

# AI とマインクラフトの世界 と、昔の話をしよう



# AI とマイクラフトの世界と、昔の話をしよう

## Agenda

- Part 1** DeepMind DreamerV3とは何か
- Part 2** 昔の話をしよう
- Part 3** DreamerV3でのAI技術の飛躍
- Part 4** DeepMindの研究動向とDreamerV3

# Part 1

DeepMind DreamerV3とは何か

マルレク 2025/07/05

「AIとマイクラの世界と、昔の話をしよう」

# Part 1 DeepMind DreamerV3とは何か

## Agenda

- はじめに
- DeepMind DreamerV3とは何か
- DeepMind DreamerV3の何が画期的なのか

# AI とマインクラフトの世界



はじめに

# AI とマイクラフトの世界

6月のマルレクは、「AIとマイクラフトの世界」というテーマで開催しようと思います。

AIの利用者は、この一年で大きく拡大しました。

IT系の方は、コード生成にAIを普通に使うようになりまし、ジブリ画像の生成はちょっとしたブームになりました。学生のレポートでもSNSへの投稿でも、AIの利用は日常の風景の一部になりつつあります。僕も、最近AIによる「音声概要」を愛用しています。

# AIとゲーム？

マイクラフトは、多くの人をご存知だと思いますが、有名なゲームです。僕もアカウントを持っています。現在の開発元は、Microsoft社です。

「Minecraftは史上最も売れたゲームであり、2025年時点で3億5000万本以上を販売し、月間アクティブプレイヤーは約1億7000万人に達している。(wikipediaより)

問題は、急発展を続けるAIの世界とゲームの世界とに、どのようなつながりがあるかということです。

# ゲームをするAI

今回のセミナーで紹介するAIは、役に立つプログラムを教えてくれるわけでも、面白い画像を作ってくれるわけでも、レポートを書いてくれるわけでもありません。

DeepMind社のAI DreamerV3は、ひたすらゲームをするAIなのです。

次のDreamerV3のGitHubのトップページを見てください。  
DreamerV3は、Minecraftに限らず、数多くのゲームをプレイすることのできるAIなのです。



# DreamerV3 Minecraftで ダイヤモンド発掘に挑戦し成功する

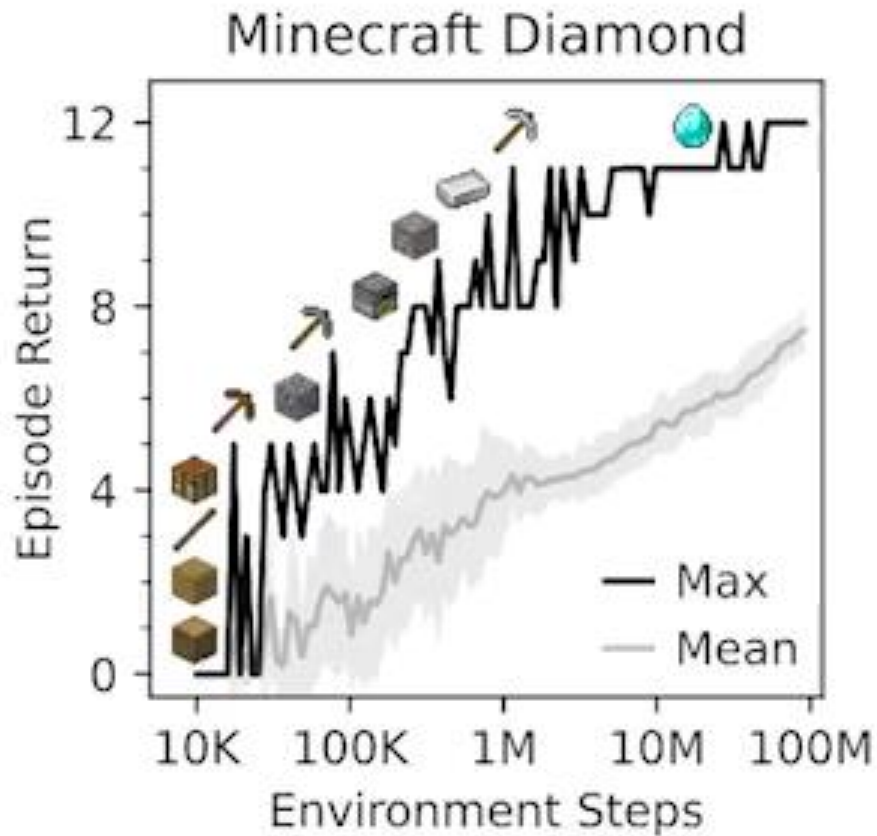
DreamerV3のゲームでの強さをよく示すのが、Minecraftで難しい課題の一つである、「ダイヤモンドの発掘」に成功したことです。

ダイヤモンドは、深い地層に生成される鉱石ブロックを壊すか、村などの自動生成された建造物のチェストの中身から入手することができます。

地中で手に入れる場合、ネザライト・ダイヤモンド・鉄のいずれかのツルハシを使って水色斑点模様の石を壊すと手に入りますが、石や木のツルハシを使って壊すと、アイテムがドロップせずにブロックが壊れてしまうので注意してください。

地表に近い浅い部分をいくら探しても見つかることはなく、マグマのある深い場所を探索したり掘り進む必要があることから、危険も伴う作業になります。

# ダイヤモンドを発見！



他の場所から始めても、発見できる

<https://danijar.com/project/dreamerv3/>

# AI Agentの自律性

## 何に注目しているのか 1

本当は、Minecraftゲームの紹介をもう少し詳しくしないと、DreamerV3の達成の意味が分かりにくいのですが、それだけだとつまらないですね。

ここでは、もう少し一般的なコンテキストから、DreamerV3の行ったことの意味を考えてみようと思います。

現在のAIの進化をAIのAgent化、大雑把に言って、AIの「自律性の拡大」として捉えることは、正しいと思います。

ただ、AIの自律性が何を意味するかについては、よく考える必要があります。

前回のセミナーで見た、ソフトウェア開発サイクルの変化の中でのAI Agentの役割は、AIがLLMモデルの成功によって獲得した人間のことばの意味を理解する能力に基づいています。

そこで見たAI Agentの中核には、全てLLMが存在していました。そこでは、人間の自然言語による荒い指示の残余を、AIがその意味理解能力によって埋めることが「AIの自律性」と考えられているように思います。

今回取り上げるDreamerV3は、まぎれもなく自律して行動するAI Agentですが、ただ、そこには、LLMは存在しません。それはLLM登場以前の「強化学習モデル」に基づいています。

それは、Agentの自律性という概念が、その言語能力とは独立したよりプリミティブなものであることを意味していると思います。

# AIのMulti Modal化

## 何に注目しているのか 2

言語能力以前から存在しうるAgentの自律性？  
知性の無いAI Agentに自律的な行動ができるのか？  
と考えるかもしれませんが。

でも、いったんAIの世界を離れて見れば、人間のような言語能力を持たないバクテリアや昆虫が、自律的に行動しているのは明らかだと思います。

抽象的なレベルでAgentの自律性を構成しているものを考えてみましょう。

第一に、自律性の前提として、Agentとその外部(「環境」と呼ぶことにしましょう)とが、へだてられていることが必要です。

第二に、Agentは外部から完全に切り離されて孤立しているわけではなく、その環境と相互に双方向に、次のような関係を持ちます。

- 環境 → Agent:

Agentは、外部の環境から「情報」を受け取ることができます。

- Agent → 環境:

Agentは、環境に対して「反応」して、環境と環境との関係を変えることができます。

「自律的なAgent」の成立には、Agentが環境からの多様な情報を受け取る能力の拡大が必要なのです。この能力は、AIの世界では、「AIのMulti Modal化」と呼ばれています。

現在、熱い注目を集めている「AI のAgent化」は、は発生史的には、この 2023年9月の「AIのMulti Modal化」を起源にしていると僕は考えています。

このことは、今回紹介するDeepMindのDreamerV3を、こうした「自律的AI Agent」の発展のもう一つのあらわれとして捉える上で、注目すべきことです。

# マルチモーダルなAIのモデル Agent Base Model

ChatGPT can  
now see, hear,  
and speak

AIの感覚  
能力の拡大



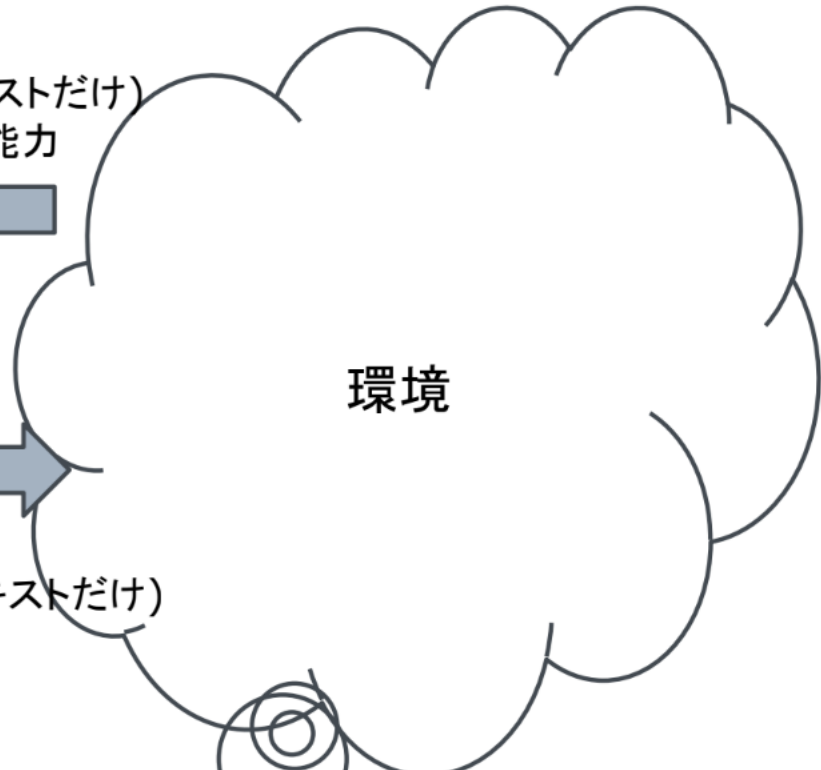
(今はテキストだけ)  
AIの感覚能力



AIの反応  
(今はテキストだけ)



環境



2023年9月の段階での展望

# 人間の力

## 何に注目しているのか 3

僕がDeepMind DreamerV3に注目する理由を最後に述べてみたいと思います。

ゲームをするAIは、珍しいものではありません。

特に、同じDeepMindが開発したAlpha GO は、画期的なものでした。画期的すぎて、人間はプロを含めて誰もAlpha GOに勝てなくなって、DeepMindはそれ以上の開発を中止しました。

あの衝撃的な事件以来、多くの人がゲームでは人間はAIに勝てないと心に刷り込まれたかもしれません。

でも、違うんです。

DreamerV3は、確かに素晴らしい！

ただ、彼は、高価なGPUを数十台並べ、おそらく膨大な電力を消費して、Minecraftで一個のダイヤモンドを発見すると探索をやめてしまいゲームを終了します。

人間は、Minecraftで小学生だってダイヤモンドを採掘できます。でも、ダイヤを見つけたからといって、Minecraft のゲームをやめる小学生はいないと思います。ダイヤモンド製のツルハシを作って、新しい冒険に向かうはずです。

Alpha GOとは違って、DreamerV3は、コンピュータには解けないけど人間が楽しめるゲームがあることを示しているのだと思います。

こんな論文があります。

arXiv > cs > arXiv:1203.1895

Search...

Help | Adv

Computer Science > Computational Complexity

[Submitted on 8 Mar 2012 (v1), last revised 8 Feb 2015 (this version, v3)]

## Classic Nintendo Games are (Computationally) Hard

Greg Aloupis, Erik D. Demaine, Alan Guo, Giovanni Viglietta

We prove NP-hardness results for five of Nintendo's largest video game franchises: Mario, Donkey Kong, Legend of Zelda, Metroid, and Pokemon. Our results apply to generalized versions of Super Mario Bros. 1-3, The Lost Levels, and Super Mario World; Donkey Kong Country 1-3; all Legend of Zelda games; all Metroid games; and all Pokemon role-playing games. In addition, we prove PSPACE-completeness of the Donkey Kong Country games and several Legend of Zelda games.

「スーパー・マリオ」も「ドンキーコング」も「ゼルダの伝説」「ポケモン」も、計算複雑性理論的には NP-Hardだと言っています。

# DeepMindのAI開発のビジョン

今回のセミナーでは、後半でDeepMindのAI開発のビジョンを紹介したいと思います。

具体的には、次の三つのプロジェクトを紹介します。

- DreamerV3
- Alpha Geometry
- Alpha Evolve

このパートの音声による概説は、[次のURLからアクセス](https://www.marulabo.net/wp-content/uploads/2025/05/DeepMind-%E3%80%8C%E3%83%9E%E3%82%A4%E3%83%B3%E3%82%AF%E3%83%A9%E3%83%95%E3%83%88%E3%81%AE%E4%B8%96%E7%95%8C%E3%80%8D.mp3)できます。お聞きください。

<https://www.marulabo.net/wp-content/uploads/2025/05/DeepMind-%E3%80%8C%E3%83%9E%E3%82%A4%E3%83%B3%E3%82%AF%E3%83%A9%E3%83%95%E3%83%88%E3%81%AE%E4%B8%96%E7%95%8C%E3%80%8D.mp3>



# DeepMind DreamerV3 の何が画期的なのか

# DreamerV3の何が画期的なのか

DreamerV3は、昨今話題のAI Agentの一種です。ただ、このAI Agentは、皆さんの身近にいる「コードを書いてくれる」AI Agentとは、一味違ってきます。

前回のセッションでは、DreamerV3を「ゲームをするAI Agent」として紹介してきました。それだけでは彼が可哀想です。

今回のセッションでは、なぜ、この「ゲームをするAI Agent」が「コードを書いてくれるAI Agent」以上に画期的なのかを考えてみたいと思います。

ひたすらゲームをするだけのAIが、コードを書いてくれるAIより「画期的」だと思ふには、視点の切り替えが必要かもしれません。

それは、簡単なことです。我々がAIに何を期待しているのかを改めて考えること、一言で言えば、AIを作る人の視点を持つことです。そのためには、AIを作るとか何かのAIのプロジェクトに参加する必要はありません。そうした視点を「想像」するだけで十分です。

技術が関わる世界には、その技術を使う人とその技術を作る人がいます。IT技術者は、これまでは、技術を作る側の一端に立っていました。ただ、AI技術に関しては、IT技術者は最も熱心なAI技術の利用者になろうとしています。ひたすらAIを使うだけだと、IT技術者にとっては、あまりいいことないと僕は思います。

実は、「想像」は、夢見るDreamerV3のキーワードの一つです。

# 世界のモデルと未来を想像する力

DreamerV3の基本的な論文のタイトルは、“Mastering diverse control tasks through world models”です。  
「多様な制御タスクを世界モデルを通じてマスターする」

世界モデルってなんでしょう？

この論文に、とても印象的な一節があります。

"The algorithm is based on the idea of learning a world model that equips the agent with rich perception and the ability to imagine the future"

「このアルゴリズムは、エージェントに豊かな知覚と未来を想像する能力を付与する世界のモデルを学習するというアイデアに基づいている」

皆さんの身近で働いているAI Agentは、「世界」のことを知る（たとえば、それがゲームの画面上のバーチャルな世界だとしても）「豊かな知覚」を持っていますか？

あなたのAI Agentは、「知覚」を持ちません。それは、RAGで拡張されたLLMが管理する文字情報の世界で生きています。

皆さんのAI Agentは、「未来を想像する」能力を持っていますか？多分、ないと思います。もしもあったとしても、それは「ハルシネーション」と呼ばれるでしょう。

幸いなことに、現時点では、我々がどのようなAIを構想するかによって、AIはその未来の姿を変えます。DreamerV3が、「画期的」なのは、まず、そのアイデアにあると、僕は考えています。

# AIの中に「世界のモデル」を作る

あらためて、そのアイデアを見ておきましょう。

「このアルゴリズムは、エージェントに豊かな知覚と未来を想像する能力を付与する世界のモデルを学習するというアイデアに基づいている」

重要なのは、AIの内部に「世界のモデル」 -- これをworld modelと呼んでいます -- を作るというアイデアです。

minecraftの場合には、ゲームのピクセル画面を取り込んで、autoencoderで圧縮して、その圧縮した画像情報を自分の内部に時系列順に記憶していきます。

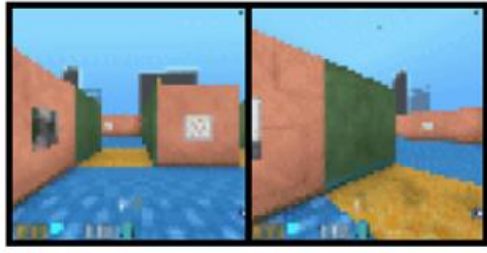
# DreamerV3が「想像」する「未来」

単に「世界」の画像情報を取り込むだけだったら、それは、ビデオデッキや監視カメラのメモリーと変わりありません。

DreamerV3は、world model をRNNのニューラル・ネットワークとして構築します。このworld model ニューラル・ネットワークは、例えば、5フレームの画像データを受け取ると、その5フレームの情報のみを元にして45フレームの画像データを生成します。

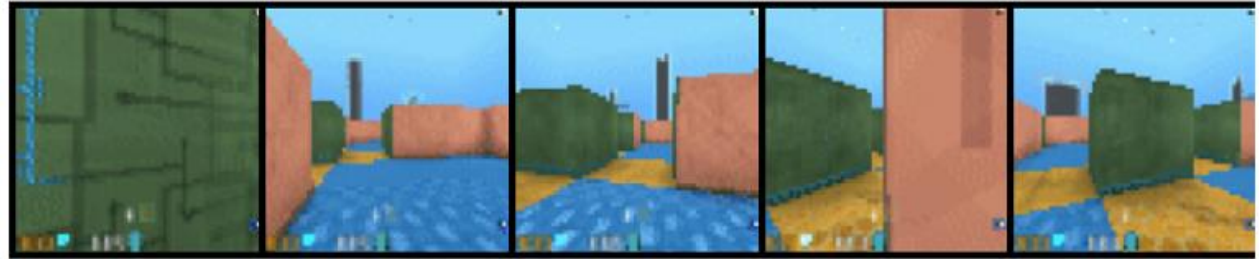
このworld modelが5フレームの現実の画像データから生成した45フレームの画像データが、DreamerV3が「想像」した「未来」の世界なのです。

