非同期AMI基盤システム αステージ

Geson Anko 2024/03/27

自己紹介

- げそん (GesonAnko)

  𝕏 (Twitter)@GesonAnkoVR

- ML集会 主催

  自律機械知能の研究開発

  PythonでML関係ツールの作成

  VRChatに **P-AMI<Q>**（ぱみきゅー） っていう自律機械知能を造ってるよ。

概要

- 深層モデルの学習と推論を **非同期に並行して行う** 自律機械知能システム（フレームワーク）を実装
  - **実時間上で**学習しながら動作するAIが構築可能に。
    （特にVRChatと相性が良い。）
  - メモリの**同期処理は隠蔽**。開発を容易かつ安全に。制約はある…
    → 同期処理ラッパークラスを通して使用する。
- Pythonで非同期システム作るのしんどい。
  … **しんどい。**

目次

ぱみきゅー

ML Shukai

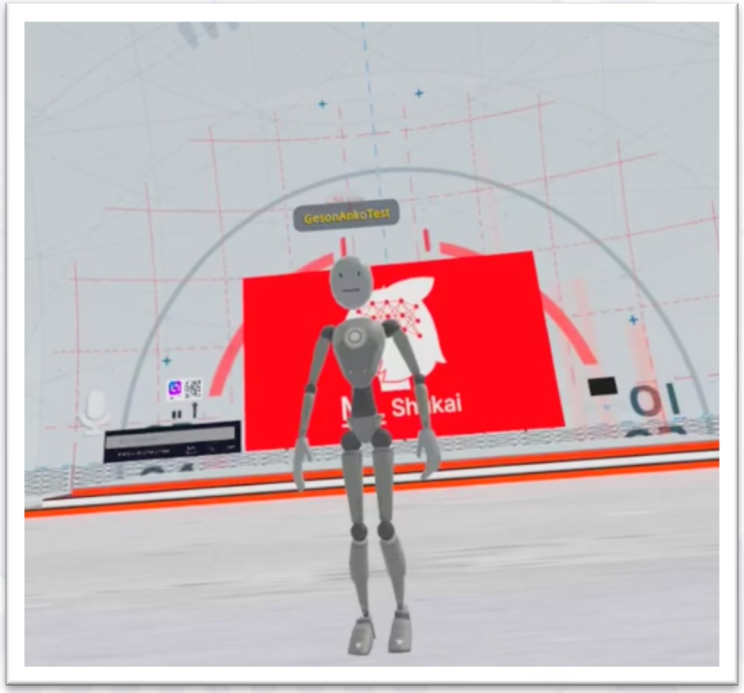
1. おさらい
   1. 自律機械知能 P-AMI<Q>とは
   2. 現在動作中のシステムについて
   3. システムの課題
2. 新システムの **What's New.**
3. システムの全体像
4. 実際に動かしてみると
5. 開発の**難所**
6. Next Step: 一緒に作りませんか？
7. 資料など

おさらい

自律機械知能 ”P-AMI<Q>” とは？

好奇心ベースの原始自律機械知能　2023年9月 誕生

Primitive **A**utonomous **M**achine **I**ntelligence based on **Q**(Cu)riosity.



