Chair of Computer Science 6 (Machine Learning and Reasoning)

RWTH AACHEN UNIVERSITY

The present work was submitted to the Chair of Machine Learning and Reasoning.
Diese Arbeit wurde vorgelegt am Lehrstuhl für Maschinelles Lernen und Inferenz.

Action Classification Through Dynamic Scene Graph Generation From Image Sequences

Klassifikation von Aktionen durch die Generierung von dynamischen Szenengraphen aus Bildsequenzen

Bachelor Thesis
Bachelorarbeit

Presented by / Vorgelegt von

Paul Schulte
445231

Supervised by / Betreut von Ulzhalgas Rakhman, M.Sc.

1st Examiner / 1. Prüfer Univ.-Prof. Ph.D. Hector Geffner

2nd Examiner / 2. Prüfer Juniorprof. Dr. rer. nat. Christopher Morris

Aachen, June 9, 2025

Acknowledgements

I am deeply grateful to my supervisor, Ulzhalgas Rakhman, for her invaluable guidance, support, and patience throughout the development of this thesis. I also extend my sincere thanks to Prof. Hector Geffner and Prof. Christopher Morris for serving as examiners and for giving me the opportunity to pursue this work in my area of interest.

I'm grateful to my family and friends for their steady support and encouragement throughout this journey. I especially want to thank Gereon Geuchen — not only for his consistent support during my studies but also for being a great friend and companion throughout the process. Your support and understanding made a real difference.

I'm also thankful to Emily Overlack for her ongoing support and encouragement throughout the writing of this thesis. Your patience and thoughtful perspective helped me stay focused and balanced during some of the more challenging moments.

Zu guter Letzt gilt mein tiefster Dank meinen Eltern, Heike und Rainer Schulte, sowie meiner Schwester Hannah, die mich mein ganzes Leben lang unterstützt haben und bei allen Entscheidungen hinter mir standen. Ohne euer Verständnis, eure Geduld und beständige Ermutigung wäre es mir nicht möglich gewesen, diese Arbeit zu schreiben oder mein Studium zu verfolgen.

Abstract

Accurate action classification in image sequences is essential for real-world applications such as autonomous driving and human-robot interaction. While existing approaches often rely on pixel-based features effective for single-image tasks, they typically fail to capture the temporal dynamics and structural relationships inherent in dynamic scenes. This thesis focuses on improving action classification by leveraging structured graph representations that encode object interactions over time. These dynamic scene representations — generated from the image sequences — explicitly encode evolving inter-object relationships and serve as input to our action classification module. To this end, we introduce a graph-based neural architecture[1] that processes these structured inputs using a dual-head classification module — one head predicts the action type, while the second predicts the participating objects (arguments). This dual-head approach enables finer-grained action understanding by disentangling the action name from its semantic arguments across consecutive frames. We evaluate our method on synthetically generated image sequences derived from planning domain benchmarks that test spatial and temporal reasoning. Experimental results demonstrate that our approach achieves high accuracy in action classification, highlighting the benefits of structure-aware learning for sequential visual understanding.

[1]The codebase of our framework and the generated datasets will be made publicly available at https://github.com/p-schulte/ac_dsg.

Contents

1 Introduction

Deep learning has demonstrated significant accomplishments in single-image tasks such as classification, segmentation, and object detection [1, 2]. These developments have been driven by increasingly sophisticated architectures, large annotated datasets, and improved training techniques, enabling machines to match or even surpass human-level performance in many cases.

A key factor underlying this success is that single-image inputs provide only static spatial information, which allows models to focus exclusively on learning spatial patterns to identify objects and their attributes. In this context, pixel-based feature extractors such as Convolutional Neural Networks (CNNs) have demonstrated strong effectiveness in image classification and related tasks [3].

However, extending this success to image sequences and videos continues to be a substantial research challenge. Unlike single images, sequences introduce a temporal dimension, where not only object attributes but also their pairwise relationships can evolve over time. This dynamic nature requires models to capture both spatial and temporal dependencies, substantially increasing the complexity of image sequence classification tasks [4].

One popular example of an image sequence task is action classification, where the objective is to recognize and categorize actions over a series of frames. In contrast to single-image tasks, the temporal nature of image sequences introduces several unique and intricate problems, such as occlusion and motion ambiguity.

To address these and other challenges that emerge from introducing the temporal dimension, we need to develop a method that can successfully capture both the spatial and temporal relationships between objects in image sequences — ideally through a structured and semantically meaningful representation.

1.1 Motivation

In this thesis, we focus on a more fine-grained version of the aforementioned action classification task in image sequences. In contrast to conventional approaches that assign a flat label to an entire frame or clip, we aim to infer structured actions that include both the action type and the specific objects involved in each interaction (e.g., ”stack(block B, block C)”). This task plays a critical role in various real-world applications, including autonomous driving, surveillance, human-robot interaction, video analytics, and healthcare. These fields often

require interpretable and reliable predictions, as they operate in safety-critical environments where understanding the reasoning behind a decision is just as important as the decision itself. As such, developing robust methods for sequence-level visual understanding is a key step towards closing the gap between single-image analysis and dynamic scene interpretation.

However, this transition is far from trivial — structured action classification in sequences introduces unique challenges that static image models are not equipped to handle. Figure 1.1 illustrates an example from a controlled block manipulation domain, where understanding the temporal context and the evolving relationships of objects is essential. In this scenario, a robot interacts with objects, and the goal is to classify the actions that occur between consecutive frames — identifying both the action type and the participating entities. A key challenge arises when objects are temporarily invisible — for instance, ball B is hidden inside block C in frame 2, making it unobservable to the model at that moment. Nevertheless, the action ”unhide(B, C)” must still be correctly inferred between frames 2 and 3, based on information from earlier frames. Single-image classification methods would fail in this situation, as they lack access to temporal context and cannot detect ball B at all in the occluded frame. This example highlights the necessity for temporal reasoning, where models must accumulate and interpret context across time steps to infer actions that are not immediately observable from a single frame.

