第一回
フィジカルAIハンズオン資料

平塚謙良(京都大学 工学部 B4)

アジェンダ

1. イントロダクション&ゴール設定
2. 模倣学習ミニレクチャー
3. データ収集
4. 学習&ACT解説
5. 評価と発展

アジェンダ

1. イントロダクション&ゴール設定
2. 模倣学習ミニレクチャー
3. データ収集
4. 学習&ACT解説
5. 評価と発展

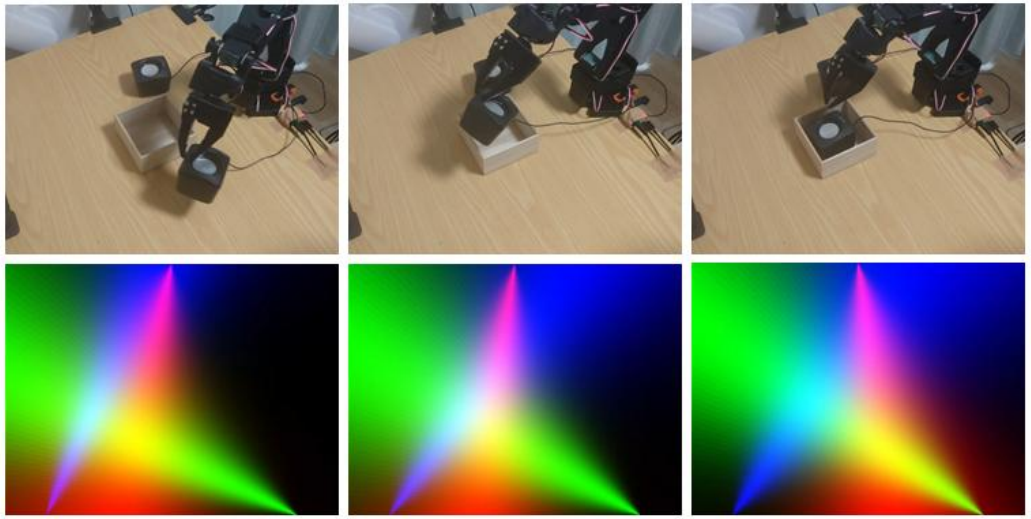
イントロダクション

自己紹介

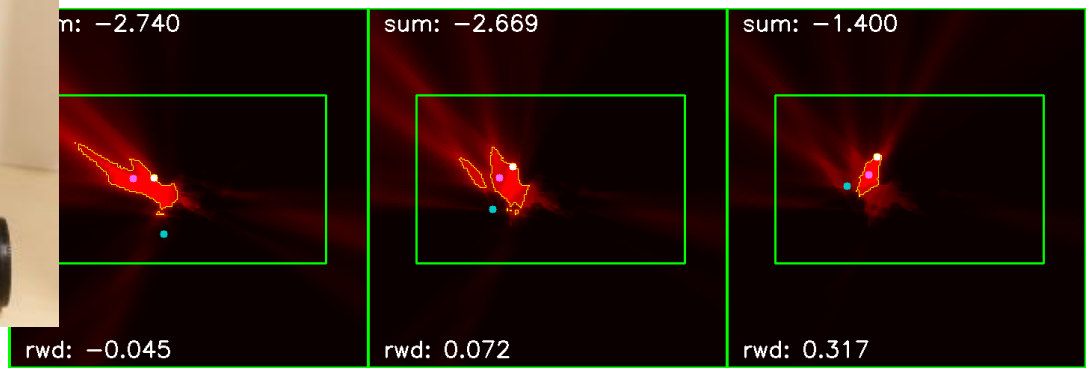
氏名：平塚謙良（Kaneyoshi Hiratsuka）

研究領域：
- 強化学習
- 模倣学習
- ロボット聴覚

インターン等：
- 松尾研究所
  →世界モデルやVLAの研究開発
- AIRoAコンペ
  →VLAの強化学習
- Sony SSUP



音環境情報に基づくピックアンドプレースの模倣学習



世界モデルベースの深層強化学習による音源追跡の検討

アンケート

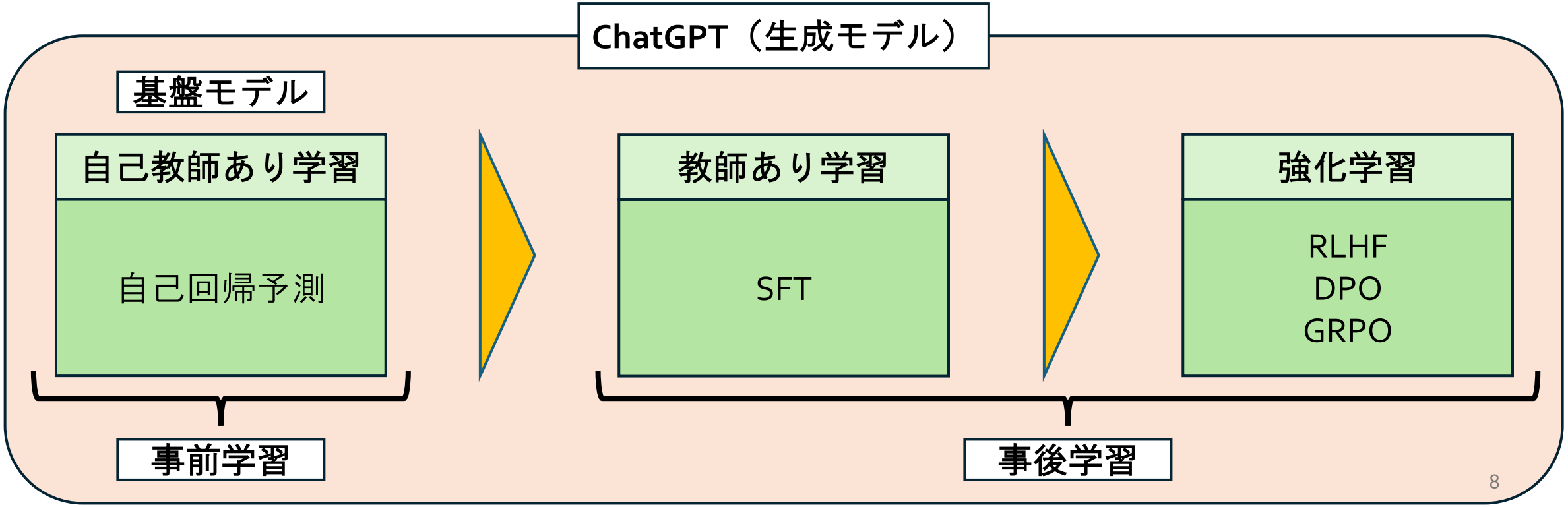
- ロボットを動かしたことがある？

- 機械学習の経験がある？

フィジカルAIについて

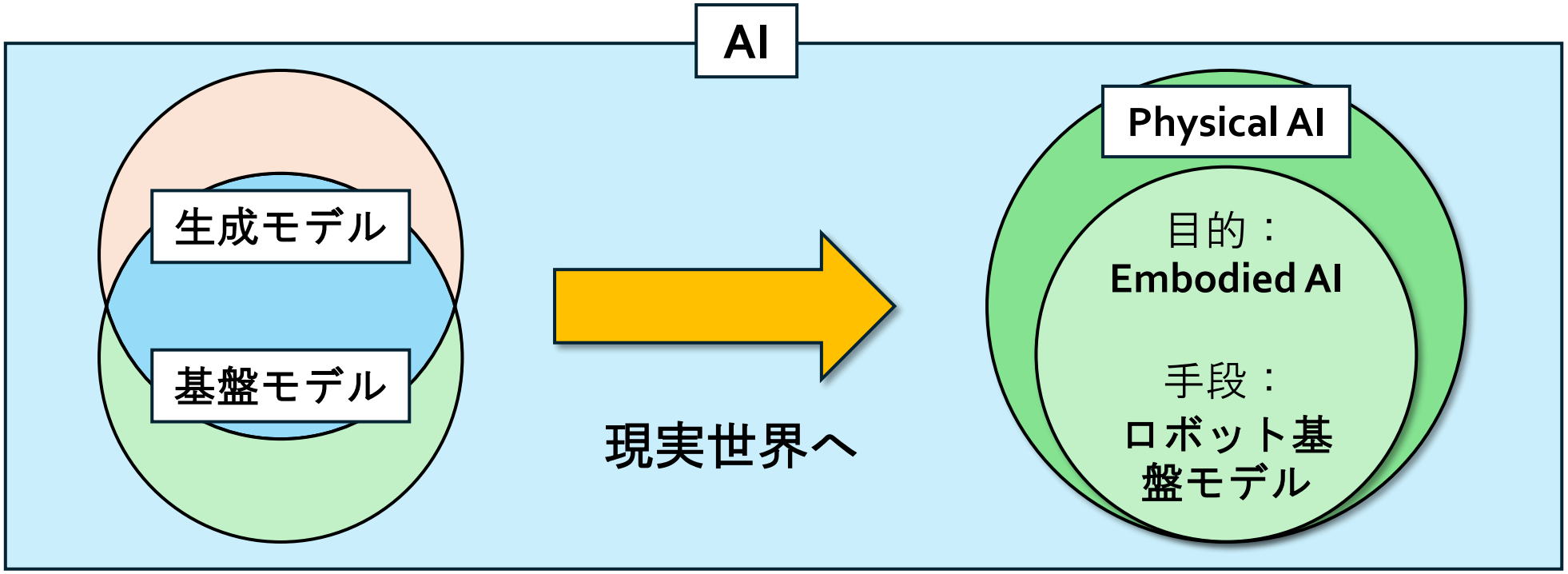
ChatGPTにみるAIの主要概念

- ChatGPTはテキストを生成する**生成モデル・生成AI**の一種
- **事前学習**：
  大規模データを使って**自己教師あり学習**し，幅広く知識を獲得 (**基盤モデル**)
- **事後学習**：基盤モデルを**教師あり学習**や**強化学習**で使いやすく調整



概念の整理 (1/2)

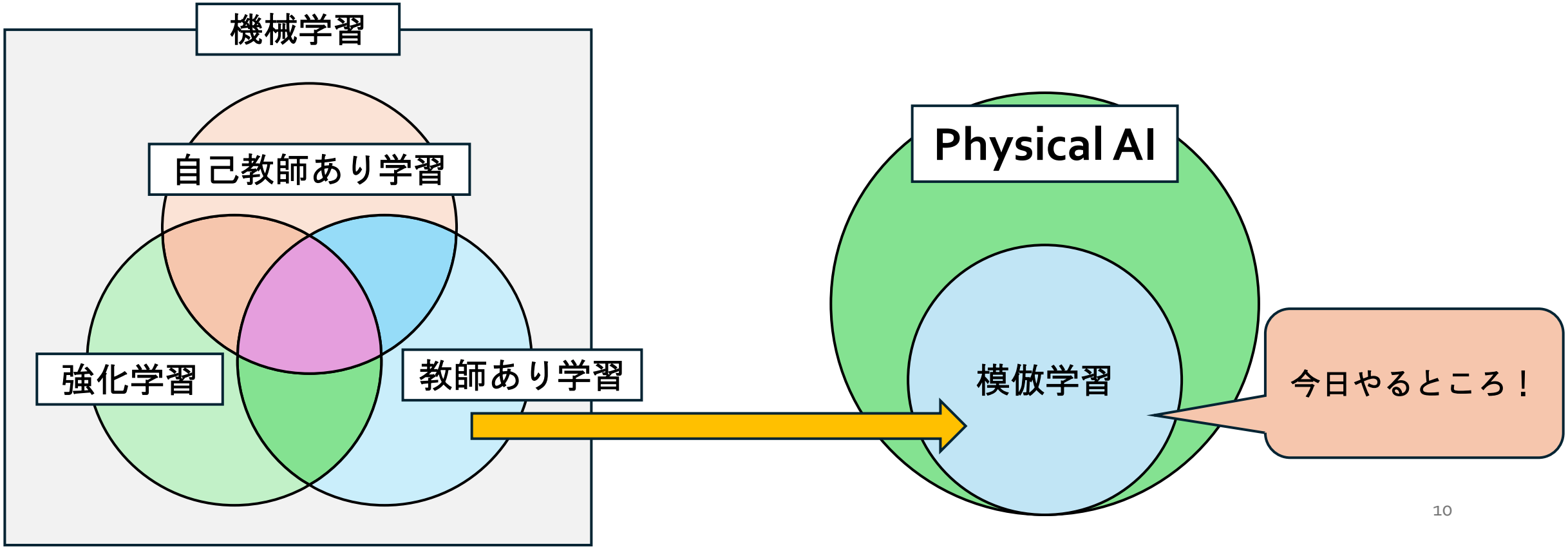
- **生成モデル / AI**：新しいコンテンツを生成するAI
- **基盤モデル**：大規模なデータで事前学習され，特定機能を持つモデルの基盤となる
- **Embodied AI**：身体性を持って環境に作用できるAI. その手段の１つとしてのロボット基盤モデル.
- **Physical AI**：**現実**世界で知的・自律的に振る舞うシステム全般



概念の整理 (2/2)

機械学習の3つのフレームワーク

- **教師あり学習**：正解ラベル付きデータから学習（模倣学習など）
- **自己教師あり学習**：ラベルなしデータから，疑似ラベルを生成して学習
- **強化学習**：試行錯誤と報酬を通じて学習