## 現実の世界



$T = 0$

5



10

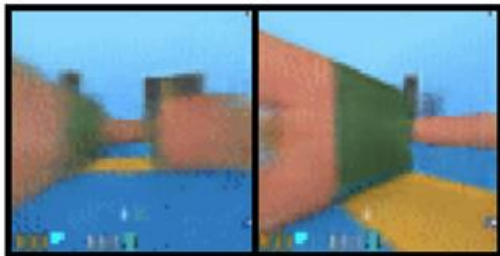
15

20

25

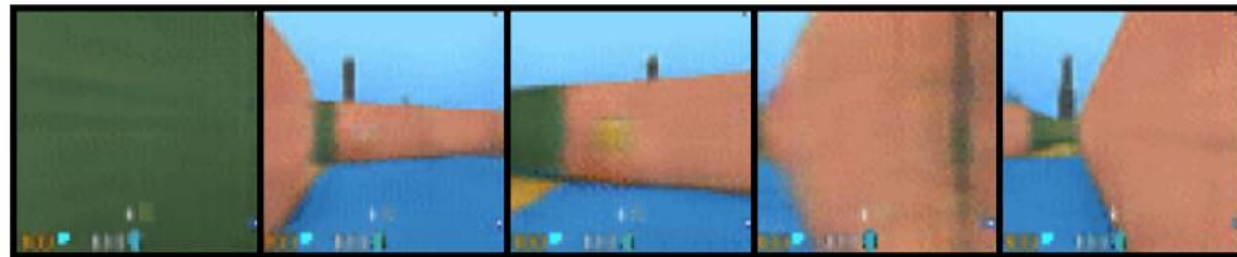
30

## World Model



$T = 0$

5



10

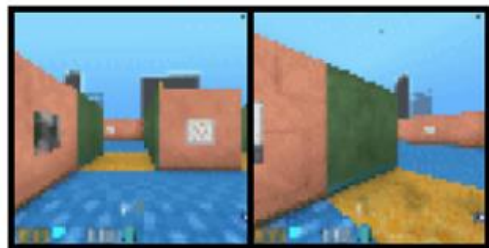
15

20

25

30

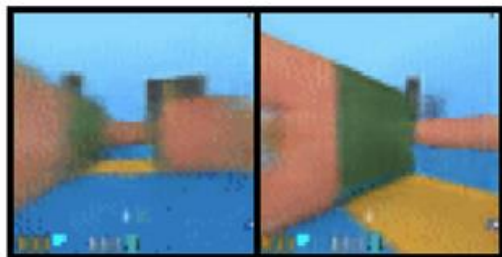
## 現実の世界



$T=0$

5

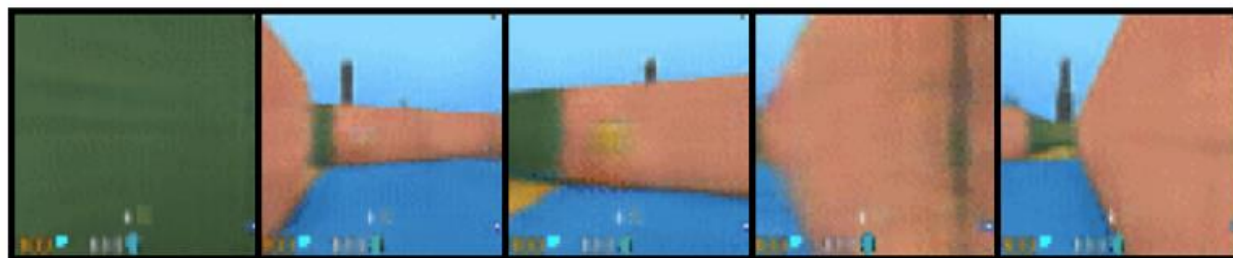
## World Model



$T=0$

5

World Modelがフレームを生成する



10

15

20

25

30

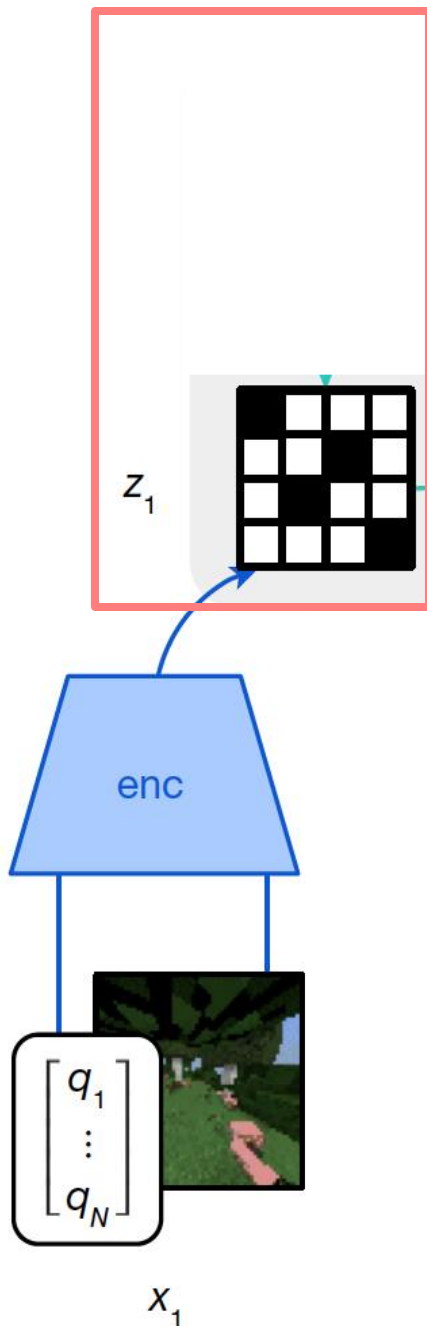
World Modelが想像した未来

world modelは、エンコーダ(enc)を使って、N次元の視覚からの入力  $x_t$  を圧縮されたデジタル表現  $z_t$  にエンコードします。

その  $z_t$  から world model の RNN の内部状態  $h_t$  が生成されます。

この  $h_t$  には、world model が「想像した未来」の情報が含まれています。

world model の本体は、この  $z_t$  と  $h_t$  のペアと考えると問題ありません。



こうした world model は、すぐ先に見た表現やDreamerV3論文 Fig. 3の次のような表現(赤枠の部分)とは違ったものに見えるかもしれません。

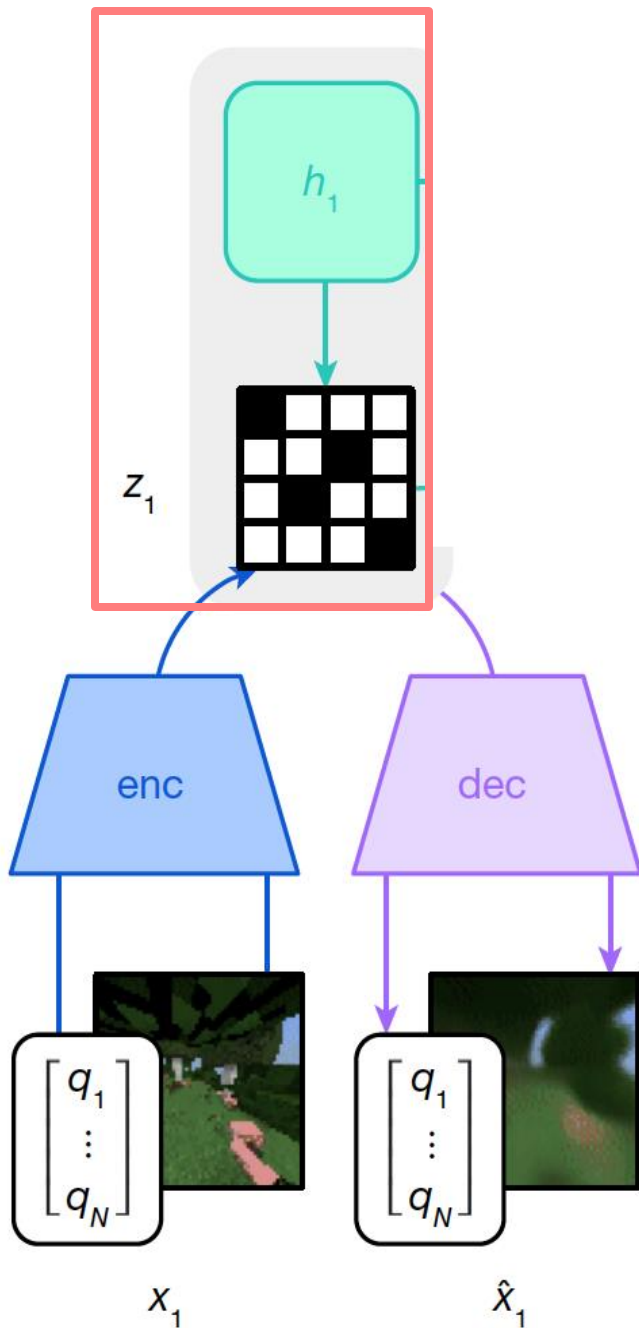
それには理由があります。



world modelは、エンコーダ(enc)を使って、N次元の視覚からの入力  $x_t$  を圧縮されたデジタル表現  $z_t$  にエンコードします。

圧縮されたデジタル表現  $z_t$  は、デコーダ(dec)を使用して N次元の画像データ  $\hat{x}_t$  として再構築できます。

先のようなworld model の表現として利用されているのは、このように形成された画像データ  $\hat{x}_t$  です。



# world modelとaction

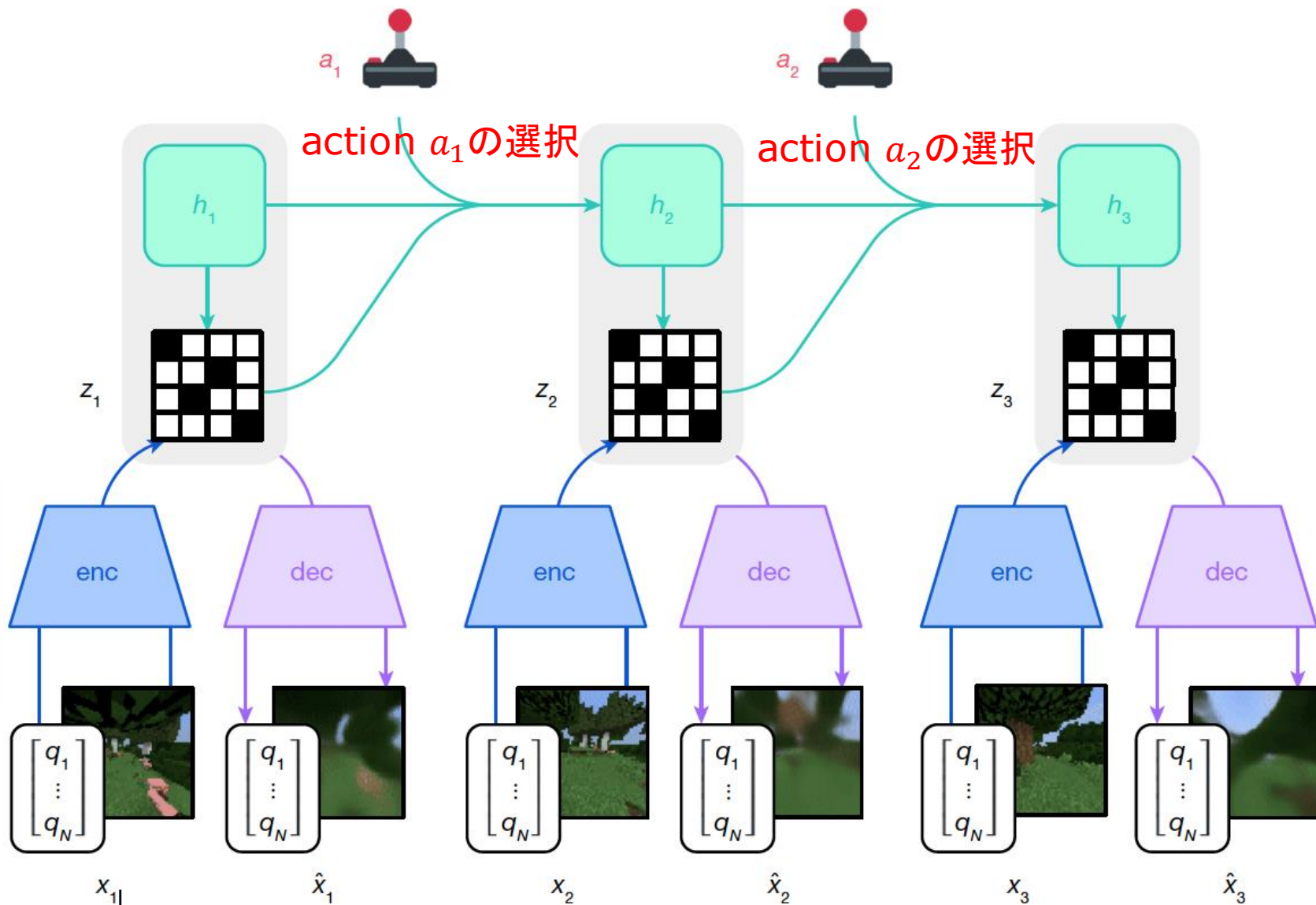
DreamerV3のworld model で興味深いのは、world model の「未来への想像」を含んだRNNの状態をベースに、一つの行動(action)を選択することです。

次の図のように。

actionの実行によって、世界は変わります。あるいは、Agent であるDreamerV3と世界の間接関係は変わります。

この変化をDreamerV3は視覚情報の変化として感知しworld model の状態を変化させます。もちろん、この時点での新しい「未来への想像」を行います。

# world modelでのactionの選択と状態の遷移



# DreamerV3は、どのように行動を選択するのか？

それでは、DreamerV3はどのように行動を選択するのでしょうか？

DreamerV3 world model の中核であるRNNニューラルネットワークに加えて、次の二つのネットワークを備えています。

- **Critic ニューラルネットワーク:**

想像された未来で受け取ることができる報酬の価値を $v_t$ を評価する。

- **Actor ニューラルネットワーク:**

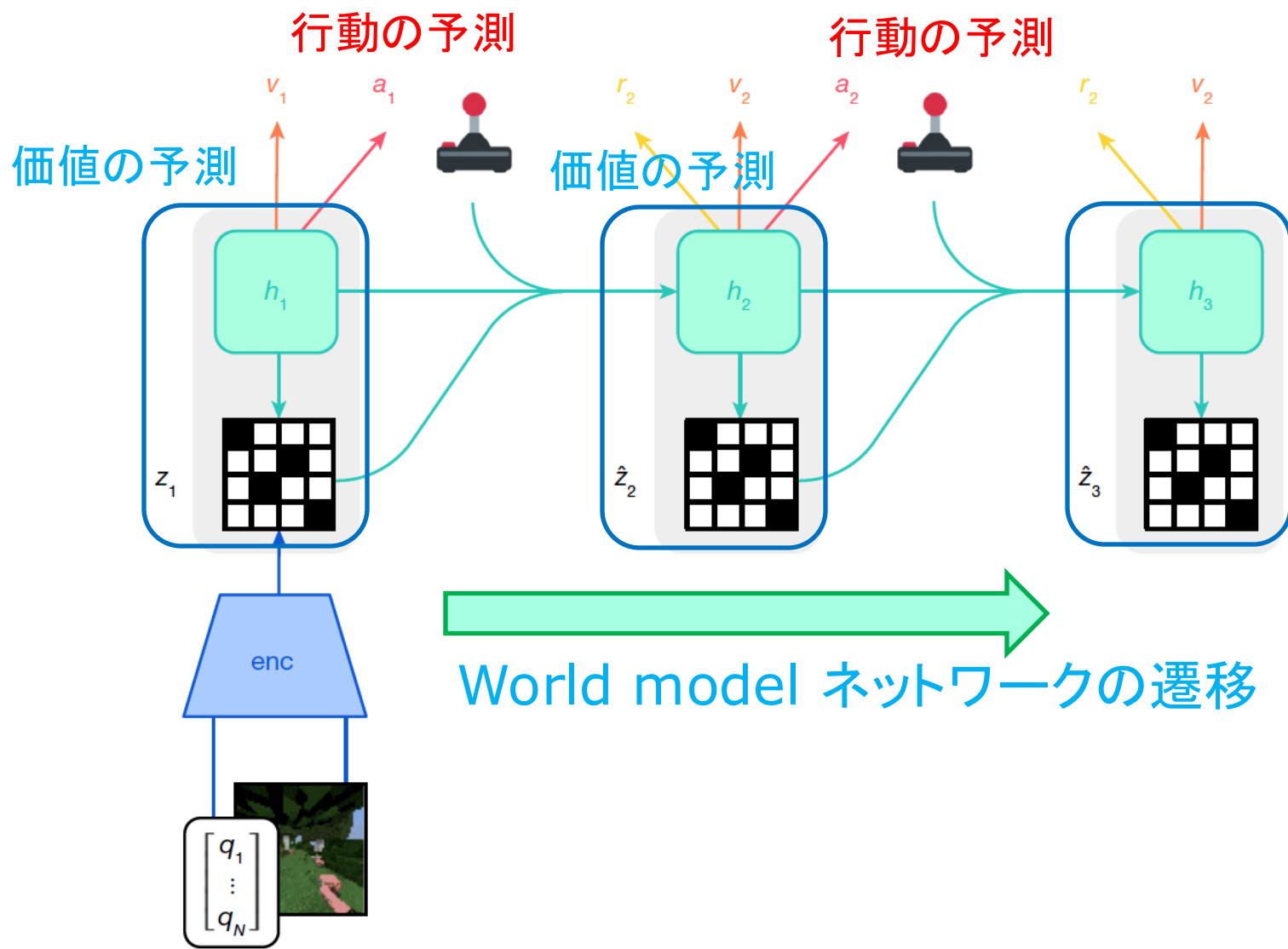
最も価値の高い結果に到達するための行動 $a_t$ を選択する。

# Actor-Criticの働き

ActorとCriticニューラル・ネットワークは、world modelによって予測される表現  $z_t$  と報酬  $r_t$  の軌跡を学習し、そこから行動  $a_t$  と価値  $v_t$  を予測します。

次の図のように。

# Actor と Critic ネットワークの働き



# DreamerV3を構成する 三つのニューラルネットワーク

DreamerV3は3つのニューラルネットワークで構成されています。

world modelは潜在的な行動の結果を予測し、criticは各結果の価値を判断し、actorは最も価値のある結果に到達するための行動を選択します。

これらのコンポーネントは、エージェントが環境と相互作用する間に、再生された経験から同時に訓練されることとなります。

## 次回のセッション

今回のセッションでは、DreamerV3の特徴である World model の概略を見てきました。

次回は、AI技術の発展の中で、「世界のモデル」がどのように扱われてきたかを振り返ってみようと思います。

ウィノグラードの「積み木の世界」と、ロドニー・ブルックスの「世界の内部モデル不要論」を紹介します。





# Part 2

昔の話をしよう

マルレク 2025/07/05

「AIとマイクラフトの世界と、昔の話をしよう」

# Part 2 昔の話をしよう

## Agenda

- AIと「世界モデル」(1)  
Winograd – SHRDLU
- AIと「世界モデル」(2)  
Brooks – Intelligence without representation



# AIと「世界モデル」(1)

昔の話をしよう

Winograd – SHRDLU

# AIと「世界モデル」

先のセッションで、DreamerV3の「世界モデル」の概略を見てきました。今回のセッションは、もちろん、DreamerV3の「世界モデル」にインスパイアされたものですが、その画期性へのアプローチを変えてみようと思います。

今回のセッションでは、AIがその発展の歴史の中で、どのように「世界」に関わろうとしたのかを、その「世界モデル」論にフォーカスして振り返ってみようと思います。

# 昔の話をしよう

今回と次回のセッションで、次の二つのトピックを取り上げます。

- Winograd – SHRDLU
- Brooks – Intelligence without representation

前者は 1970年、後者は 1987年の論文の話ですので、50年から 30年近く昔の話です。

老人の昔話は、一般的には、あまり役には立ちません。でも、今日のような激しい時代の変わり目には、少しは、役立つだろうと思っています。今回と次回のセッションのテーマは、「昔の話をしよう」です。

A computer system  
for understanding language  
specifically English

terry  
winograd  
under-  
standing  
natural  
language  
edinburgh  
university  
press

It answers questions, executes  
commands, and accepts information  
in an instructive English dialogue.

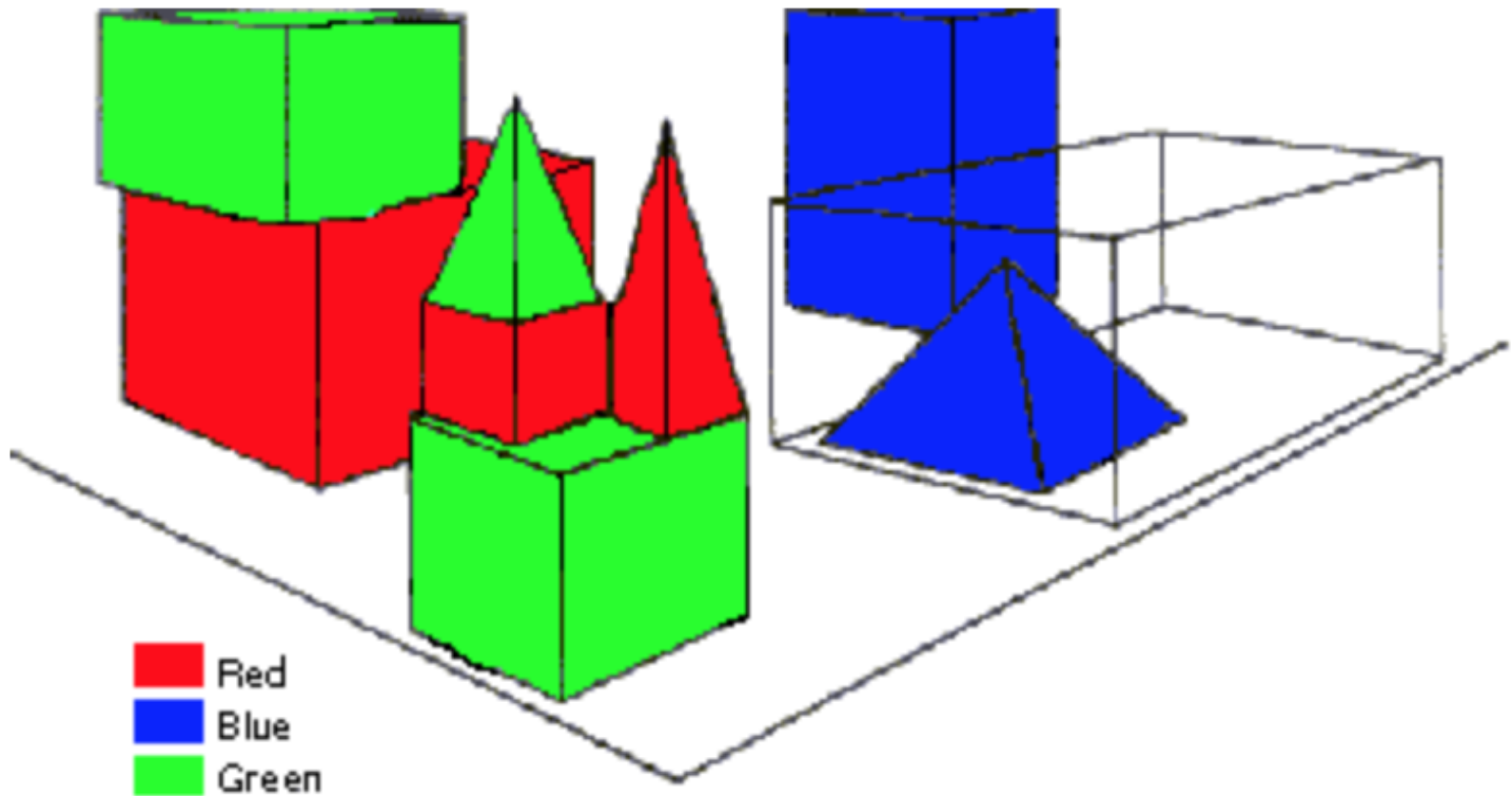
In modelling language understanding, integration of all the  
aspects of language—syntax, semantics, and inference—is  
necessary. This system contains a parser, a recognition  
grammar of English, programs for semantic analysis, and  
a general problem-solver. A computer cannot deal reasonably  
with language unless it understands the subject matter.  
This program is given a detailed model of a particular domain.  
It has a simple model of its own mentality, and can  
remember and discuss its plans and actions as well as  
carrying them out. It enters into a dialogue, responding to  
English sentences with actions and English replies,  
seeking clarification when its heuristic programs cannot  
understand a sentence. Edinburgh University Press £4

# Terry Winograd



## SHRDLU

# ウィノグラードのSHRDLU 「積み木の世界」



<https://science.slc.edu/~jmarshall/courses/2005/fall/cs151/lectures/intro/SHRDLU.htm>

