Diffusion Policies with Multimodal Conditioning

Maurice Rahme

March 17, 2025

1 Introduction

This document presents a detailed explanation of a generic diffusion policy architecture that integrates multimodal conditioning. The model is designed to transform a noisy action sequence into a clean output (e.g., end-effector poses or motor commands). A key aspect of the architecture is the distinction between:

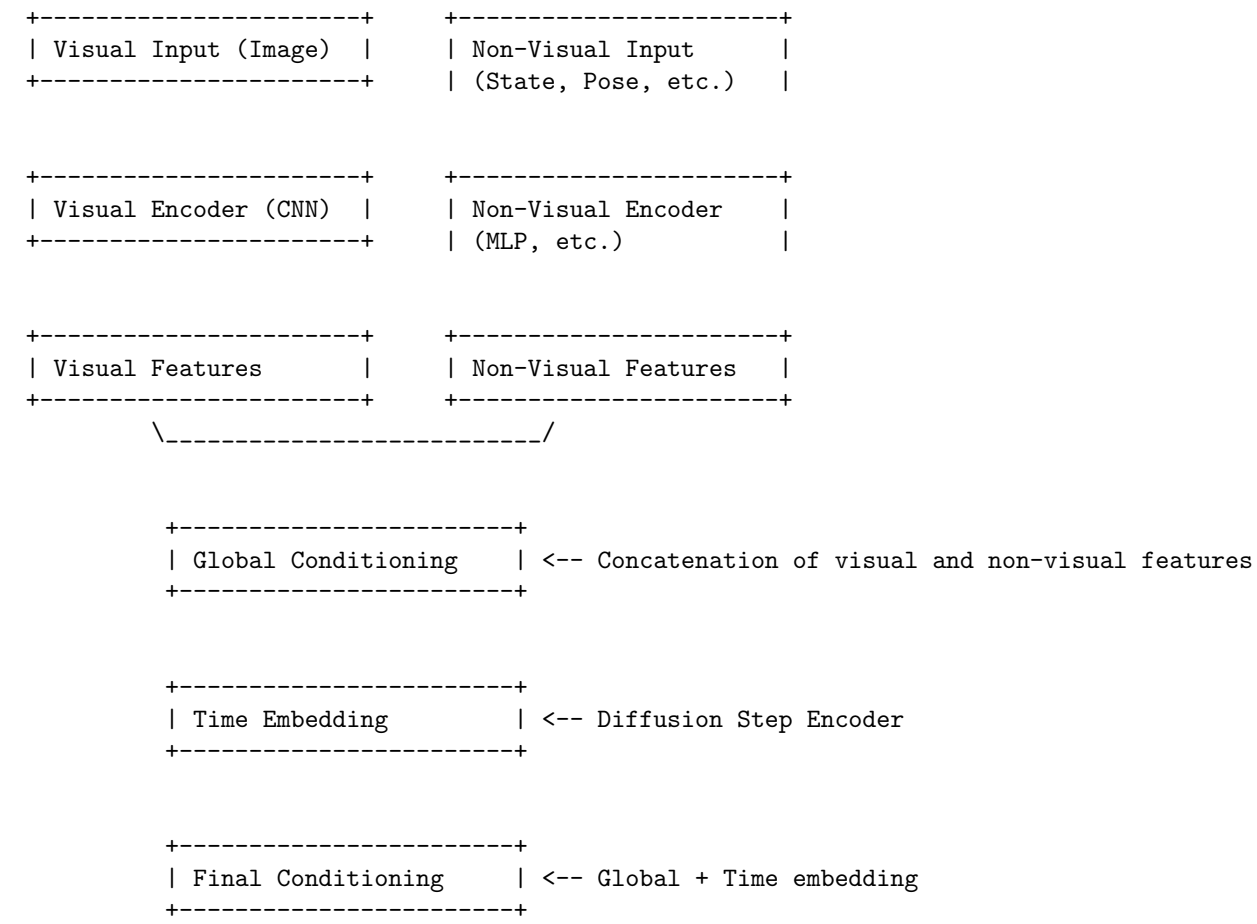
- **Noised Action Processing:** The noised action sequence is directly processed by the U-Net (via convolutional layers).
- **Conditioning Signal Processing:** Visual inputs (e.g., images) and non-visual inputs (e.g., state or pose) are processed via dedicated encoders. Their outputs, along with a time embedding from a diffusion step encoder, are concatenated into a conditioning vector that is injected into the U-Net via FiLM modules.

