいたベースラインで初期化され，イテレーションごとに生成したものを追加

# Automated Design of Agentic Systems

Framework Codeの一部

```python
# Named tuple for holding task information
Info = namedtuple('Info', ['name', 'author', 'content', 'iteration_idx'])

# Format instructions for FM response
FORMAT_INST = lambda request_keys: f"Reply EXACTLY with the following JSON format.\n{str(request_keys)}\nDO NOT MISS ANY FIELDS AND MAKE SURE THE JSON FORMAT IS CORRECT!\n"

# Description of the role of the FM Module
ROLE_DESC = lambda role: f"You are a {role}."

@backoff.on_exception(backoff.expo, openai.RateLimitError)
def get_json_response_from_gpt(msg, model, system_message, temperature):
    \"""
    Function to get JSON response from GPT model.

    Args:
    - msg (str): The user message.
    - model (str): The model to use.
    - system_message (str): The system message.
    - temperature (float): Sampling temperature.

    Returns:
    - dict: The JSON response.
    \"""
    ...
    return json_dict

class FM_Module:
    \"""
    Base class for an FM module.

    Attributes:
    - output_fields (list): Fields expected in the output.
    - name (str): Name of the FM module.
    - role (str): Role description for the FM module.
    - model (str): Model to be used.
    - temperature (float): Sampling temperature.
    - id (str): Unique identifier for the FM module instance.
    \"""

    def __init__(self, output_fields: list, name: str, role='helpful assistant', model='gpt-3.5-turbo-0125', temperature=0.5) -> None:
        ...

    def generate_prompt(self, input_infos, instruction) -> str:
        \"""
        Generates a prompt for the FM.

        Args:
        - input_infos (list): List of input information.
        - instruction (str): Instruction for the task.

        Returns:
        - tuple: System prompt and user prompt.

        An example of generated prompt:
```

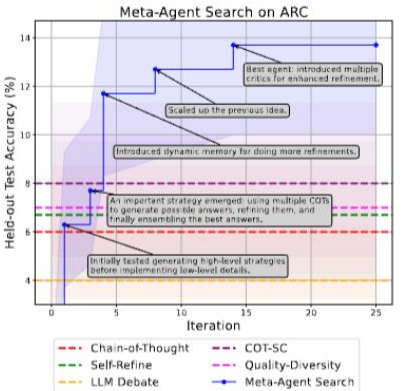
# Automated Design of Agentic Systems

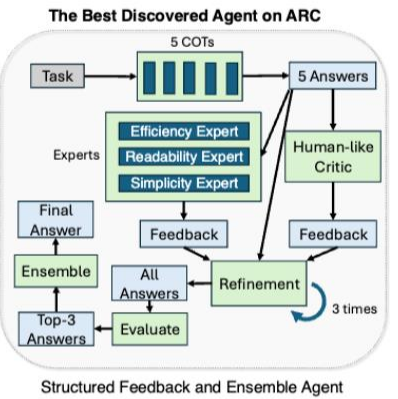
実験概要

- 評価タスクとデータセット
  - ARC (Abstraction and Reasoning Corpus): 推論能力を評価する論理パズルタスク。
  - DROP: 読解力を評価するデータセット。
  - MGSM: 多言語の数学問題解決能力を評価するベンチマーク。
  - MMLU: 多様な分野の知識を問う問題解決能力を評価するベンチマーク。
  - GPQA: 科学分野の大学院レベルの難問を問うデータセット。
- 評価指標: 各タスクにおいて、正確性やF1スコアなどの指標
- 結果概要: Meta Agent Searchによって発見されたエージェントは、既存の手設計のエージェントと比較して、すべてのタスクにおいて優れた性能を示した。特に、DROPやMGSMなどのタスクでは、大幅な性能向上。さらに、発見されたエージェントは、他のモデルやドメインにもうまく転移できることが示された。

# Automated Design of Agentic Systems

実験詳細(ARC)