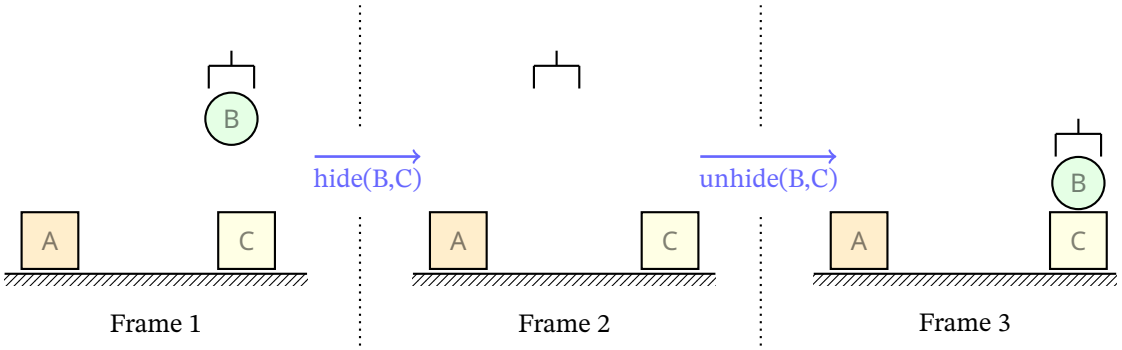


Figure 1.1: Example image sequence from a symbolic planning domain. Ball B is first placed inside block C, making it temporarily unobservable (”hide(B, C)”). In the subsequent step, it is revealed again (”unhide(B, C)”). This scenario illustrates a case of temporal occlusion, where correct action classification requires reasoning over multiple frames rather than relying on single-frame observations.

To address this challenge, we require intermediate representations that explicitly capture both spatial and temporal relationships while supporting interpretability. Scene graphs [5], and their temporally extended variant — dynamic scene graphs (DSGs) [6] — provide a structured, symbolic model of object interactions over time. These representations serve not only as interpretable summaries of image sequences but also as input to supervised learning pipelines. However, their dynamic and evolving nature poses new challenges: the graph structure itself changes over time, complicating the design of models that can leverage such structure for

fine-grained action prediction. Overcoming this limitation forms the central focus of this thesis.

In the following section, we outline the key technical challenges that arise in this setting and summarize our main contributions to addressing them.

1.2 Main Challenges and Contributions

Using DSGs for structured action classification in image sequences presents several challenges that need to be addressed to achieve effective and interpretable results. We identify three main challenges that are particularly relevant to our work:

A: **Dynamic Scene Graph Processing.** Incorporating graph structures into supervised learning tasks presents unique challenges due to their non-Euclidean nature. This is especially relevant for DSGs, where both the node set and adjacency structure evolve over time. Existing Graph Neural Network (GNN) architectures are typically optimized for static graphs, limiting their direct applicability to DSGs. For example, the Spatial-Temporal Graph Convolutional Network (ST-GCN) introduced by Yan *et al.* [7] was originally designed for human-skeleton data, where the graph structure — i.e., the adjacency matrix representing joint connections — remains fixed throughout the sequence. Such a constraint significantly restricts the model's ability to capture the dynamic inter-object relationships characteristic of DSGs. Furthermore, many GNNs assume a homogeneous edge type, which is unsuitable for DSGs that encode multiple types of semantic relationships between objects. These limitations necessitate architectural adaptations to support time-varying graph topologies and heterogeneous edge semantics to fully leverage DSGs for downstream tasks like action classification.

B: **Entity Selection for Action Argument Grounding.** Structured action classification requires not only identifying the action type but also correctly selecting the specific entities that participate in the action. This process — grounding abstract action arguments in concrete scene elements — is substantially more complex than flat action prediction. It demands that the model combines high-level semantic understanding with fine-grained spatial and relational reasoning. This challenge becomes even more intricate in dynamic scenes where objects may be temporarily occluded in certain frames. In such cases, the model must leverage temporal information and graph-based context to infer the correct arguments.

C: **Lack of Benchmarks.** While numerous datasets exist for action classification in image sequences, they primarily focus on the aforementioned flat action labels (e.g., "stack", "unstack") without specifying the interacting objects involved in each action. For instance, the Action Genome (AG) dataset [8] provides annotations of actions and scene graphs, yet it lacks structured action labels that include argument-level information (e.g., "stack(B,

C)"). Moreover, although AG includes DSG annotations, they are not aligned with symbolic representations or temporally consistent object identities required for structured reasoning. To the best of our knowledge, no existing benchmark provides both structured action annotations and aligned DSGs, making it difficult to train and evaluate models that operate on temporally evolving, graph-based representations.

To address these challenges, we propose a supervised learning framework for action classification in image sequences that leverages the semantic structure of DSGs. We enhance the ST-GCN architecture to explicitly handle the dynamics of DSGs, allowing the model to learn from temporally evolving inter-object relationships. Our approach combines symbolic reasoning with deep graph-based representations to improve both classification performance and interpretability. The main contributions of this thesis are as follows:

A: **Dynamic Scene Graph Processing.**

(i) We extend the DSG generation pipeline of Feng *et al.* [9] to incorporate not only human-object but also object-object interactions, enabling a more sophisticated and temporally-aware semantic representation of the scene.

(ii) We leverage so-called *tracklets* [9] — temporally consistent object identities that map detections to specific objects across frames — to maintain consistent node indexing within the DSG over time. This temporal alignment enables our model to effectively extract and utilize temporal context through temporal convolution.

(iii) We design an improved ST-GCN architecture capable of dynamic spatial graph convolution, allowing the model to understand the evolving inter-object relationships of the generated DSGs. This new convolution formulation is specifically tailored to handle changing adjacency matrices and heterogeneous edge types, enabling the model to effectively process DSGs as input.

B: **Entity Selection for Action Argument Grounding.** We introduce a dual-head classification module as part of our improved ST-GCN architecture that jointly predicts action types (e.g., "stack", "unstack") and their corresponding arguments, i.e., the specific objects involved. This multi-task formulation improves both the interpretability and explainability of structured action understanding.

C: **Symbolic Planning Domain Benchmarks.** To facilitate future research in structured action classification in image sequences, we develop a synthetic dataset generator based on PDDLgym [10]. Our generator produces image sequences, annotated with DSGs and lifted actions (e.g. "stack(A,B)") for classical planning domains, enabling a systematic evaluation and training of our method.

Building on these contributions, we investigate how graph-based representations of image sequences can improve action classification performance. To this end, we aim to answer the following research questions:

- **RQ1:** How can DSGs derived from image sequences be leveraged as structured, temporally-aware representations to enhance the performance of action classification?
- **RQ2:** How can we design the action classification head to improve the interpretability and explainability of the predicted actions?
- **RQ3:** How can we effectively evaluate the performance of action classification methods on structured action understanding tasks, particularly in the context of image sequences?

2 Background

This chapter reviews the theoretical and technical foundations necessary to understand the methods developed in this thesis. First, we introduce DSGs as a structured representation of image sequences. Then, we provide an overview of GNNs, with a particular focus on Graph Convolutional Networks (GCNs), which are central to modeling interactions within DSGs.