なぜ今Physical AIなのか

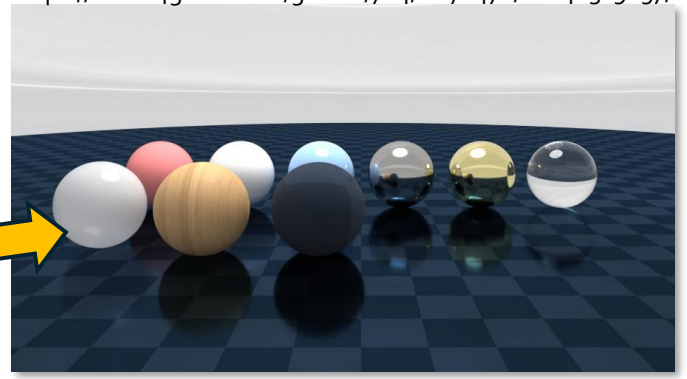
技術的成熟

アーキテクチャや学習法の進化：
- 大規模，高精度のモデルの登場

ハードウェアの進化：
- 高性能なGPU
- 安価なセンサーやアクチュエータ

シミュレータの進化：
- 高速に動作
- 物理的に忠実

https://www.4gamer.net/games/784/G078478/20240319057/


https://genesis-world.readthedocs.io/en/latest/user_guide/getting_started/visualization.html


https://www.figure.ai/news/helix-loads-the-dishwasher

社会的要求

- 労働力不足
- 社会インフラの老朽化

ゴール設定

本日のゴール

ゴール

みんなで集めたデータで，お菓子を掴むAIロボットを学習させる

https://mognavi.jp/product/258119

ハンズオンを通じて体験できること

- 模倣学習の一連のプロセス（データ収集→学習→評価）
- 最新のTransformerベースの手法（ACT）の威力
- フィジカルAIの面白さと難しさ

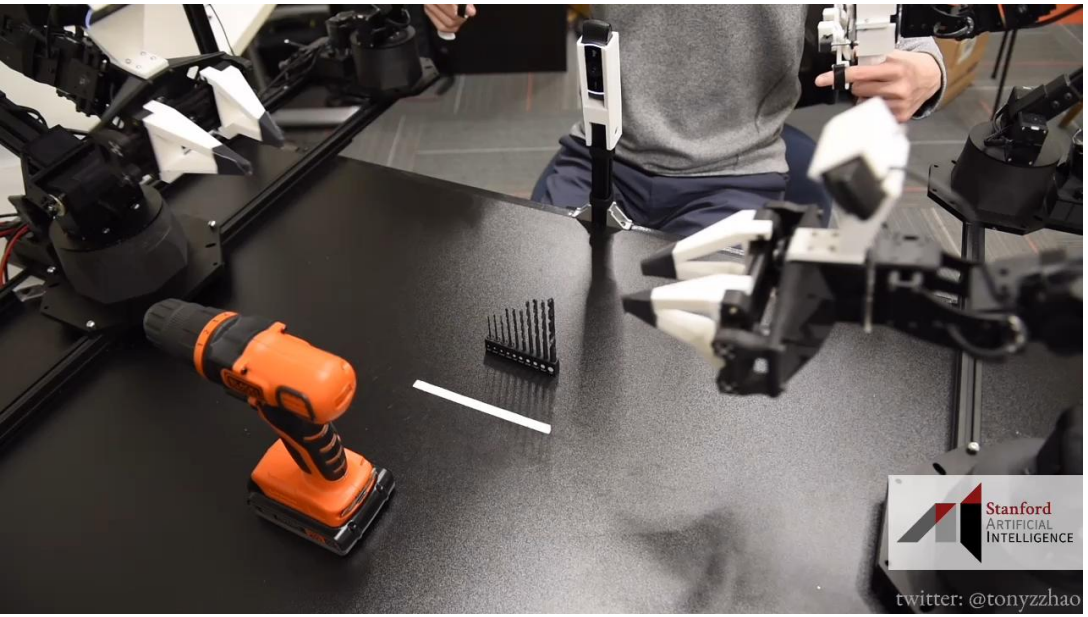
完成形のイメージ

アジェンダ

1. イントロダクション&ゴール設定
2. **模倣学習ミニレクチャー**
3. データ収集
4. 学習&ACT解説
5. 評価と発展

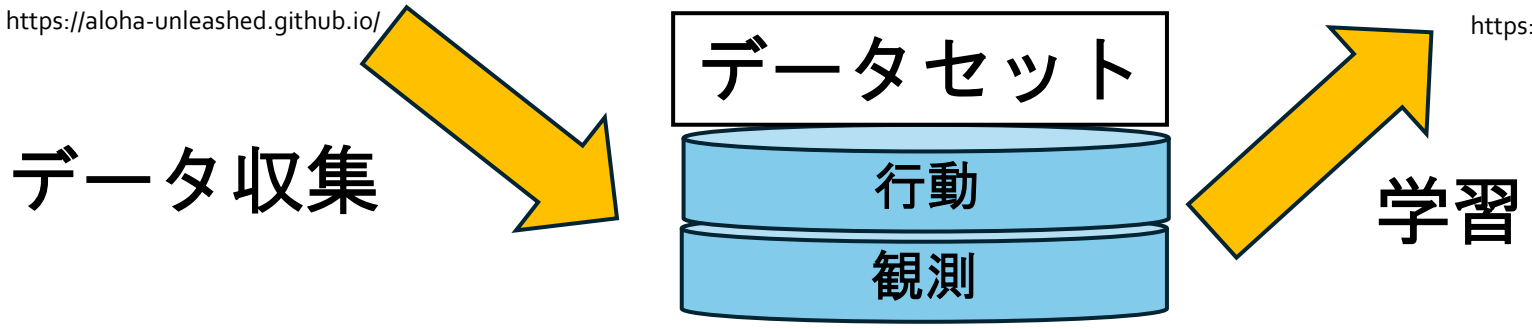
模倣学習：どんなことができる？

- 人間等のデモンストレーションから学習して動作を模倣
- マニピュレータ（腕型ロボット）の制御や精密な動作が得意



https://aloha-unleashed.github.io/

https://aloha-unleashed.github.io/



模倣学習：概要

基本手法

- ロボット操作時の観測と行動をペアとして，データセットを収集
- 観測から行動を予測するPolicyを，教師あり学習で学習

※Policy (方策)：ロボットを動かすための行動を出力する機械学習モデルの事

メリット

- データセットを集めれば学習可能
- 評価関数や報酬関数の設計が不要

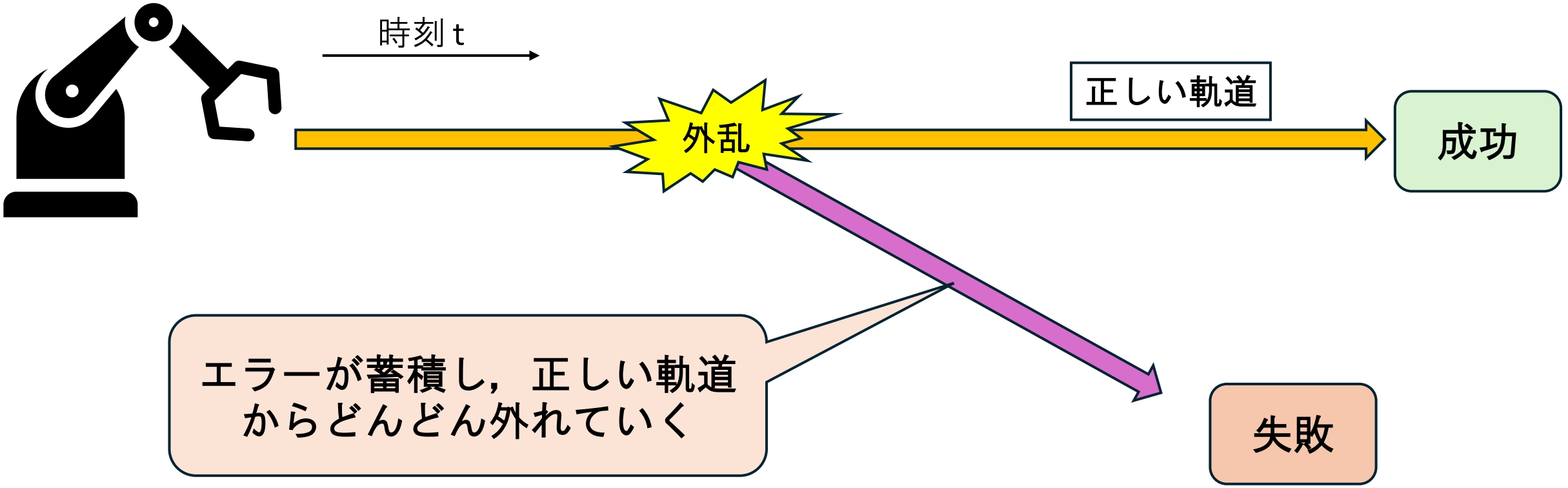
デメリット

- 外乱に対して脆弱
  →**OOD問題**が顕著

模倣学習：OOD (Out of Distribution) 問題とは？

課題

データセットにない未知の状況に陥ると，学習していないため，
どう動けばよいのか分からなくなる．



模倣学習：最近のトレンド

生成AI技術を応用して性能向上！

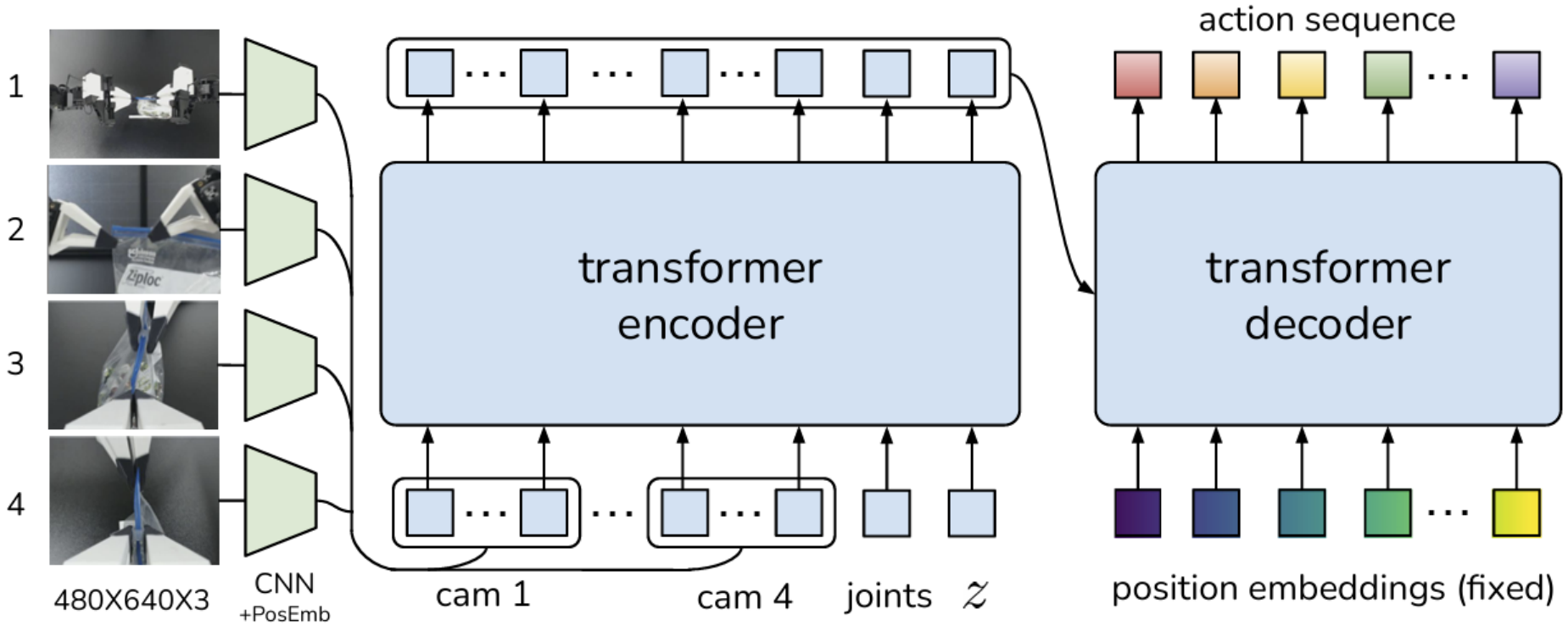
大規模言語モデル (LLM) に使われているTransformerを応用
→**Action Chunking with Transformer (ACT)**

今回使用するモデル！

画像生成モデルに使われている拡散モデル (Diffusion Model) を応用
→**Diffusion Policy**

模倣学習：Action Chunking with Transformers (ACT)