- メタエージェントはGPT-4, 発見されたエージェントやベースラインの実行はGPT-3.5
- エージェントを５回評価し，それぞれ段階的に性能が向上（左）
- 発見されたベストエージェントのイメージ（右）

# Automated Design of Agentic Systems

発見されたエージェントのコードの一部

# Automated Design of Agentic Systems

実験結果



| Agent Name | F1 Score | Accuracy (%) | | |
|---|---|---|---|---|
| | Reading Comprehension | Math | Multi-task | Science |
| **State-of-the-art Hand-designed Agents** | | | | |
| Chain-of-Thought (Wei et al., 2022) | 64.2 ± 0.9 | 28.0 ± 3.1 | 65.4 ± 3.3 | 29.2 ± 3.1 |
| COT-SC (Wang et al., 2023b) | 64.4 ± 0.8 | 28.2 ± 3.1 | **65.9 ± 3.2** | 30.5 ± 3.2 |
| Self-Refine (Madaan et al., 2024) | 59.2 ± 0.9 | 27.5 ± 3.1 | 63.5 ± 3.4 | **31.6 ± 3.2** |
| LLM Debate (Du et al., 2023) | 60.6 ± 0.9 | **39.0 ± 3.4** | 65.6 ± 3.3 | 31.4 ± 3.2 |
| Step-back Abstraction (Zheng et al., 2023) | 60.4 ± 1.0 | 31.1 ± 3.2 | 65.1 ± 3.3 | 26.9 ± 3.0 |
| Quality-Diversity (Lu et al., 2024c) | 61.8 ± 0.9 | 23.8 ± 3.0 | 65.1 ± 3.3 | 30.2 ± 3.1 |
| Role Assignment (Xu et al., 2023) | **65.8 ± 0.9** | 30.1 ± 3.2 | 64.5 ± 3.3 | 31.1 ± 3.1 |
| **Automated Design of Agentic Systems on Different Domains** | | | | |
| Best Agents from Meta Agent Search | **79.4 ± 0.8** | **53.4 ± 3.5** | **69.6 ± 3.2** | **34.6 ± 3.2** |

- 全てのドメインにおいてベースラインと比較して優れたエージェントを発見

# Automated Design of Agentic Systems

実験結果

| Agent Name | Accuracy on ARC (%) | | | |
|---|---|---|---|---|
| | GPT-3.5 | Claude-Haiku | GPT-4 | Claude-Sonnet |
| **Manually Designed Agents** | | | | |
| Chain-of-Thought (Wei et al., 2022) | 6.0 ± 2.7 | 4.3 ± 2.2 | 17.7 ± 4.4 | 25.3 ± 5.0 |
| COT-SC (Wang et al., 2023b) | **8.0 ± 3.2** | 5.3 ± 2.5 | 19.7 ± 4.5 | 26.3 ± 4.9 |
| LLM Debate (Du et al., 2023) | 4.0 ± 2.2 | 1.7 ± 1.5 | 19.0 ± 4.5 | 24.7 ± 4.8 |
| Self-Refine (Madaan et al., 2024) | 6.7 ± 2.7 | **6.3 ± 2.8** | **23.0 ± 5.2** | **39.3 ± 5.5** |
| Quality-Diversity (Lu et al., 2024c) | 7.0 ± 2.9 | 3.3 ± 2.2 | **23.0 ± 4.7** | 31.7 ± 5.3 |
| **Top Agents Searched with GPT-3.5** | | **Transferred to Other FMs** | | |
| Structured Feedback and Ensemble Agent | **13.7 ± 3.9** | 5.0 ± 2.5 | 30.0 ± 5.2 | 38.7 ± 5.5 |
| Hierarchical Committee Reinforcement Agent | 13.3 ± 3.8 | 8.3 ± 3.2 | 32.3 ± 8.9 | 39.7 ± 5.5 |
| Dynamic Memory and Refinement Agent† | 12.7 ± 3.9 | **9.7 ± 3.3** | **37.0 ± 5.3** | **48.3 ± 5.7** |