2.1 Scene Graph Representation

Understanding images often requires more than identifying individual objects — it also involves reasoning about how these objects relate to one another. Scene graphs address this challenge by providing a structured, graph-based representation of single images, where nodes denote objects, and edges encode relationships between them. This format allows models to not only detect objects in an isolated manner, but to reason about their interactions, and a higher-level context.

Formally, a scene graph is defined as a directed graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes representing detected objects, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{R} \times \mathcal{V}$ is the set of directed, labeled edges denoting pairwise relationships between them [5, 6]. Each node $v_i \in \mathcal{V}$ is a tuple $v_i = \langle \mathbf{b}_i, \mathbf{c}_i, \mathbf{f}_i \rangle$, where $\mathbf{b}_i \in [0,1]^4$ is the normalized bounding box, $\mathbf{c}_i \in \{0,1\}^{|\mathcal{O}|}$ is a one-hot vector encoding the object class from a predefined set $\mathcal{O}$, and $\mathbf{f}_i \in \mathbb{R}^C$ is a C-dimensional feature vector extracted from the image region corresponding to the bounding box [9].

Each edge $e_{ij} \in \mathcal{E}$ connects nodes v_i and v_j, and is defined as $e_{ij} = (v_i, r_{ij}, v_j)$, where $r_{ij} \in \mathcal{R}$ is the relationship type of the edge. Equivalently, for each relationship type $r \in \mathcal{R}$, the corresponding adjacency matrix $\mathbf{A}_r \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ encodes the presence or absence of relation r between each pair of nodes. That is,

$$(\mathbf{A}_r)_{ij} = \begin{cases} 1 & \text{if } (v_i, r, v_j) \in \mathcal{E}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

This structured representation jointly captures the entities present in an image and the semantic relations between them, supporting higher-level reasoning over the scene's composition.

To enable structured representations of image sequences, static scene graphs are extended into DSGs. A DSG, denoted as $\mathcal{G}_{\text{dyn}} = \{G_t = (\mathcal{V}_t, \mathcal{E}_t)\}_{t=1}^{T}$, is a temporally indexed sequence of static scene graphs, where $\mathcal{V}_t$ and $\mathcal{E}_t$ denote the set of nodes and edges at time step $t \in [T]$ for a horizon of length $T \in \mathbb{N}$, respectively [6]. In this formulation, each frame is modeled independently via its scene graph, while the overall DSG captures the temporal dynamics and

evolving structure of the scene across the sequence. Figure 2.1 illustrates an example of a DSG consisting of three timesteps, showing how the object relationships develop over time.

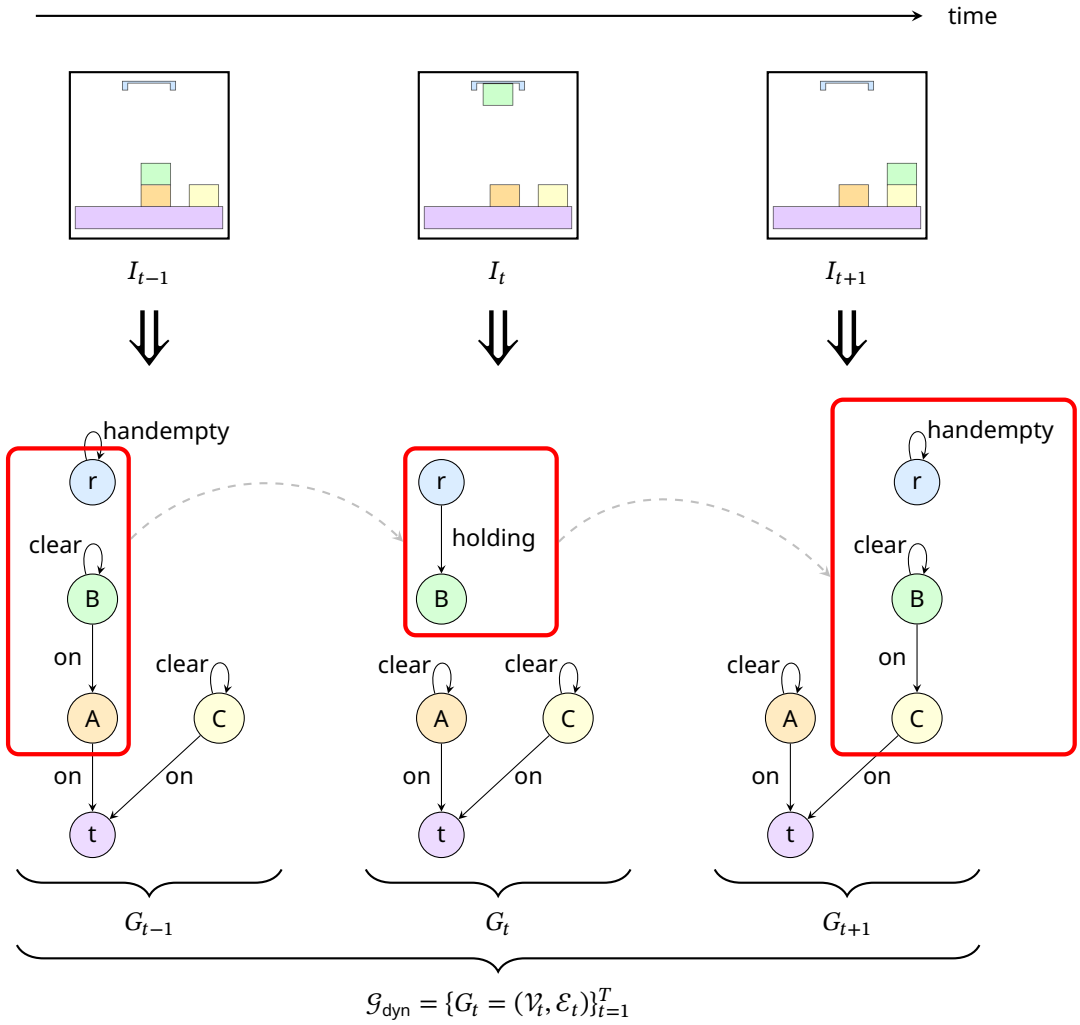


Figure 2.1: Illustration of a DSG over three consecutive image frames I_i for $i \in \{t-1, t, t+1\}$. Each frame is represented by a static scene graph $G_i = (\mathcal{V}_i, \mathcal{E}_i)$, where nodes denote objects and edges encode their relationships. The nodes labeled `t` and `r` correspond to the table and robot, respectively, while `A`, `B`, and `C` represent blocks. Initially, the robot is "handempty", and block `B` is stacked on block `A`. In the next frame, the robot picks up `B`, and finally, places it on block `C`. Dashed arrows indicate the temporal correspondence of the highlighted nodes across frames. The highlighted nodes in each graph mark the objects whose relational contexts change across the sequence.