# SHRDLUのアーキテクチャと能力

SHRDLUは、1968年から1970年にかけてMITのテリー・ウィノグランドによって開発された、自然言語理解のための統合システムです。このシステムは、自然言語パーサー、意味解釈器、推論器、そして「積木の世界(Blocks World)」を操作するためのエフェクターから構成されています。

「積木の世界」は、色付きのブロックやピラミッドなどで構成される単純な仮想環境であり、SHRDLUはユーザーの命令に基づいてこれらのオブジェクトを操作します。

SHRDLUの最も注目すべき能力は、積木の世界に関連する英語の命令や質問を理解し、応答することです。

大きな赤いブロックを取って。

OK。

(画面上でロボットアームが動き出す。図のように、小さな赤いブロックと大きな赤いブロックの2つが見える。大きなブロックの上には緑色の立方体が積み重ねられている。ロボットはまず緑色の立方体をテーブルの上に移動させ、次に赤いブロックを拾う。)

あなたが持っているものよりも背の高いブロックを見つけて、箱に入れて。

「それ」とは、私が持っているものよりも背の高いブロックのことだね。

(SHRDLU は赤いブロックを置き、より背の高い青いブロックを拾い、箱に入れる。)

箱の中には何が入っている？

青いピラミッドと青いブロック。

ピラミッドは何で支えられている？

箱。

(実際にはテーブルの上に赤いピラミッドと緑のピラミッドの2つがあります。SHRDLUは質問が先ほど言及されたピラミッドを指すものと正しく推測しています。)

箱に入っていないブロックはいくつありますか？

4つです。

そのうちの少なくとも1つは、私が捨てるように言ったものより細いですか？

はい、赤い立方体です。

支えられていますか？

はい、緑のブロックによって支えられています。

テーブルはブロックを持ち上げられるか？

いいえ。

ピラミッドはブロックで支えられるか？

はい。

ピラミッドはピラミッドを支えられるか？

わからない。

2つのピラミッドを積み重ねて。

(ロボットは試すが失敗する。)

できない。

# SHRDLUのアーキテクチャと能力

SHRDLUの推論は、オブジェクト、プロパティ、関係の明示的な記号表現に基づいて行われました。また、この世界には基本的な物理法則も組み込まれていて、ブロックは積み重ね可能だが、ピラミッドの上にピラミッドを重ねることはできないといった可能性を推論することができました。

先の対話例では、「PICK UP A BIG RED BLOCK. (大きな赤いブロックを持ち上げなさい)」といった直接的な命令から、「FIND A BLOCK WHICH IS TALLER THAN THE ONE YOU ARE HOLDING AND PUT IT INTO THE BOX. (あなたが持っているブロックよりも背の高いブロックを見つけて、それを箱に入れなさい)」といったより複雑な指示まで、SHRDLUが的確に応答することができました。

# AIエージェントとしてのSHRDLU

SHRDLUは、現代的な意味での学習能力は限定的だったものの、初期の統合AIエージェントの多くの側面を体現していました。

- **知覚**: 入力された自然言語を感覚入力として解釈した。
- **知識表現**: 積木の世界の状態に関する動的な知識ベースを維持した。
- **意思決定/計画**: 命令で述べられた目標を達成するための一連の行動を計画することができた。  
例えば、「stack up two pyramids(2つのピラミッドを積み上げなさい)」という命令に対し、不可能であれば「I CAN'T(できません)」と応答し、計画と制約チェックの能力を示した。
- **行動**: 仮想的な積木の世界で行動を実行した。

# SHRDLUの歴史的意義と限界

SHRDLUは、機械が特定の文脈内で言語を「理解」し、有意義な対話を行うことができることを示した、AIの画期的なデモンストラーションでした。NLP(自然言語処理)およびAI研究に大きな影響を与えました。

しかし、SHRDLUには重大な限界がありました。その知能は、極めて単純化された積木の世界に限定されており、一般的な世界の知識を持たず、現実世界の複雑さにスケールアップすることはできませんでした。

また、手作業で作られたルールに依存していたため、柔軟性に欠け、拡張が困難でした。

SHRDLUの自然言語理解での成功は、「積木の世界」の単純によって初めて可能になりました。例えば、「積木の世界」を記述するのに「おそらく50語程度」しか必要としないことが、統合された自然言語理解、推論、行動という複雑なタスクを扱いやすいものにしました。

ウィノグラード自身、次のように語っています。「SHRDLUや同様のプログラムがマイクロワールド(小さな世界)について推論するために採用したアプローチは、現実世界を理解するために必要な膨大な量の知識を扱うように容易にスケールアップできなかった」

このことは、初期のAIが統合された知能を実証するためにマイクロワールド(小さな世界)が不可欠であったことを示しています。

# SHRDLU以降のAI研究

SHRDLUは、自然言語理解、記号的推論、計画、行動を組み合わせて、言語、知覚、推論、行動という、ある意味で非常に統合されたエージェント能力を示しました。その構成要素はルールベースで限定的だったものの、歴史的にはこうしたSHRDLUによる統合自体が重要な成果であったと考えられます。

SHRDLU以降、AI研究は長年にわたりしばしば専門化されたサブ問題(例えば、視覚、計画、NLPを個別に扱う)に焦点を当ててきました。別の言い方をすれば、AI研究の中で、統合された能力を持つAIエージェントに対する関心は、相対的に低下します。

# エージェント論への新しい関心

そうした停滞状態が数十年続いたのですが、変化は突然訪れました。

2012年に始まるDeep Learning (AlexNetでの画像認識、Googleの猫、音声認識)の知覚分野での鮮烈な成功は、LLMでの革命的な自然言語理解能力の獲得へと引き継がれました。

LLMの自然言語理解の能力は、自然言語と同様にSyntaxとSemanticsを持つ人工言語であるプログラム言語の理解へと容易に拡張されました。

また、AIのMulti Modal化は、改めて「統合されたAIエージェント」への関心を高めることになりました。

# SHRDLUとDreamerV3を比較する

SHRDLUとDreamerV3という、50年の時を経て開発された二つのAIシステムを比較することで、AIの進歩の軌跡と、エージェントという概念自体の変化が明らかになると僕は考えています。

個別の論点での比較に入る前に、最初に確認しておきたいことが一つあります。

それは、SHRDLUの「世界モデル」の中核を担っていた、言語理解能力をDreamerV3は欠いていることです。これは、LLMベースのエージェント・モデルが当たり前になっている現在のエージェントのイメージとは、かなり違うものです。それは、DreamerV3エージェントの大きな特徴の一つです。

# 人間とエージェントの対話

SHRDLUの対話は人間からの命令ベースでした。エージェントは人間からの自然言語による指示に反応し、その「目標」は直接与えられました。

一方、DreamerV3は、強化学習を通じて事前定義された目標を達成するためのポリシーを学習します。その対話は、動作中に人間から直接命令を受けるのではなく、報酬を最大化することを目指す学習されたポリシーによって駆動されます。

# 世界の表現

SHRDLUの世界のモデルの表現は、自然言語によるもので、明示的なものでした。世界のモデルは静的であり、行動によって更新されました。

対照的に、DreamerV3の世界表現は、彼の world model である RSS ニューラル・ネットワークによって学習された暗黙的なものです。

あえて言えば、このworld modelを構成するニューロンの個々の重みの総体が、彼が学習した世界の表現だということになります。このモデルでは、SHRDLUでの自然言語で明示的に表現されたモデルのように、人間が直接にその世界の表現を解釈できるものではありません。

# 自然言語と視覚

HRDLUにとって自然言語理解は中心的機能でした、文を解析し、参照を解決し、言語構造をその記号的な世界表現にマッピングしました。視覚は抽象化されており、生の視覚データを処理することなくブロックやピラミッドについて「知って」いたと言えます。

DreamerV3は、SHRDLUとは明確に異なって、本質的に自然言語処理を必要としません。世界のピクセル入力から学習するため、その「視覚処理」は、予測と行動のために画像から関連する特徴を学習することに関するものです。彼は、言語によって媒介された世界理解を必要としません。

# 学習と適応

SHRDLUの学習は最小限かつ言語的でした。試行錯誤から学習したり、経験に基づいて中核的な行動を適応させたりすることは、ありませんでした。その知識はほぼ固定されています。

それに対して、DreamerV3にとって学習は基本的な要素です。経験から world model、価値関数(critic)、ポリシー(actor)を学習し、強化学習を通じて継続的に適応します。

## 汎用性とスケーラビリティ

SHRDLUは、そのマイクロワールドと手作業で作られた知識に高度に制約されていました。より複雑なドメインや異なる種類の知識へのスケーリングは、未解決問題でした。

DreamerV3は、汎用性とスケーラビリティのために設計されています。単一のハイパーパラメータセットが、広範囲の多様なドメイン(連続的/離散的、視覚的/低次元、2D/3D)で機能します。

# 説明可能性 と パフォーマンス

SHRDLUは高度な説明可能性を提供しました。その推論プロセスは、言語的かつルールベースであるため、追跡可能でした。ユーザーの要求や(単純な)目標に基づいて、その行動に関する「なぜ」という質問に答えることができました。

DreamerV3は説明可能性が低いと言わざるを得ません。深層学習システムとして、その意思決定プロセスは不透明(「ブラックボックス」)です。高いパフォーマンスを達成する一方で、特定の行動を特定の時点で選択した理由を詳細に理解することは困難です。

## この50年のAIエージェントの進化？

この50年の進化は、狭く深い説明可能な知能から、広く高性能だがしばしば説明可能性の低い知能へと優先順位が変化したことを示しているようにも見える。

SHRDLUの記号的な性質は、その「思考」(推論ステップ)をアクセス可能にした。しかし、この明示的で手作業でコード化された知識への依存が、汎化におけるアキレス腱であった。

DreamerV3は、データから表現を学習することで、手作業による知識エンジニアリングの必要性を回避し、はるかに複雑で多様な問題に取り組むことを可能にしている。しかし、これらの学習された表現はしばしば不可解である。

## また、別の視点から 昔の話をしてみよう

僕は、実は、この50年間のAIエージェントの進化を、SHRDLからDreamerV3への進化として特徴づける議論の枠組みに、必ずしも賛成ではないのです。

DreamerV3の達成は素晴らしいし、両者の対比は明確で示唆に富んでいるのですが、それは少し単純化を含んでいて、現在の到達を過大評価する危険があるとも感じています。

基本的には、エージェントが自分の外部の世界と相互作用して、それを通じて、エージェントが世界の理解を何らかの形で深めていくこと。

そうした視点から、エージェント論の現在を振り返るには、もう少し、異なったアプローチを知った方がいいとも感じています。

次回のセッションでは、「エージェントに、世界の内部モデルはいらない。世界のモデルは、世界そのものだ。」という、Brooksの表象なき知性 Intelligence without representation」論を紹介しようと思います。



# AIと「世界モデル」(2)

昔の話をしよう

Brooks – Intelligence without representation

# Rodney Brooks



## Intelligence without representation

<https://people.csail.mit.edu/brooks/papers/representation.pdf>

# Rodney Brooks

このセッションでは、Rodney Brooks の 1987年の論文  
“Intelligence without representation” を紹介します。

Rodney Brooks は、SHRDLUの「成功」以降のAI研究の「停滞」の原因を鋭く指摘し、新しいAI研究の道を示した重要なAI研究者です。

もっとも、彼の理論は、当時のAI研究の主流派に受け入れられた訳ではありませんでした。

ただ、彼の理論は、ロボット研究の理論と実践に広く深い影響を与え、その影響は今日も続いています。Brooksは、iRobot社の最高技術責任者で、ルンバの開発者です。

# Intelligence without representation

## Abstract

その論文のAbstractで彼は、次のように述べます。

「人工知能の研究は、表現の問題で停滞している。  
知能を段階的にアプローチし、知覚と行動を通じて現実世界とインターフェースすることに厳密に依存すると、表現への依存は消える。」

人工知能研究の停滞の原因は、「表現の問題」であるとされています。

「知能システムの根本的な分解は、相互に表現を介してインターフェースする必要がある独立した情報処理ユニットへの分解ではない。」

「代わりに、知能システムは、相互に特に多くのインターフェースを必要とせず、知覚と行動を通じて世界と直接インターフェースする独立した並列の活動生成ユニットに分解される。」

「中央システムと周辺システムの概念は消滅し、すべてが中央であり周辺でもある。」

こうした見方を、“Subsumption Architecture” “SA”と略される)といいます。

# Use the world as its own model

## この論文のメッセージ

この論文に込めたBrooksのメッセージは、論文冒頭のIntroduction に明確に示されています。

「我々は、このアプローチに従って、一連の自律移動ロボットを構築してきた。その結果、予想外の結論(C)に達し、かなり過激な仮説(H)を立てた。

(C) 非常に単純なレベルの知能を調べると、世界に関する明示的な表現やモデルは単に邪魔になるだけであることがわかる。世界そのものをそれ自身のモデルとして使うほうが良いことがわかった。」

「(H) 表現は、知能システムの最も大規模な部分を構築する際の抽象化の単位として適切ではない。

表現が過去15年間の人工知能研究の中心的課題であったのは、それだけが孤立したモジュールや学会論文の間でインターフェースを提供していたからに過ぎない。」

SHRDLUのように、知能システムの中心に言語表現による「世界のモデル」を置くのは、システムの発展にとって邪魔になるだけだ。それが停滞の原因だ。知覚と行動を通じて世界と直接インターフェースをとるようにして、世界のモデルは世界自身を使うのがいい。ということです。かなり痛烈な、SHRDLU流のAI批判です。

## 彼の着想の源

彼のの着想の源は生物学と進化にありました。単純な生物は、精巧な内部表現なしに複雑な行動を示します。

彼は進化のタイムスケール(単純な反応は何十億年もかけて進化したが、書記や象徴主義はごく最近のものでしかない)や昆虫の知能に言及しています。10万個のニューロンしか持たない昆虫が、ナビゲートしたり飛んだりできることから、それが複雑な記号数学計算を行っているのか疑問を呈しました。

そして、「世界をそれ自身のモデルとして使う」ことを提唱したのです。こうしたアプローチは、ロボット工学に革命をもたらしました。

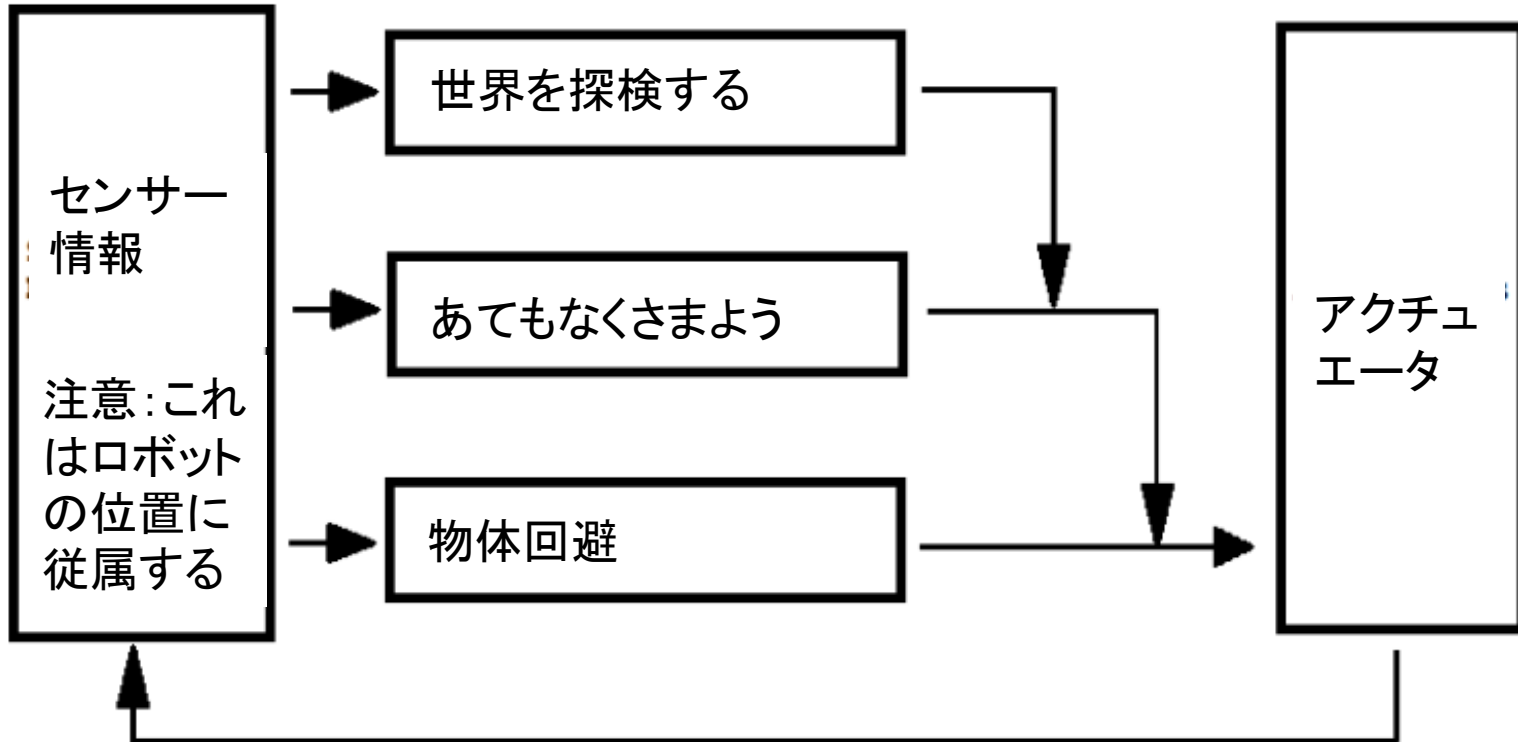
# SA:サブサンクション・アーキテクチャ

## 階層的な制御と拡張有限状態機械

SAの最も特徴的な側面の一つは、その階層的な制御と拡張有限状態機械(AFSM)の使用です。SAでは、知能制御プログラムが、拡張有限状態機械のネットワークの層を定義することによって構築されます。これらのAFSMは、特定の時間経過後に状態変化を開始するように設定できるタイマーを備えた有限状態機械です。

各層は特定の行動能力レベルを実装し、上位の層は下位の層を「サブサンクション」(統合または結合)して、より包括的な行動を生み出します。例えば、「物体回避」の最も基本的な層の上に、「あてもなくさまよう」層が構築され、さらにその上に「世界を探検する」層が追加されるといった具合です。この開発プロセスは反復的であり、最も基本的で低レベルの層から段階的に構築されていきます。

# Subsumption Architecture



# サブサンクション・アーキテクチャ

「身体性」(embodiment)と「状況性」(situatedness)

SAの哲学のもう一つの柱は、グローバルな世界モデルと従来のAIプランニングの否定です。

ブルックスは、「世界そのものが最良のモデルである」と主張し、ロボットは内部の記号表現を構築・維持するのではなく、知覚から行動への直接的な結合を通じて環境と相互作用すべきだと考えました。

この分散型アプローチでは、計算要素間に「自由な」通信ネットワークや共有メモリは存在せず、すべての通信経路は明示的に「ワイヤー」として指定される必要があります。これにより、中央の世界モデルを維持することが困難になり、必要に応じて世界から関連する側面を直接感知する方が容易になるという考え方です。

「状況性」(situatedness)は、ロボットが人間のような時間枠内で環境に反応できるべきであるという重要な概念です。これは、内部モデルに基づく遅い、熟慮的な処理ではなく、動的な環境との直接的でリアルタイムな相互作用を意味します。

また、「身体性」(embodiment)もSAの中心的な要素です。ブルックスは、身体を持つエージェントを構築することで、設計者はシミュレーションだけでなく物理世界で機能する統合された物理制御システムを開発せざるを得なくなると主張しました。身体性は、感覚データと意味のある行動を直接結合させることで、多くの従来のAIが直面する哲学的問題である記号接地問題を解決するのにも役立つと考えられました。

# サブサンプリング・アーキテクチャが もたらした影響とパラダイムシフト

サブサンプリング・アーキテクチャはロボティクスに革命をもたらし、抽象的な問題解決から物理的世界と相互作用する身体化されたエージェントへと焦点を移しました。

それは、行動ベースロボティクスとリアルタイムAIに強い影響を与えました。また身体性、すなわち環境との物理的相互作用が知能にとって不可欠であるという考え方を強調しました。

# サブサンクション・アーキテクチャの困難

## 長期的計画と複雑な目標指向行動における課題:

主に反応的であるため、複雑な多段階計画の策定と実行、または即座に知覚できない抽象的な目標に関する推論が困難でした。「将来の目標を達成する方法を計画したり、潜在的(かつ回避可能)な危険を予測したりすることはできない」。

## 限定的な表現と推論:

「表現なき知能」が核心的な信条であったのですが、これは抽象的な知識や、存在しないエンティティ/状況に関する推論を必要とするタスクの困難さも意味しました。

# サブサンプリング・アーキテクチャと DreamerV3

WinogradのSHRDLUとBrooksのSAとDeepMindのDreamerV3の三つを並べると、AI エージェントの「進化」の一つの方向が見えてきます。

純粹に記号的なシステムであるSHRDLUと純粹に反応的なシステムであるSAの限界は、経験から学習し適応できるエージェントへの関心を高めることになります。

それは、「データ駆動型のエージェント」と呼んでいいものです。

# データ駆動型のエージェントへの 移行を可能にしたもの

このデータ駆動型のエージェントへの移行を可能にしたものは、次のような技術のシフトです。これは周知のことだと思います。

- **ニューラル・ネットワーク**: かつて注目を集めていたが、訓練アルゴリズムの改良とマシンの計算能力の向上で、AI研究の第一線に復活。
- **コンポーネントの中核としての機械学習(ML)**: 明示的なルールをプログラミングすることから、システムがデータからパターンや行動を学習できるようにすることへのシフト。
- **強化学習(RL)**: 相互作用と報酬を通じて学習するエージェントへ