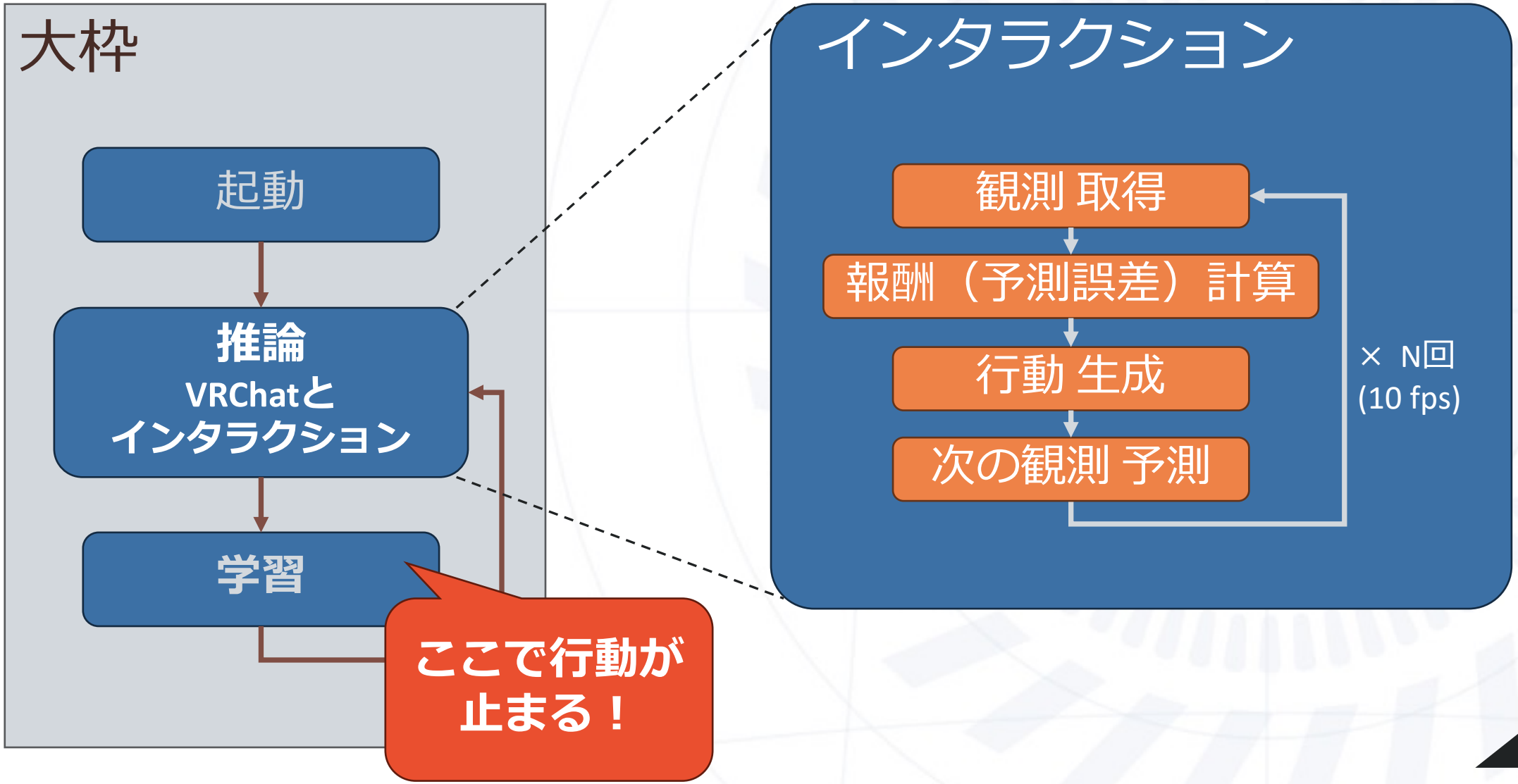
好奇心に従って ワールド上を動き回る。

好奇心とは、未学習や未探索の領域に向かう性質

バーチャル学会2023 で発表したよ。

VRC Group **“VAE.1646”** のGroup+インスタンスにいる。
Japan Street をうろついている。

現在動作中のシステム（処理手続き）



システムの課題

- **定期的に止まる**
  推論と学習を交互に行うため。

- 問題
  - **経験の連続性**が切れる：現実の時間進行との不一致
    → プランニングアルゴリズムなどに悪影響
  - **モデルサイズを大きくできない**：学習時間が増加 → 停止時間も増加
    → 大規模化は深層モデルの要
  - **計算リソースの非効率的使用**

What’s New.