2.2 Graph Convolutional Networks

Many neural methods like CNNs only perform well on Euclidean, fixed-size inputs and struggle with non-Euclidean data such as scene graphs, a limitation addressed by GNNs [6, 11]. The main idea behind GNNs is to iteratively update node representations by aggregating information

from neighbors. Various aggregation methods are used in GNNs, and in this section, we focus on GCNs, being one of the foundational and widely used GNN models due to their simplicity and effectiveness. Other popular GNN architectures include GraphSAGE [12], Graph Attention Network (GAT) [13], and Message Passing Neural Networks [14], each with its unique aggregation functions and design principles.

Graph Convolutional Layers. GCNs update node representations by aggregating information from local neighborhoods [15]. The update rule for a single GCN layer is given by:

$$\mathbf{X}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}^{(l)}\mathbf{W}^{(l)}\right), \tag{2.2}$$

where $\mathbf{X}^{(l)}$ is the node feature matrix at layer l, with $\mathbf{X}^{(0)}$ representing the input features. The adjacency matrix $\mathbf{A}$ is augmented with self-loops by adding the identity matrix $\mathbf{I}_N$, yielding $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$. To stabilize training and prevent numerical instabilities such as vanishing or exploding features, the adjacency matrix is symmetrically normalized using the degree matrix $\tilde{\mathbf{D}}$, where $\tilde{\mathbf{D}}_{i,i} = \sum_j \tilde{\mathbf{A}}_{i,j}$. $\mathbf{W}^{(l)}$ is a learnable weight matrix for layer l, and σ denotes a non-linear activation function, such as ReLU. Finally, $\mathbf{X}^{(l+1)}$ is the updated feature matrix at layer $l+1$.

This formulation allows GCN layers to propagate and integrate information across fixed graph structures, capturing node attributes and local neighborhoods. After stacking multiple such layers, the model produces node-level embeddings, which can be used for downstream tasks such as node or graph classification [15].

However, standard GCNs face several limitations when applied to more complex graph structures. First, they assume a fixed graph topology, which requires the number and connectivity of nodes to remain constant throughout the aggregation. This constraint poses challenges when applying GCNs on dynamic graph structures, such as DSGs, where the nodes and edges can vary over time. Second, classical GCN formulations typically handle only homogeneous edge types, i.e., all edges are treated equally during message passing. This is insufficient for heterogeneous graphs like DSGs, which encode different semantic relationships — such as ”on”, ”clear”, or ”holding” — as distinct edge types. These limitations motivate the development of more expressive models capable of handling temporally evolving and semantically rich graph structures.

3 Related Works

This section reviews foundational and recent works on DSG generation and action classification in image sequences. We will discuss the limitations of existing methods and highlight the significance of our proposed approach in addressing these challenges.

3.1 Scene Graph Generation

Scene graphs, first proposed by Johnson *et al.* in 2015 [5], have since been the focus of extensive research, leading to the development of numerous approaches for their generation. Current scene graph generation (SGG) methods can be broadly categorized into two groups: *two-stage methods* and *one-stage methods* [16]. Two-stage methods first detect bounding boxes and objects, followed by the detection of pairwise relationships [6, 9]. In contrast, one-stage methods simultaneously detect and recognize objects and their relationships in a unified process [17].

3.1.1 Static Scene Graph Generation

Early works on static scene graph generation aimed to detect and classify objects and their respective relationships from single images. Xu *et al.* proposed to leverage a Recurrent Neural Network (RNN) to perform iterative message passing over detected objects in images and their pairwise relationships [18]. This enables joint reasoning over objects and interactions, instead of treating them as independent entities.

Zellers *et al.* highlighted the importance of repetitive subgraphs (motifs) in scene graphs, demonstrating that relationships are highly predictable given the object categories present in the scene [19]. Building on this insight, they proposed the Stacked Motif Network (MOTIFNET) architecture, which leverages Long Short-Term Memory (LSTM)-based global context encoding to classify objects and, in particular, pairwise relationships. Their architecture implicitly captures higher-order motifs, improving the model's ability to capture recurring relational patterns within complex visual scenes.

Yang *et al.* introduced a Relation Proposal Network (RePN) that efficiently reduces the quadratic relationship search space by scoring and selecting likely object pairs based on their feature representations [20]. To refine predictions, they apply an attentional GCN, which selectively propagates contextual information across the graph to capture higher-order dependencies.

This design reduces the computational complexity of relationship inference and enables more scalable and accurate scene graph generation.

Following these advances in static scene graph generation, research has progressively shifted towards DSG generation, which aims to capture dynamic relationships and interactions between objects across sequences of images.

3.1.2 Dynamic Scene Graph Generation

Most DSG generation methods extend the principles of static scene graph generation to image sequences. Typically, an initial prediction is made for each frame individually, which is subsequently refined using recurrent architectures — such as RNNs, LSTMs, or transformers [21] — to capture temporal dependencies and ensure classification consistency over time.

Teng *et al.* introduced TRACE, a detect-to-track framework designed to decouple relation prediction from the complexities of low-level entity tracking [22]. By isolating context modeling for relation prediction, TRACE reduces sensitivity to tracking error propagation. However, the method remains limited to short-term dependencies and suffers from the *aggregation problem*, where conflicting predictions for the same frame arise from different image sequence segments.

Cong *et al.* proposed STTran, a baseline model that employs a spatial encoder and a temporal decoder to extract spatial-temporal contexts implicitly [6]. This approach highlights the importance of temporal correlations for inferring dynamic visual relationships. However, STTran relies heavily on adjacent keyframes, limiting its ability to ensure long-term consistency and fully capture extended temporal dependencies.

STKET improves upon STTran by explicitly embedding spatial and temporal priors into the attention mechanism [23]. This allows it to better capture spatial-temporal relationships, but like STTran, it still struggles with long-term consistency, as it relies on adjacent keyframes.

To address these limitations, the Dynamic Scene Graph Detection Transformer (DSG-DETR) introduces inter-frame trajectory construction for object instances and pairs using bipartite graph matching, i.e. aligning objects and their relationships across different frames, enhancing long-term temporal dependencies in video data [9]. By building on the STTran framework, DSG-DETR significantly improves multi-relation classification performance and demonstrates the benefits of modeling extended temporal relationships.

Recent work by Wang *et al.* proposed OED [17], which directly integrates object detection and relationship reasoning, further advancing one-stage SGG methods. OED employs a unified transformer-based architecture that uses a Progressive Refined Interaction Module (PRM). This module incrementally selects and aggregates relevant information across frame sequences, enabling the model to efficiently process DSG generation tasks in an end-to-end manner.