# DreamerV3での「世界モデル」

DreamerV3 の「世界モデル」は、環境がどのように振る舞うかを学習した予測モデルです。これにより、DreamerV3は、「もしxをしたら何が起こるか？」と問いかけることができます。学習されたモデル内で将来の軌道をシミュレートすること(「想像」)によって「計画」を可能にします。

「世界モデル」はサンプル効率を向上させることができます。なぜなら、DreamerV3は現実世界の相互作用よりも生成コストの低いシミュレートされた経験から学習できるからです。

DreamerV3は、内部で結果を探求できるようにすることで、疎な報酬や長期的な時間軸といった課題に対処することができるようになりました。

# DreamerV3

## エージェント技術の歴史的「統合」

DreamerV3のような「世界モデル」を持つ学習ベースのエージェントの登場は、先行したエージェント技術のある種の歴史的統合を表していると考えられます。

SHRDLUのような初期の記号主義AIは、手作業で作られた明示的な表現に依存していました。ブルックスはこれらを硬直的すぎると拒絶しました。

DreamerV3は、ブルックスが批判した内部「表現」(世界モデル)を再導入するのですが、その表現はSHRDLUのようなプログラムされた静的なものではなく、学習された動的なものです。

初期のRLは、BrooksのSAのように、内部に「世界モデル」を持たず、反応的な方策を学習していました。

DreamerV3に見られるようなモデルベースRLは、内部モデルのアイデアを復活させるのですが、このモデルはデータから学習される、確率論的に振る舞い予測を可能にするなエンジンです。

こうして、DreamerV3は、静的で決定論的な表現に関するブルックスの懸念に対処しつつ、SAのような純粹に反応的なシステムには欠けていた計画能力を可能にしました。

# DreamerV3における「想像」の概念

僕は、DreamerV3における「想像」の概念に注目しています。

それは、人間の先見性に対する強力な計算論的アナロジーを提供するものです。それは、エージェントが高価な現実世界の試行錯誤なしに潜在的な未来を探求することを可能にするものです。

それは、しばしば完全なモデルに依存していたSHRDLUの純粋に記号的な計画と、SAの純粋に反応的な行動の両方を超えた重要なステップであると思います。

# DreamerV3は、複雑さにどう挑戦したか？

もっとも、DreamerV3が、エージェントに課せられたすべての課題をクリアした訳ではありません。

次回のセッションでは、DreamerV3が、複雑さの問題に、どのように挑戦したのかを見ていきたいと思います。





# Part 3

## DreamerV3でのAI技術の飛躍

マルレク 2025/07/05

「AIとマイクラフトの世界と、昔の話をしよう」

# Part 3 DreamerV3でのAI技術の飛躍

## Agenda

- DreamerV3はEnder Dragonを倒せるか？
- DreamerV3でのAI技術の飛躍

# DreamerV3



Ender Dragonを倒せるか？

# MinecraftでのDiamond 採掘の難しさ

DreamerV3の作者は、マインクラフトでのダイヤモンド集めの難しさについて、次のように語っています。

「人気ゲーム『Minecraft』で人間の手を借りずにダイヤモンドを収集することは、報酬が疎で探索が困難、かつ手続き的に生成されたオープンワールド環境における長い時間軸という特徴から、人工知能の重要なマイルストーンとして広く認識されています。

DreamerV3は、**専門家によるデモやカリキュラムなしで**、希薄な報酬からダイヤモンドを収集する最初のアルゴリズムであり、この課題解決に成功しました。動画では、3000万環境ステップ(プレイ時間17日)で収集した最初のダイヤモンドが紹介されています。」

<https://danijar.com/project/dreamerv3/>

ネットには、こんな期待の声があります。

「DreamerV3 は、Minecraft でダイヤモンドを収集する方法を完全に独学で習得した最初のアルゴリズムだ。

これにより、強化学習の適用範囲がさらに拡大した。ネットユーザーたちが言うように、DreamerV3 はすでに成熟した汎用アルゴリズムだ。

次は、自分でモンスターをアップグレードして倒し、究極のボスであるEnder Dragonと戦う方法を学ぼう！」

<https://hyper.ai/en/news/23036>

# DreamerV3は、 Ender Dragon を倒せるのか？

Ender Dragonは、マインクラフトの世界のラスボスです。ゲームとしてマインクラフトを見れば、その最終目標は、Ender Dragonを倒すことです。

DreamerV3は素晴らしいという話をしてきたのですが、このセッションでは、「DreamerV3は、Ender Dragon を倒せるのか？」という問題を考えてみようと思います。

ゲームに関心のない人には、どうでもいいことに思えるかもしれませんが。

ただ、先のコメントには、DreamerV3の達成について、いくつかの誤解が含まれていると思います。

# DreamerV3 は、「完全に独学」で ダイヤモンドを収集する方法を習得したのか？

問題は、DreamerV3 は、「完全に独学」でダイヤモンドを収集する方法を習得したのか？ ということです。

DreamerV3の論文でもGitHubでも、「非常に疎な報酬」に対する強化学習での成功が語られ、それが「専門家によるデモやカリキュラムなし」で行われたことが強調されています。

ただ、DreamerV3のソースコードを読んでわかったことがあります。

それは、現在の実装では、「ダイヤモンドの収集」という最終ゴールに至るまで、12個のサブゴールが設定され、それぞれのサブゴールに報酬が設定されています。それは、「隠れたカリキュラム」と言ってもいいと思います。

# DreamerV3のソースを読む

このセッションでは、DreamerV3のソースコードを読みたいと思います。

コードの基本的な部分は、次に公開されています。

“Mastering Diverse Domains through World Models”














<https://github.com/danijar/dreamerv3/tree/main?tab=readme-ov-file>

# DreamerV3のディレクトリー構造

GitHubでのDreamerV3のルートディレクトリーは、次のような構造をしています。

このうち、今回のセッションで重要なのは、次の三つです。

- dreamerv3/dreamerv3 ディレクトリー
- dreamerv3/embodied ディレクトリー
- baseline.yaml ファイル

 dreamerv3	Update Dockerfile
 embodied	Fix parallel driver argument
 scores	Update agent and infrastructure
 .dockerignore	Update agent and infrastructure
 .gitignore	Initial commit
 Dockerfile	Update Dockerfile
 LICENSE	Update
 README.md	Update readme
 baselines.yaml	Update
 entrypoint.sh	Update
 plot.py	Update
 requirements.txt	Pin Atari version
 setup.py	Update

## ./baseline.yaml ファイル

DreamerV3が、150ものタスクをハイパー・パラメーターを変更することなく、実行できることは論文でも強調されていました。baseline.yaml ファイルには、そのタスクの一覧の一部が収められています。






```
1   atari57_gamer:
2     atari_alien: [227.8, 7127.7]
3     atari_amidar: [5.8, 1719.5]
4     atari_assault: [222.4, 742.0]
5     atari_asterix: [210.0, 8503.3]
6     atari_asteroids: [719.1, 47388.7]
7     atari_atlantis: [12850.0, 29028.1]
8     atari_bank_heist: [14.2, 753.1]
9     atari_battle_zone: [2360.0, 37187.5]
10    atari_beam_rider: [363.9, 16926.5]
11    atari_berzerk: [123.7, 2630.4]
12    atari_burris: [22.1, 160.7]
```

```
92     procgen_hard:
93         procgen_bigfish: [0, 40]
94         procgen_bossfight: [0.5, 13]
95         procgen_caveflyer: [2, 13.4]
96         procgen_chaser: [0.5, 14.2]
97         procgen_climber: [1, 12.6]
98         procgen_coinrun: [5, 10]
99         procgen_dodgeball: [1.5, 19]
100        procgen_fruitbot: [-0.5, 27.2]
101        procgen_heist: [2, 10]
102        procgen_jumper: [1, 10]
103        procgen_leaper: [1.5, 10]
104        procgen_maze: [4, 10]
105        procgen_miner: [1.5, 20]
106        procgen_ninja: [2, 10]
107        procgen_plunder: [3, 30]
108        procgen_starpilot: [1.5, 35]
```

minecraft ないですね

# ./dreamerv3/dreamerv3ディレクトリー

dreamerv3/dreamerv3ディレクトリーは、次のような構造をしています。

 <code>__init__.py</code>	Update
 <code>agent.py</code>	Update
 <code>configs.yaml</code>	Update Dockerfile
 <code>main.py</code>	Update
 <code>rssm.py</code>	Update

この中で、`configs.yaml` と `main.py` に注目しましょう。

# ./dreamerv3/dreamerv3/configs.yaml

./dreamerv3/configs.yaml は、先の ./baseline.yaml の情報を補って、汎用の実行環境である DreamerV3 が対象とする様々な環境とタスクの情報の一覧を与えています。

env:

```
atari: {size: [96, 96], repeat: 4, sticky: True, gray: True, actions: all, lives: unused, noops:
procgen: {size: [96, 96], resize: pillow}
crafter: {size: [64, 64], logs: False}
atari100k: {size: [64, 64], repeat: 4, sticky: False, gray: False, actions: needed, lives: unused}
dmlab: {size: [64, 64], repeat: 4, episodic: True, use_seed: True}
minecraft: {size: [64, 64], break_speed: 100.0, logs: False, length: 36000}
dmc: {size: [64, 64], repeat: 1, proprio: True, image: True, camera: -1}
loconav: {size: [64, 64], repeat: 1, camera: -1}
```

**minecraft は、ここに出ています。**

```
149
150     size400m: &size400m
151     .*\.rsm: {deter: 12288, hidden: 1536, classes: 96}
152     .*\.depth: 96
153     .*\.units: 1536
154
155     minecraft:
156         task: minecraft_diamond
157
158     dmlab:
159         task: dmlab_explore_goal_locations_small
160         run: {steps: 2.6e7, train_ratio: 32}
161
162     atari:
163         task: atari_pong
164         run: {steps: 5.1e7, train_ratio: 32}
---
```

もう一箇所、minecraft は、ここにも出てきます。

# ./dreamerv3/dreanerv3/main.py

./dreamerv3/dreanerv3/main.py はDreamerV3のメインプログラムです。

その主要な働きは、汎用のゲーム・シミュレータであるDreamerV3に、次のような形で与えられるコマンド引数を解釈して、特定のゲーム環境の構築と、そこでのタスクの指定を行うことです。

main.py **-configs** minecraft **-task** minecraft\_diamond



構築すべきゲーム環境



そこで達成すべきタスク

## main.py の make\_env(...) 関数

```
def make_env(config, index, **overrides):
    suite, task = config.task.split('_', 1)
    if suite == 'memmaze':
        from embodied.envs import from_gym
        import memory_maze # noqa
    ctor = {
        'dummy': 'embodied.envs.dummy:Dummy',
        'gym': 'embodied.envs.from_gym:FromGym',
        'dm': 'embodied.envs.from_dmenv:FromDM',
        'crafter': 'embodied.envs.crafter:Crafter',
        'dmc': 'embodied.envs.dmc:DMC',
```

```
'atari': 'embodied.envs.atari:Atari',
'atari100k': 'embodied.envs.atari:Atari',
'dmlab': 'embodied.envs.dmlab:DMLab',
'minecraft': 'embodied.envs.minecraft:Minecraft',
'loconav': 'embodied.envs.loconav:LocoNav',
'pinpad': 'embodied.envs.pinpad:PinPad',
'langroom': 'embodied.envs.langroom:LangRoom',
'procgen': 'embodied.envs.procgen:ProcGen',
'bsuite': 'embodied.envs.bsuite:BSuite',
'memmaze': lambda task, **kw:
from_gym.FromGym(
    f'MemoryMaze-{task}-v0', **kw),
}[suite]
```

```
if isinstance(ctor, str):
    module, cls = ctor.split(':')
    module = importlib.import_module(module)
    ctor = getattr(module, cls)
kwargs = config.env.get(suite, {})
kwargs.update(overrides)
if kwargs.pop('use_seed', False):
    kwargs['seed'] = hash((config.seed, index)) % (2
** 32 - 1)
if kwargs.pop('use_logdir', False):
    kwargs['logdir'] = elements.Path(config.logdir) /
f'env{index}'
env = ctor(task, **kwargs)
return wrap_env(env, config)
```

```
if isinstance(ctor, str):
```

```
    module, cls = ctor.split(':')
```

```
ctor = 'embodied.envs.minecraft:Minecraft'
```

```
module, cls = ctor.split(':')
```

```
module = 'embodied.envs.minecraft'
```

```
cls = 'Minecraft'
```

```
** 32 - 1)
```

```
    if kwargs.pop('use_logdir', False):
```

```
        kwargs['logdir'] = elements.Path(config.logdir) /  
f'env{index}'
```

```
    env = ctor(task, **kwargs)
```

```
    return wrap_env(env, config)
```

```
if isinstance(ctor, str):  
    module, cls = ctor.split(':')  
    module = importlib.import_module(module)
```

```
module = 'embodied.envs.minecraft'
```

```
module = importlib.import_module(module)
```

```
module =  
importlib.import_module('embodied.envs.minecraft')
```

```
if kwargs.pop('use_logdir', False):  
    kwargs['logdir'] = elements.Path(config.logdir) /  
f'env{index}'  
env = ctor(task, **kwargs)  
return wrap_env(env, config)
```

```
if isinstance(ctor, str):
    module, cls = ctor.split(':')
    module = importlib.import_module(module)
    ctor = getattr(module, cls)
```

```
cls = 'Minecraft'
module =
importlib.import_module('embodied.envs.minecraft')

ctor = getattr(module, 'Minecraft' )
```

```
kwargs['logdir'] = elements.Path(config.logdir) /
f'env{index}'
env = ctor(task, **kwargs)
return wrap_env(env, config)
```

# Minecraft環境の構築

## embodied.envs.minecraft モジュール

```
import importlib
import embodied
```

```
ctor = getattr(module, 'Minecraft')
```

```
class Minecraft(embodied.Wrapper):
    def __init__(self, task, *args, **kwargs):
        module, cls = {
            'wood': 'minecraft_flat:Wood',
            'climb': 'minecraft_flat:Climb',
            'diamond': 'minecraft_flat:Diamond',
        }[task].split(':')
```

```
        module = importlib.import_module(f'.{module}', __package__)
        cls = getattr(module, cls)
        env = cls(*args, **kwargs)
        super().__init__(env)
```

# ./embodied/envs/minecraft\_flat.py

## class Diamond(embodied.Wrapper):

```
def __init__(self, *args, **kwargs):
    actions = {
        **BASIC_ACTIONS,
        'craft_planks': dict(craft='planks'),
        'craft_stick': dict(craft='stick'),
        'craft_crafting_table': dict(craft='crafting_table'),
        'place_crafting_table': dict(place='crafting_table'),
        'craft_wooden_pickaxe': dict(nearbyCraft='wooden_pickaxe'),
        'craft_stone_pickaxe': dict(nearbyCraft='stone_pickaxe'),
        'craft_iron_pickaxe': dict(nearbyCraft='iron_pickaxe'),
        'equip_stone_pickaxe': dict(equip='stone_pickaxe'),
        'equip_wooden_pickaxe': dict(equip='wooden_pickaxe'),
        'equip_iron_pickaxe': dict(equip='iron_pickaxe'),
        'craft_furnace': dict(nearbyCraft='furnace'),
        'place_furnace': dict(place='furnace'),
        'smelt_iron_ingot': dict(nearbySmelt='iron_ingot'),
    }
```

```
self.rewards = [  
    CollectReward('log', once=1),  
    CollectReward('planks', once=1),  
    CollectReward('stick', once=1),  
    CollectReward('crafting_table', once=1),  
    CollectReward('wooden_pickaxe', once=1),  
    CollectReward('cobblestone', once=1),  
    CollectReward('stone_pickaxe', once=1),  
    CollectReward('iron_ore', once=1),  
    CollectReward('furnace', once=1),  
    CollectReward('iron_ingot', once=1),  
    CollectReward('iron_pickaxe', once=1),  
    CollectReward('diamond', once=1),  
    HealthReward(),  
]
```

```
length = kwargs.pop('length', 36000)
env = MinecraftBase(actions, *args, **kwargs)
env = embodied.wrappers.TimeLimit(env, length)
super().__init__(env)
```

# Diamond Env レポジトリ

[https://github.com/danijar/diamond\\_env](https://github.com/danijar/diamond_env)

「このDiamond Envは、DreamerV3 が使用する Minecraft ダイヤモンド環境です。

DreamerV3 は、人間のデータや手動で作成されたカリキュラムを使用せずに Minecraft でダイヤモンドを収集する最初の強化学習アルゴリズムです。

私たちは、この環境を、既存の多くの人気のあるベンチマークよりも興味深い課題を提供する、強化学習研究のための標準的なベンチマークとして提案します。」

# Diamond Env Overview

ダイヤモンド環境では、エージェントは Minecraft をプレイして、希少な報酬のみから、ダイヤモンドを収集するまで 12 のマイルストーンを達成します。これは探検の挑戦課題となります。さらに、各エピソードはランダムに生成されたユニークな 3D 世界で展開されるため、エージェントは一般化する能力が必要となります。

この環境は、抽象的なクラフトアクションのサポートを含む最新バージョンである MineRL バージョン 0.4.4 (コミット 204130f) に基づいています。私たちは、アルゴリズムの比較を容易にするため、バグ修正と標準化されたカテゴリ別アクション空間および観測空間を提供しています。。

# 観測空間

各観測値は、以下のキーと、対応する配列のdtypesおよびshapeを持つ辞書です。

image:	uint8 (64, 64, 3)
inventory:	float32 (391,)
inventory_max:	float32 (391,)
equipped:	float32 (393,)
breath:	float32 ()
health:	float32 ()
hunger:	float32 ()

# アクション空間

アクション空間は、以下の 25 のアクションを持つ、平坦なカテゴリ空間です。

noop,  
attack turn\_up, turn\_down, turn\_left, turn\_right,  
forward, back, left, right, jump,  
place\_dirt, craft\_planks, craft\_stick,  
craft\_crafting\_table, place\_crafting\_table,  
craft\_wooden\_pickaxe, craft\_stone\_pickaxe,  
craft\_iron\_pickaxe  
equip\_stone\_pickaxe, equip\_wooden\_pickaxe,  
equip\_iron\_pickaxe,  
craft\_furnace, place\_furnace, smelt\_iron\_ingot

# 報酬関数

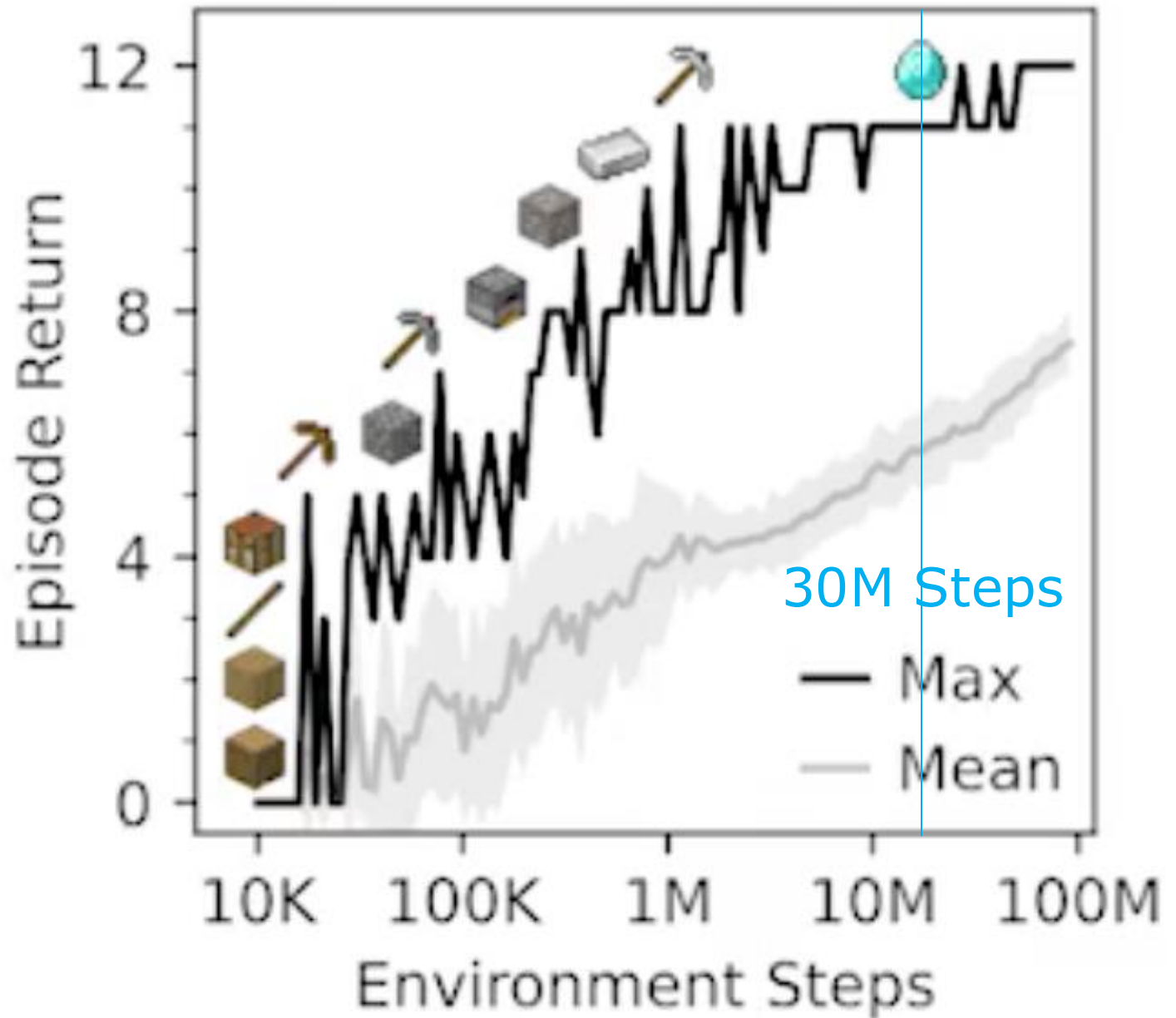
報酬関数は疎です。以下の 12 のマイルストーンは、現在のエピソード中にそのアイテムが初めて取得されたときに 1 の報酬を生成します。

log, planks, stick, crafting\_table, wooden\_pickaxe, cobblestone, stone\_pickaxe, iron\_ore, furnace, iron\_ingot, iron\_pickaxe, diamond