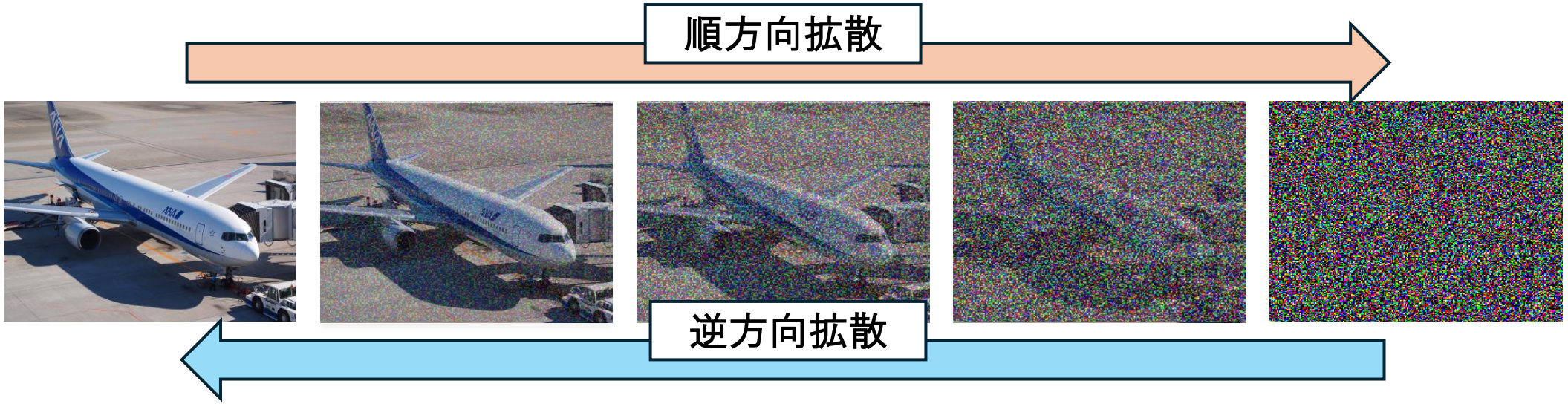
- 畳み込みニューラルネットワーク (ResNet-18) を用いて画像特徴量を抽出
- 注意機構 (Transformer) を利用し，入力の重要な要素に注目
- 一度の生成で複数step分の行動を出力し，高い時間的一貫性



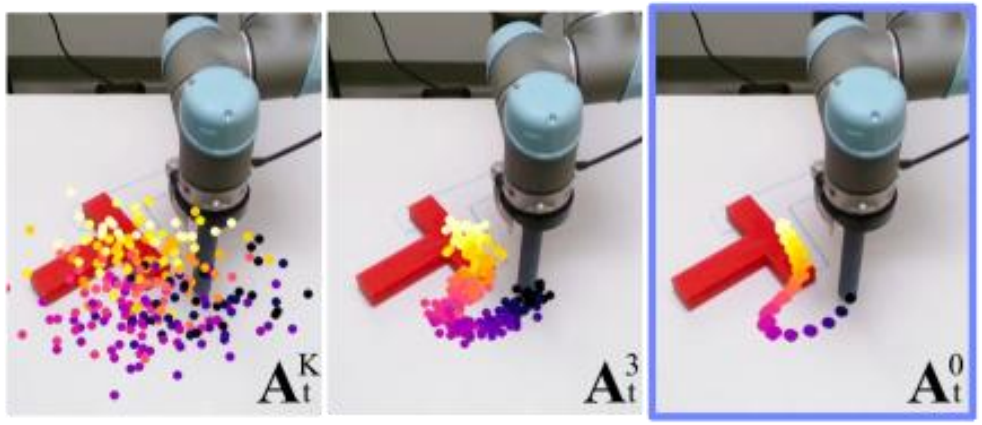
後ほど詳しく説明

模倣学習：Diffusion Policy

拡散モデル：画像生成で成功．ノイズ除去を学習．



Diffusion Policy：現在の観測を条件づけて，ノイズから行動を生成



多峰性分布（複数の正解が存在するタスク）をモデル化可能

アジェンダ

1. イントロダクション&ゴール設定
2. 模倣学習ミニレクチャー
3. **データ収集**
4. 学習&ACT解説
5. 評価と発展

ハンズオン環境について

ハンズオン環境の全体像

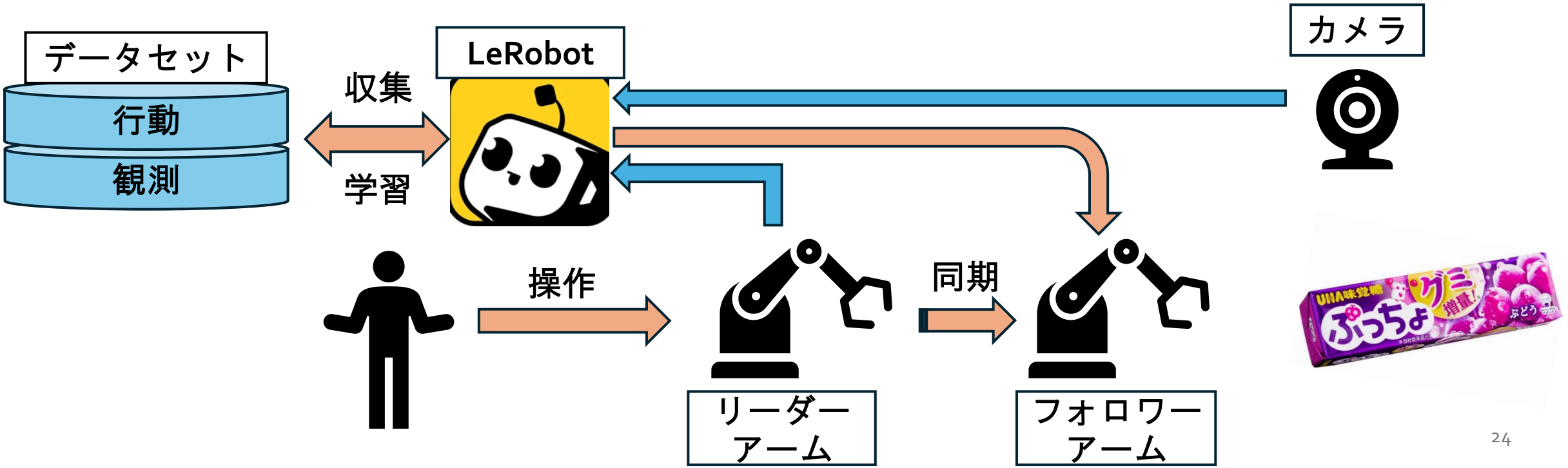
- **ハードウェア：SO-100, SO-101**

オープンソース化されている安価な5軸マニピュレータ

- **ソフトウェア：LeRobot**

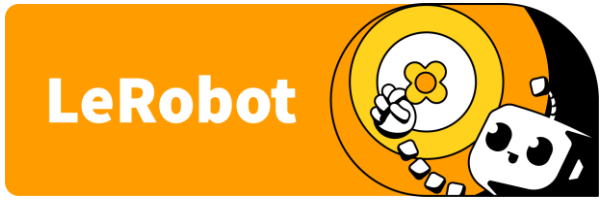
ロボット学習においてデファクトスタンダードとなりつつあるオープンソースライブラリ

https://analyticsindiamag.com/ai-news-updates/hugging-face-launches-open-source-so-101-robot-arm-for-ai-builders/



LeRobotライブラリ紹介

LeRobotとは



Hugging Face と連携したPhysical AIの研究開発プラットフォーム

課題

- データセットごとにフォーマットがバラバラ
- ロボットが高価であるなど，新規参入のハードルが高い

LeRobotによる解決策

- Hugging Face連携によりデータセットやモデル重みの共有が用意に
- LeRobot Dataset形式でデータセット規格を統一
- 安価なオープンソースアームとその制御コードを公開
- Diffusion PolicyやACTなど，メジャーな模倣学習アルゴリズムに対応

→ **新規参入ハードルが低下**

ハードウェア紹介

カメラ

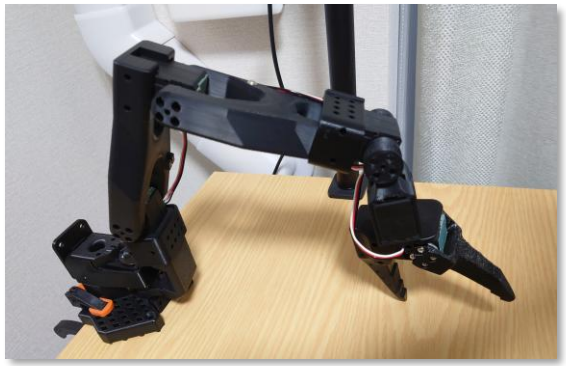
- UGREEN ウェブカメラ
- REALSENSE D455

ロボット

- SO-100
- SO-101

その他

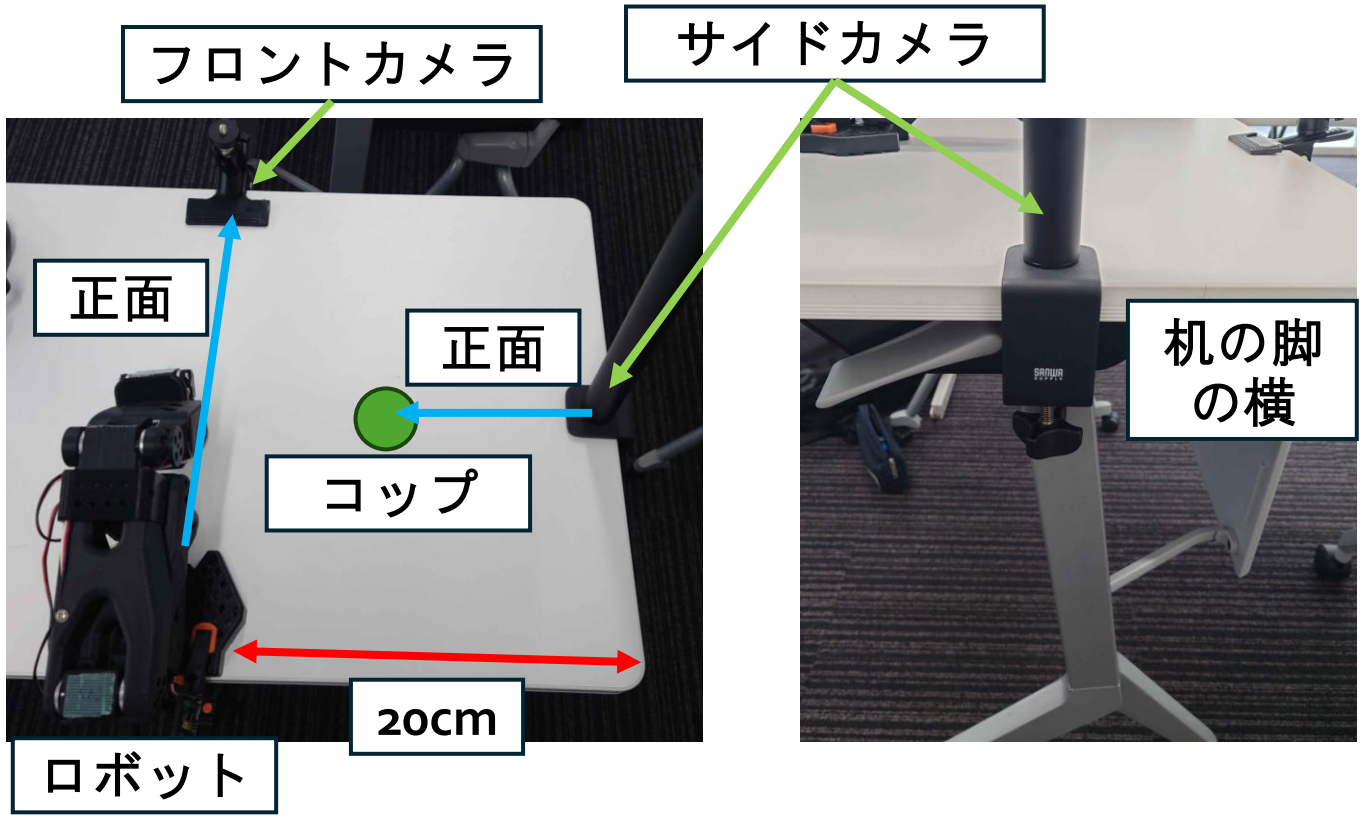
- atolla USBハブ
- クランプ
- カメラデスクマウント
- クリップカメラスタンド



環境セットアップ

環境セットアップ

- ロボットのベースの右端が、机の右端から20cm
- フロントカメラはロボットの正面
- サイドカメラは机の足の横
- コップはロボットの右、サイドカメラの正面に置く
- コップは養生テープで机に固定



資料共有

コマンドが多いので，以下のページにアクセスして資料ページを開いておいて下さい.