- GPT3.5で発見されたエージェントは別のLLMでも高い性能
- （Claude-Sonnetが高い）

# Automated Design of Agentic Systems

実験結果

| Agent Name | Accuracy (%) | F1 Score | Accuracy (%) | |
|---|---|---|---|---|
| | Math | Reading Comprehension | Multi-task | Science |
| **Manually Designed Agents** | | | | |
| Chain-of-Thought (Wei et al., 2022) | 28.0 ± 3.1 | 64.2 ± 0.9 | 65.4 ± 3.3 | 29.2 ± 3.1 |
| COT-SC (Wang et al., 2023b) | 28.2 ± 3.1 | 64.4 ± 0.8 | **65.9 ± 3.2** | 30.5 ± 3.2 |
| Self-Refine (Madaan et al., 2024) | 27.5 ± 3.1 | 59.2 ± 0.9 | 63.5 ± 3.4 | **31.6 ± 3.2** |
| LLM Debate (Du et al., 2023) | **39.0 ± 3.4** | 60.6 ± 0.9 | 65.6 ± 3.3 | 31.4 ± 3.2 |
| Step-back Abstraction (Zheng et al., 2023) | 31.1 ± 3.2 | 60.4 ± 1.0 | 65.1 ± 3.3 | 26.9 ± 3.0 |
| Quality-Diversity (Lu et al., 2024c) | 23.8 ± 3.0 | 61.8 ± 0.9 | 65.1 ± 3.1 | 30.2 ± 3.1 |
| Role Assignment (Xu et al., 2023) | 30.1 ± 3.2 | **65.8 ± 0.9** | 64.5 ± 3.3 | 31.1 ± 3.1 |
| **Top Agents Searched on Math (MGSM)** | | **Transferred beyond Math Domains** | | |
| Dynamic Role-Playing Architecture | **53.4 ± 3.5** | 70.4 ± 0.9 | 62.4 ± 3.4 | 28.6 ± 3.1 |
| Structured Multimodal Feedback Loop | 50.2 ± 3.5 | 70.4 ± 0.9 | **67.0 ± 3.2** | 28.7 ± 3.1 |
| Interactive Multimodal Feedback Loop | 47.4 ± 3.5 | **71.9 ± 0.8** | 64.8 ± 3.3 | **29.9 ± 3.2** |

- 別ドメイン（数学タスクから別タスク）を渡った検証

# Automated Design of Agentic Systems

まとめとFuture work

- ADASではエージェントシステムの設計を自動化
- Meta Agent Searchによってコード空間を効率的に探索することで、人間が設計したものよりも優れた性能を持つエージェントを自動的に発見
- メタエージェント自身もエージェントであるため、ADASを自己参照的に適用し、メタエージェントの学習を自動化できる可能性
- LangChainのようなエージェントフレームワークや、RAGなどの既存の構成要素をMeta Agent Searchに取り入れることで、より効率的な探索が可能になる可能性
- 性能だけでなく、コスト、レイテンシ、堅牢性など、複数の目的を考慮したエージェントシステムの設計
- より高度な探索アルゴリズムを導入することで、より多様で革新的なエージェントシステムを発見できる可能性
- 実行ログの分析や主観的な評価によってより評価関数を高度化できる可能性
- 複数ドメインを考慮したエージェント探索

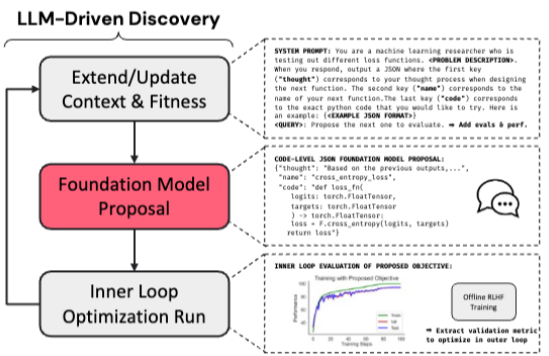
# Discovering Preference Optimization Algorithms with and for Large Language Models