Despite the simplicity and efficiency of OED as a single-step approach, our work primarily relies on the DSG-DETR. We hypothesize that temporal object consistency plays a critical role in understanding semantically complex relationships, essential for accurate DSG generation in the context of action classification. DSG-DETR's use of inter-frame trajectory construction and bipartite graph matching enables it to model extended temporal dependencies, making it well-suited for this task.

3.2 Action Classification in Image Sequences

Action classification in image sequences presents significant challenges, as outlined in Section 1.1. To address these, previous research has explored a wide range of approaches. They range from low-level pixel-based methods to techniques that incorporate structured intermediate representations, such as scene graphs. In this section, we review both directions and argue in favor of structured representations as a more effective solution for our task.

3.2.1 Action Classification via Pixel-Based Features

Many state-of-the-art action classification models rely on direct processing of pixel-based features extracted from raw video frames [4, 24]. These methods typically use convolutional or transformer-based architectures to encode spatio-temporal patterns from RGB frames or optical flow — often without explicit scene-level reasoning.

While such models have demonstrated strong performance on large-scale benchmarks like Kinetics-400 [25], they face significant limitations when applied to scenarios requiring fine-grained object interactions or long-term temporal dependencies. One fundamental challenge lies in the difficulty of maintaining consistent representations of individual objects across frames, especially when visual appearance varies due to occlusion, motion blur, or changes in viewpoint. This hinders the model's ability to reason about object identity and track inter-object relationships over time. Additionally, the absence of structured representations often leads to a lack of interpretability, making it difficult to analyze why certain actions were predicted. Pixel-based models also generalize poorly in complex scenes with multiple interacting entities, where understanding actions demands more than visual texture or motion cues.

In contrast, our approach introduces an intermediate structured representation in the form of DSGs, which captures both spatial and temporal relationships between objects across frames. By first generating DSGs using the DSG-DETR framework [9], we explicitly separate object detection and interaction modeling from the downstream action classification task. This allows our graph-based classifier to focus on high-level semantic reasoning — tracking the involved entities and how their relationships evolve over time. We hypothesize that this explicit relational modeling leads to better generalization, interpretability, and performance in action

classification, especially in complex domains where interactions between objects are crucial for understanding behavior.

3.2.2 Action Classification via Structured Representations

In contrast to pixel-based methods, recent work increasingly explores structured representations to improve action classification in image sequences. These methods first construct an intermediate representation that encodes the context of the scene, which is then used as input to a downstream classification model.

Kochakarn *et al.* proposed a self-supervised learning framework to learn embeddings from sequences of scene graphs using an encoder network with attention mechanisms [11]. They evaluated these embeddings and attention masks for future driver-action classification on real-world data and used an LSTM-based decoder to predict the next action class. Their approach emphasizes interpretability through attention masks and structured reasoning, though it does not directly address DSG generation from images.

Yan *et al.* introduced ST-GCN to classify actions based on skeleton data, combining spatial and temporal graph convolution [7]. Their model effectively captures spatio-temporal dependencies in skeleton sequences but its reliance on fixed graphs limits its applicability to more complex, object-centric scenes.

A key limitation of both approaches [7, 11] is that they classify actions from a fixed set of labels without identifying the involved entities (e.g., objects or nodes). Our proposed method addresses this gap by leveraging DSGs generated via the DSG-DETR model and applying a GNN-based architecture for action classification. These DSGs enable not only the prediction of action types but also the identification of the relevant objects involved in each action. By capturing object interactions and their temporal evolution, our approach offers a more accurate and interpretable representation of dynamic scenes. This allows us to overcome limitations such as suboptimal action modeling [7, 11] and lack of interpretability (cf. Section 3.2.1) while enabling fine-grained, entity-aware action classification in dynamic scenes.

Together, these gaps motivate our use of DSGs and GNNs to perform structured, temporally consistent action classification that goes beyond label prediction to identify the objects involved.

4 Approach

After formally introducing the problem formulation for action classification in image sequences, this chapter presents the foundational concepts underlying our method and details the design of the proposed framework.

4.1 Problem Formulation

The objective of this work is to classify structured symbolic actions of arity $k \in \mathbb{N}$ that occur between consecutive frames in an image sequence. Each action is defined not only by an action type (e.g., "stack", "pickup") but also by an ordered set of k participating entities (e.g., "stack(B, C)"). In contrast to forecasting approaches such as [11], which predict future actions, our focus lies on retrospective action classification. By leveraging symbolic representations of scenes, we classify past transitions between frames in terms of both their type and participants.

Formally, given a sequence of images $\mathcal{I} = \{I_1, I_2, \dots, I_T\}$ for a fixed time horizon $T \in \mathbb{N}$ as well as a finite set of labels for object types $\mathcal{O}$, relationship types $\mathcal{R}$, and action types $\mathcal{A}$, the goal is to classify, for each time step $t \in \{1, \dots, T\}$, the action type $a_t \in \mathcal{A}$ that occurred between the images I_t and I_{t+1}, as well as the set of involved entities $\mathcal{V}_t^{\text{involved}} \subseteq \mathcal{O}$ that participated in the action. The final output is a symbolic structured action prediction of the form $a_t(v_1, \dots, v_k)$, where a_t is the predicted action type, and $v_1, \dots, v_k \in \mathcal{V}_t^{\text{involved}}$ are the identified entities involved in the action. This problem can be split into the two following subproblems:

Subproblem 1: DSG Generation. We define a function

$$f_{\text{DSG}} : (\mathcal{I}, \mathcal{O}, \mathcal{R}) \mapsto \left(\mathcal{G}_{\text{dyn}} = \{G_t = (\mathcal{V}_t, \mathcal{E}_t)\}_{t=1}^{T},\ \mathbf{F}\right), \tag{4.1}$$

where each scene graph G_t represents the symbolic scene structure of frame I_t, with nodes $\mathcal{V}_t \subseteq \mathcal{O}$ representing detected objects and edges $\mathcal{E}_t \subseteq \mathcal{R}$ representing predicted relationships. The matrix $\mathbf{F} \in \mathbb{R}^{T \times |\mathcal{V}_t| \times C}$ denotes the visual feature representations extracted by an object detector for each detected object in every frame.

Subproblem 2: Action Classification From DSGs. Furthermore, we define a second function

$$f_{\text{action}} : (G_t, G_{t+1}, \mathcal{A}, \mathbf{F}) \mapsto \left(a_t, \mathcal{V}_t^{\text{involved}}\right), \tag{4.2}$$

that classifies the action occurring between frames I_t and I_{t+1}. The output consists of the predicted action type $a_t \in \mathcal{A}$ and an ordered set of involved entities $\mathcal{V}_t^{\text{involved}} \subseteq \mathcal{V}_t$ of cardinality k.