推論と学習を、

非同期的に **同時並行**して

実行可能に！

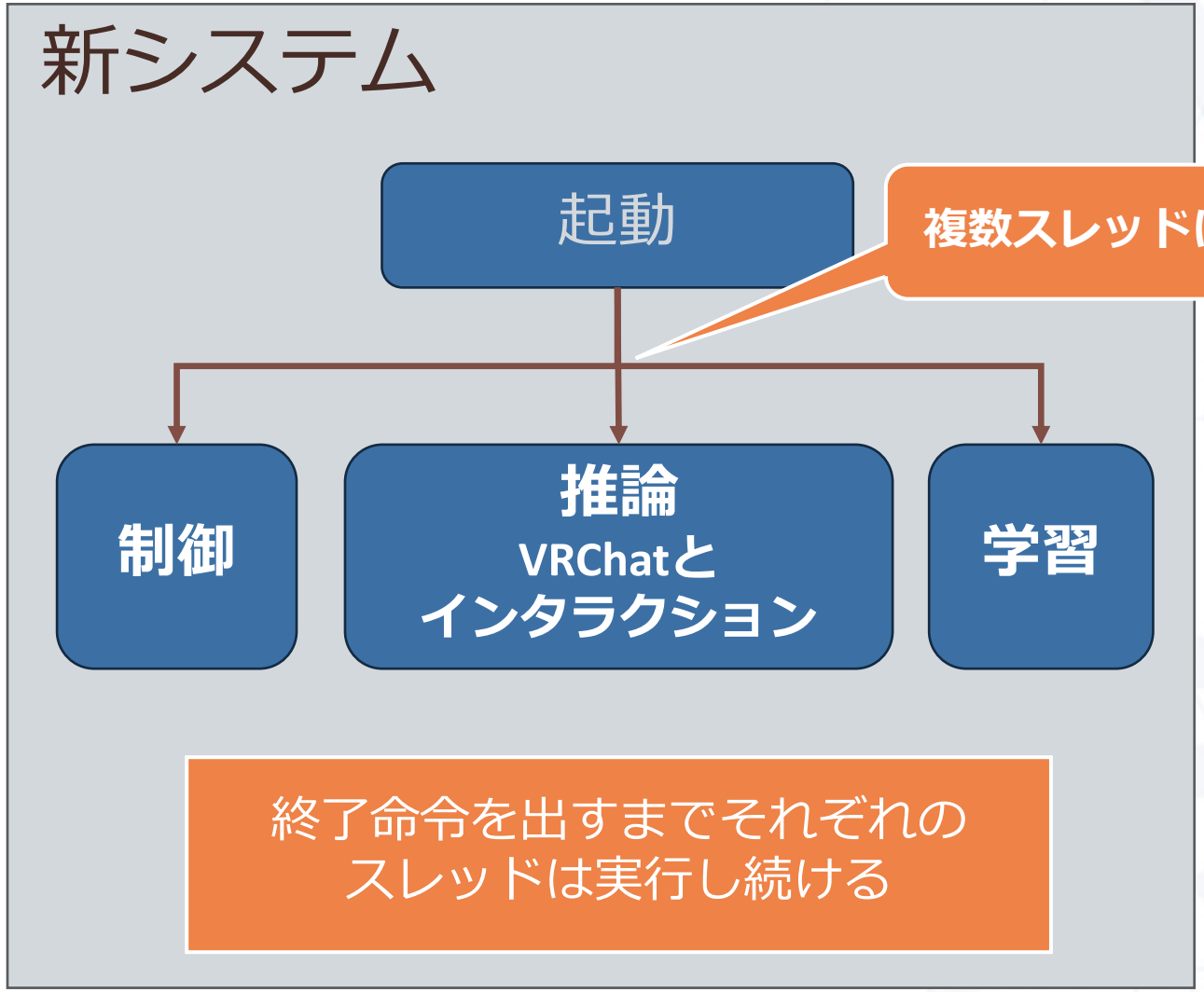
What's New. 概要

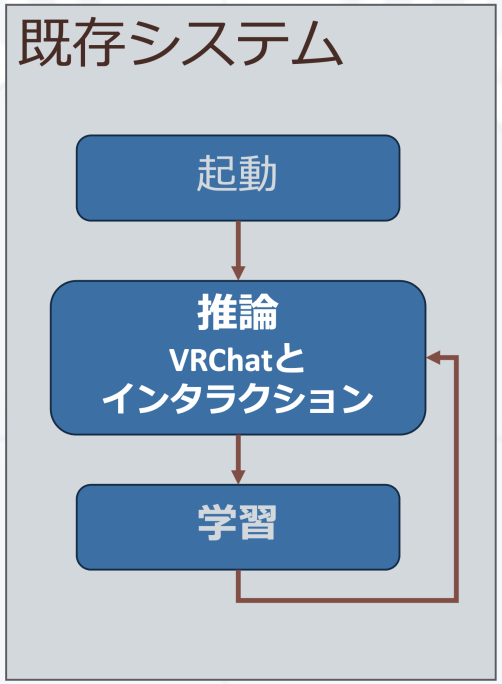
- 大きく変わったことは3つ。
  1. 処理手続き
     システムが**マルチスレッド化**
     Mainスレッド、Inferenceスレッド、Trainingスレッド