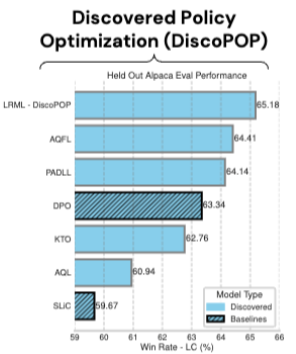
背景・概要

- Preference Optimization（選好最適化）は人間の選好をよりよく反映した出力となるよう、LLMを微調整するための重要な手法
- 通常、手動で作成された損失関数を用いた、オフライン教師あり学習タスクとして扱われる.
- これらの手法は人間の創意工夫に制約される
- LLMを用いた目的関数発見を行い、専門家の介入なしに、自動的に新しい最先端の選好最適化アルゴリズムを生成することを目指す

# Discovering Preference Optimization Algorithms with and for Large Language Models

手法




**Algorithm 1** LLM-Driven Objective Discovery

1: Initialize LLM with established loss functions and their performance *in context*.
2: **repeat** for each generation $i$
3: LLM proposes a new candidate objective function $f_i$
4: Run unit tests to check the validity of the candidate and resample if needed.
5: Evaluate the objective function using the performance metric $\eta$
6: Update the LLM context with the performance data
7: LLM refines generation strategy based on the feedback
8: **until** convergence criteria are met or maximum generations are reached

- LLM（GPT-4）で新しい選好最適化損失関数を提案させ、その性能を評価するプロセスを反復的に行う
- 提案された損失関数で訓練されたLLMを、マルチターン対話評価ベンチマークであるMT-Benchで評価し、そのスコアを使用
- LLMへのプロンプトに過去の提案と性能評価の結果を含めることで、LLMはより良い損失関数を生成するように誘導

# Discovering Preference Optimization Algorithms with and for Large Language Models

システムプロンプト（一部）

```
You are a machine learning researcher who is testing out different
    RLHF loss functions. When you respond, output a JSON where the
    first key ("thought") corresponds to your thought process when
    designing the next function. The second key ("name") corresponds
    to the name of your next function. Finally, the last key ("code")
    corresponds to the exact python code that you would like to try.
    Here is an example:

{
"thought": "Based on the previous outputs, I should try the direct
    preference optimization algorithm.",
"name": "dpo",
"code": "def sigmoid_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    losses = -F.logsigmoid(self.beta * logits)
    return losses"
}

You are deeply familiar with binary classification losses from the
    literature. Be creative and reference prior literature when
    possible.

You must use the exact function interface used above. Feel free to
    define extra hyperparameters within your function as constants. Do
     not make them attributes of self.

Note that ‘self.beta = 0.05‘.

RLHF loss functions train on a dataset of pairs of preferred and
```

# Discovering Preference Optimization Algorithms with and for Large Language Models

ユーザプロンプト（一部）

```
Here are some results we've obtained:

[
{
    "code": "
def logistic_log_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
```

16

```
def kto_pair_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    chosen_KL = (policy_chosen_logps - reference_chosen_logps).mean().
    clamp(min=0)
    rejected_KL = (policy_rejected_logps - reference_rejected_logps).
    mean().clamp(min=0)

    chosen_logratios = policy_chosen_logps - reference_chosen_logps
    rejected_logratios = policy_rejected_logps -
    reference_rejected_logps
    # As described in the KTO report, the KL term for chosen (rejected
    ) is estimated using the rejected (chosen) half.
```

17

```
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    losses = -F.logsigmoid(self.beta * logits)
    return losses
    ",
    "fitness": 7.8875
},
{
    "code": "
```

```
    losses = torch.cat(
        (
            1 - F.sigmoid(self.beta * (chosen_logratios - rejected_KL)
    ),
            1 - F.sigmoid(self.beta * (chosen_KL - rejected_logratios)
```