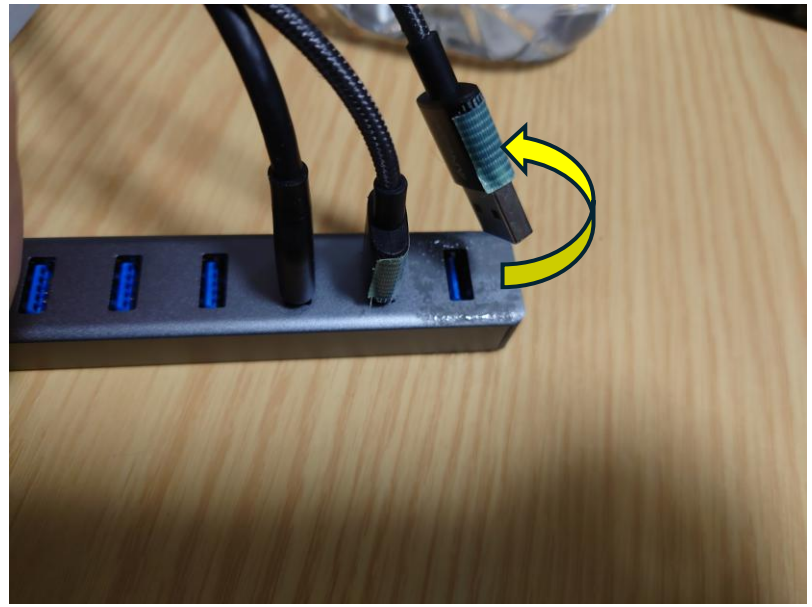
https://x.gd/iPTgF

SO-10xセットアップ (1/3)

- 資料中「1. LeRobotのセットアップ」は実行済み

- **ポートの検出**

1. VSCodeで，LeRobotのディレクトリを開く
2. Ctrl + j でターミナルを開く
3. **$ uv run lerobot-find-port** を実行
4. リーダー側のUSBコードを抜いてEnter
5. 表示されたポートを覚えておく
6. フォロワー側でも同様の操作を行う



- **ポートに権限を付与**

$ sudo chmod 666 /dev/ttyACM0

$ sudo chmod 666 /dev/ttyACM1

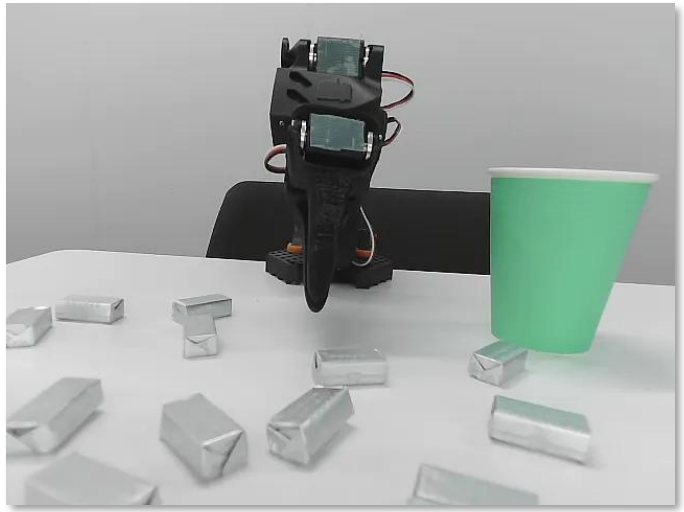
SO-10xセットアップ (2/3)

- 資料中「キャリブレーション」は実行済み
- カメラセットアップ

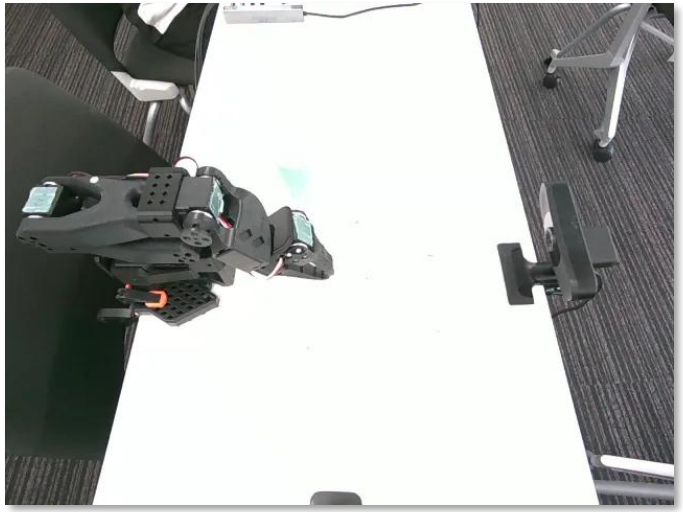
Webカメラ（フロント側）とRealSense（サイド側）のパスを調べる

$ uv run lerobot-find-cameras opencv

outputs/captured_images/ に出力された画像を見て、カメラのパスを特定



Webカメラ（フロント側）



RealSense（サイド側）

画像の名前が
opencv__dev_video8.png
ならパスは
/dev/video8

SO-10xセットアップ (3/3)

テレオペレーション

- SO-101を使っている場合はrobot.type=so101_follower, teleop.type=so101_leaderとする
- カメラやロボットのポートを適宜修正
- しっかり動いて，カメラ映像も表示されていたらOK

修正可能性のある箇所を赤で強調しています

コマンド

```
uv run lerobot-teleoperate ¥
        --robot.cameras="{ front: {type: opencv, index_or_path: /dev/video8, width: 640, height: 480, fps: 30},
                           side: {type: opencv, index_or_path: /dev/video6, width: 640, height: 480, fps: 30}}" ¥
        --robot.type=so100_follower ¥
        --robot.port=/dev/ttyACM1 ¥
        --robot.id=follower ¥
        --teleop.type=so100_leader ¥
        --teleop.port=/dev/ttyACM0 ¥
        --teleop.id=leader ¥
        --display_data=true
```

データ収集

良いデモ

良い例

- 滑らかでゆっくりとした操作
- 多様性のあるデータ
  - ぷっちょをランダムに置く
- 時間内にタスクを完遂する
- 再現性の高い動き
  →グリッパーを大きく開いて掴む

悪いデモ

悪い例

- タスクに最適化し過ぎた動き
  - グリッパーを必要最低限しか開かない
  - 操作が速すぎる
- 時間内にタスクが終わらない
- タスクと関係のない動き

  →ロボットは，リセットタイムに初期位置に戻す

操作練習

- 良い例と悪い例を意識して操作練習をしてみてください

良い例

- 滑らかでゆっくりとした操作
- 多様性のあるデータ
  - ぷっちょをランダムに置く
- 時間内にタスクを完遂する
- 再現性の高い動き
  →グリッパーを大きく開いて掴む

悪い例

- タスクに最適化され過ぎた動き
  - グリッパーを必要最低限しか開かない
  - 操作が速すぎる
- 時間内にタスクが終わらない
- タスクと関係のない動き
  →ロボットは，リセットタイムになってから初期位置に戻す

データ収集 (30分程度）

SO-100を使っているチームはrecordo,
SO-101を使っているチームはrecord1
としてください

コマンド

```
export DATASET_NAME=record0

uv run lerobot-record ¥
        --robot.cameras="{ front: {type: opencv, index_or_path: /dev/video8, width: 640, height: 480, fps: 30},
                           side: {type: opencv, index_or_path: /dev/video6, width: 640, height: 480, fps: 30}}" ¥
        --robot.type=so100_follower ¥
        --robot.port=/dev/ttyACM1 ¥
        --robot.id=follower ¥
        --teleop.type=so100_leader ¥
        --teleop.port=/dev/ttyACM0 ¥
        --teleop.id=leader ¥
        --display_data=true ¥
        --dataset.repo_id=local/${DATASET_NAME} ¥
        --dataset.root=datasets/${DATASET_NAME} ¥
        --dataset.num_episodes=60 ¥
        --dataset.push_to_hub=false ¥
        --dataset.episode_time_s=12 ¥
        --dataset.reset_time_s=8 ¥
        --dataset.video_encoding_batch_size=20 ¥
        --dataset.single_task="Pick up puccho"
```

- 各チーム60エピソードずつ収集
- 20エピソードごとにエンコード処理が入るので，そのタイミングで操作者を交代
- PCの音量を上げておく

データ回収

- **datasets/** 下のデータセットフォルダ (recordo, record1) を USBメモリにコピーしてください

アジェンダ

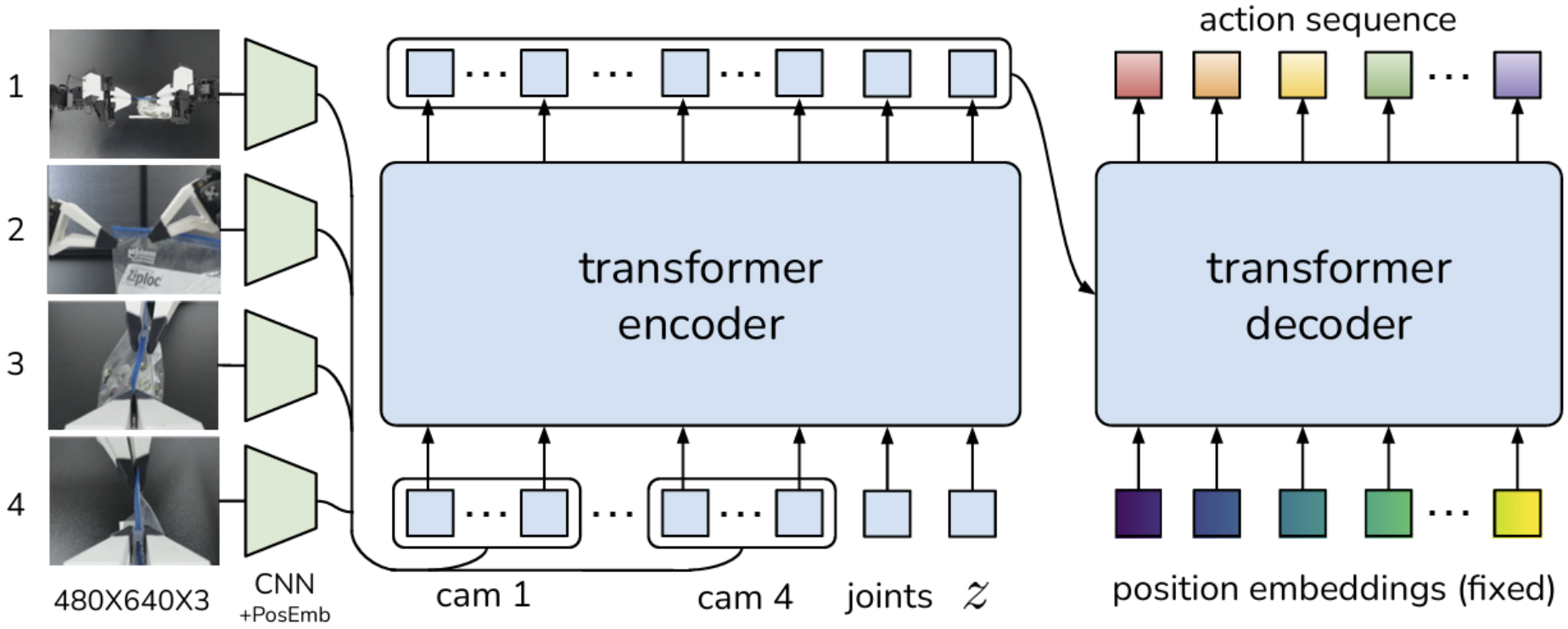
1. イントロダクション&ゴール設定
2. 模倣学習ミニレクチャー
3. データ収集
4. **学習&ACT解説**
5. 評価と発展

学習の実行

ACT解説

ACT概要（再掲）

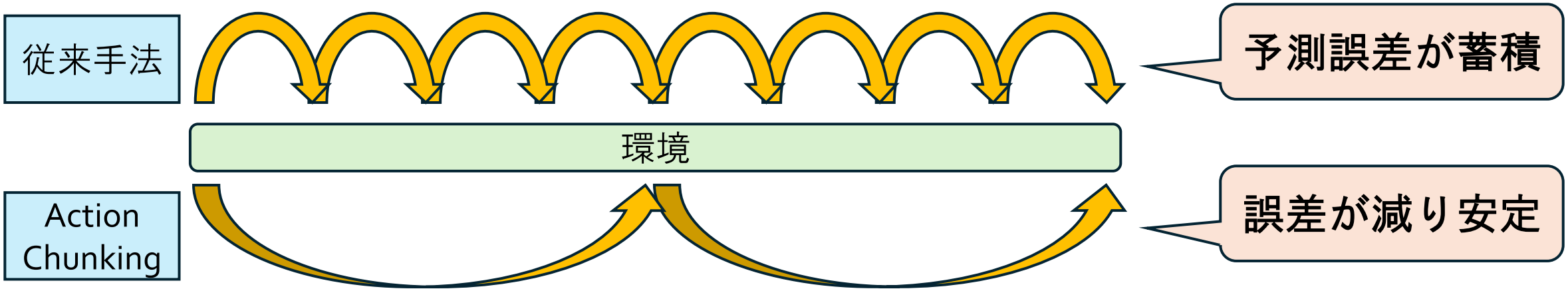
- 畳み込みニューラルネットワーク (ResNet-18) を用いて画像特徴量を抽出
- 注意機構 (Transformer) を利用し，重要な情報に注目
- 一度の生成で複数step分の行動を出力し，高い時間的一貫性