# 報酬関数

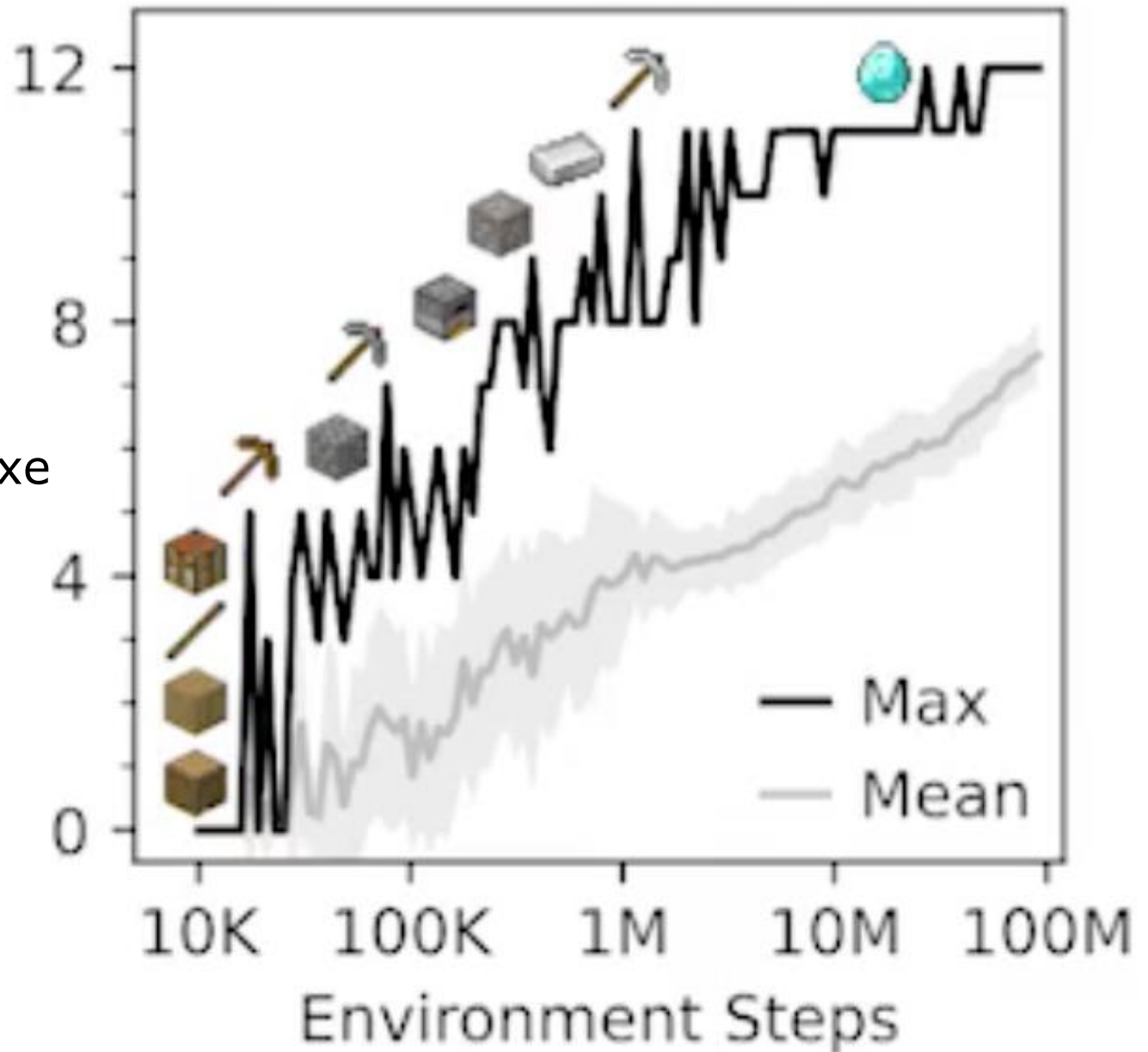
1. log
2. planks
3. stick
4. crafting\_table
5. wooden\_pickaxe
6. cobblestone
7. stone\_pickaxe
8. iron\_ore
9. furnace
10. iron\_ingot
11. iron\_pickaxe
12. diamond

# Minecraft Diamond



# Minecraft Diamond

diamond  
iron\_pickaxe  
iron\_ingot  
furnace  
iron\_ore  
stone\_pickaxe  
cobblestone  
wooden\_pickaxe  
crafting\_table  
stick  
planks  
log



# 12個のsub-goal 「隠れたカリキュラム」

DreamerV3 論文 (Nature, 2024) では以下のように述べられています:

“DreamerV3 is the first algorithm to solve the challenging Minecraft Diamond task from scratch, without expert demonstrations or curriculum, using only sparse rewards.”

この文言からは、

- 「完全なスパース報酬環境」
- 「報酬はダイヤモンド取得のみ」
- 「途中の段階に対しては報酬が一切無い」

という印象を受けます。

# DreamerV3は、Ender Dragon を倒せるのか？

でも、実際には、ダイヤモンド採取という最終ゴールに対して、12個のサブゴールを設定して、それぞれのサブゴールに報酬を与えています。それは「隠れたカリキュラム」と呼んでいいものです。

DreamerV3はダイヤモンドをゲットするのに、30M step (3000万ステップ)を要しました。

はるかに複雑な「Ender Dragon打倒」というタスクを、「Ender Dragon打倒」した時だけ報酬を与えるというスタイルで、DreamerV3で実行するのは無理だと思います。

## サブゴールを与えるという戦略は 複雑な問題に取り組むためには必要である

ただ、DreamerV3が、Ender Dragon を倒す方法は、あるかもしれません。

複雑な問題に取り組むために、サブゴールを考えるという戦略は、不当なアプローチでは、まったくありません。例えば、数学の問題を解く時には、日常的に行われていることです。

非常に疎な報酬構造でも動く汎用の強化学習エンジンとそのアルゴリズム開発を志向するという開発目標からは離れるかもしれませんが、DreamerV3のアイデアの延長上で、「Ender Dragon を倒す」ことは、効率的なサブゴールの設計を通じて可能かもしれません。それはそれで、チャレンジングだと僕は思います。

# DreamerV3でのAI技術の飛躍



# AI技術としてのDreamerV3

これまで、DreamerV3 を、主に、Minecraft でダイヤモンド採掘もできる「ゲームをするAI」として紹介してきた。

このセッションでは、もう少し一般的な視点で、DreamerV3 を見てみようと思う。

DreamerV3が、AI技術としてどのような特徴を持っているかを整理することで、DeepMindのAI開発のビジョンの一端を知ることができると考えている。

# DreamerV3の主要目的

人工知能(AI)研究における長年の課題の一つは、広範な応用分野にわたる多様なタスクを学習し解決できる汎用アルゴリズムの開発であった。

既存の強化学習(RL)アルゴリズムは、開発されたタスクと類似のタスクには容易に適用できるものの、新たな応用領域に合わせて設定するには、依然として人間の専門知識と多くの実験が必要とされる。

DreamerV3は、この課題に対処することを主要目的としている。具体的には、単一の固定された設定で150以上の多様なタスクにおいて専門的な手法を凌駕する性能を発揮する汎用アルゴリズムの実現を目指した。

# DreamerV3の提案手法

DreamerV3の核心的な手法は、環境の「ワールドモデル」を学習することにある。

このワールドモデルは、エージェントが潜在的な行動の結果を予測し、将来のシナリオを「想像」することで行動を改善することを可能にする。

システムは、並行して訓練される3つのニューラルネットワークで構成される。

### ワールドモデル:

感覚入力 $x_t$ をエンコーダ(enc)を用いて離散表現 $z_t$ に符号化し、これらの表現の系列を予測する。入力はデコーダ(dec)によって $\hat{x}_t$ として再構成され、表現を形成するのに役立つ。具体的には、自己符号化を通じて感覚入力のコンパクトな表現を学習する。

### クリティックネットワーク:

想像された各結果の価値を判断する。

### アクターネットワーク:

最も価値の高い結果に到達するための行動を選択する。

# DreamerV3の新規性

## 汎用性と固定ハイパーパラメータ:

DreamerV3は、150以上の多様なタスクにおいて、単一の固定されたハイパーパラメータセットで専門的な手法を上回る性能を達成した。これは、真に汎用的な学習エージェントに向けた重要な一歩である。

## ロバスト性技術:

正規化、バランシング、変換に基づく技術を導入することで、多様なドメインや変動する信号の大きさに対して安定した学習を可能にした。例えば、報酬の正規化においては、観測された報酬の5パーセンタイルから95パーセンタイルの範囲を計算し、外れ値に対してロバストな学習を実現している。これは、異なる環境で遭遇する広範なデータ分布や報酬スケールを扱う上で不可欠である。

## スケーラビリティ:

より大きなモデルサイズは、より高いスコアを達成するだけでなく、タスク解決に必要なインタラクションも少なくなる傾向があり、性能とデータ効率を予測可能な形で向上させる方法を提供する。

# DreamerV3の成果

DreamerV3の顕著な成果としては、人間によるデータやカリキュラムなしに、ゼロからMinecraftでダイヤモンドを収集することに初めて成功した点が挙げられる。これは、疎な報酬、探索の困難さ、長い時間軸、そしてオープンワールドにおける手続き的な多様性のために、AIにとって実質的な挑戦と認識されてきた課題である。

歴史的に見れば、複数のタスクをこなすAIシステムであっても、新しいタスクやドメインごとに大幅な再調整やアーキテクチャの変更が必要となることが多かった。DreamerV3が「単一構成」または「固定ハイパーパラメータ」で「150以上の多様なタスク」をこなすという事実は、タスク特化型知能から、より本質的で適応的な学習能力への移行を示している。

これは、将来のAI開発が、特殊化されたソリューションの集合体ではなく、本質的にロバストで柔軟な学習アーキテクチャの創出に重点を置く可能性を示唆しており、新たな問題へのAI適用に必要な専門知識の度合いを低減させ、AI技術の民主化に繋がらうる。

さらに、ワールドモデルの学習への依存は、エージェントが環境の内部的な「理解」またはシミュレーションを構築する方向への動きを意味しており、これはより高度な推論と計画の基礎となる。

DreamerV3は、AIエージェントが再設定されることなく新たなドメインを学習し習得するという、AIにおける中心的な課題に向けて大きな進歩を示している。これにより、強化学習の適用範囲が大幅に広がる可能性がある。広範な実験やドメイン固有のチューニングの必要性をなくすことで、DreamerV3は強化学習を様々な分野の困難な制御問題を解決するための容易に適用可能なツールにすることを目指している。

## DreamerV3の課題

しかしながら、DreamerV3にも限界は存在する。例えば、Minecraftにおいては、ダイヤモンドを収集できるものの、1億環境ステップ未満では、人間のプレイヤーのように全てのシナリオで収集できるわけではない。

また、DreamerV3のようなリカレント・ステート・スペース・モデル (RSSM) ベースの世界モデルは、一般に単一ステップの統計的推論に依存しており、これが長期的な想像タスクにおいて予測誤差の蓄積を引き起こす可能性がある。

この「長期想像」のボトルネックは、DreamerV3がRSSMベースであることから、そのロバスト性技術によってある程度緩和されているとしても、根本的な課題として残っている可能性が高い。

これは、DreamerV3が成功を収めている一方で、その想像されるシナリオにおける非常に長期的な予測の質と信頼性が依然として弱点である可能性を示唆している。

このことから、次世代のワールドモデルは、純粹に統計的な系列予測を超えて、より一貫性のある正確な長期計画を保証するためのメカニズム(例えば、DMWMで提案されている論理的推論の統合)を組み込む必要が出てくるだろう。





# Part 4

## DeepMindの研究動向と DreamerV3

マルレク 2025/07/05

「AIとマイクラの世界と、昔の話をしよう」

# Part 4 DeepMindの研究動向とDreamerV3

## Agenda

- DeepMindの三つのプロジェクト
- DeepMindのAI開発のビジョン

A photograph of a pond with several large, green lily pads floating on the water. Several yellow flowers are in various stages of bloom, with some fully open and others as buds. The water is dark and reflects the surrounding foliage. The text "DeepMindの三つのプロジェクト" is overlaid in the center of the image.

# DeepMindの三つのプロジェクト

# DeepMindの三つのプロジェクト

このセッションでは、DeepMindによる近年の画期的なAI研究、特に汎用制御エージェント「DreamerV3」、数学的証明システム「AlphaGeometry」、そして科学的・アルゴリズム的発見コーディングエージェント「AlphaEvolve」の三つに焦点を当てて、その概要を見ていきたいと思います。

- DreamerV3
- AlphaGeometry
- AlphaEvolve

これらの研究は、異なるアプローチを取りながらも、AIのフロンティアを拡大するという共通のビジョンを追求しています。

## 三つのプロジェクトの概要

- **DreamerV3**は、多様なタスクを単一の固定構成で習得する能力を示し、AIの汎用性における新たな地平を切り開きました
- **AlphaGeometry**は、神経言語モデルと記号的演繹エンジンを融合させ、オリンピックレベルの幾何学問題を解くことで、AIによる高度な数学的推論の可能性を実証しました。
- **AlphaEvolve**は、大規模言語モデル(LLM)と進化的アルゴリズムを組み合わせ、新たなアルゴリズムを発見し、既存の計算システムを最適化することで、AIによる自律的な科学的発見と技術革新の時代の到来を予感させました。

# DreamerV3

## **“Mastering Diverse Domains through World Models”**

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba,  
Timothy Lillicrap

[Submitted on 10 Jan 2023 (v1), last revised 17 Apr  
2024 (this version, v2)]

<https://arxiv.org/abs/2301.04104>

# DreamerV3 論文 Abstract

幅広いアプリケーションにおけるタスクを解決する能力を学習する汎用アルゴリズムの開発は、人工知能における根本的な課題の一つです。

現在の強化学習アルゴリズムは、開発された目的と類似したタスクには容易に適用可能ですが、新たな応用分野への適応には、多大な人間による専門知識と実験が必要です。

私たちは、単一の構成で150を超える多様なタスクにおいて専門的な手法を凌駕する汎用アルゴリズム「DreamerV3」を提案する。

Dreamerは環境のモデルを学習し、未来のシナリオを想像することで行動を改善する。正規化、バランス調整、変換に基づく頑健性技術により、ドメインを越えた安定した学習を実現する。

Dreamerは、人間によるデータやカリキュラムなしでMinecraftからダイヤモンドを収集する最初のアルゴリズムです。

この成果は、オープンワールドにおけるピクセルと希薄な報酬からはるか遠くを展望する戦略を探索する必要がある人工知能における重大な課題として提示されてきました。

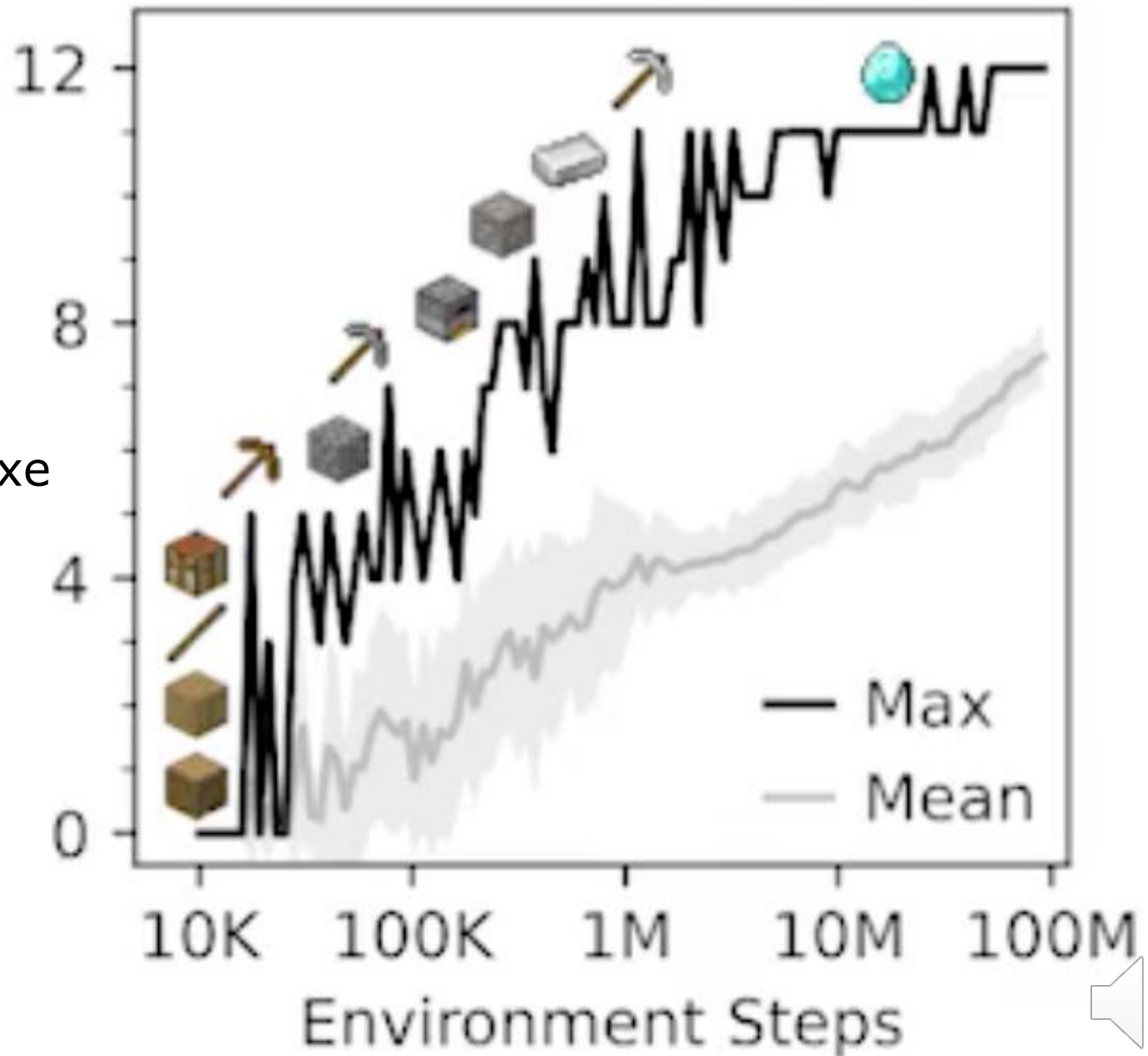
私たちの研究は、広範な実験を必要とせずに困難な制御問題を解決可能にし、強化学習の汎用性を高めています。

# DreamerV3



# Minecraft Diamond

diamond  
iron\_pickaxe  
iron\_ingot  
furnace  
iron\_ore  
stone\_pickaxe  
cobblestone  
wooden\_pickaxe  
crafting\_table  
stick  
planks  
log



# AlphaGeometry

## **“Solving olympiad geometry without human demonstrations”**

Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He & Thang Luong

Published: 17 January 2024

<https://www.nature.com/articles/s41586-023-06747-5>

# AlphaGeometry 論文 Abstract

オリンピックレベルの数学の定理を証明することは、人間レベルの自動推論において重要なマイルストーンだ。これは、大学入学前の数学の分野において、世界トップレベルの才能を持つ者たちにとっても非常に難しいとされているためだ。

しかし、現在の機械学習のアプローチは、人間の証明を機械で検証可能な形式に変換するコストが高いため、ほとんどの数学の分野には適用できない。幾何学の場合、その翻訳の難しさにより、問題はさらに深刻で、トレーニングデータの不足が深刻だ。