  2. 学習データの収集、使用の 手続き
     Inferenceスレッドで集めて、Trainingスレッドで使う

  3. モデルの推論と学習
     Inferenceスレッドで推論
     Trainingスレッドで学習

1. 処理手続き

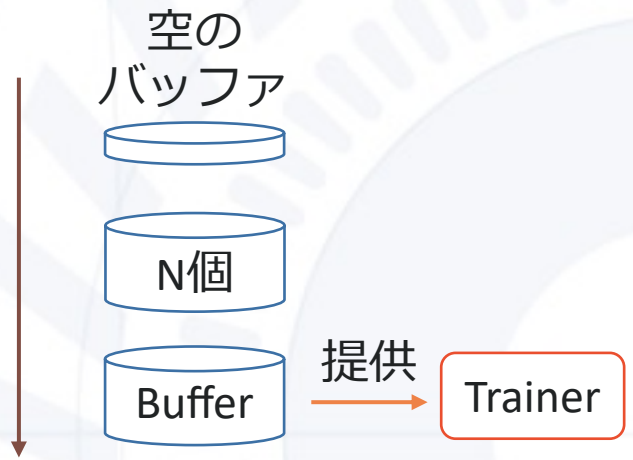

新システム
起動
複数スレッドに分岐
制御
推論
VRChatと
インタラクション
学習
終了命令を出すまでそれぞれの
スレッドは実行し続ける


既存システム
起動
推論
VRChatと
インタラクション
学習

2. 学習データの収集・使用

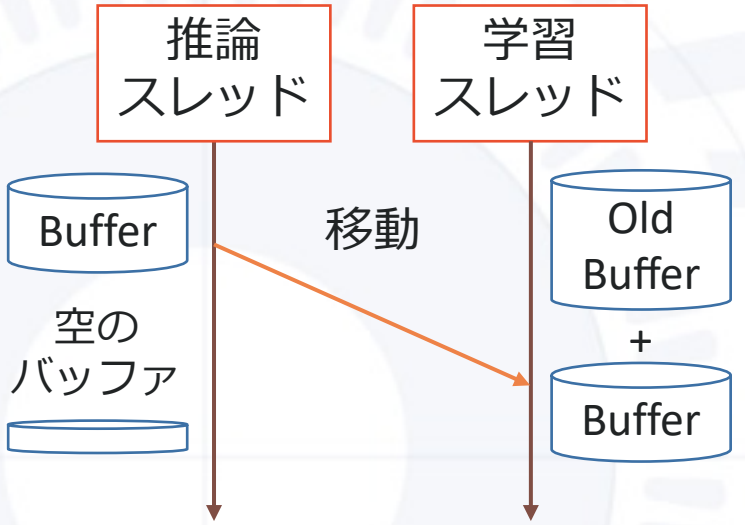
- 既存システム
  1. 空のバッファを用意
  2. データを N 個 収集
  3. 集めたデータを学習に提供
  4. 2に戻る

  全て同期的に行われるよ。



2. 学習データの収集・使用

- 新システム
  Data Collector / Data User システム



- Data Collector
  （推論スレッド）
  1. 空のバッファを用意
  2. データを集める。（無限ループ）
  …
  ?. Userに渡したら新しい空のバッファにセット

同期
処理

- Data User
  （学習スレッド）
  1. 空のバッファを用意
  2. データを Collectorから受け取る
     この時 Collectorに空のバッファをセット
  3. 古いバッファと結合
  4. 学習処理へ提供
  5. 2へ戻る。

2. 学習データの収集・使用

ML Shukai

- Re-constructable Class

  「空のバッファが必要なら、新しく作り直せば良いではないか。」

  - コンストラクタをラップ
  - 引数をディープコピーして保存
  - `new( )` で新規バッファ を生成

  - 引数にでかいオブジェクト渡すとメモリコピーがヤバい。
  - そもそもコンストラクタが重い場合ヤバい…

```python
class BaseDataBuffer(ABC):
    _init_args: tuple[Any, ...]
    _init_kwds: dict[str, Any]

    @classmethod
    def reconstructable_init(cls, *args: Any, **kwds: Any) -> Self:
        """Stores constructor arguments for renewing the data buffer, and throw
        them to :meth:`__init__`."""
        instance = cls(*copy.deepcopy(args), **copy.deepcopy(kwds))
        instance._init_args = args
        instance._init_kwds = kwds
        return instance

    @property
    def is_reconstructable(self) -> bool:
        return hasattr(self, "_init_args") and hasattr(self, "_init_kwds")

    def new(self) -> Self:
        if self.is_reconstructable:
            return self.__class__.reconstructable_init(*self._init_args, **self._init_kwds)
        else:
            raise RuntimeError(
                "Can not create new instance! Did you forget to use `reconstructable_init` "
                "instead of `__init__` when creating a instance?"
            )
```

```python
class ThreadSafeDataCollector(Generic[BufferType]):
    def move_data(self) -> BufferType:
        """Move data's pointer to other object."""
        with self._lock:
            return_data = self._buffer
            self._buffer = self._buffer.new()
            return return_data
```