キーポイント１：Action Chunking

従来手法：現在の状態を見て，次の１ステップの行動を予測

Action Chunking：未来のkステップ分の行動をまとめて予測



人間のデモ操作には「流れ」や「間」が存在する

→１ステップ毎のモデル化では解釈が難しい

→行動をチャンクで学習する事で時間的依存関係を捉えやすくなる

キーポイント２：Temporal Ensembling

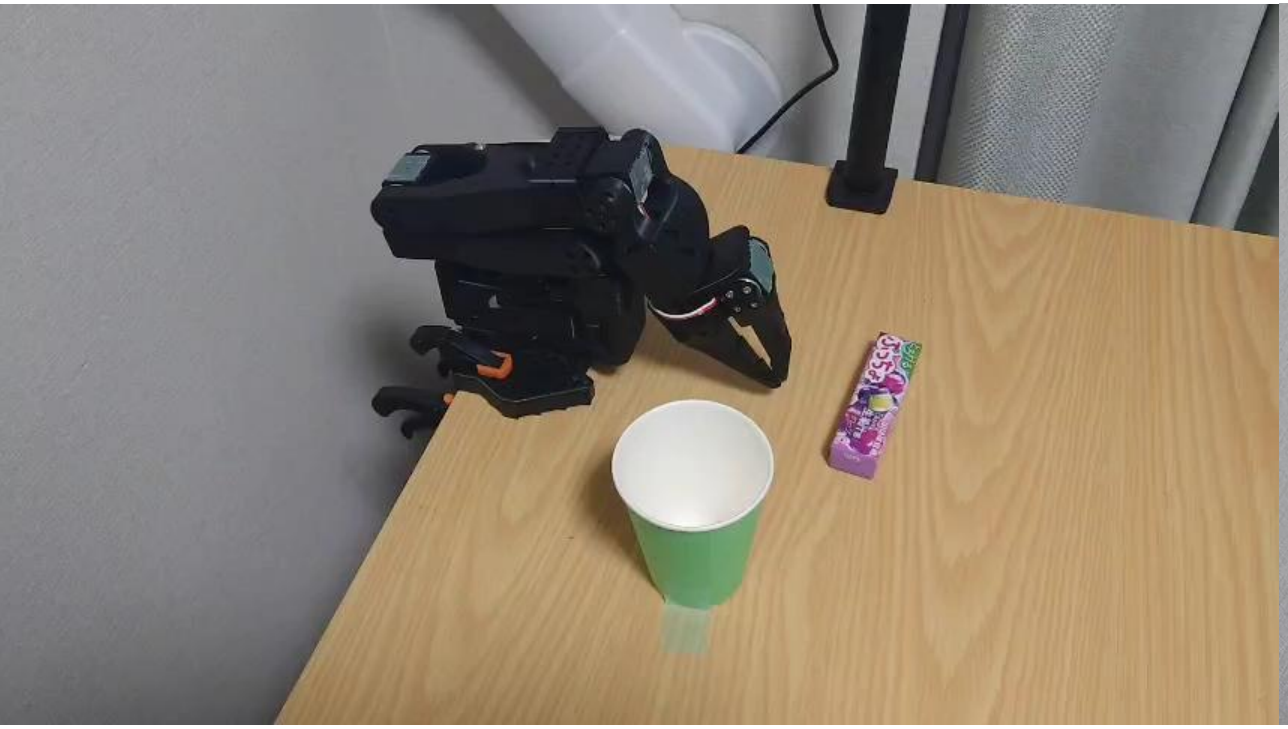
- 単純なAction Chunkingでは，kステップごとに行動計画が切り替わり，動きがぎこちなくなる
- Temporal Ensemblingでは，毎ステップ最新の観測で行動のチャンクを予測し，重みづけて滑らかに更新

→環境変化に素早く応答

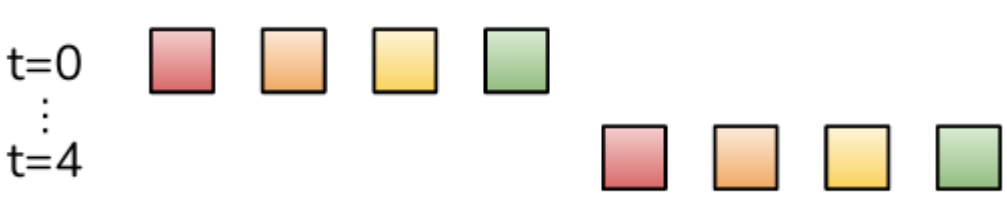
$$w_i = \exp(-m \times i)$$

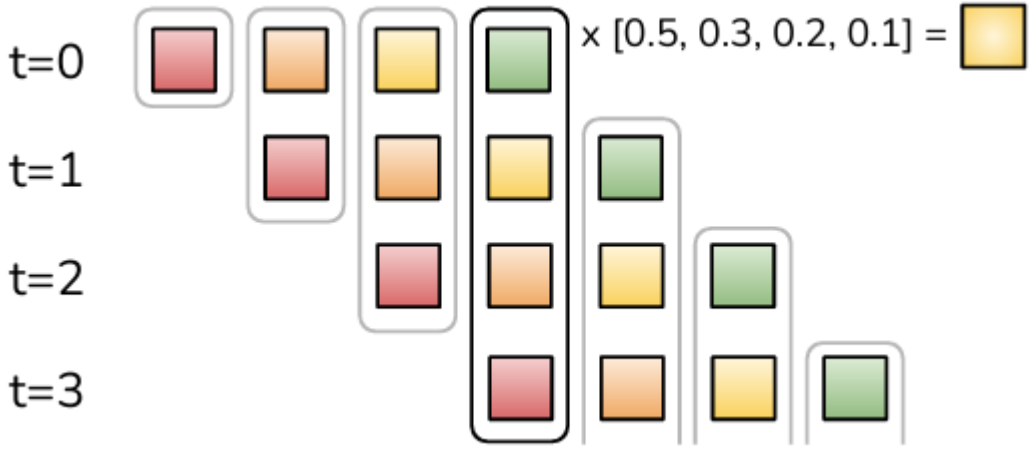
比較

AC



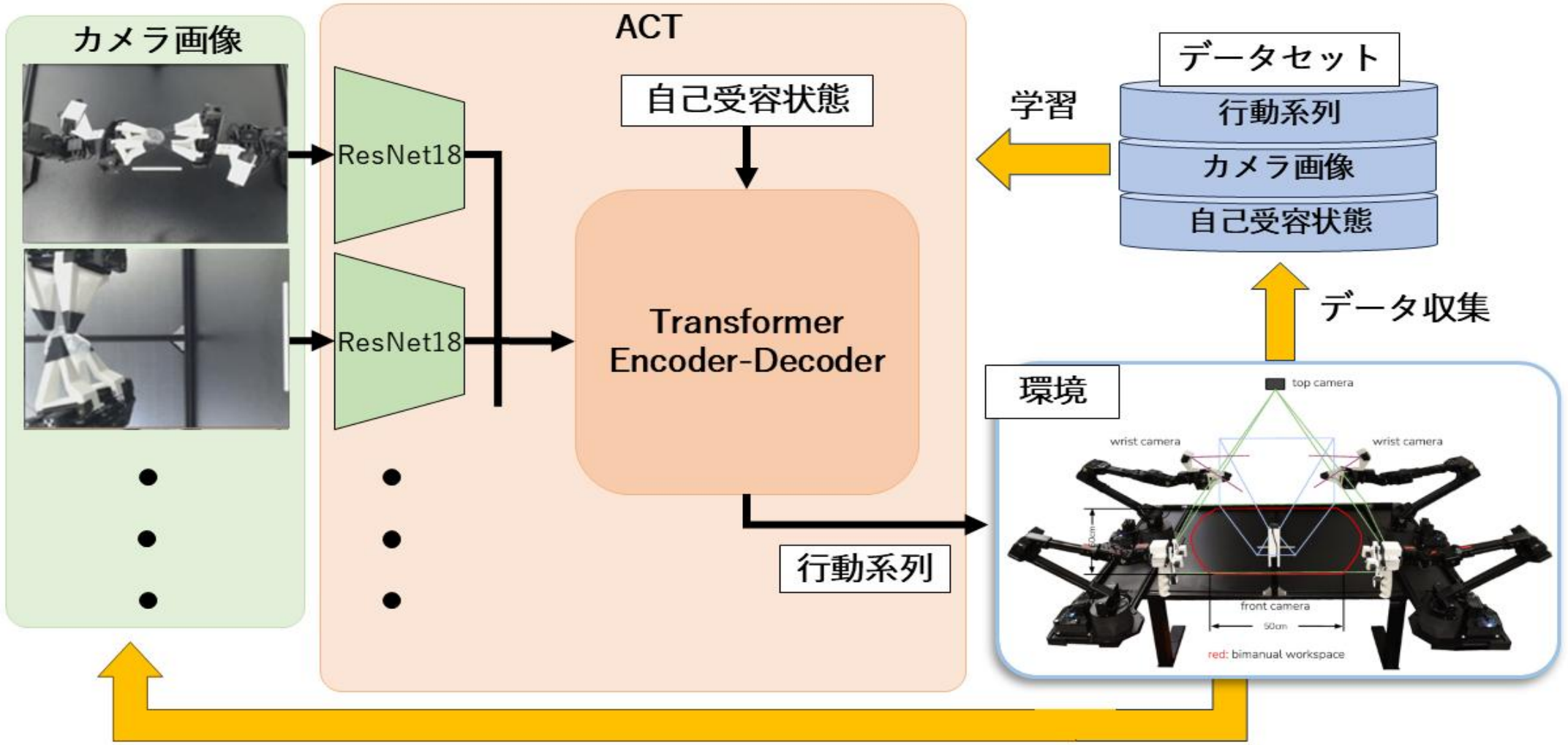
AC+TE





アーキテクチャ概要

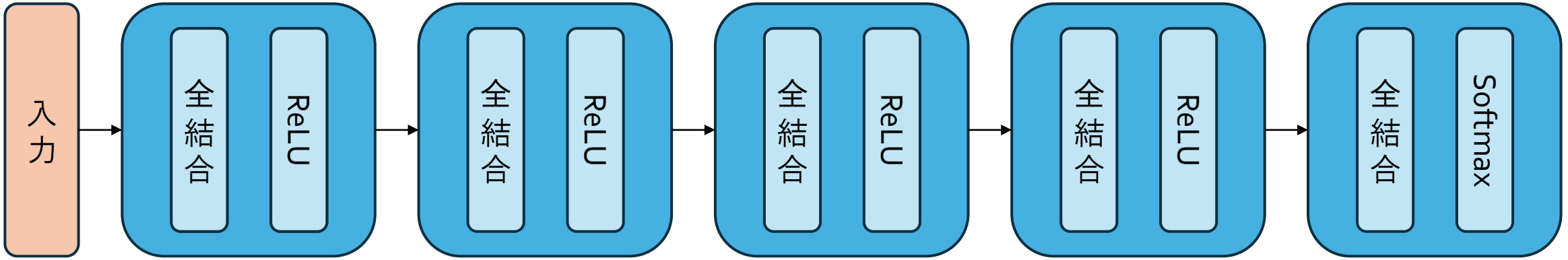
- ResNet-18やTransformerとは？



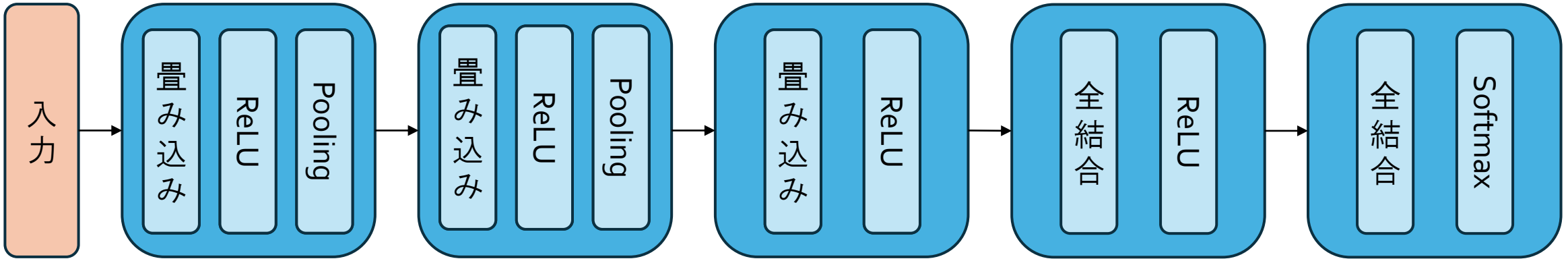
ResNet-18

CNNについて：全体の構造 (1/8)

- ResNet-18はCNNの一種
- CNNと通常のアーキテクチャ（MLP）の違いは？



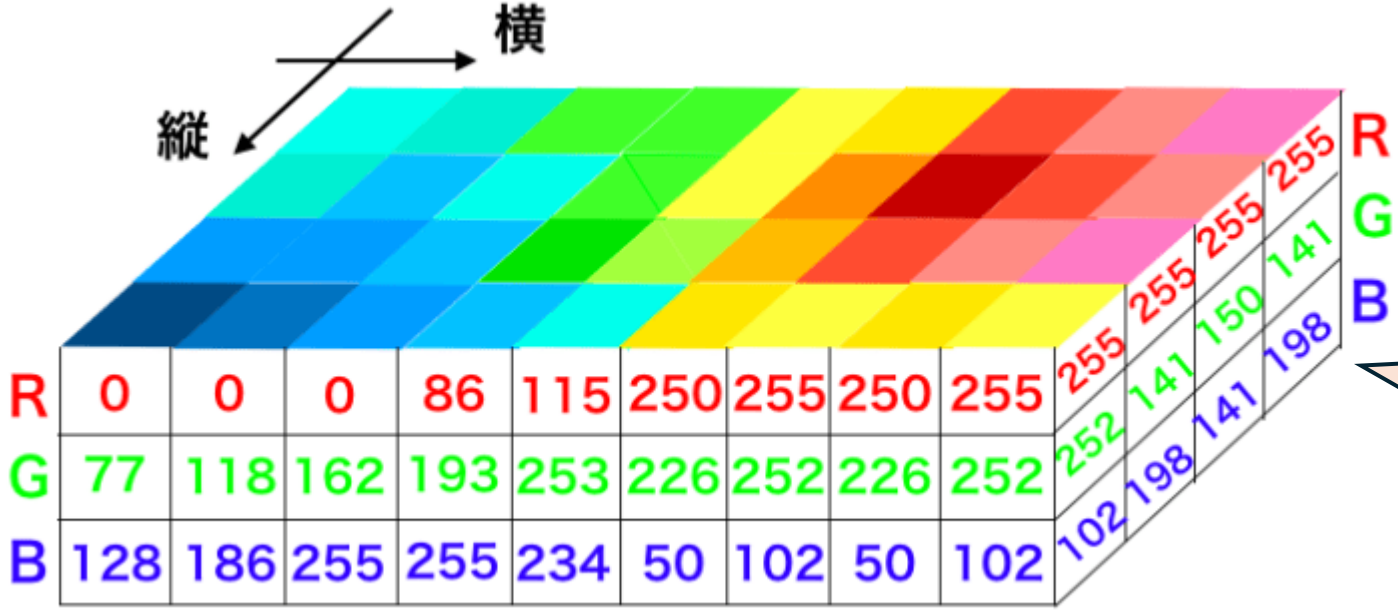
多層パーセプトロン（MLP）の例



畳み込みニューラルネットワーク（CNN）の例

CNNについて：畳み込み層 (2/8)