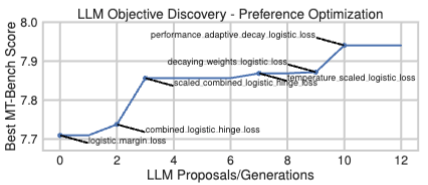
# Discovering Preference Optimization Algorithms with and for Large Language Models

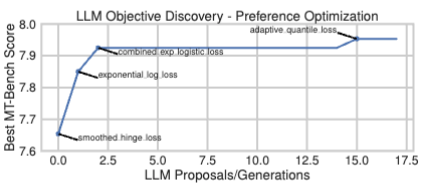
実験

- タスク
  - MT-Benchを用いたマルチターン対話
  - Alpaca Eval2.0を用いたシングルターン対話
  - Reddit TL;DRデータセットを用いた要約
  - IMDbデータセットを用いた肯定的な感情生成
- 結果の概要
  - 発見された目的関数のうち，一貫して高い性能を示したLog Ratio Modulated Lossを本論文で発見した損失関数DiscoPOPと呼ぶ.
  - DiscoPOPが、いずれのタスクにおいても既存の最先端手法と同等以上の性能を達成
  - DiscoPOPは単一ターン対話とテキスト要約タスクにおいて優れた性能
  - IMDbの感情生成タスクでは、DiscoPOPは特定のβ値において収束が困難になるなど、改善の余地も見られた

# Discovering Preference Optimization Algorithms with and for Large Language Models

実験結果詳細: MT-Benchでのマルチターン会話




| Name | Full Name | Objective $f$ Function | Score (/ 10) ↑ |
|---|---|---|---|
| DPO | Direct Preference Optimization | $\log(1+\exp(-\beta\rho))$ | 7.888 |
| DPO* | Official HuggingFace 'zephyr-7b-gemma' DPO model | $\log(1+\exp(-\beta\rho))$ | 7.810 |
| SLiC | Sequence Likelihood Calibration | $\text{ReLU}(1-\beta\rho)$ | 7.881 |
| KTO | Pairwise Kahneman-Tversky Optimization | see [Ethayarajh et al., 2024] | 7.603 |
| DBAQL | Dynamic Blended Adaptive Quantile Loss | $\sigma(\text{Var}[\beta\rho/\tau]) \cdot f_{dpo}(\beta\rho/0.9) + (1-\sigma(\text{Var}[\beta\rho/\tau])) \cdot f_{exp}(\beta\rho \cdot 0.9)$ | 7.978 |
| AQL | Adaptive Quantile Loss | $q \cdot f_{dpo}(\beta\rho) + (1-q) \cdot f_{stic}(\beta\rho)$ | 7.953 |
| PADLL | Performance Adaptive Decay Logistic Loss | $0.9 \cdot (1 - 0.5 \cdot \mathbb{1}[\rho < 0]) \cdot f_{dpo}(\beta\rho)$ | 7.941 |
| AQFL | Adaptive Quantile Feedback Loss | $r \cdot f_{dpo}(\beta\rho) + (1-r) \cdot f_{stic}(\beta\rho)$ | 7.931 |
| CELL | Combined Exponential + Logistic Loss | $0.5 \cdot f_{dpo}(\beta\rho) + 0.5 \cdot f_{exp}(\beta\rho)$ | 7.925 |
| LRML (**DiscoPOP**) | Log Ratio Modulated Loss | $(1-\sigma(\beta\rho/\tau)) \cdot f_{dpo}(\beta\rho) + \sigma(\beta\rho/\tau) \cdot f_{exp}(\beta\rho)$ | 7.916 |
| PFL | Policy Focused Loss | $1/2 \cdot f_{dpo}(\beta\rho) \cdot \mathbb{1}[\pi_w > \pi_r] + 2 \cdot f_{stic}(\beta\rho) \cdot \mathbb{1}[\pi_w \leq \pi_r]$ | 7.900 |