そこで、我々は、人間による証明を必要としない、ユークリッド平面幾何学の定理証明システム「AlphaGeometry」を提案する。

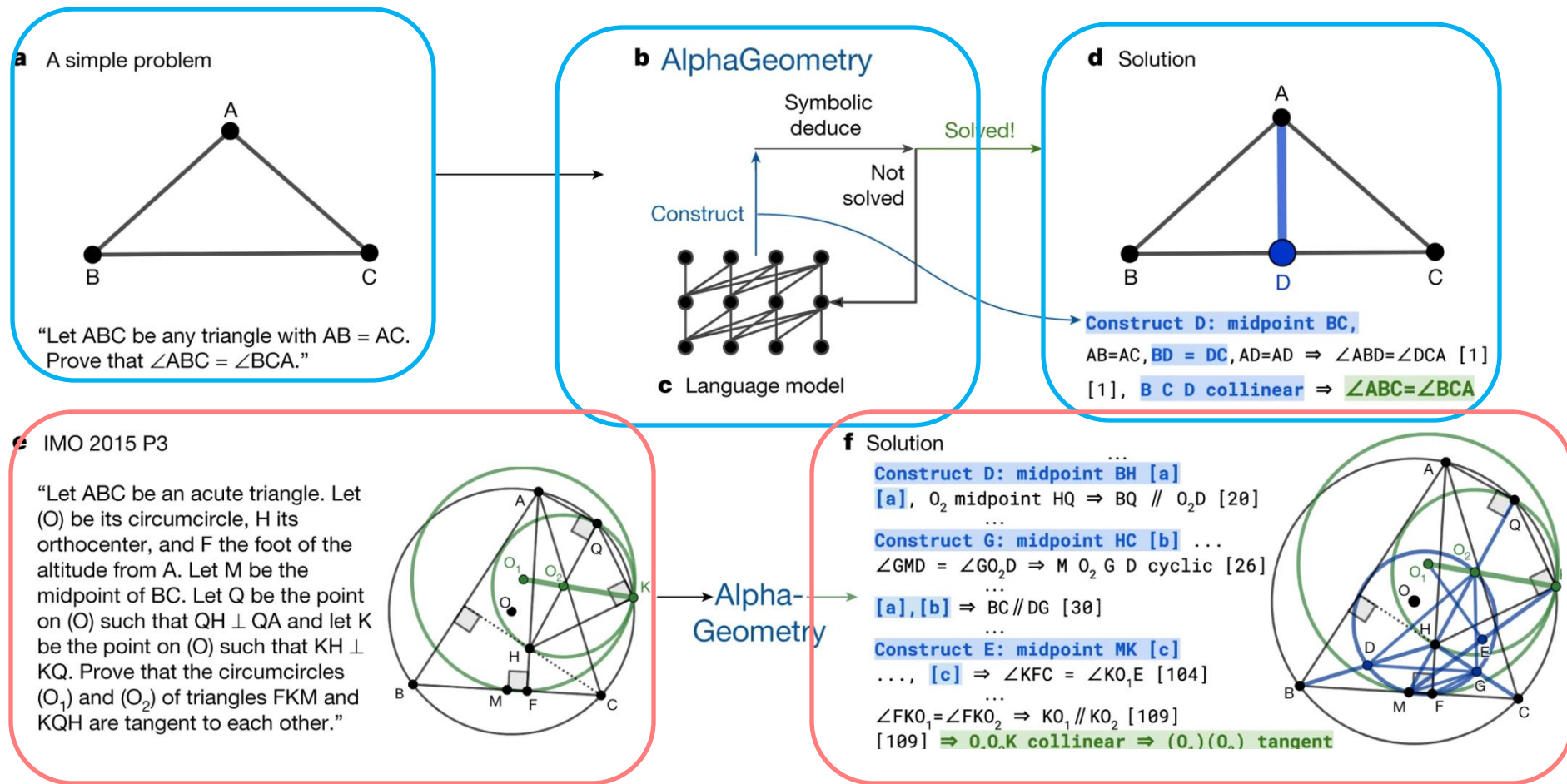
このシステムは、さまざまな複雑さの定理と証明を何百万も合成することで、人間による証明の必要性を回避している。

AlphaGeometryは、大規模な合成データから一から訓練された神経言語モデルを使用し、困難な問題の無限の分岐点をシンボリック推論エンジンに導くニューロ・シンボリックシステムだ。

30の問題からなる最新のオリンピックレベルの問題セットにおいて、AlphaGeometryは25の問題を解決し、10問題しか解決できなかった従来の最良手法を凌駕し、国際数学オリンピック(IMO)金メダリストの平均性能に迫る性能を示した。

特に、AlphaGeometryは人間が読める証明を生成し、IMO 2000と2015のすべての幾何学問題を人間専門家による評価で解き、2004年に翻訳されたIMO定理の一般化バージョンを発見した。

# 図 1: ニューロシンボリック AlphaGeometry の概要と、それが単純な問題と IMO 2015 問題 3 の両方をどのように解決するか。



上段は、AlphaGeometry が単純な問題を解く様子を示しています。a は、単純な例とその図です。b は、AlphaGeometry が記号推論エンジンを実行して証明探索を開始します。エンジンは、定理が証明されるか、新しい文が尽きるまで、定理の前提から新しい文を網羅的に推論します。

c は、記号エンジンが証明を見つけられなかったため、言語モデルが 1 つの補助点を構築し、記号エンジンが再試行する前に証明状態を拡大しています。このループは解決策が見つかるまで続きます。

d、単純な例では、最初の補助点「D を BC の中点とする」の構築後にループが終了します。

証明は、青でハイライトされた「 $BD = DC$ 」と「B、D、C は共線的」という中点の性質を利用する他の2つのステップからなります。

下段は、AlphaGeometry が IMO 2015 問題 3(IMO 2015 P3)を解く方法を示しています。

e、IMO 2015 P3 の問題文と図。

f、IMO 2015 P3 の解には3つの補助点があります。

両方の解では、言語モデルの出力をシンボリックエンジンの出力と交互に配置し、実行順序を反映しています。

f の IMO 2015 P3 の証明は、説明のために大幅に短縮・編集されています。その完全版は補足情報に記載されています。

# IMO 2015 P3

**原問題:**  $ABC$  を  $AB > AC$  である三角形とする。Gamma をその外接円、 $H$  をその垂心、 $F$  を  $A$  から下った高さの足点とする。  $M$  を  $BC$  の中点とする。  $Q$  を Gamma 上の点で  $\angle HQA = 90^\circ$  となる点とし、  $K$  を Gamma 上の点で  $\angle HKQ = 90^\circ$  となる点とする。点  $A, B, C, K, Q$  はすべて異なり、この順序で Gamma 上に存在すると仮定する。三角形  $KQH$  と  $FKM$  の外接円が互いに接することを証明せよ。

**翻訳:** 三角形  $ABC$  がある。三角形  $CBA$  の直交中心を  $H$  とする。直線  $AH$  と  $BC$  の交点を  $F$  とする。  $BC$  の中点を  $M$  とする。三角形  $CBA$  の外接円の中心を  $O$  とする。点  $Q$  を円  $(O;A)$  上に、 $AQ$  が  $HQ$  に垂直になるように定義する。点  $K$  を円  $(O;A)$  上に、 $HK$  が  $KQ$  に垂直になるように定義する。点  $O_1$  を三角形  $KHQ$  の外接円の中心と定義する。点  $O_2$  を三角形  $FMK$  の外接円の中心と定義する。  $K;O_1;O_2$  が共線であることを証明せよ。

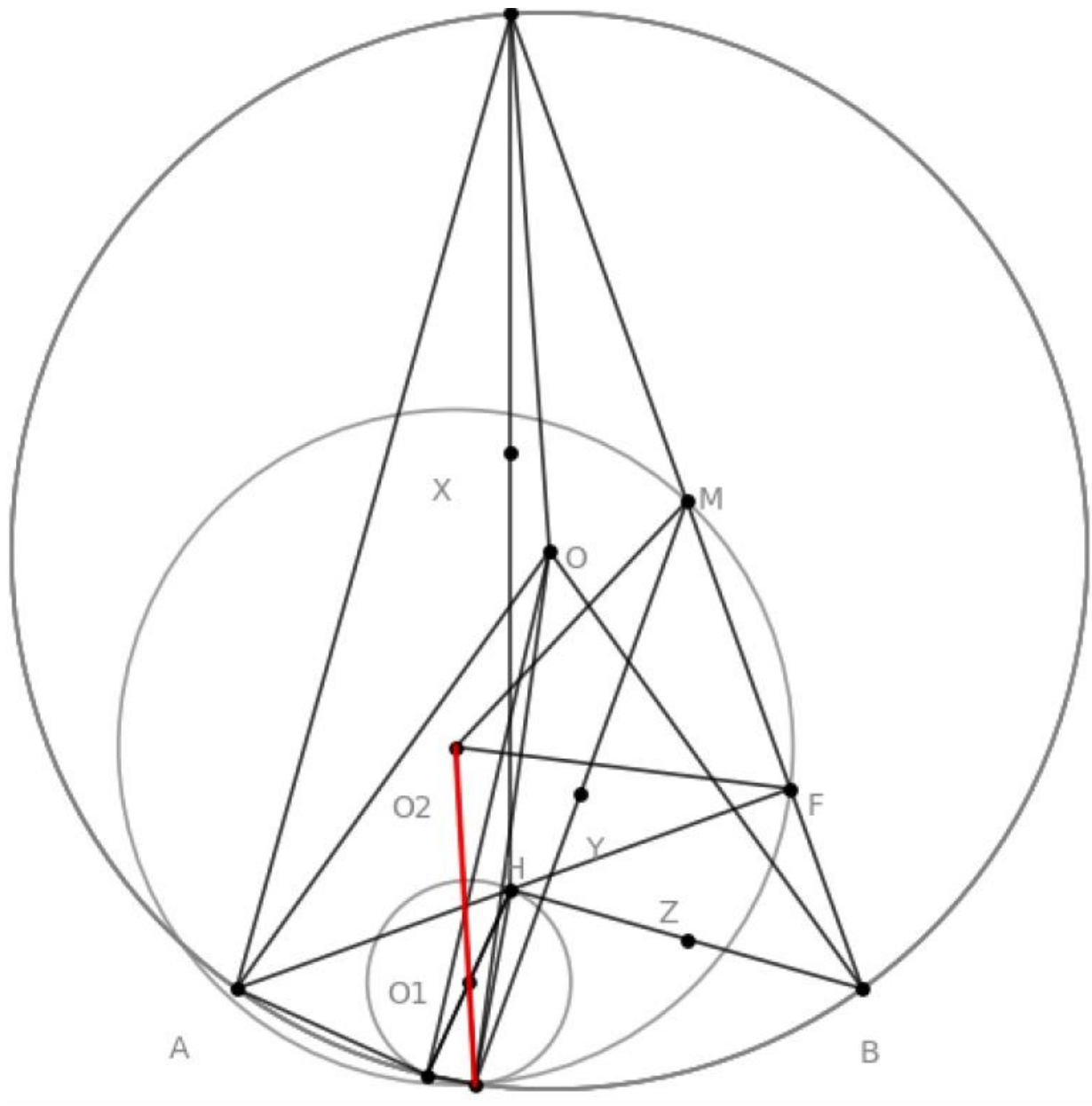


Figure 21: imo 2015 p3

## 証明:

CH の中点に点 X を、KM の中点に点 Y を、BH の中点に点 Z をそれぞれ作ります。

ステップ 1. B、C、M は一直線上にあり、AH は BC に垂直なので、AH は CM に垂直です。

ステップ 2. B、H、Z は共線的で、AC は BH に垂直  $\Rightarrow$  AC は HZ に垂直。

ステップ 3. AC は HZ に垂直で、AH は CM に垂直  $\Rightarrow$   $\angle HAC = \angle (CM, HZ)$  および  $\angle AHZ = \angle MCA$ 。

ステップ 4. A、F、H は一直線上にあり、B、C、F は一直線上にあり、B、C、M は一直線上にあり、B、H、Z は一直線上にあり、 $\angle HAC = \angle (CM, HZ)$  である  $\Rightarrow \angle CAF = \angle HBF$ 。

ステップ 5. A、F、H は一直線上にあり、B、C、F は一直線上にあり、AH は BC に垂直である  $\Rightarrow \angle AFC = \angle BFH$ 。

ステップ 6.  $\angle CAF = \angle HBF$  および  $\angle AFC = \angle BFH \Rightarrow \angle A = \angle C$ 。FB FH ...

...

...

ステップ 105.  $B, C, M$  は共線であり、 $K, M, Y$  は共線であり、 $AH$  は  $BC$  に垂直であり、 $KM$  は  $O_2Y$  に垂直である  $\Rightarrow L(AH, CM) = LKY O_2$ 。

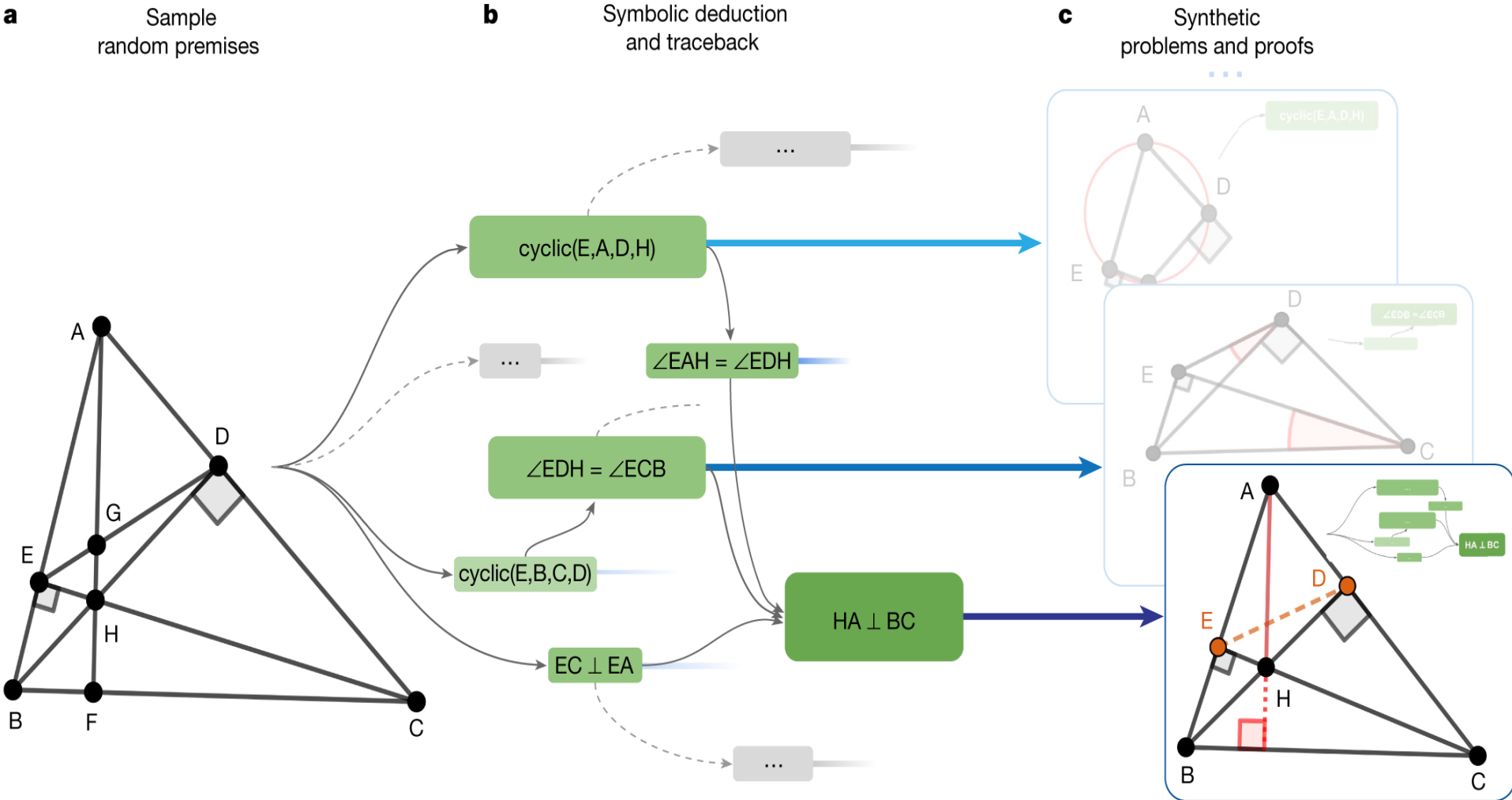
ステップ 106.  $B, C, F$  は共線であり、 $B, C, M$  は共線であり、 $LKFM = LKO_2Y \Rightarrow L(CM, FK) = LY O_2K$ 。

ステップ 107.  $L(AH, CM) = LKY O_2$  および  $L(CM, FK) = LY O_2K \Rightarrow L(AH, FK) = LY KO_2$ 。

ステップ 108.  $K, M, Y$  は共線的であり、 $L(AH, FK) = LY KO_2$ 、 $LFKO_1 = LOMK$ 、 $AH$  は  $BC$  に垂直であり、 $BC$  は  $MO$  に垂直である  $\Rightarrow LFKO_1 = LFKO_2$ 。

ステップ 109.  $LFKO_1 = LFKO_2 \Rightarrow KO_1$  は  $KO_2$  に平行である。ステップ 110.  $KO_1$  は  $KO_2$  に平行である  $\Rightarrow K, O_1, O_2$  は共線的である。

# 図 3: AlphaGeometry 合成データ生成プロセス。



a、まず、ランダムに選択した大規模な定理の前提をサンプリングする。

b、記号推論エンジンを使用して推論の閉包を取得する。これにより、文の有向非循環グラフが返される。

グラフ内の各ノードについて、トレースバックを実行して、そのノードに必要な前提と依存推論の最小セットを見つける。

例えば、右端のノード「 $HA \perp BC$ 」の場合、トレースバックは緑のサブグラフを返す。

c、最小前提と対応するサブグラフが合成問題とその解を構成する。下の例では、ポイント E と D は HA と BC の構築には無関係にもかかわらず証明に参加しているため、言語モデルによって補助構造として学習される。

## 図 5: AlphaGeometry は、翻訳された IMO 2004 P1 よりもより一般的な定理を発見した。

**原問:**  $ABC$  を  $AB = AC$  の三角形とする。直径  $BC$  の円は、辺  $AB$  と  $AC$  をそれぞれ  $M$  と  $N$  で交わる。辺  $BC$  の中点を  $O$  とする。角  $LBAC$  と  $LMON$  の二等分線は  $R$  で交わる。三角形  $BMR$  と  $CNR$  の外接円は、辺  $BC$  上にある共通点を持つことを証明せよ。

**翻訳:**  $ABCO$  を四角形とする。点  $M$  を円  $(O, B)$  と直線  $AB$  の交点として定義する。点  $N$  を円  $(O, B)$  と直線  $AC$  の交点として定義する。  $AR$  が  $LBAC$  の二等分線であり、  $OR$  が  $LMON$  の二等分線である点  $R$  を定義する。三角形  $RBM$  の外心として点  $O_1$  を定義する。三角形  $RCN$  の外心として点  $O_2$  を定義する。円  $(O_1, R)$  と  $(O_2, R)$  の交点を点  $P$  と定義する。  $B, C, P$  が共線であることを証明せよ

**IMO 2004 P1:**

“Let  $ABC$  be an acute-angled triangle with  $AB \neq AC$ . The circle with diameter  $BC$  intersects the sides  $AB$  and  $AC$  at  $M$  and  $N$  respectively. Denote by  $O$  the midpoint of the side  $BC$ . The bisectors of the angles  $\angle BAC$  and  $\angle MON$  intersect at  $R$ . Prove that the circumcircles of the triangles  $BMR$  and  $CNR$  have a common point lying on the side  $BC$ .”

Translate  
↓

**Premise**

$A B C O M N R P$  : Points

$\text{mid\_point}(O, B, C)$  [--]

$\text{same\_line}(B, M, A)$  [00]  $OM=OB$  [01]

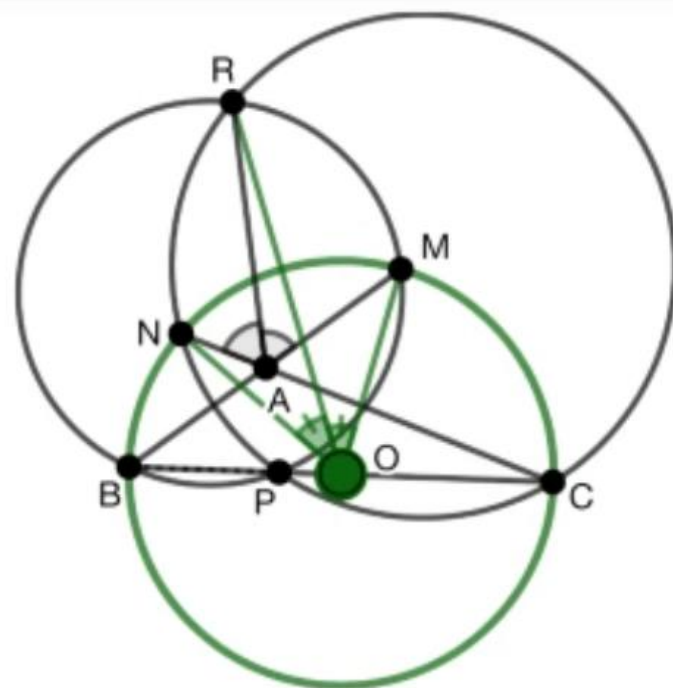
$\text{same\_line}(N, C, A)$  [02]  $ON=OB$  [03]

$\angle BAR = \angle RAC$  [04]  $\angle MOR = \angle RON$  [05]

$\text{circle}(B, M, R, P)$  [06]  $\text{circle}(C, N, R, P)$  [07]

**Goal**

$\text{same\_line}(P, B, C)$



Generalize  
↓



左から上から、IMO 2004 P1 の自然言語による記述、その翻訳文、および AlphaGeometry の解法を示しています。

証明に必要な最小前提を抽出するためのトレースバックアルゴリズムにより、AlphaGeometry は証明に不要な前提を特定しました：

P、B、C が共線であるためには、O は BC の中点である必要はありません。

右、上から、元の定理図；下、一般化された定理図。

O は中点の位置から解放され、P は依然として BC 上に留まっている。元の問題では P が B と C の間に位置する必要があるが、この条件は一般化された定理と解決策では保証されない点に注意。

# AlphaEvolve

**“AlphaEvolve: A coding agent for scientific and algorithmic discovery”**

Alexander Novikov et al. Alpha Evolve teak

[Submitted on 16 Jun 2025]

<https://arxiv.org/abs/2506.13131>

# AlphaEvolve 論文 Abstract

このホワイトペーパーでは、AlphaEvolve という進化型コーディングエージェントを紹介する。

AlphaEvolve は、オープンな科学問題の解決や計算インフラの重要な部分の最適化など、極めて困難なタスクにおいて、最先端の LLM の能力を大幅に向上させる。

AlphaEvolve は、LLM の自律的なパイプラインを調整し、コードに直接変更を加えることでアルゴリズムを改善するタスクを実行します。進化的なアプローチを採用し、1 つまたは複数の評価者から継続的にフィードバックを受けながら、AlphaEvolve はアルゴリズムを反復的に改善し、新たな科学的・実践的な発見につながる可能性があります。