- 多層パーセプトロン（MLP）は入力データに含まれる要素の関連性を無視して，１次元配列として扱う
- 畳み込みニューラルネットワーク（CNN）は，入力データに含まれる要素の関連性を解釈できる



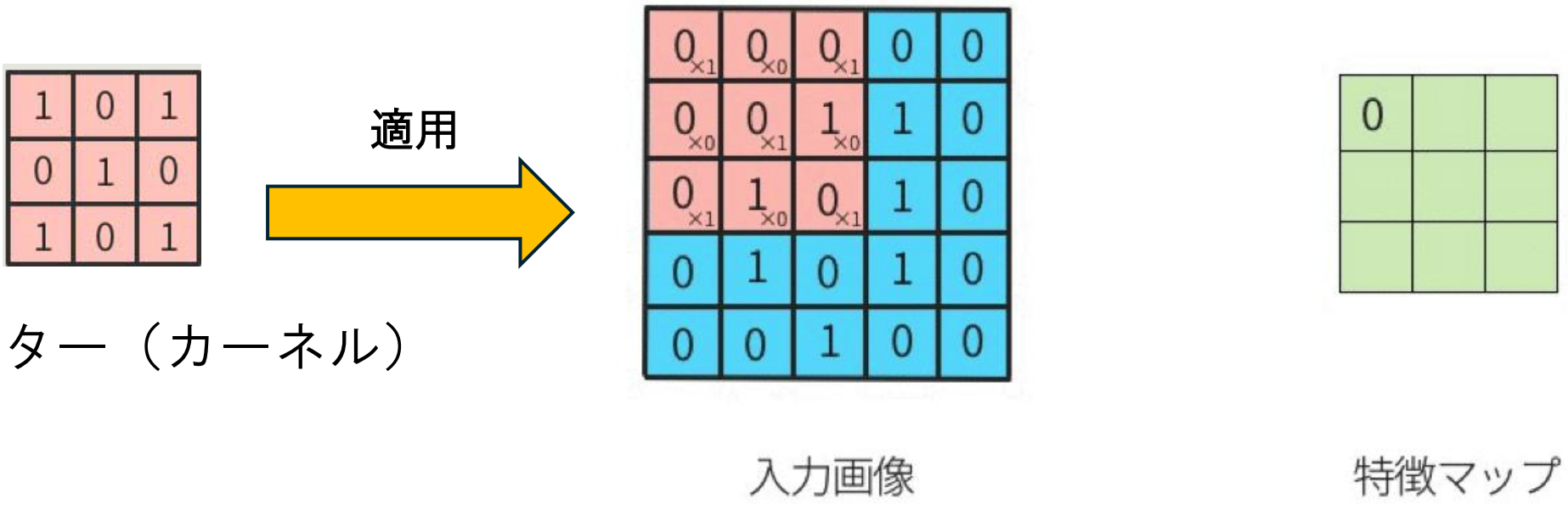
画像データなどは，隣接するデータに明らかに関連性がある

画像の引用元 https://daeudaeu.com/bitmap/

CNNについて：畳み込み演算 (3/8)

畳み込み演算

- 入力データに対してフィルターを一定間隔でスライドさせて，その積の和を求める（積和計算）
- フィルターはMLPにおける重みWに対応し，学習により更新される



画像の引用元 https://deepage.net/deep_learning/2016/11/07/convolutional_neural_network.html

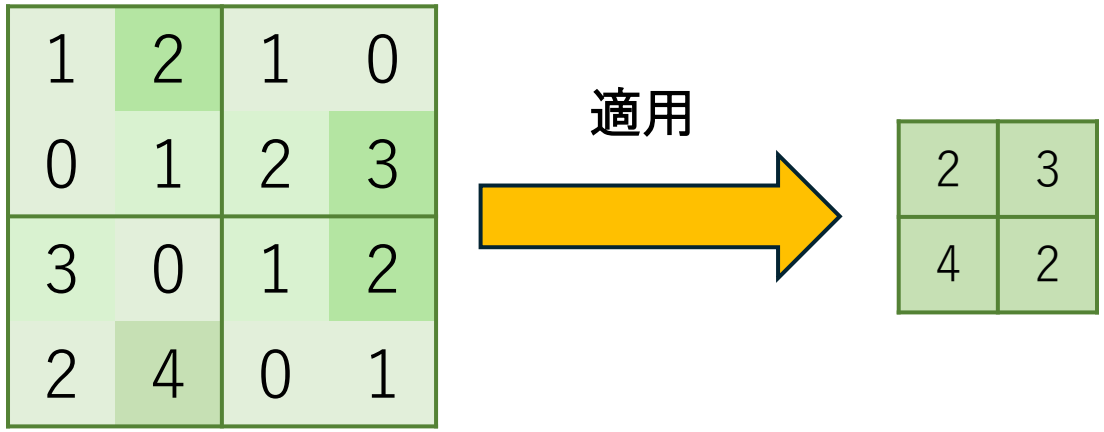
CNNについて：プーリング層 (4/8)

プーリング層

入力データの一定領域内の最大値や平均値を取り出して出力とする

特徴

- パラメータを持たない
- チャンネル数が変化しない
- 入力データの微小変化に対して頑健（ロバスト）

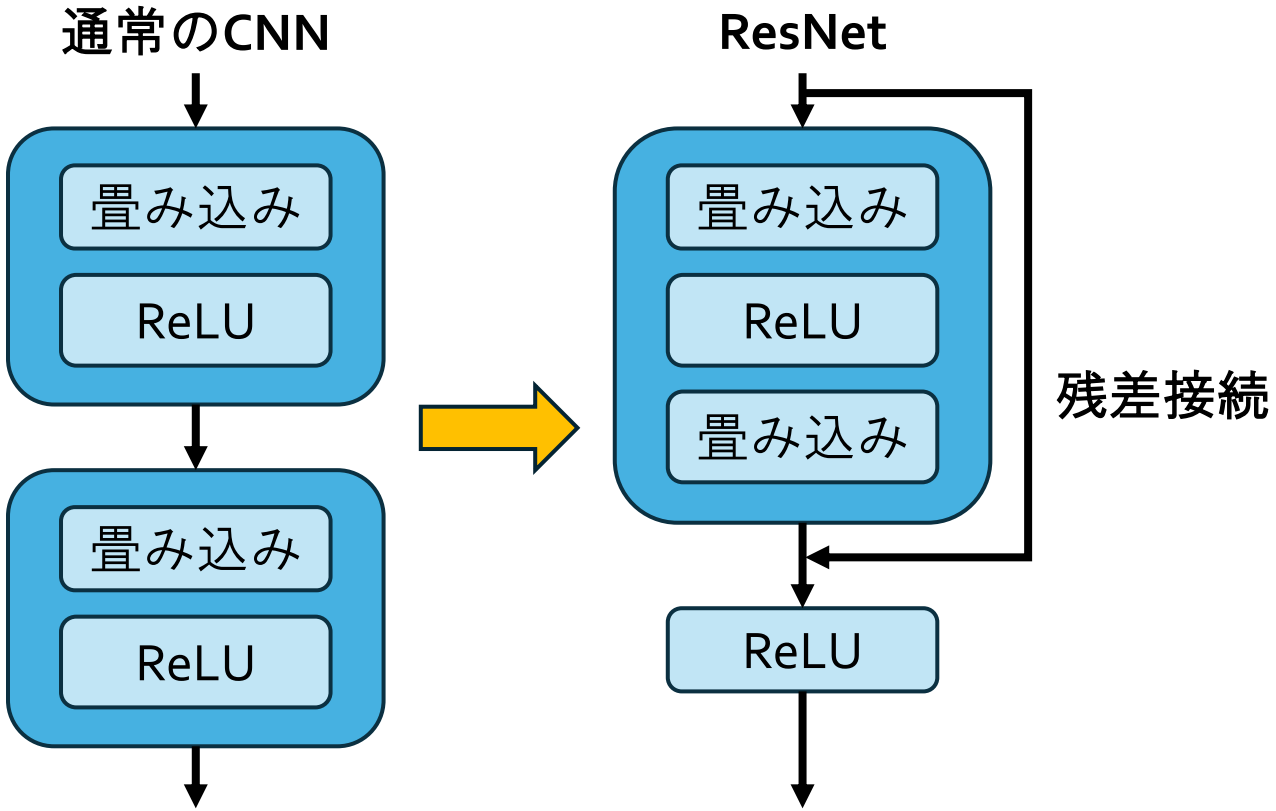


2×2のMaxプーリングを行った場合の例

ResNet-18

- CNNは理論的には層を増やすほど細かい特徴を抽出できる
- 現実的には**勾配消失問題**などが発生

⇒ ResNetは残差接続を導入して層を増やしても学習可能に



- ResNet-18は合わせて18層構造
- ACTはResNet-18を用いて(15, 20, 728)の画像特徴量を抽出

Transformer

Transformerの背景

これまでの言語・系列変換モデル（RNNなど）の課題

- 入力を逐次処理するため，並列計算できない
- 長期的依存関係を捉えられない

Transformerの登場

- 2017年に「Attention Is All You Need」という論文で機械翻訳のモデルとして登場
- Attention機構などを組み合わせ技術の複合体
- 入力の長期的依存関係を捉えられるように

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

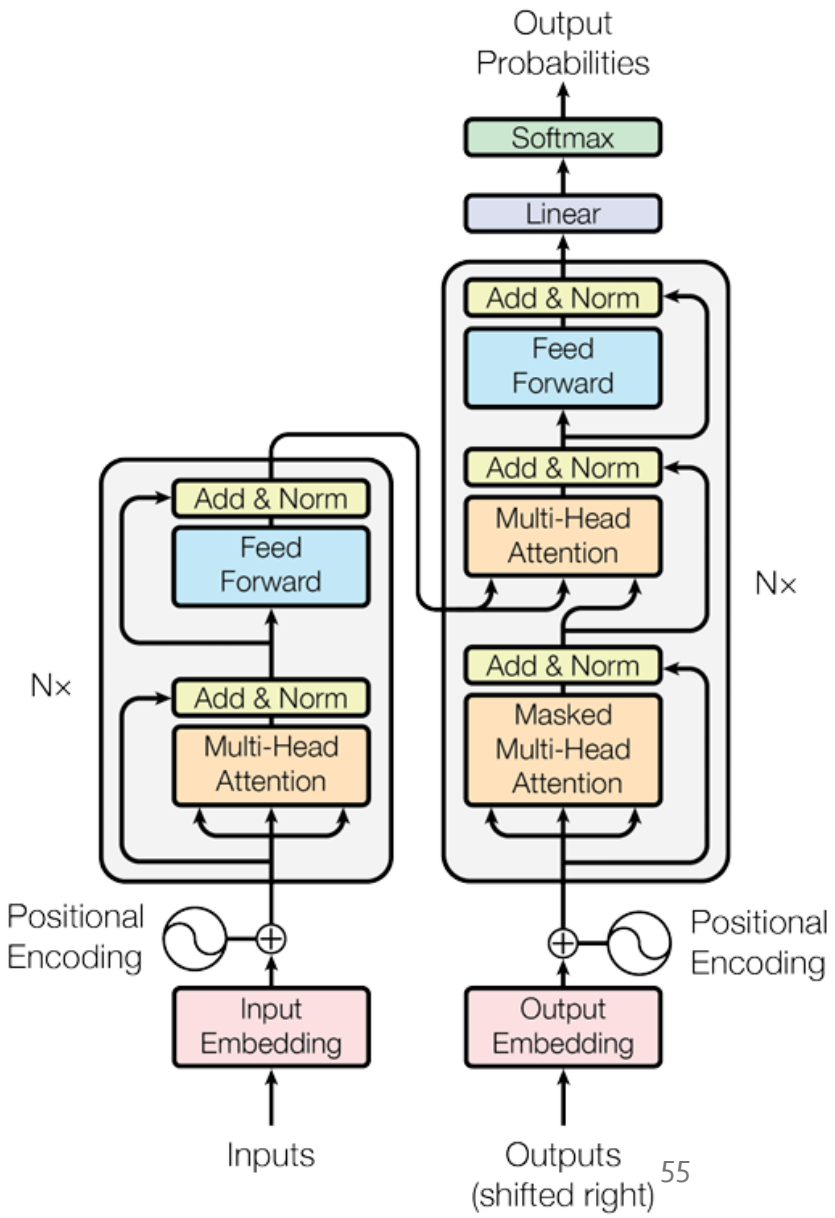
Transformerの構造

全体的にはEncoderとDecoderに分けられ，
それぞれ複数の構成要素からなる

<u>構成要素</u>

- QKV Attention
- Multi-head Attention
- Self-Attention, Cross-Attention
- フィードフォワード層
- 位置符号 (positional encoding)
- 残差接続 (residual connection)