- LLMを用いた目的関数発見により生成された新しいオフライン選好最適化アルゴリズムの評価
- 70億パラメータのgemmaモデル「zephyr-7b-gemma-sft」を使用
  - gemmaのベースモデルを「deita-10k-v0-sft」データセットで教師ありファインチューニングしたもの
  - 「Argilla DPO Mix 7K」というペアワイズ選好データセットで学習
    - 新しい目的関数を評価するとき、この最後のステップでDPOを生成された目的関数に置き換える．ハイパーパラメータは同じものを使用
- 左図: LLMによる目的関数発見の例（１回目と２回目）.
- 約100の目的関数を評価（右図は一部）

# Discovering Preference Optimization Algorithms with and for Large Language Models

実験結果詳細: Alpaca Eval 2.0でのシングルターン会話

| Function | Win Rate (%) ↑ vs. GPT-4 | Win Rate - LC (%) ↑ vs. GPT-4 | Win Rate (%) ↑ vs. SFT Checkpoint | Win Rate - LC (%) ↑ vs. SFT Checkpoint |
|---|---|---|---|---|
| DPO | $11.23 \pm 0.97$ | $12.81 \pm 0.66$ | $78.72 \pm 1.26$ | $63.34 \pm 0.30$ |
| DPO* | $11.99 \pm 1.00$ | $14.73 \pm 0.71$ | $75.75 \pm 1.31$ | $59.88 \pm 0.41$ |
| SLiC | $10.67 \pm 0.94$ | $13; 16 \pm 0.69$ | $75.05 \pm 1.34$ | $59.67 \pm 0.42$ |
| KTO | $12.57 \pm 1.00$ | $13.58 \pm 0.67$ | $78.81 \pm 1.25$ | $62.76 \pm 0.31$ |
| DBAQL | $10.68 \pm 0.92$ | $11.41 \pm 0.57$ | $72.06 \pm 1.42$ | $54.40 \pm 0.38$ |
| AQL | $11.11 \pm 0.96$ | $13.63 \pm 0.68$ | $76.34 \pm 1.30$ | $60.94 \pm 0.36$ |
| PADLL | $\mathbf{14.07 \pm 1.04}$ | $\mathbf{14.89 \pm 0.66}$ | $\mathbf{81.10 \pm 1.21}$ | $64.14 \pm 0.28$ |
| AQFL | $\mathbf{13.63 \pm 1.05}$ | $\mathbf{15.55 \pm 0.71}$ | $\mathbf{79.32 \pm 1.23}$ | $64.41 \pm 0.34$ |
| CELL | $10.27 \pm 0.93$ | $12.26 \pm 0.61$ | $71.75 \pm 1.39$ | $57.48 \pm 0.34$ |
| LRML | $\mathbf{13.21 \pm 1.02}$ | $\mathbf{14.78 \pm 0.67}$ | $\mathbf{79.27 \pm 1.24}$ | $\mathbf{65.18 \pm 0.32}$ |
| PFL | $8.15 \pm 0.83$ | $10.67 \pm 0.57$ | $68.27 \pm 1.44$ | $56.14 \pm 0.43$ |