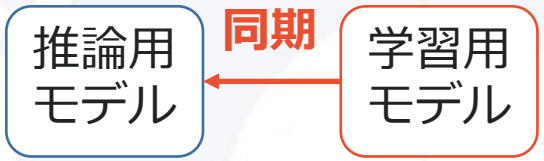
用法を守って安全に使えばとっても便利！

3. モデルの推論と学習

- 大枠

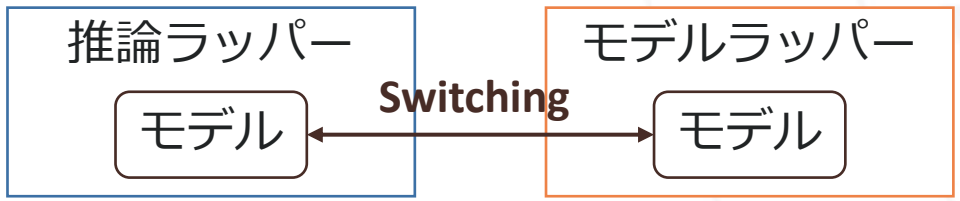
  学習スレッド用モデル、推論スレッド用モデルが存在する。

  1. 学習スレッドでパラメータを更新
  2. 推論スレッドのモデルと同期（**高速に**）



- 同期方式：内部モデルスイッチング

  ラッパークラスを介してモデルを扱う。




```
# 学習されたモデルを推論用にセットアップ。
model_wrapper.freeze_model()

# 学習モデルと推論モデルをスイッチ。
model_wrapper.model, inference_wrapper.model = (
    inference_wrapper.model, model_wrapper.model
)       You, 1 秒前 • Uncommitted changes

# model_wrapperには推論に使われていた古いモデルが渡されるため、パラメータを同期。
model_wrapper.model.load_state_dict(inference_wrapper.model.state_dict())

# 推論に使われていたモデルを学習モードにする。
model_wrapper.unfreeze_model()
```