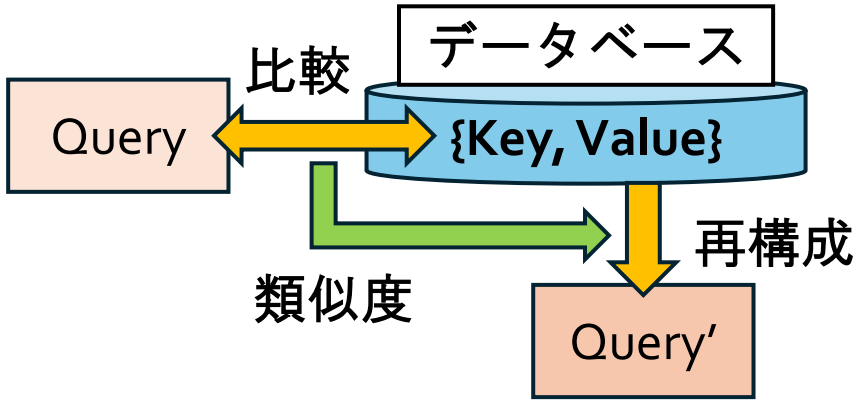
など



QKV Attention

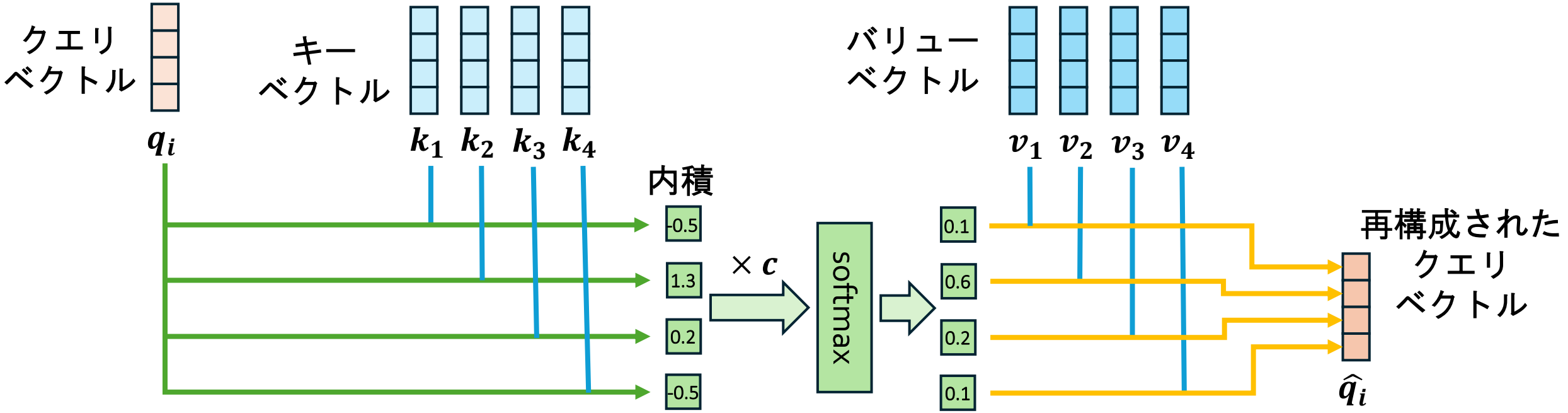
QKVとは？ → Query, Key, Valueの頭文字

イメージ



QKV Attentionの流れ

QueryとKeyの類似度を計算し，その値に応じて
Keyと紐づけられたValueを混合して出力



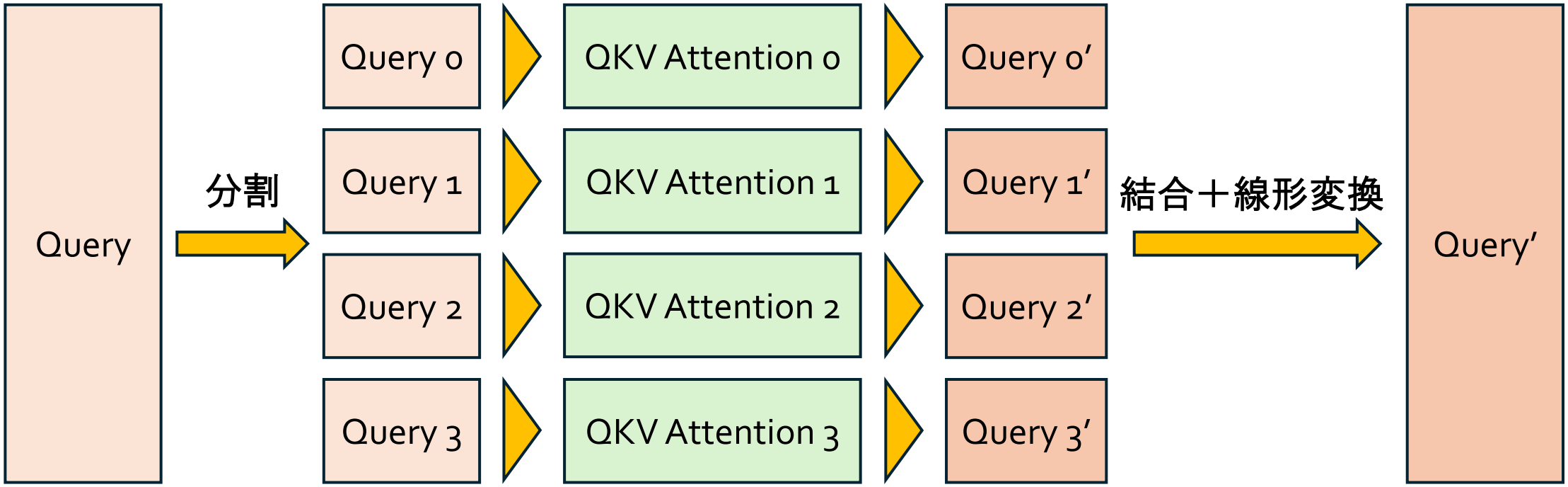
$$\hat{Q} = Attention(Q, K, V) = V \cdot softmax(cK^T Q)$$

Multi-head Attention

QKV Attentionを複数組み合わせたもの

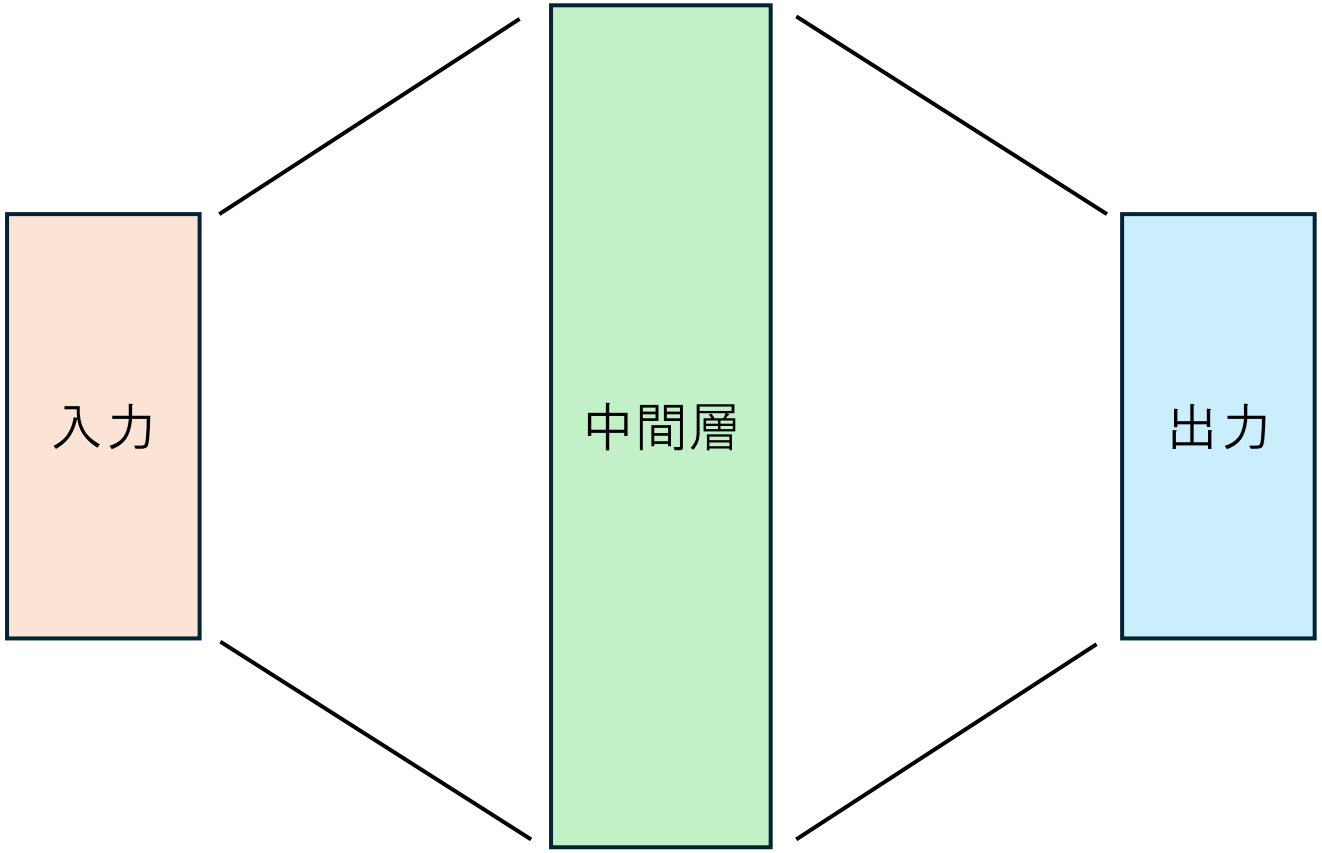
入力の次元が大きくなるとSoftmax関数の影響で上手く情報を取り出せなくなる

⇒ 入力を分割してしまえばいい！



フィードフォワード層

- ２つの巨大な線形層と活性化関数を組み合わせたシンプルな構成
- LLMにおいては知識を保持していると言われる



古典的な２層型ニューラルネットワークと比較すると，中間層が入出力層より大きくなっているのが特徴

位置符号 (positional encoding)

Attention機構の課題

- 入力系列の入れ替えに対して結果が不変

⇒ 位置情報を伝える必要

位置符号

- 正弦関数で位置情報を表現

$$\text{位置符号 } e_{t,k} = \begin{cases} \sin\left(\frac{t}{T^{(k-1)/d}}\right) \\ \cos\left(\frac{t}{T^{(k-2)/d}}\right) \end{cases}$$

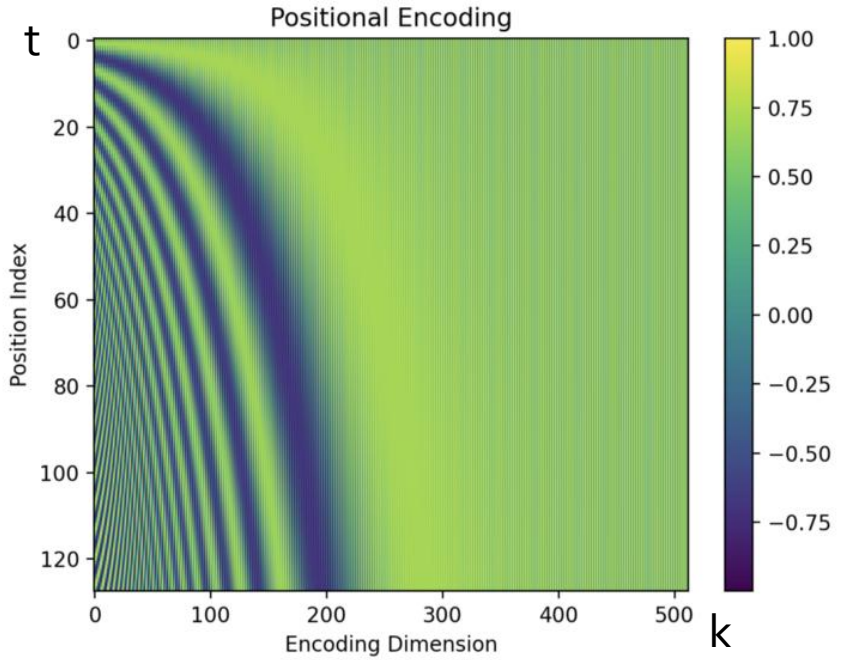
t: 系列上の位置
T: 仮想的な最大系列長
d: ベクトルサイズ
k: 要素番号

イメージ

[I, am, cat] → [吾輩,は,猫である]

[am, I, cat] → [は, 吾輩, 猫である]