The overall action classification task is thus decomposed into learning the two functions f_{DSG} and f_{action}, which are trained independently in a supervised learning setting. The first function, f_{DSG}, operates directly on the image sequence $\mathcal{I}$ to generate a DSG. The second function, f_{action}, is trained on the output of f_{DSG} and performs structured action classification based on the extracted scene representations. Both functions are supervised using annotated datasets that provide ground-truth scene graphs and action labels, respectively.

To reduce ambiguity in determining which entities are involved in an action, we introduce the following constraint: At each time step t, only a single object is actively moved between frames. This simplifying assumption reflects the structure of the environments we consider for evaluation, where actions are atomic and involve manipulating one object at a time. It ensures that action labels and participating entities can be clearly attributed during both annotation and prediction.

4.2 Preliminaries

Before introducing our proposed approach, we briefly review the two key components it builds upon: the DSG-DETR [9] and ST-GCNs [7]. The DSG-DETR is used for DSG generation, while ST-GCNs are employed for action classification.

4.2.1 Dynamic Scene Graph Detection Transformer

The objective of the DSG-DETR method, proposed by Feng *et al.*, is to construct DSGs that capture both spatial and temporal dependencies across sequences of images. The model combines object detection with a transformer-based refinement pipeline consisting of three main stages: (1) object detection in individual frames, (2) temporal refinement of object classifications through tracklet construction, and (3) pairwise relationship prediction via a dedicated transformer module. This section highlights the central innovation of DSG-DETR — tracklet construction — which enables temporally consistent object representation across frames. In our framework, we additionally leverage these tracklets to ensure consistent indexing in the adjacency matrices of the generated DSGs. An overview of the complete pipeline is provided in Figure 4.1.

Object Detection. In the first stage, the DSG-DETR employs a pre-trained Faster R-CNN model [1] with a ResNet-101 backbone [26] to detect objects independently in each frame $I_i \in \mathcal{I}$ of the input sequence. For every detected object, the model outputs a tuple $\langle \mathbf{b}_i, \tilde{\mathbf{c}}_i, \mathbf{f}_i \rangle$, where $\mathbf{b}_i \in [0,1]^4$ denotes the normalized bounding box in the format $[x_{\min}, y_{\min}, x_{\max}, y_{\max}]$,

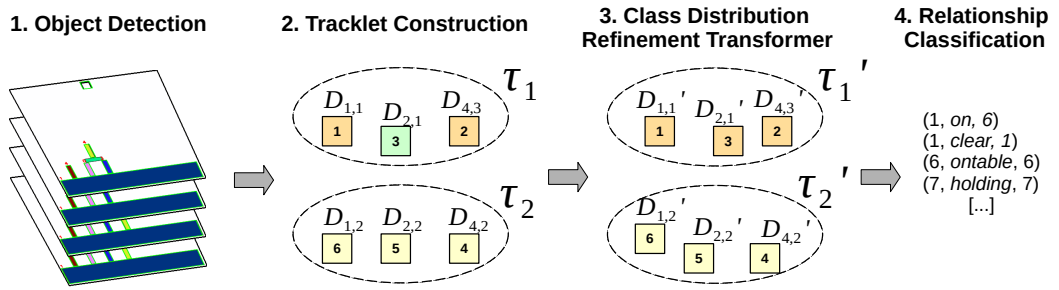


Figure 4.1: Illustration of the DSG-DETR architecture, adapted from [9]. In the first step, the model detects objects in each frame of the input image sequence. Then, it constructs object tracklets by linking detections across frames. The object transformer refines the predicted class distributions of the detected and classified objects. Finally, the relationship transformer predicts pairwise relationships between the detected objects.

$\tilde{\mathbf{c}}_i \in [0,1]^{|\mathcal{O}|}$ represents the class probability distribution over the set of object classes $\mathcal{O}$, and $\mathbf{f}_i \in \mathbb{R}^C$ is a high-dimensional appearance feature vector extracted from the final layer of the ResNet-101 backbone. The problem with these per-frame detections is the decoupled nature of the prediction, which can lead to inconsistencies in object classification across frames. To address this, the DSG-DETR introduces a refinement mechanism that links detections across frames, ensuring coherent object identities over time.

Tracklet Construction. To achieve temporally consistent object classification, Feng *et al.* proposed to construct tracklets by linking detections across frames. This association is performed using the Hungarian matching algorithm [27], which finds an optimal assignment between current frame detections and previously established tracklets based on a matching cost. Formally, the assignment $\sigma_i(j) = k$ indicates that the j-th detection in frame i is associated with the k-th tracklet.

Each detection is described by a tuple $D_{ij} = \langle \mathbf{b}_{ij}, \tilde{\mathbf{c}}_{ij}, \mathbf{f}_{ij} \rangle$ consisting of its bounding box, class distribution, and visual features, while the full detection set for frame i is denoted as $D_i = \{D_{ij} \mid j\}$. A tracklet $T_{i,k}$ comprises all previous detections matched to the k-th tracklet, i.e.,

$$T_{i,k} = \{D_{i'j} \mid i' \leq i, j \leq |D_{i'}|, \sigma_{i'}(j) = k\}, \tag{4.3}$$

while the set of all tracklets up to frame i is defined as $T_i = \{T_{i,k} \mid k\}$. To describe each tracklet, the latest bounding box $\hat{\mathbf{b}}_{i,k}$ is retained, while the average class distribution and visual features

are computed as:

$$\hat{\mathbf{c}}_{i,k} = \frac{1}{|T_{i,k}|} \sum_{i'=1}^{i} \sum_{j=1}^{|\mathcal{D}_{i'}|} \mathbb{1}_{[\sigma_{i'}(j)=k]} \tilde{\mathbf{c}}_{i'j}, \text{ and} \tag{4.4}$$

$$\hat{\mathbf{f}}_{i,k} = \frac{1}{|T_{i,k}|} \sum_{i'=1}^{i} \sum_{j=1}^{|\mathcal{D}_{i'}|} \mathbb{1}_{[\sigma_{i'}(j)=k]} \mathbf{f}_{i'j}. \tag{4.5}$$