このアプローチの広範な適用可能性を、複数の重要な計算問題への適用を通じて示した。

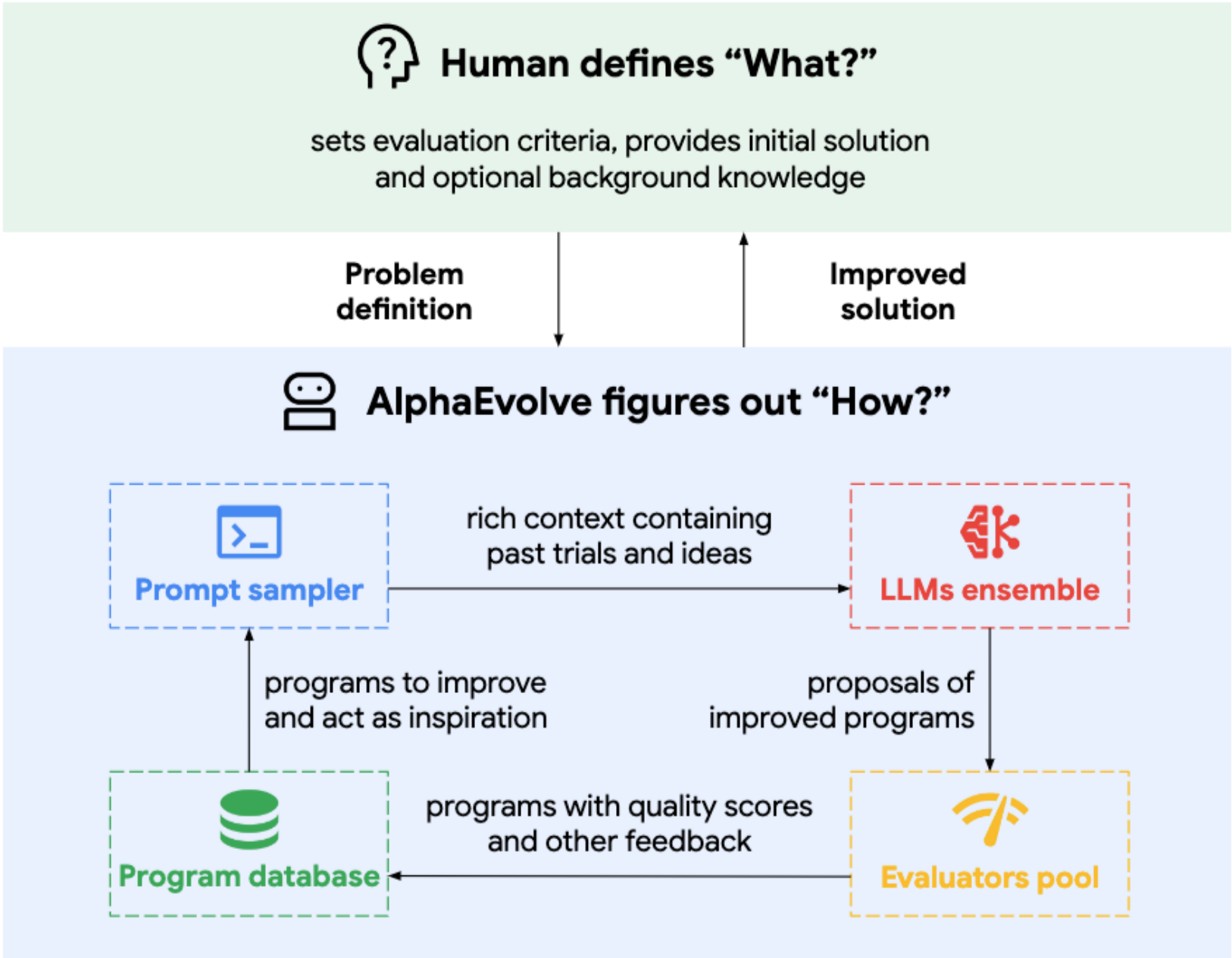
Googleの大型計算スタックの重要なコンポーネントの最適化に適用した場合、AlphaEvolveはデータセンターのスケジューリングアルゴリズムの効率化を実現し、ハードウェアアクセラレーターの回路設計における機能的に同等の簡素化を発見し、AlphaEvolve自体を支えるLLMのトレーニングを加速させた。

さらに、AlphaEvolveは、数学とコンピュータサイエンスの幅広い問題において、最先端の解決策を超える新規で証明可能な正しいアルゴリズムを発見し、従来の自動発見手法の適用範囲を大幅に拡大しました (Romera-Paredes et al., 2023)。

特に、AlphaEvolve は、スカラー乗算を用いて 2 つの複素数値行列を乗算する手順を発見した検索アルゴリズムを開発しました。

これは、この設定において、56 年ぶりにストラッセンのアルゴリズムを改良した最初の成果です。AlphaEvolve や同様のコーディングエージェントは、科学や計算の多くの分野における問題の解決に、大きな影響を与える可能性があると考えています。

# Figure 1 | AlphaEvolve high-level overview.



# Scientist / Engineer

Prompt template and configuration

Choice of existing or custom LLMs

Evaluation code

Initial program with components to evolve



Prompt sampler



LLMs ensemble



Evaluators pool



Program database

## Distributed Controller Loop

```
parent_program, inspirations = database.sample()
prompt = prompt_sampler.build(parent_program, inspirations)
diff = llm.generate(prompt)
child_program = apply_diff(parent_program, diff)
results = evaluator.execute(child_program)
database.add(child_program, results)
```



AlphaEvolve

## 図 2 | AlphaEvolve の発見プロセスの拡大図。

ユーザーは、初期プログラム（進化させるコンポーネントをマークしたもの）、評価コード、およびオプションの設定（セクション 2.1）を提供する。

その後、AlphaEvolve は進化ループを開始する。

プロンプトサンプラーは、プログラムデータベースからプログラムを使用して、豊富なプロンプトを構築する（セクション 2.2）。

これらのプロンプトに基づき、LLMはコードの修正（diff）を生成し、これらが適用されて新しいプログラムが作成される（セクション 2.3）。

これらのプログラムは評価器によって評価され（セクション2.4）、有望なソリューションはプログラムデータベースに登録される（セクション2.5）。

これにより、より良いプログラムを反復的に発見するプロセスが駆動される。

## 図 3 | 教師あり学習の進化に AlphaEvolve を適用した例。

すべてのスニペットは省略されており、省略記号 (...) は省略された行を示している。

(a) ユーザーが提供した、進化の対象となるブロックがマークされたファイル、および現在のバージョンのコードのスコアを評価するために呼び出すことができる特別な評価関数。

(b) LLMs に提供する組み立てられたプロンプトの例。

(c) LLM によって生成された出力の例。

(c) で提案された差分(diff)は、プロンプト (b) に示されている「現在のプログラム」に適用され、その結果変更されたプログラムは、評価者に送信される。

評価者は、(a) の evaluate 関数を呼び出して、新規提案されたプログラムのスコアを取得する。

(a)ユーザーが提供した、  
進化の対象となるブロックがマークされたファイル、

```
"""Image classification experiment in jaxline."""  
import jax  
  
...  
# EVOLVE-BLOCK-END  
  
...  
# EVOLVE-BLOCK-START  
class ConvNet(hk.Module):  
def __init__(self, num_classes): ...  
def __call__(self, inputs, is_training): ...  
def sweep():  
return hyper.zipit([...])  
# EVOLVE-BLOCK-END  
  
...  
def evaluate(eval_inputs) -> dict[str, float]:  
  
...  
return metrics
```

## LLMへのプロンプト (b)

Act as an expert software developer. Your task is to iteratively improve the provided codebase. [...]

### - Prior programs

Previously we found that the following programs performed well on the task at hand:

```
top_1_acc: 0.796; neg_eval_log_loss: 0.230;  
average_score: 0.513
```

```
"""Image classification experiment in jaxline."""
```

```
[...]
```

```
class ConvNet(hk.Module):

    """Network."""

    def __init__(self, num_channels=32,
                 num_output_classes=10):
        super().__init__()

        self._conv1 = hk.Conv2D(num_channels,
                                 kernel_shape=3)
        self._conv2 = hk.Conv2D(num_channels * 2,
                                 kernel_shape=3)
        self._conv3 = hk.Conv2D(num_channels * 4,
                                 kernel_shape=3)
        self._logits_module = hk.Linear(num_output_classes)

    [...]
```

## - Current program

Here is the current program we are trying to improve (you will need to propose a modification to it below).

```
top_1_acc: 0.862; neg_eval_log_loss: 0.387;  
average_score: 0.624
```

```
"""Image classification experiment in jaxline."""
```

```
[...]
```

```
class ConvNet(hk.Module):
```

```
    """Network."""
```

```
    def __init__(self, num_channels=32,  
                 num_output_classes=10):
```

```
        super().__init__()
```

```
self._conv1 = hk.Conv2D(num_channels,  
kernel_shape=3)  
self._block1 = ResNetBlock(num_channels)  
self._block2 = ResNetBlock(num_channels * 2,  
stride=2)  
self._block3 = ResNetBlock(num_channels * 4,  
stride=2)  
self._logits_module = hk.Linear(num_output_classes)  
[...]
```

SEARCH/REPLACE block rules:

[...]

other. For example, if you refer to a new config variable somewhere, you should also propose a change to add that variable.

Example:

[...]

Task

Suggest a new idea to improve the code that is inspired by your expert knowledge of optimization and machine learning.

Describe each change with a SEARCH/REPLACE block.

## (c) LLM によって生成された出力の例。

現在のモデルは、3つのResNetブロックのみからなるシンプルなResNetアーキテクチャを使用しています。

モデルの容量を増やし、正則化を追加することで、そのパフォーマンスを向上させることができます。

これにより、モデルはより複雑な特徴を学習し、未見のデータに対してより優れた汎化能力を発揮するようになります。

また、モデルの正則化をさらに進め、過学習を防ぐため、 옵ティマイザーにウェイトデカヤを追加しています。AdamWは、特にウェイトデカヤを使用する場合、Adamよりも一般的に優れた選択肢です。

<<<<<<< SEARCH

```
self._block1 = ResNetBlock(num_channels)
```

```
self._block2 = ResNetBlock(num_channels * 2, stride=2)
```

```
self._block3 = ResNetBlock(num_channels * 4, stride=2)
```

=====

```
self._block1 = ResNetBlock(num_channels)
```

```
self._block2 = ResNetBlock(num_channels, stride=1)
```

```
self._block3 = ResNetBlock(num_channels * 2, stride=2)
```

```
self._block4 = ResNetBlock(num_channels * 2, stride=1)
```

```
self._block5 = ResNetBlock(num_channels * 4, stride=2)
```

```
self._block6 = ResNetBlock(num_channels * 4, stride=1)
```

>>>>>>> REPLACE

<<<<<<< SEARCH

```
def optimizer(self, learning_rate):
```

```
return optax.adam(learning_rate)
```

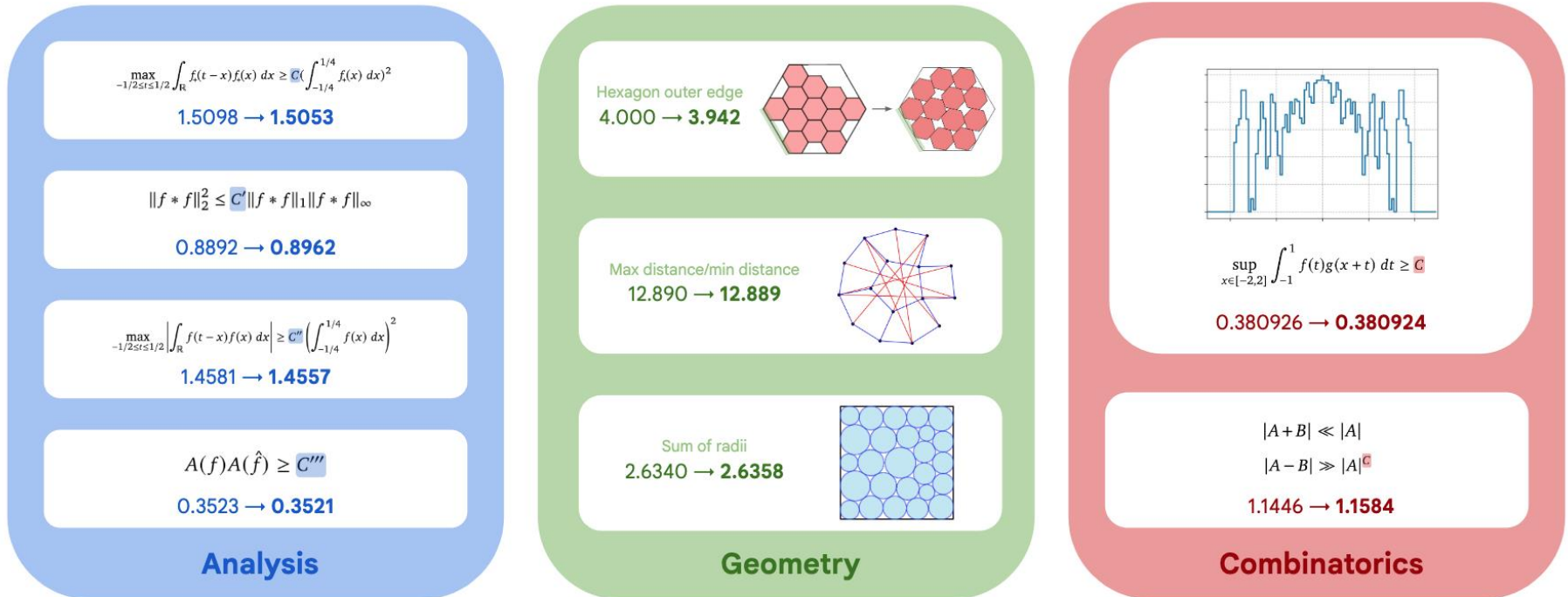
=====

```
def optimizer(self, learning_rate):
```

```
return optax.adamw(learning_rate, weight_decay=1e-4)
```

>>>>>>> REPLACE

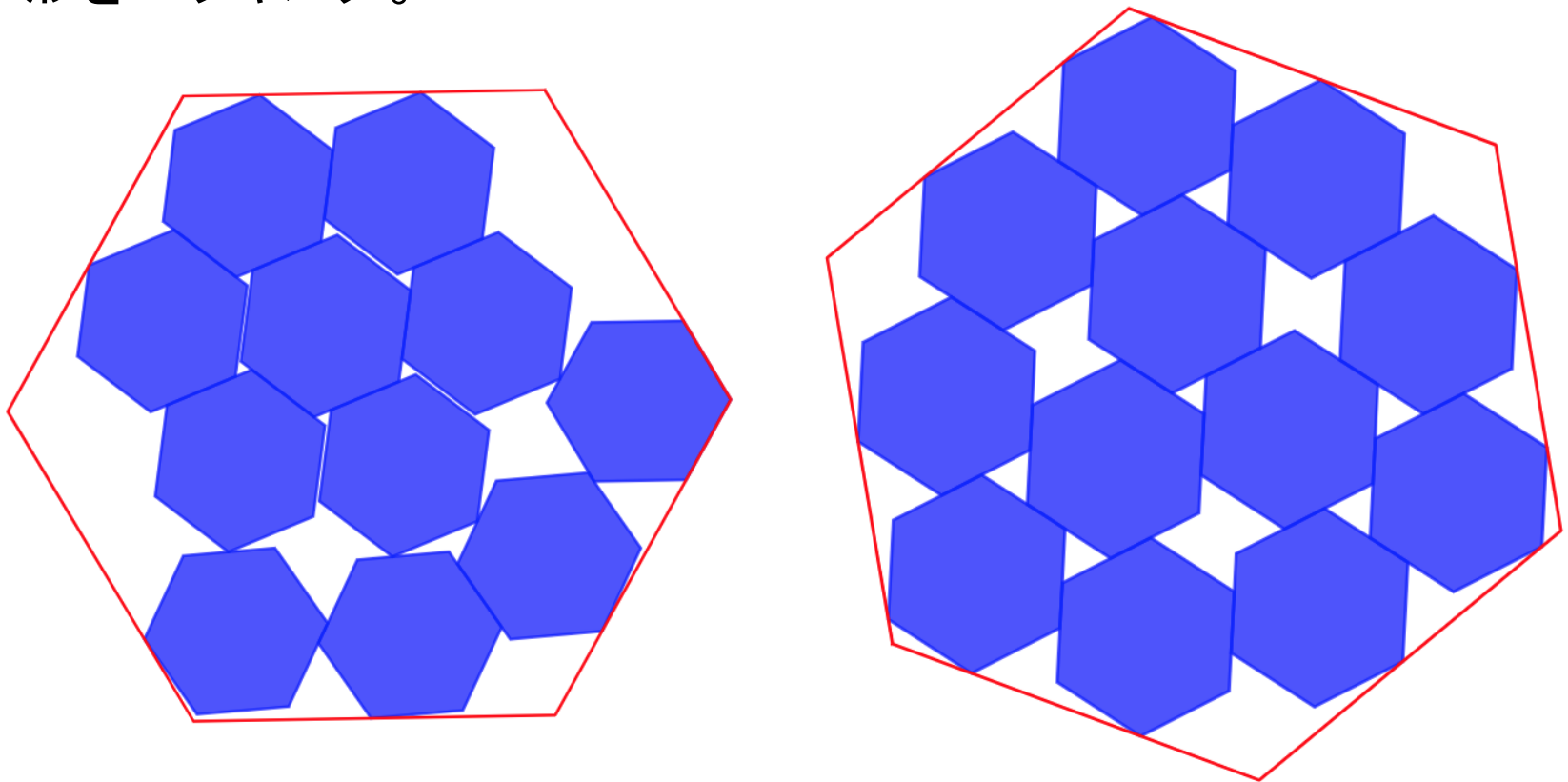
# Figure 5 | Examples of SOTA-breaking mathematical constructions discovered with AlphaEvolve.



The versatility of AlphaEvolve allows us to tackle problems in **analysis** (autocorrelation and uncertainty inequalities), **geometry** (packing and minimum/maximum distance problems) and **combinatorics** (Erdős's minimum overlap problem and sums and differences of finite sets).

# 図 11 | AlphaEvolve によって発見された パッキング問題の構造。

左: 辺の長さが 3.931 の正六角形に 11 個の単位六角形をパッキング。右: 辺の長さが 3.942 の正六角形に 12 個の単位六角形をパッキング。

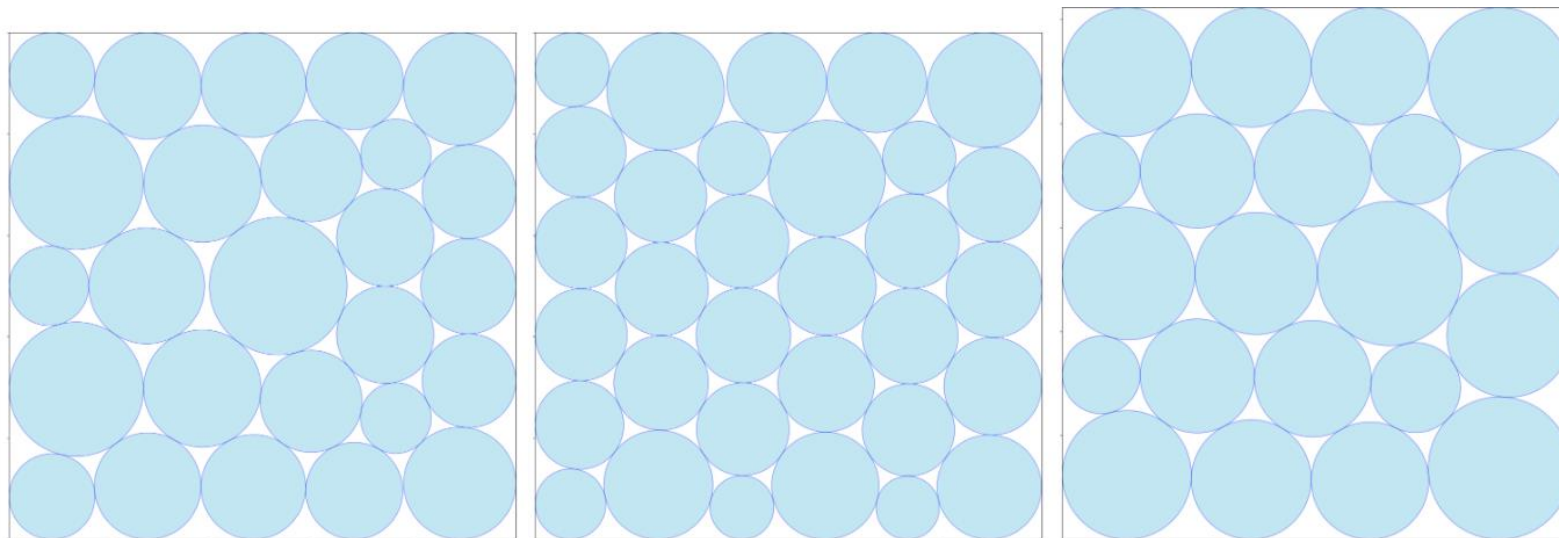


# 図 14 | AlphaEvolve によって発見された、半径の和を最大化する円詰め問題の最良解を改善する新しい構造。

左: 単位正方形に半径の和が 2.635 以上の 26 個の円。

中央: 単位正方形に半径の和が 2.937 以上の 32 個の円。

右: 周長4の矩形内に配置された21個の円(半径の和  $\geq$  2.365)。



## 6. Discussion

AlphaEvolveは、最先端のLLMと自動評価メトリクスを進化フレームワーク内で組み合わせることで、数十年にわたる数学的問題の新たな発見や、高度に最適化された計算スタックの実用的な改善につながる驚異的な力を示しています。

興味深いことに、AlphaEvolveは同じ問題に対して異なるアプローチを可能にします: 解決策を直接探す、それを一から構築する関数を見つける、または検索アルゴリズムを進化させて見つける、などです。AlphaEvolveを異なる方法で適用すると、異なるバイアス(例えば、構築的な関数を見つけることは、高度に対称的なオブジェクトの発見を好む可能性がある[83])が生じ、そのため異なる問題に適している可能性があります。

AlphaEvolveは、その進化プロセスを通じてベースLLMの能力を大幅に向上させる(例えば反復サンプリングと比較して)テスト時計算エージェントとしても捉えることができます。

一方、これは、機械フィードバックがテスト実行時の計算スケールアップを、新たな科学的発見や高価値な実践的最適化が実現される領域まで持続可能にする能力を示す説得力のある事例と言えます。

他方、自然な次のステップは、AlphaEvolveで強化されたベースLLMの性能を、次世代のベースモデルに蒸留することです。これは内在的な価値を有し、おそらく次バージョンのAlphaEvolveの性能向上にもつながるでしょう。

蒸留を超えて、AlphaEvolveが自身のインフラストラクチャおよび(将来のバージョンの)ベースLLMの効率を向上させる実践的な発見を行うことができる点も興味深い。

現在、改善効果は moderate で、AlphaEvolve の次期バージョンを改善するためのフィードバックループは数ヶ月単位です。

しかし、これらの改善により、堅牢な評価関数を備えたより多くの環境(問題)を設定する価値がより広く認識されるようになり、その結果、今後より高価値の実用的な発見がもたらされるものと予想しています。

AlphaEvolve の主な制限は、自動評価器を設計可能な問題のみを扱う点です。

これは数学や計算科学の多くの問題に当てはまりますが、自然科学のような分野では、一部の experiment しかシミュレーションや自動化できない場合があります。

AlphaEvolve は LLM によるアイデアの評価を可能にしていますが、この設定は最適化されていません。しかし、並行する研究ではこれが可能であることが示されている[34]。

自然な次のステップは、LLMが実装段階に移行する前に高レベルなアイデアに対するフィードバックを提供し、実装段階ではコード実行を通じて機械からのフィードバックが得られるように、両方の設定を連携させることだ。

A photograph of a pond with several large, green lily pads floating on the water. Several yellow flowers are in various stages of bloom, some fully open and some as buds. The water is dark and reflects the surrounding foliage. The text 'DeepMindのAI開発のビジョン' is overlaid in white in the center of the image.