入力順を入れ替えると出力順も入れ替わるだけ



https://naokishibuya.github.io/blog/2021-10-31-transformers-positional-encoding/

Encoder-Decoderモデル

入力に位置符号を加えてEncoder, Decoderに入力

Encoder

1. Multi-head Attention
2. 層正規化
3. フィードフォワード層
4. 層正規化

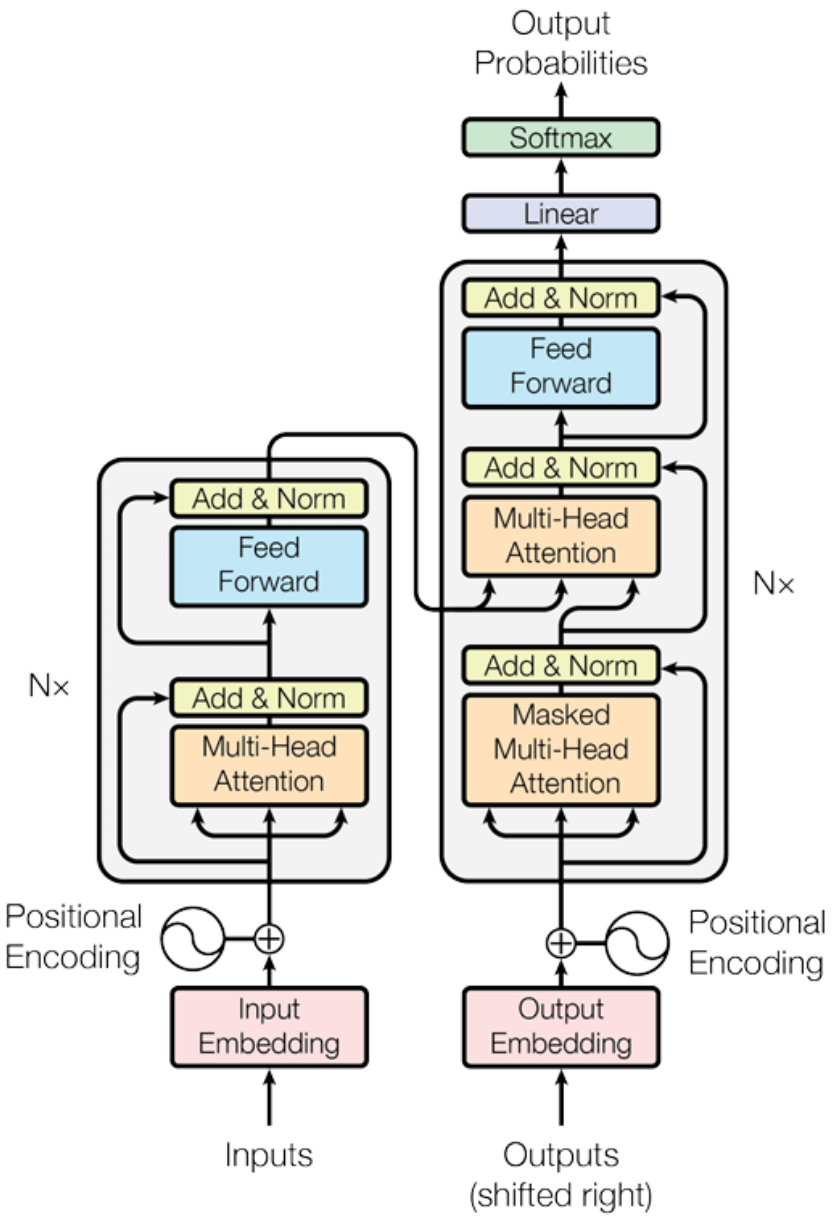
×N

Decoder

1. Multi-head Attention
2. 層正規化
3. Multi-head Attention
4. 層正規化
5. フィードフォワード層
6. 層正規化

×N

１つ目はSelf-Attention
２つ目はCross-Attention



Self-Attention vs Cross-Attention

そもそもQKV AttentionのQuery, Key, Valueってどうやって作る？

⟹ 入力Xを線形変換して作成

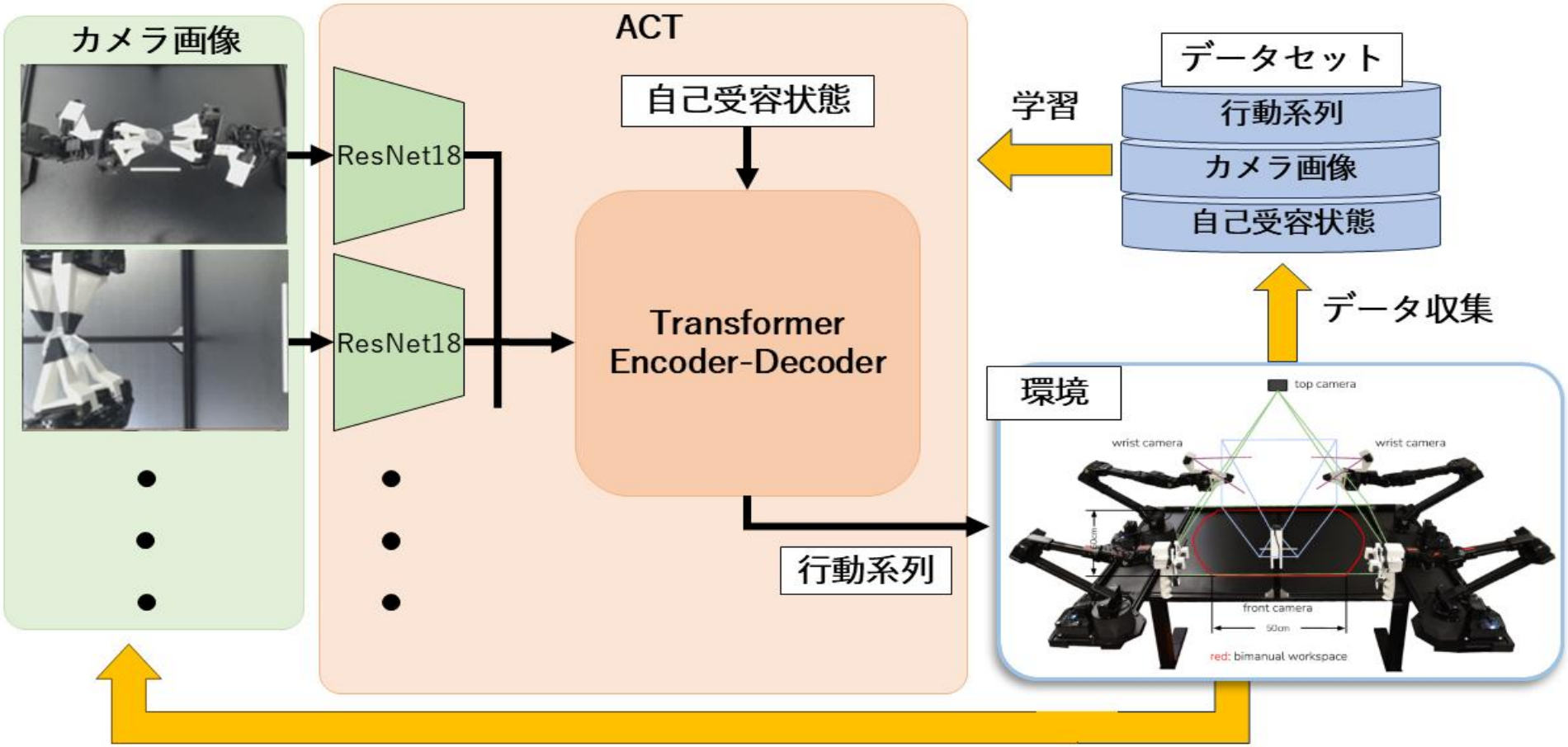
$$\begin{cases} Q = W_Q X \\ K = W_K X \\ V = W_V X \end{cases}$$

Self-Attention：すべて同じXを使う

Cross-Attention：QはDecoderからの入力X_D,
K・VはEncoderからの入力X_Eを使う

アーキテクチャ概要（再掲）

- ResNet18とTransformerのEncoder-Decoderモデルを組み合わせて，効率よく動作を学習



ACT：質疑応答

学習が終わるまで休憩

アジェンダ

1. イントロダクション&ゴール設定
2. 模倣学習ミニレクチャー
3. データ収集
4. 学習&ACT解説
5. **評価と発展**

動作テスト

実機評価

- 学習後のモデル重み (**pretrained_model**) を**output/**下に配置
- **pretrained_model/config.json**の**n_action_steps**を**40**にする

コマンド

```
export DATASET_NAME=eval_act0
uv run lerobot-record ¥
        --robot.cameras="{ front: {type: opencv, index_or_path: /dev/video8, width: 640, height: 480, fps: 30},
                           side: {type: opencv, index_or_path: /dev/video6, width: 640, height: 480, fps: 30}}" ¥
        --robot.type=so100_follower ¥
        --robot.port=/dev/ttyACM1 ¥
        --robot.id=follower ¥
        --display_data=false ¥
        --dataset.repo_id=local/${DATASET_NAME} ¥
        --dataset.root=datasets/${DATASET_NAME} ¥
        --dataset.num_episodes=1 ¥
        --dataset.push_to_hub=false ¥
        --dataset.episode_time_s=30 ¥
        --dataset.single_task="Pick up puccho" ¥
        --policy.path=outputs/pretrained_model
```

修正可能性のある箇所を赤で強調しています

Temporal Ensemblingのテスト

- pretrained_model/config.jsonのn_action_stepsを1にする
- temporal_ensemble_coeffを0.01にする

Temporal Ensembling有効化

Out-of-Distribution (OOD) テスト

いろいろ条件を変えて，動作がどう変化するか観察しましょう

- ぷっちょの色を変えるとどうなる？
- ぷっちょを複数置くとどうなる？
- コップの位置を変えるとどうなる？
-
-
-

Attention Map可視化

以下の２パターンでAttention Mapにどのような差がある？

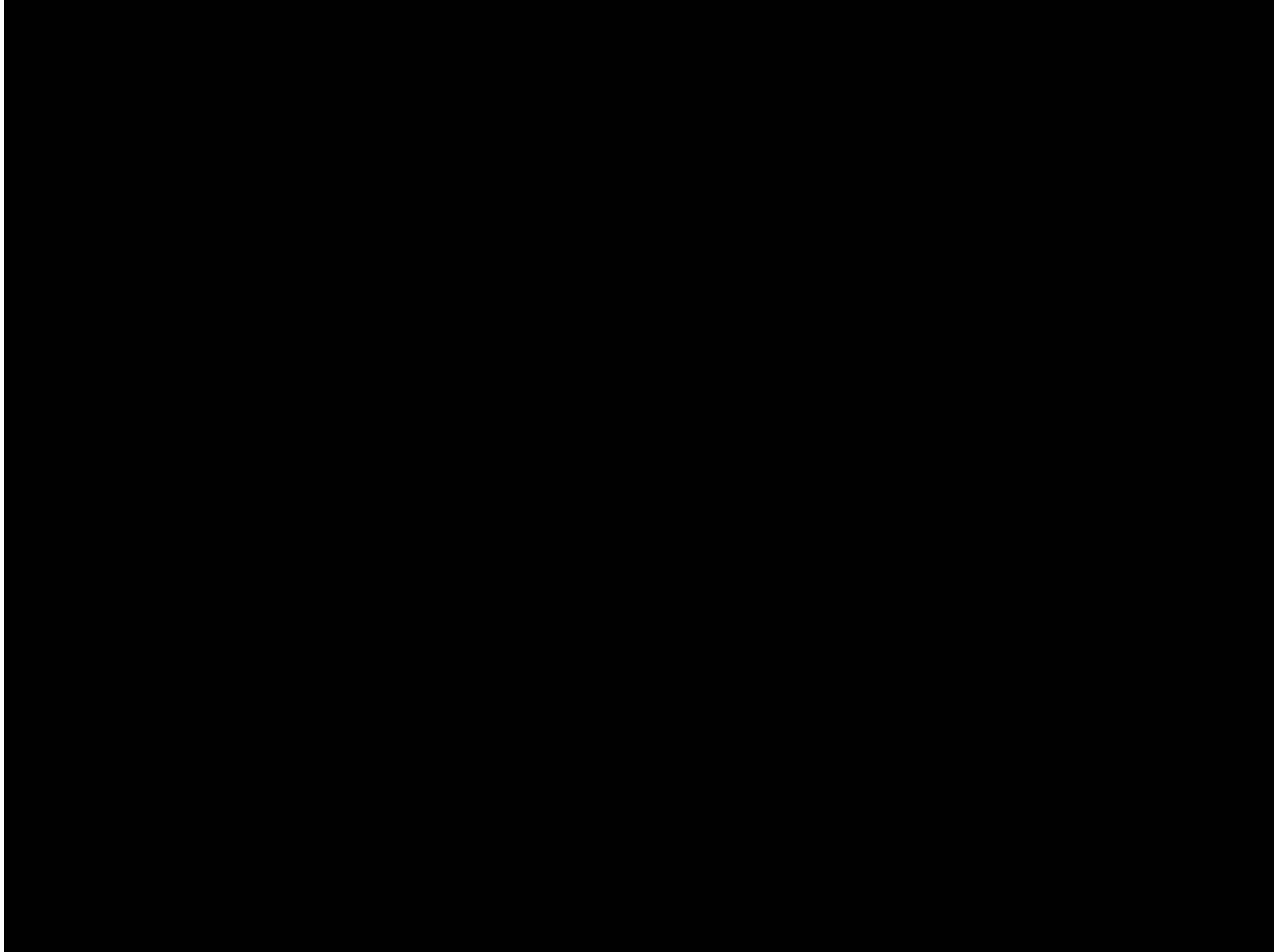
- 正常に動作しているとき
- OODが生じているとき

コマンド

```
source .venv/bin/activate
export HF_LEROBOT_HOME=~/SourceCode/lerobot/datasets
export DATASET_NAME=eval_act0
cd physical-AI-interpretability
python -m examples.visualise_original_data_attention ¥
        --dataset-repo-id=${DATASET_NAME} ¥
        --policy-path=../outputs/pretrained_model
```

評価したいデータセット名に変更

physical-AI-interpretability/output/analysis_output/に動画が出力されます



まとめ

本日のまとめ

- 模倣学習の一連のプロセス (データ収集→学習→評価) を実践
- Action Chunking with Transformer (ACT) について学んだ
- ACTの性能と，OOD問題を体験した
- Attention Mapを可視化し，タスクの成功/失敗要因を分析した

フィジカルAIの面白さと難しさ

Discussion

もっと性能を上げるには

- どんなデータがいい？
- ハイパーパラメータは？
- 他のアーキテクチャは？
- OOD問題を回避するには？

ハンズオンイベントへの要望など

本日はご参加いただきありがとうございました

ハッシュタグ #KUPAC で，ぜひ感想をXに投稿してください