The optimal assignment $\sigma_i(\cdot)$ minimizes a cost function $\mathcal{L}_{\text{HM}}$, which combines the deviation of the predicted class distributions, appearance features, and bounding boxes between every tracklet and the new detections of frame i:

$$\sigma_i := \arg\min_{\sigma_i' \in \mathfrak{S}_{n_i}} \mathcal{L}_{\text{HM}}(\mathcal{D}_i', T_{i-1}'), \tag{4.6}$$

where

$$\begin{aligned} \mathcal{L}_{\text{HM}}(\mathcal{D}_i', T_{i-1}') = \sum_{j=1}^{|\mathcal{D}_i|} \mathbb{1}_{\left[T'_{(i-1)\sigma_i(j)} \neq \varnothing\right]} \Big[& \left(1 - \cos\left(\tilde{\mathbf{c}}'_{ij}, \hat{\mathbf{c}}'_{(i-1)\sigma_i(j)}\right)\right) \\ & + \lambda_{\text{feat}} \left(1 - \cos\left(\mathbf{f}'_{ij}, \hat{\mathbf{f}}'_{(i-1)\sigma_i(j)}\right)\right) \\ & + \lambda_{\text{iou}} \mathcal{L}_{\text{iou}}\left(\mathbf{b}'_{ij}, \hat{\mathbf{b}}'_{(i-1)\sigma_i(j)}\right) + \lambda_{L_1} \left\| \mathbf{b}'_{ij} - \hat{\mathbf{b}}'_{(i-1)\sigma_i(j)} \right\|_1 \Big]. \end{aligned} \tag{4.7}$$

That is, $\mathcal{L}_{\text{HM}}$ aggregates four terms: a classification loss based on the cosine similarity between class distributions, a feature loss based on the cosine similarity of appearance embeddings, and two bounding box losses — one using the Intersection-over-Union (IoU) [28] and the other using the standard L1 distance. The set $\mathfrak{S}_{n_i}$ denotes all possible permutations of length n_i, which corresponds to the number of detections in frame i.

To ensure that each detection can be matched with a corresponding tracklet — even in cases where the sets differ in size — the detection set $\mathcal{D}_i$ and the previous tracklet set $T_{(i-1)}$ are each padded with dummy entries $\varnothing$ such that $|\mathcal{D}_i'| = |T'_{(i-1)}|$.

The non-negative weights λ_{feat}, λ_{iou}, and λ_{L1} control the relative contribution of each individual term to the total cost. Once the optimal assignment $\sigma_i(\cdot)$ is computed via this cost minimization, any unmatched detections are used to initiate new tracklets. The resulting tracklet set is then updated by appending the assigned detections to their corresponding tracklet histories.

To improve classification consistency within each tracklet, the DSG-DETR applies a transformer-based refinement module. By aggregating bounding box, class distribution, and appearance features across all elements of a tracklet, this object transformer refines noisy or ambiguous predictions. In doing so, it corrects misclassifications based on the temporal context, ensuring coherent object identity across frames.

Relationship Prediction. To infer inter-object relations, the DSG-DETR constructs relationship embeddings by combining visual, spatial, and semantic features of object pairs. These embeddings are first processed within each frame using a spatial transformer, and then temporally aggregated across frames via a second transformer. The resulting representations are used to classify relationship predicates, enabling the construction of temporally coherent DSGs.

4.2.2 Spatial-Temporal Graph Convolutional Networks

Yan *et al.* proposed ST-GCNs to model spatial and temporal dependencies of graphs in action classification tasks. Their research focuses on skeleton-based action recognition, where the input is a sequence of skeleton graphs. Each graph represents a single frame, and the nodes correspond to body joints. The edges represent the spatial relationships between these joints. The goal is to classify the action performed by the human in the sequence of frames. After representing the skeleton as a graph, Yan *et al.* proposed to use 9 layers of graphical and temporal convolution to extract meaningful features from the graph for action classification.

Graph Representation. Here, $G_{\text{skeleton}} = (\mathcal{V}, \mathcal{E})$ describes the *fixed* graph representing the human body, where $\mathcal{V}$ is the set of nodes (joints) and $\mathcal{E}$ is the set of edges (connections between joints). The node features $\mathbf{X} \in \mathbb{R}^{T\times|\mathcal{V}|\times C_l}$ for a time horizon of length T and size C_l in layer l initially consist of joint position and class. Since the graph structure is fixed, the adjacency matrix $\mathbf{A} \in \{0,1\}^{K\times|\mathcal{V}|\times|\mathcal{V}|}$ is also fixed. K describes the number of partitions of the adjacency matrix, meaning that every node's neighborhood is partitioned into K structurally meaningful subsets before the convolution to learn different weights for different types of neighbors. Therefore, before applying the convolution, the adjacency matrix is weighted with a learnable attention matrix $\mathbf{U}^{(l)} \in \mathbb{R}^{K\times|\mathcal{V}|\times|\mathcal{V}|}$:

$$\tilde{\mathbf{A}}^{(l)} = \mathbf{A} \odot \mathbf{U}^{(l)}, \tag{4.8}$$

where $\odot$ is the element-wise product and $l \in [L]$ is the layer index. These weights allow the model to learn the importance of the different fixed edges in the graph.

Spatial Graph Convolution. To efficiently process the constructed node features $\mathbf{X}$ and adjacency matrix $\tilde{\mathbf{A}}$, ST-GCNs employ spatial graph convolution, aggregating information among the neighborhoods of each node. At first, the input features are partitioned using K subsets of the adjacency matrix:

$$\mathbf{Z}^{(l)}_{t,k,v,c'} = \sum_{c=1}^{C_l} \sum_{\tau=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor} \mathbf{W}^{(l)}_{k,c',c} \cdot \mathbf{X}^{(l)}_{t+\tau,v,c}, \tag{4.9}$$

where $\mathbf{Z}^{(l)} \in \mathbb{R}^{T\times K\times|\mathcal{V}|\times C_{(l+1)}}$, and $\mathbf{W}^{(l)} \in \mathbb{R}^{K\times C_{(l+1)}\times C_l}$ is a learnable weight matrix. This allows the model to learn different patterns for different graph partitions. This augmented

feature representation matrix $\mathbf{Z}^{(l)}$ is then used to perform the spatial graph convolution, which aggregates information from the neighbors of each node. The spatial graph convolution is defined as follows:

$$\mathbf{X}_{t,v}^{(l+1)} = \sum_{k=1}^{K} \sum_{u=1}^{|\mathcal{V}|} \mathbf{Z}_{t,k,u}^{(l)} \cdot \tilde{\mathbf{A}}_{k,u,v}^{(l)}, \tag{4.10}$$

where $\mathbf{X}_{t,v}^{(l+1)}$ is the updated feature vector of node v at layer $l+1$ and timepoint t. Applying spatial graph convolution allows the model to learn spatial dependencies between nodes.