スイッチ後、推論用モデルから学習用モデルにパラメータコピー

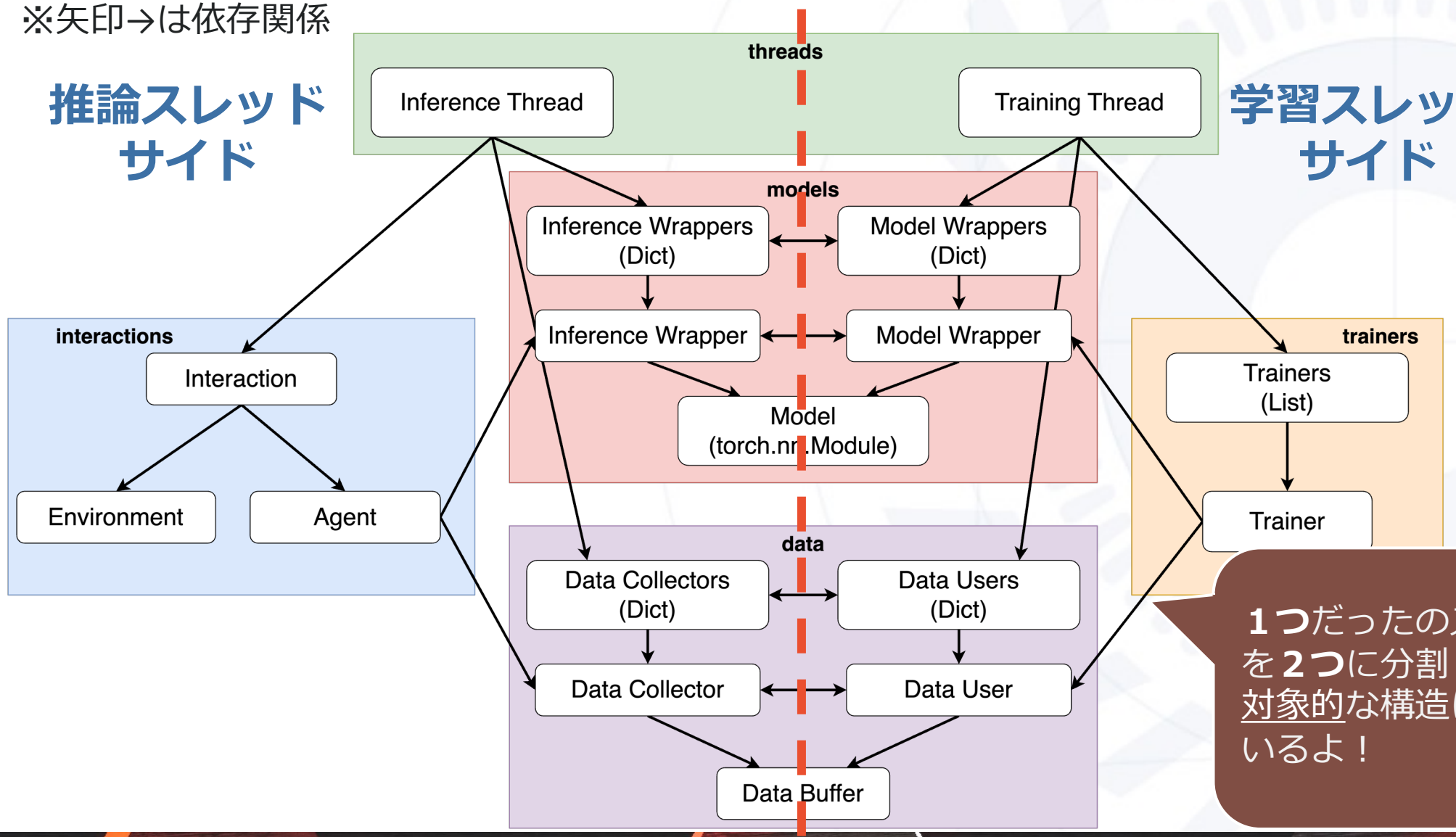
大きい深層モデルのコピーには時間がかかる。推論スレッドを待たせないため。

システムの全体像（クラス関係図）

※矢印→は依存関係

推論スレッドサイド

学習スレッドサイド



1つだったのスレッドを**2つ**に分割したので対象的な構造になっているよ！

その他 新機能

- システムの一時停止、再開
  - Web API でシステム制御（by ゆんたん）
- スレッド毎のログ分離
- Mypyでちゃんと型チェック入れる。
  - 厳密な型チェック（strict) モード
  - ついにGenericsを導入した。
- などなど…

実際に動かしてみると

実際に動かしてみると：設定

- ダミータスク

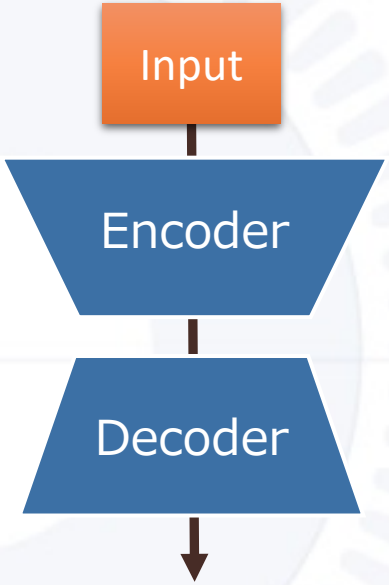
  画像 VAE の推論と学習。

  Decoderは推論に用いない。

  - 推論スレッド
    - 適当に生成した画像をEncoderで圧縮 (10 FPS)
    - 画像をバッファに保存
  - 学習スレッド
    - バッファがバッチサイズ以上溜まったら 1 epoch 学習 (インターバル無し)
  - モデル

    CNN, 比較的小規模なモデル (~ 300万パラメータ)