# DeepMindのAI開発のビジョン

# DeepMindのAI開発のビジョン

DreamerV3、AlphaGeometry、AlphaEvolveの同時並行的な開発は、DeepMindがAIに対して単一の画一的なアプローチを追求しているのではなく、知能の異なる側面に合わせて明確に区別されたアーキテクチャと学習パラダイムを探求していることを示していると思います。

DreamerV3は生の感覚データと相互作用から学習し、AlphaGeometryは神経的な直感と論理的な推論能力を構造化された論理ドメインのために組み合わせ、AlphaEvolveは広大なアルゴリズム空間を探索するために進化的フレームワーク内でLLMを使用します。

# AIの未来を考える

## それは、[知能]とは何かを考えること

これらは根本的に異なる「知る」方法であり、「問題解決」の方法です。

このことは、異なる種類の問題が根本的に異なるAIソリューションを必要とするかもしれないこと、あるいは将来のより汎用的なAIがこれらの多様な能力を統合する必要があるかもしれないという認識を示唆しています。

AGIにおけるブレークスルーは、これらの現在別々の研究の流れの統合から生まれる可能性があることには注意が必要です。

AIの未来を考えること、それは、[知能]とは何かを考えることが基本だちだと僕は考えています。

## エージェント概念の進化

また、「エージェント」という概念自体も進化し、多様化しています。

DreamerV3とAlphaEvolveはどちらも「エージェント」と表現されるのですが、そのエージェントは異なる形で現れます。

DreamerV3のエージェントは、シミュレートされた世界または現実世界。すなわちエージェントから見れば「外部環境」内の知覚、予測、行動に関するものです。

AlphaEvolveのエージェントは、コードという抽象的な空間における自律的な探索と最適化に関するものです。もちろん、ここでも、AlphaEvolveは、「受動的に応答するだけでなく、目的を持って行動するエージェント的システムとして動作する」と規定されています。

# AlphaEvolveのエージェント概念

このことは、「AIエージェント」の概念を、具現化された、あるいは環境に位置づけられた知能を超えて、情報構造自体を操作し変換するエージェントを含むように拡張することになります。

これにより、純粹にデジタルまたは抽象的なドメインでの創造的または発見的なタスクのために設計された新しいクラスのAIエージェントが生まれる可能性があります。

# 人間の学びも進化する

AlphaEvolveがAIトレーニングで使用されるアルゴリズムを含むアルゴリズムを改善する能力をもち、AlphaGeometryが合成データ生成に依存していることは、AIシステムがデータから学習するだけでなく、学習と発見のプロセスとツール自体を積極的に形成し改善する未来を示唆しているのかもしれない。

そのことは、人間が、単に外部の環境から知覚を通じて直接学ぶだけではなく、ことばの獲得以降は、文字の発明・印刷技術・さまざまなメディアを通じて学習し、図書館や学校という社会的なツールを通じて、「知識」を形成していくのとよく似た過程なのかもしれません、

# 進化とは何か？

より強力なAIは、さらに強力なAIを構築するためのより良い方法につながる可能性があります。DeepMindが、AIシステム自体の「進化」「発展」に、関心を持っていることは、重要なことだと思います。

ただ、AlphaEvolveで、我々人間は、進化とは何かを明確に、原理的一貫性を持ってシステムに提示できていないように思います。何が生き残り、何が淘汰されるのか？ それを考えることは、評価関数を通じて、LLMに丸投げされています。

原理性一貫性というのは、生物学的進化での、何も説明していないかもしれませんが「適者生存」、認識の発展での「エントロピー最大原理」、スマートなプログラムの「最小コード長」等々。をイメージしています。

# AlphaGeometryとAI4Mathの重要性

AlphaGeometryの取り組みは、数学的証明のような高い信頼性と説明可能性を必要とするタスクにとって、形式的検証と記号的推論の重要性を強調しています。

これは、DreamerV3のようなエンドツーエンドの強化学習システムよりもブラックボックス的な性質とは対照的なものです。

AI4Mathにおいて、形式手法の重視に広い関心が集まりつつあることを、僕は歓迎しています。この分野での、いわゆる「ニューロ・シンボリック・アプローチ」は、今後のAI研究の主流の一つになると僕は考えています。

# AIの未来へのビジョン

DeepMindの三つのプロジェクトは、それぞれ異なるアプローチを取りながらも、より自律的で、汎用的で、有能なAIシステムの開発という共通の方向性を示しています。

これらのシステムは複雑なタスクを習得するだけでなく、新しい知識やツールの生成にも貢献し始めていると考えることができるかもしれません。

AIの科学における役割は、データ分析ツールから、仮説生成 (DreamerV3の世界モデル)、形式的証明 (AlphaGeometry)、さらには基本的なアルゴリズム的発見 (AlphaEvolve) における潜在的な協力者へと進化しつつあるということかもしれません。

# 広がる期待

ある人は、こう考えるかもしれません。

「これらの能力が成熟し統合されれば、AIは科学プロセス自体を最適化し始め、新しい方法論を提案し、大規模な仮説の生成とテストを自動化し、科学者が使用する数学的または計算的フレームワークを進化させることさえ可能にするかもしれない。」

また、こういう考えも出てくるかもしれません。

「これは、実験助手としてのAIを超え、科学事業の共同設計者としてのAIへの一歩である。DeepMindがDreamerV3のような高度に汎用的なシステムとAlphaGeometryのような高度に専門化されたシステムに同時に投資していることは、AGIへの道がこれら二つの極の間の洗練された相互作用を含むかもしれないことを示唆している」と。

## 現状の評価

DeepMindのAIの未来についての多面的なアプローチが、こうした未来予測に一つの根拠をを与えてくれる可能性はあります。

ただ、その予測が現実のものになるためには、多面的な探求がそれぞれの分野で確実に深化・発展して、かつ、それが一つの方向に統合されることが必要です。

残念ながら、DreamerV3の期待の「想像力」は長期的見通しに欠け Ender Dragonを倒せず、AlphaGeometryは代数幾何を知らずユークリッド幾何の問題しか解けず、AlphaEvolveの要の評価関数はタスクごとにad hocに設定され、原理的なレベルでの一貫性を持ちません。

# AIの未来と科学の未来

AI技術の現在の到達点はさておき、興味深いのは、DeepMindのさまざまな取り組みから派生しうるこれらのAIの未来像が、基本的には科学の未来像として語られていることだと思います。

こうした「AIの未来と科学の未来」を結びつける議論の射程は、当面、現実的・実践的には大きな問題となるであろう「プログラマーの役割をAIが補完するのか代替するのか」という議論の射程とは確かに異なっています。

ただ、現在プログラマーが行なっている仕事の大部分が、AIによって代替されるという歴史の過程が進行したとしても、そのことをもって、科学の世界で人間の役割がAIによって代替されるだろうと予測することは出来ないと僕は考えています。

# デイドロのオーム

昔の話をしよう (3)





## ドウニ・ディドロ

(Denis Diderot、1713年  
10月5日 - 1784年7月31  
日)

フランスの哲学者、美術批評家、作家。主に美学、芸術の研究で知られる。

18世紀の啓蒙思想時代にあつて、ジャン・ル・ロン・ダランベールとともに百科全書を編纂した、いわゆる百科全書派の中心人物であり、多様な哲学者と交流した。

-- wiki

もし、すべてに答えるオウムがいたなら、  
私は迷わず、それは思考する存在だと言うだろう...

s'il se trouvait un perroquet qui répondît à tout, je  
prononcerais sans balancer que c'est un être  
pensant ...

この文章は、ドゥニ・デイドロの最初の個人著作であり、1746年に匿名で出版された『哲学思想』(Pensées philosophiques)に含まれています。この文章は『哲学思想』の「Pensée XX」(第20思想)に登場します。

[https://fr.wikisource.org/wiki/Pens%C3%A9es\\_philosophiques/Pens%C3%A9es](https://fr.wikisource.org/wiki/Pens%C3%A9es_philosophiques/Pens%C3%A9es)

actes extérieurs et la pensée, il n'y a point de liaison essentielle, me dit-elle ; il est possible que ton antagoniste ne pense non plus que sa montre : fallait-il prendre pour un être pensant le premier animal à qui l'on apprit à parler ? Qui t'a révélé que tous les hommes ne sont pas autant de perroquets instruits à ton insu ?... » « Cette comparaison est tout au plus ingénieuse, me répliqua-t-il ; ce n'est pas sur le mouvement et les sons, c'est sur le fil des idées, la conséquence qui règne entre les propositions et la liaison des raisonnements, qu'il faut juger qu'un être pense : s'il se trouvait un perroquet qui répondît à tout, je prononcerais sans balancer que c'est un être pensant... Mais qu'a de commun cette question avec l'existence de Dieu ? quand vous m'aurez démontré que l'homme en qui j'aperçois le plus d'esprit n'est peut-être qu'un automate, en serai-je mieux disposé à reconnaître une intelligence dans la nature ?... » C'est mon affaire, repris-je : convenez cependant qu'il y aurait de la folie à refuser à vos semblables la faculté de penser. « Sans doute ; mais que s'ensuit-il de là ?... » Il s'ensuit que si l'univers, que dis-je l'univers ! que si l'aile d'un papillon m'offre des traces mille fois plus distinctes d'une intelligence que vous n'avez d'indices que votre semblable est doué de la faculté de penser, il serait mille fois plus fou de nier qu'il existe un Dieu que de nier que votre semblable pense. Or, que cela soit ainsi, c'est à vos lumières, c'est à votre conscience que

## この引用の文脈

『哲学思想』の第20思想において、この引用はデイドロと無神論者との間の対話の中で、無神論者の発言として登場します。

無神論者は、思考する存在であることの確信を論じる中で、思考は表面的な行動や音(デカルトが思考能力を否定した動物もこれらを生成する)によって判断されるのではなく、「**観念の連鎖、命題間の論理的帰結、そして推論の結合**」によって判断されると主張します。

この文脈で、無神論者は「あらゆる質問に答えるオウム」という仮説的な例を挙げ、真の思考が論理的かつ首尾一貫した推論によって証明されることを強調しています。

私は無神論者に対して学問の精緻な論理を駆使して反論を試みたが、無駄だった。彼はその論理の弱点を突いて、かなり説得力のある反論を返してきた。

「多くの無意味な真実が反論の余地なく示されているが、神の存在、道徳的な善と悪の現実、魂の不滅は、私にとっては依然として問題だ。何ということだ！ 三角形の三つの角が二つの直角に等しいことを確信することよりも、これらの問題について理解することが、私にとってそれほど重要ではないということか？」

巧みな弁論家として、彼はこの考えの苦味をすべて私に飲み込ませた。私は、最初の成功に酔っている男には奇妙に聞こえるであろう質問で反撃した。

「あなたは思考する存在か？」と私は尋ねた。「それを疑うことができるか？」と彼は満足そうに答えた。「なぜできないのか？ 私がそれを確信する根拠は何だ？ ...音や動きか？ ...しかし、哲学者は、思考能力を奪われた動物にも同じものを見出す。なぜ私は、デカルトがアリに否定するものをあなたに認めるべきなのか？ あなたは、私を納得させるような行動を外に表している。私は、あなたが確かに思考していると断言したくなる。しかし、理性は私の判断を保留する。」

「外的な行為と思考の間には本質的な関連性はない、と彼女は言った。あなたの敵対者は、自分の時計と同じように思考していないかもしれない。最初に言葉を教わった動物を、思考する存在とみなすべきだろうか？すべての人が、あなたの知らないうちに訓練されたオウムではないことを、誰があなたに教えたのか？

「その比較はせいぜい巧妙なだけだ、と彼は答えた。「思考するかどうかを判断するのは、動きや音ではなく、考えの流れ、命題間の論理的関連性、推論の連鎖だ。もし、すべてに答えるオウムがいたなら、私は迷わず、それは思考する存在だと言うだろう... しかし、この問題と神の存在とは何の関係があるのだろうか？私が最も知性があると思う人間が、実は単なる自動機械に過ぎないことをあなたが証明したとしても、私は自然の中に知性を認める気にはなれないだろう... それは私の問題だ、と私は続けた。

しかし、あなたの同類に思考能力がないと否定するのは愚かなことだと認めなさい。確かにそうだが、そこから何が導き出されるのか？「それは、宇宙、いや、宇宙どころか、蝶の羽が、あなたが同類が思考能力を持っていることを示す証拠よりも千倍も明確な知性の痕跡を見せてくれた場合、神の存在を否定するよりも、同類が思考能力を持っていることを否定するほうが千倍も狂気だということだ。

さて、それが事実であるかどうかは、あなたの知性、あなたの良心に問う。あなたは、どんな人間の思考、行動、行動にも、昆虫のメカニズムよりも、より多くの知性、秩序、賢明さ、一貫性を見たことがあるか？神は、ニュートンの偉大な著作に表れている思考能力と同じように、ダニの目に明確に刻印されているのではないのか？何だ！形成された世界は、説明された世界よりも知性を証明していないというのか？なんと大胆な主張だ！...

「しかし、あなたは反論するだろう、私は自分自身で考えることができるから、他者に思考能力があることをより容易に認める、と。その通りだ、私はその前提を否定しない。しかし、私の証拠があなたの証拠よりも優れていることで、その差は補われているのではないだろうか？最初の存在の知性は、その作品によって、哲学者の著作によって示される思考能力よりも、自然の中でよりよく証明されているのではないだろうか？私があなたに反対したのは、蝶の羽、シラミの眼だけだったことを考えてみてくれ。

私は宇宙の重さであなたを押しつぶすこともできたのだ。私が大いに間違っているか、この証拠は学校で教えられている最も優れた証拠に匹敵する価値がある。私はこの論理と、同様の単純な他のいくつかの論理に基づいて、神の存在を認めている。乾燥した形而上学的な考えの織り成す布ではなく、真実を明らかにするよりもむしろ嘘のように見せるのに適したものでなく。

# 外部からの観察可能性と 内部の論理的整合性の関係

この引用が提示する文脈は、単なる知性の定義を超え、思考の本質、特に外部からの観察可能性と内部の論理的整合性の関係という、哲学における根源的な問題を提起しています。

これは、現代の人工知能(AI)における「ブラックボックス問題」(内部プロセスを理解することと外部の振る舞いを評価することの対比)の中心的な議論に先行するものであり、単なる計算能力と真の知性の区別という微妙ながらも重要なニュアンスを含んでいます。

デイドロは、この対話を通じて、外部から観察可能な行動(応答能力)が、知性や思考の存在を判断するための主要な基準となりうることを示唆しました。

## 知性を「パフォーマンス」として捉える

このデイドロの引用は、知性を、単に知識を保持する能力としてではなく、質問に対して適切かつ論理的に応答する「パフォーマンス」として捉える視点を示しています。

この思考実験は、外部から観察可能な行動が、知性や思考の存在を判断するための主要な基準となりうるという考えを提示しており、これはアラン・チューリングが1950年に提唱した「模倣ゲーム」(チューリングテスト)の概念に驚くほど類似しています。

どちらも、機械が人間のような知性を持つかどうかを判断する基準として、言語的応答能力を重視する点で共通しています。

# デイドロの思想の射程

この時代に、このような思考実験が行われたことは、18世紀の啓蒙思想家たちが人間の知性の本質を深く探求し、その限界と可能性を問い直していたことを示しています。

先のオウムの例えの「第20思想」だけでなくデイドロの『哲学思想』には、現代の我々にとっても、刺激的な問題提起がたくさん含まれています。

そのいくつかを見ておきましょう。

## XVIII.

無神論に大きな打撃を与えたのは、形而上学者の手によるものではなかった。

マレブランシュやデカルトの崇高な思索は、マルピギの観察ほど物質主義を揺るがすものではなかった。

この危険な仮説が今日では揺らいでいるのは、実験物理学のおかげである。ニュートン、ムシェンブローク、ハートゾイカー、ニウエンティットの著作においてのみ、至高の知性を持つ存在の存在を証明する十分な証拠が発見された。

これらの偉大な人物たちの研究のおかげで、世界はもはや神ではなく、車輪、ロープ、滑車、バネ、重りを持つ機械となった。

## XIX.

存在論の微妙な点は、せいぜい懐疑論者を生み出したに過ぎない。真の神の存在を信じる者は、自然の知識によってのみその確信を得ることができた。

胚の発見だけで、無神論の最も強力な反論の一つが払拭された。運動が物質の本質的な性質であるか、あるいは偶発的な性質であるかにかかわらず、私は今や、その効果は発達に終結すると確信している。

すべての観察が、腐敗だけでは組織化されたものは何も生み出さないことを私に示している。

私は、最も卑しい昆虫のメカニズムも、人間のメカニズムと同じくらい素晴らしいものであることを認めることができる。

そして、分子内の運動が一方を生み出すことができるならば、他方も生み出した可能性が高いと結論付けることを恐れない。200年前に、ある無神論者が、いつか、熱せられた肉の塊から無数の昆虫が孵化するのを見るように、完全に形作られた人間が地球の深部から出現するかもしれない、と主張したとしたら、その無神論者に、形而上学者はどのような答えを返しただろうか[4]。

# デイドローの言語観

デイドローは、言語を人間の意識の進化と洗練にとって不可欠な要素であると見なしました。彼の見解では、言語能力の発展と思考の洗練の間には密接な関連性があり、言語が観念を組織化し、過去の感覚を想起させ、反省的思考の始まりを告げる上で決定的な役割を果たすと信じられていました。

デイドローはまた、ジョン・ロックと同様に、単純な観念でさえ個々の話し手によって表現方法が極めて多様であるという「表現の個別性」を重視しました。これは、話し手の現実に対する特定の把握が、彼自身の心の自由な活動によって修正される結果であると考えられました。この考え方は、言語が単一の客観的現実を反映するだけでなく、主観的な経験と創造性によって形作られるという、より複雑な理解を示唆しています。

## 『聾啞者に関する手紙』

彼の主要な著作である『聾啞者に関する手紙』(Lettre sur les sourds et muets, 1751) において、デイドロは芸術的言語の問題を探求し、聾啞者のコミュニケーション方法を研究することで言語の起源に光を当てようとしていました。

この作品では、原始言語が感情的かつ音楽的であり、その後効率性のために慣習的に規制された言語へと進化したという、ルソーと類似した見解が示唆されています。さらに、『百科全書』(Encyclopédie) の編集者として、彼は知識の正確な伝達のために言語の精密さを追求し、その正確性が最も重要であると考えました。

# 言葉の力

デイドロの言語観は、単なる記述に留まらず、言語が人間の精神活動(思考、意識、感情)を形成し、文明化のプロセスを推進する動的な力であるという、啓蒙思想の核心的な信念を反映しています。

言語が観念を組織化し、反省的思考を可能にすることは、人間の認知構造そのものに不可欠な要素であることを示します。

# 人間の発展と社会の進歩における言語の役割

また、「表現の個別性」への注目は、言語が固定された普遍的なコードではなく、各話者の独自の現実把握によって形成されることを示唆し、個人の心と表現の間の動的な相互作用を強調します。言語を「文明化のプロセス」の推進力と位置づけることは、**デイドロ**が人間の発展と社会の進歩における言語の役割を深く理解していたことを示します。

感情的な言語から合理的な言語への進化という彼の見解は、人類がより原始的な状態からより高度な知的洗練へと移行するという、より広範な哲学的物語と一致します。**『百科全書』**における**「言語の精密さ」への懸念**は、知識を正確に伝達し、ひいては「思考の一般的な様式を変える」という言語の力を強調しています。



# Performance と Competence

昔の話をしよう (4)