- LRML（DiscoPOP）が高い性能

# Discovering Preference Optimization Algorithms with and for Large Language Models

実験結果詳細: TL;DRデータセットを用いた要約

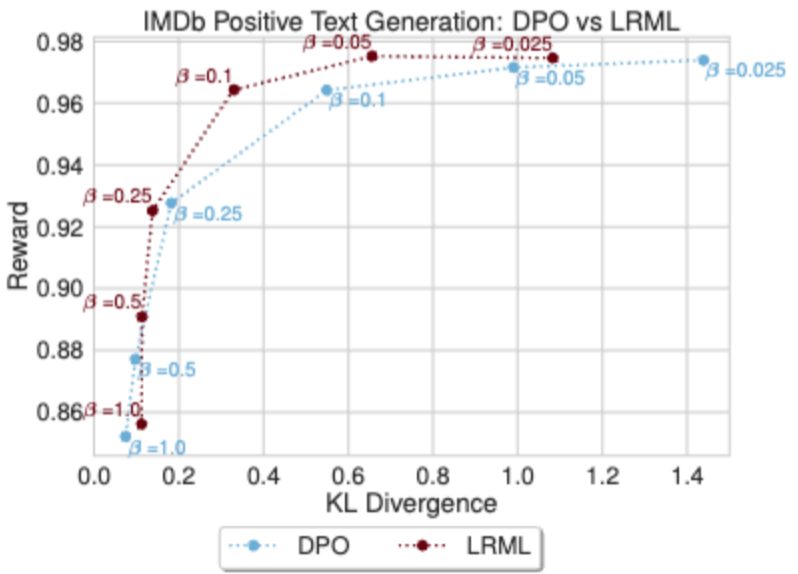
overlapping the standard error of the best score.

| Function | Win Rate (%) ↑ vs. Human Preference | Win Rate - LC (%) ↑ vs. Human Preference | Win Rate (%) ↑ vs. SFT Checkpoint | Win Rate - LC (%) ↑ vs. SFT Checkpoint |
|---|---|---|---|---|
| DPO | **88.27 ± 1.07** | **82.82 ± 0.00** | **54.38 ± 1.52** | 54.64 ± 0.00 |
| SLiC | 83.02 ± 1.29 | 63.41 ± 0.00 | 53.03 ± 1.52 | 54.11 ± 0.00 |
| KTO | 85.34 ± 1.18 | 80.26 ± 0.00 | 51.15 ± 1.54 | 50.0 ± 0.00 |
| DBAQL | 84.71 ± 1.21 | 78.68 ± 0.00 | 52.55 ± 1.52 | 55.14 ± 0.00 |
| AQL | 81.87 ± 1.32 | 68.89 ± 0.00 | 46.00 ± 1.54 | 50.0 ± 0.00 |
| PADLL | **88.54 ± 1.05** | 76.13 ± 0.00 | **55.34 ± 1.52** | **55.64 ± 0.00** |
| AQFL | 85.03 ± 1.22 | 76.23 ± 0.00 | 49.56 ± 1.53 | 50.38 ± 0.00 |
| CELL | 86.33 ± 1.14 | 73.72 ± 0.00 | 50.35 ± 1.52 | 51.90 ± 0.00 |
| LRML | **87.63 ± 1.10** | 81.88 ± 0.00 | 53.46 ± 1.52 | 55.10 ± 0.00 |
| PFL | 79.84 ± 1.35 | 69.23 ± 0.00 | 44.12 ± 1.52 | 44.57 ± 0.00 |