Variational AutoEncoder



実際に動かしてみると：ログ

ML Shukai

いい感じに動いてるっぽい。

```
[2024-03-27 15:00:25,528][main.__main__][INFO] - Launch AMI.
[2024-03-27 15:00:25,528][main.__main__][INFO] - Instantiating Interaction <ami.interactions.fixed_interval_interaction.Fixe
[2024-03-27 15:00:25,529][main.__main__][INFO] - Instantiating DataCollectors <ami.data.utils.DataCollectorsDict.from_data_b
[2024-03-27 15:00:25,530][main.__main__][INFO] - Instantiating Models <ami.models.utils.ModelWrappersDict>
[2024-03-27 15:00:25,899][main.__main__][INFO] - Instantiating Trainers <ami.trainers.utils.TrainersList>
[2024-03-27 15:00:25,901][main.__main__][INFO] - Instantiating Thread Classes...
[2024-03-27 15:00:25,901][main.__main__][INFO] - Instantiating MainThread: <ami.threads.main_thread.MainThread>
[2024-03-27 15:00:25,902][main.__main__][INFO] - Instantiating InferenceThread: <ami.threads.inference_thread.InferenceThrea
[2024-03-27 15:00:25,902][main.__main__][INFO] - Instantiating TrainingThread: <ami.threads.training_thread.TrainingThread>
[2024-03-27 15:00:25,903][main.__main__][INFO] - Sharing objects...
[2024-03-27 15:00:25,904][main.__main__][INFO] - Starting threads.
[2024-03-27 15:00:25,904][inference.InferenceThread][INFO] - Starts background thread.
[2024-03-27 15:00:25,904][training.TrainingThread][INFO] - Starts background thread.
[2024-03-27 15:00:25,904][inference.InferenceThread][INFO] - Start inference thread.
[2024-03-27 15:00:25,905][training.TrainingThread][INFO] - Starts the training thread.
[2024-03-27 15:00:25,905][main.MainThread][INFO] - Start main thread.
[2024-03-27 15:00:25,910][main.MainThread][INFO] - Serving system command at 'http://0.0.0.0:8391'
[2024-03-27 15:01:06,690][main.WebApiHandler][INFO] - Shutting down threads
[2024-03-27 15:01:06,690][main.MainThread][INFO] - End main thread.
[2024-03-27 15:01:06,720][training.TrainingThread][INFO] - End the training thread.
[2024-03-27 15:01:06,745][inference.InferenceThread][INFO] - End the trainig thread.
[2024-03-27 15:01:06,745][inference.InferenceThread][INFO] - Joined background thread.
[2024-03-27 15:01:06,746][training.TrainingThread][INFO] - Joined background thread.
[2024-03-27 15:01:06,746][main.__main__][INFO] - Terminated AMI.
```

実際に動かしてみると：リソース使用量

ML Shukai

- Desktop PC
  - CPU: **10% ほど**（Intel Core i7 14700KF）
  - GPU: **40~50% ほど**（NVIDIA RTX 4090）

- Laptop PC

  MacBook Pro, M3 Chip.
  - CPU: **100% !** (4E + 4P core)
  - GPU: **100% !** (10 core)

推論と学習が常に同時実行されているので、スペックは必要。

さらにデスクトップモードの VRChatが…

開発の**難所**

開発の難所

- 非同期処理の隠蔽

  ガチガチにフレームワーク化した
  - 同期Lockのかけ忘れを防ぐため。

  - 設計は ほぼMyxyさんだけとやった。
    - 非同期システム特有の設計を理解してもらうのが難しい。
    - 経験や知識のある少人数で設計しないとカオス化する

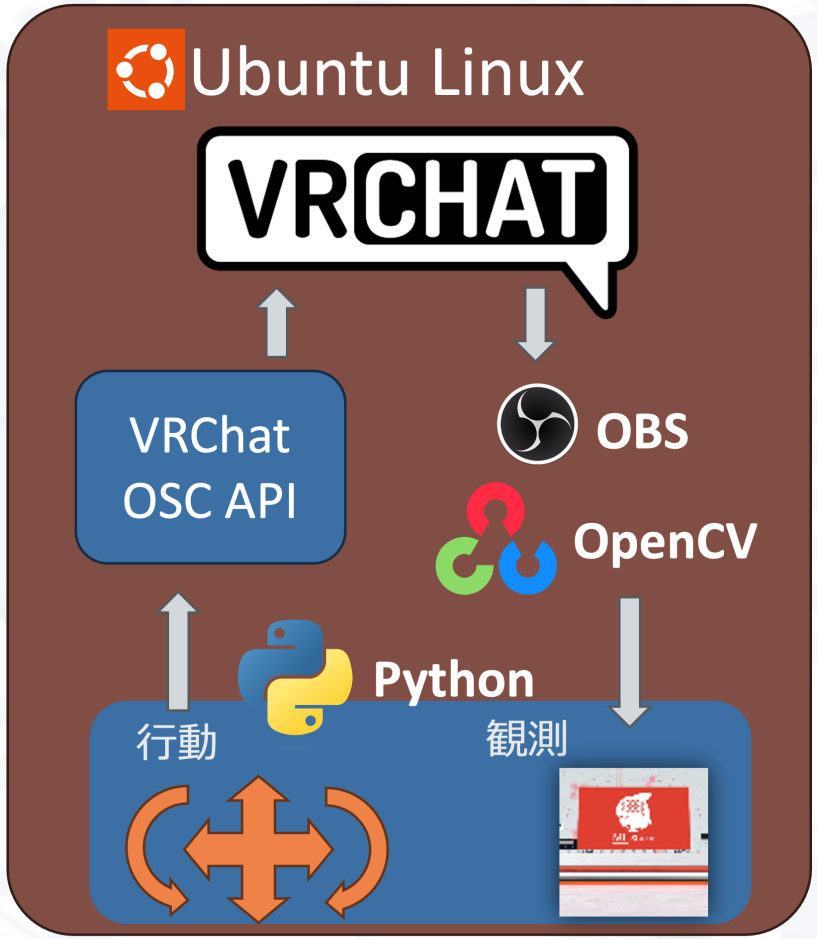
- mypy（静的型チェッカー）
  - 非同期システムを堅牢に作るため。
  - 厳密な型チェックを有効化したので そこそこ辛かった。