2 Architecture

The architecture comprises two distinct pathways:

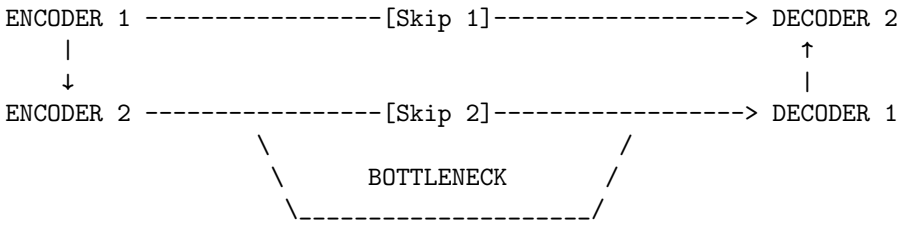
1. **Noised Action Pathway:** The noisy action sequence is processed by a U-Net that refines it over several layers of convolution.
2. **Conditioning Pathway:** Visual data is processed by a visual encoder (e.g., a CNN) and non-visual data (such as state/pose) is processed by a non-visual encoder (typically an MLP). A diffusion step encoder produces a time embedding. These outputs are concatenated to form a global conditioning vector.

Conditioning



2.1 U-Net

Here's the outline for a 2-stage U-Net.



Encoders

The encoder blocks extract hierarchical features from the noised action input. They progressively reduce the resolution, capturing more abstract representations as the network goes deeper. This dimensionality reduction helps the network focus on global structures while filtering out noise.

Residual Connections

Residual or skip connections ensure that fine-grained details, which might be lost during the downsampling in the encoders, are directly transferred to the decoders. This direct path helps in reconstructing the output with greater fidelity by reintroducing local details.

Bottleneck

The bottleneck consolidates global context and deep features extracted from the input. By processing the compressed representation further, the bottleneck forms a context-aware distillation that guides the upsampling process in the decoders. This is critical for ensuring that the denoised output maintains consistency with the overall structure of the original action.

The bottleneck is typically formed by applying one or more conditional residual blocks (similar to the ones described above) without additional downsampling. This deep processing helps in integrating information from the entire input before reconstruction.

Decoders

The decoder blocks upsample the processed representation back to the original resolution. They integrate the high-level context from the bottleneck with the detailed information provided by the skip connections. This combination enables the network to generate a refined, denoised action that accurately reflects both global and local features.

2.1.1 Integration of Conditioning via FiLM

Each encoder and decoder block internally uses conditional residual blocks to process the noised input. In these blocks, the FiLM module injects the conditioning signal (derived from visual inputs, non-visual inputs, and a time embedding) by computing per-channel scale and bias parameters. These parameters modulate the convolutional feature maps without directly mixing the conditioning data into the convolutional operations. This approach ensures that the external context influences the learning process without compromising the spatial and temporal structure of the noised action.

3 The Diffusion Process

3.1 Forward Process

Starting with a clean target x_0, noise is added progressively:

$$x_t = \sqrt{\alpha_t}\, x_{t-1} + \sqrt{1-\alpha_t}\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

This process corrupts the action until it is nearly Gaussian.

3.2 Reverse Process

The reverse process is modeled as:

$$p_\theta(x_{t-1} \mid x_t, c) = \mathcal{N}\Big(x_{t-1}; \mu_\theta(x_t, t, c),\, \Sigma_\theta(x_t, t, c)\Big),$$

where c is the conditioning signal (global conditioning $\oplus$ time embedding). The network learns to predict the noise, thereby refining the action.

3.3 Extensions

Goal conditioning (e.g goal pose for PushT) is critical for the policy to learn multi-goal scenarios.

Self-attention can improve temporal consistency for highly complex and/or long-horizon tasks.

4 Pseudocode

```
Algorithm DiffusionPolicy:
Input:
    - NoisedActionSequence X
    - Diffusion Time Step t
    - VisualInput (e.g., Image)
    - NonVisualInput (e.g., State, Pose)

Process:
    // Process conditioning inputs:
    1. VisualFeatures  <- VisualEncoder(VisualInput)
    2. NonVisualFeatures <- NonVisualEncoder(NonVisualInput)
    3. GlobalCondition <- Concatenate(VisualFeatures, NonVisualFeatures)
    4. TimeEmbed <- DiffusionStepEncoder(t)
    5. Conditioning <- Concatenate(TimeEmbed, GlobalCondition)

    // Process the noised action:
    6. PredictedNoise <- U-Net(
            InputSignal = X,
            Conditioning = Conditioning
       )

Output:
    - DenoisedAction or PredictedNoise


Module: U-Net (with Conditional Residual Blocks)
Input:
    - InputSignal: NoisedActionSequence X
    - Conditioning: Combined conditioning signal

Process:
    a. Apply Initial Convolution to project InputSignal.
    b. For each Downsampling Stage:
         i. Apply a ConditionalResidualBlock with FiLM modulation
            (using Conditioning).
         ii. Save skip connection.
         iii. Downsample via Conv1D.
    c. Apply Bottleneck Residual Blocks.
    d. For each Upsampling Stage:
         i. Upsample the current representation.
         ii. Concatenate with corresponding Skip Connection.
         iii. Apply a ConditionalResidualBlock with FiLM modulation.
    e. Apply Final Convolution to produce OutputSignal.

Output:
    - OutputSignal (Predicted Noise / Denoised Action)
```