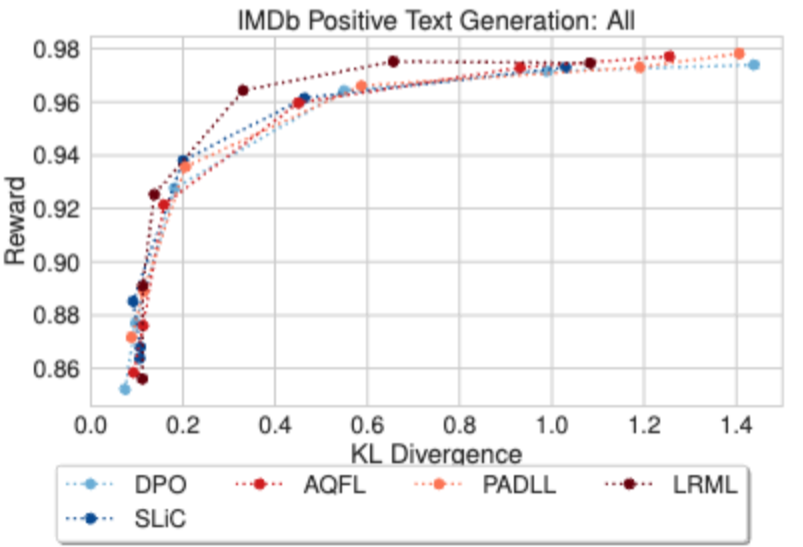
- PADLLとDPOが高い．（LRML - DiscoPOPではない）

# Discovering Preference Optimization Algorithms with and for Large Language Models

実験結果詳細: IMDbデータセットを用いた肯定的な感情生成



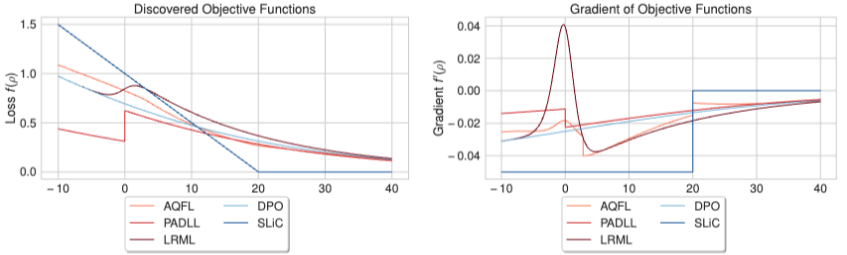

(a) DPO vs LRML




(b) Discovered vs Baseline Losses

# Discovering Preference Optimization Algorithms with and for Large Language Models

DiscoPOPの分析・限界




(a) Discovered Objective Functions (b) Gradients of the Discovered Objective Functions

$$f_{lrml}(\beta\rho) = (\sigma(\beta\rho/\tau) - 1) \cdot f_{dpo}(\beta\rho) + \sigma(\beta\rho/\tau) \cdot f_{exp}(\beta\rho) \quad (4)$$

$$= (1 - \sigma(\beta\rho/\tau)) \cdot \log(1 + \exp(-\beta\rho)) + \sigma(\beta\rho/\tau) \cdot \exp(-\beta\rho) \quad (5)$$

**E.6 LRML: Log Ratio Modulated Loss**

MT-Bench Score: 7.916

```
def log_ratio_modulated_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    # Modulate the mixing coefficient based on the log ratio
    magnitudes
    log_ratio_modulation = torch.sigmoid(logits)
    logistic_component = -F.logsigmoid(self.beta * logits)
    exp_component = torch.exp(-self.beta * logits)
    # Blend between logistic and exponential component based on log
    ratio modulation
    losses = logistic_component * (1 - log_ratio_modulation) +
```

- ロジスティック損失と指数損失の動的加重和である
- 温度パラメータτ=0.05の数式は(4)(5)

# Discovering Preference Optimization Algorithms with and for Large Language Models

まとめとFuture work

- LLMを用いた目的関数発見という新しい手法を提案し、Offline Preference Optimizationの損失関数の自動生成を実現
- 専門家の介入なしに、従来の手法よりも優れた性能を持つ新しい損失関数を発見できた
- DiscoPOPと名付けた損失関数は、ロジスティック損失と指数損失を動的に組み合わせることで、高い性能を実現
- LLMの提案プロセスをさらに改善するために、温度サンプリングや、より多くの訓練情報を利用
- DiscoPOPの安定性を向上させるために、β値の調整や、勾配クリッピングなどの手法を検討する必要

# まとめ

- LLMを使ってアルゴリズムを自動設計する２つの論文について紹介
  - Automated Design of Agentic Systems
  - Discovering Preference Optimization Algorithms with and for Large Language Models
- ADASはLLMを使ったエージェントシステム全体を対象とし，DiscoPOPの論文は選好最適化の損失関数の自動発見に焦点を当てている．
- どちらも手動設計による非効率に対する探索の効率化が主なモチベーション
- ADASはエージェントシステム全体を構築する時，DiscoPOPは選好最適化によってLLMの性能を上げたいときなのでモチベーションは異なる．
- プロセスとしてはLLMでアルゴリズム生成→評価で選抜のイテレーションを繰り返すので類似点も多い