ReconstructableにしたData Bufferクラスは未だに納得いってない。

Next Step

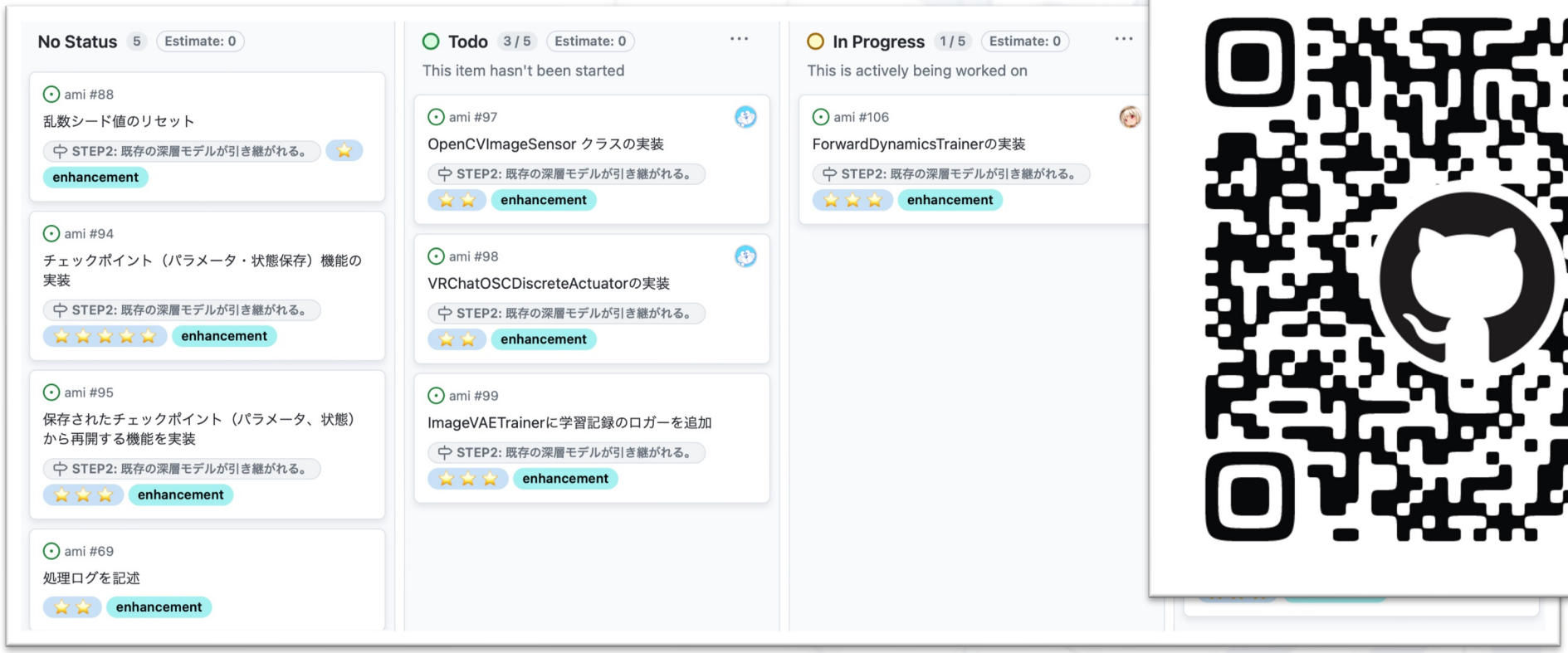
Next Step: To Do.

- 既存実装の引き継ぎ
  - Environment (Sensor, Actuator)
  - Models
  - Data Buffers
  - Agent
  - Trainers
- システムの状態セーブ・ロード機能
  - モデルパラメータ、オブジェクトの状態情報
- 学習記録の可視化
  - TensorBoard, MLFlowなど



一緒に作りませんか？

- ご興味があれば、𝕏@GesonAnkoVR まで。
- タスクの例はプロジェクトボードを参照
  - ⭐が多いほど難しいタスク



GitHub リポジトリ

- MLShukai/ami

本日のスライド

EOF