Temporal Graph Convolution. After capturing the spatial relationships, Yan *et al.* introduced a temporal graph convolution module to model dependencies across time. This is implemented via a 2D convolution applied to the spatially aggregated node features:

$$\mathbf{X}^{(l+1)} = \text{Conv2D}\big(\mathbf{X}^{(l+1)}\big). \tag{4.11}$$

Here, the 2D convolution operates across the temporal and channel dimensions for each node individually, treating its features over time as a local temporal neighborhood. This enables the model to learn motion dynamics and temporal patterns across consecutive frames. The result is a refined node feature representation $\mathbf{X}^{(l+1)}$, which captures both spatial and temporal context and can be used for downstream tasks such as action classification.

Action Type Prediction. Before classifying action types, the node features are pooled using global average pooling to obtain a single, graph-level embedding. This graph-level embedding is then passed through a Multi-Layer Perceptron (MLP) to perform the final action classification. It is important to note that the model is designed for *fixed* graphs, where the number of nodes and type of edges are constant. This is a key difference to our approach, where we aim to learn a *dynamic* graph representation that can adapt to the changing relationships between objects in the scene. Furthermore, the action set is flat, representing the action classification task as a multi-class classification problem that does not consider which objects are involved in the action.

4.3 Proposed Method

In this section, we present our proposed method for the structured action classification task, detailing the benchmark generation method (Section 4.3.1), the framework overview (Section 4.3.2), and the training and testing pipeline (Section 4.3.3).

4.3.1 Image Sequence Dataset Generation

To evaluate the performance of our proposed model, we generate synthetic datasets composed of image sequences derived from symbolic planning domains. The dataset generation process

comprises three main steps: random problem instance generation, symbolic planning, and scene rendering, each described in detail below. An overview of the proposed dataset generation pipeline is illustrated in Algorithm 1.

Algorithm 1 Synthetic Image Sequence Dataset Generation Pipeline

Require: PDDL domain definition $\mathcal{D}$, number of sequences N
Ensure: Annotated image sequences $\mathcal{I}_1, \ldots, \mathcal{I}_N$ with action and DSG annotations $\mathcal{A}_1, \ldots, \mathcal{A}_N$

$\mathcal{I} \leftarrow \emptyset, \mathcal{A} \leftarrow \emptyset$ *Initialize images and annotations*
for $i = 1$ to N **do**
  $\mathcal{Q}_i, \mathcal{G}_i \leftarrow$ `SampleRandomProblem`$(\mathcal{D})$ *Random problem instance generation*
  $\mathcal{P}_i \leftarrow$ `Planner`$(\mathcal{Q}_i, \mathcal{G}_i)$ *Fast Downward planner*
  $\mathcal{I}_i \leftarrow \emptyset, \mathcal{A}_i \leftarrow \emptyset$
  for each action $a_t \in \mathcal{P}_i$ **do**
    $Q_{t+1} \leftarrow$ `SimulateAction`(Q_t, a_t) *State transition*
    $I_{t+1} \leftarrow$ `RenderScene`(Q_{t+1}) *Rendering (with Blender-3D)*
    $A_{t+1} \leftarrow$ `Annotate`(a_t, Q_{t+1}) *Generate annotations for DSG and action*
    Append I_{t+1} to $\mathcal{I}_i$ and A_{t+1} to $\mathcal{A}_i$
  Append $\mathcal{I}_i$ to $\mathcal{I}$ and $\mathcal{A}_i$ to $\mathcal{A}$
return $\mathcal{I}$ and $\mathcal{A}$ *Return generated sequences and annotations*

For symbolic planning and rendering, we leverage PDDLgym [10], a framework that supports state modeling, action execution, and scene rendering based on Planning Domain Definition Language (PDDL) [29] domain definitions. PDDLgym offers two key advantages: it enables the generation of datasets with precise action annotations and corresponding DSGs, and it supports a wide range of PDDL domains. This flexibility allows us to evaluate our model across multiple environments, including Blocksworld [30], Towers of Hanoi [31], and custom-designed domains, facilitating a more comprehensive evaluation of our framework. An example PDDL domain file for a custom-designed domain is provided in Listing A.1.

By generating the datasets synthetically and controlling the ground-truth scene graphs, we can systematically analyze our model's performance under varying conditions and conduct detailed ablation studies to understand the model's behavior better. Furthermore, to the best of our knowledge, no publicly available benchmark exists for the structured action classification problem on image sequences. To address this gap, we intend to release our generated datasets along with the PDDLgym-based generator to support further research in this area.

Random Problem Instance Generation. To introduce diversity and complexity into the generated datasets, we randomly generate problem instances based on the given PDDL domain definition for each environment. For example, in the Blocksworld domain, the number of blocks $n \in [l, h]$ is sampled uniformly at random from a specified range, where l and h denote the lower and upper bounds, respectively. Additionally, the number of piles on which blocks can be stacked is sampled uniformly from a range between l' and h', further controlling the spatial structure of the scene. A random subset of these blocks is placed directly on the table,

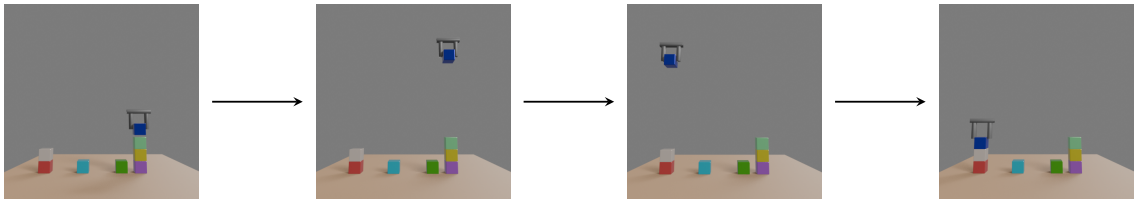
while the remaining blocks are stacked on top. A goal of the form ”and(on(A, B), ..., on(C, A))”, where A, B, and C denote block identifiers, is then selected at random. The generated problem instance is compiled into a PDDL file containing the initial state, the goal state, and the available actions. This instance is subsequently passed to PDDLgym for symbolic planning and rendering. This procedure is repeated for every image sequence in the dataset, offering the possibility to generate a large number of sequences with varying complexity and object configurations.

Symbolic Planning. Starting from the randomly generated initial state, the objective is to pick and execute actions a_t (cf. Section 4.1) until the goal state is reached. Each environment state S_t is represented as a tuple $(\mathcal{O}_t, \mathcal{R}_t)$, where $\mathcal{O}_t$ denotes the set of objects and $\mathcal{R}_t$ the relationships between them. Finding a valid sequence of actions $a_1, \dots, a_T$ that leads to the goal state is a planning problem, which we solve using the Fast Downward planner [32], implemented in PDDLgym. Fast Downward employs heuristic search strategies and advanced planning techniques, whose details are beyond the scope of this thesis.

PDDLgym’s simulation environment initializes with the generated problem instance and sequentially executes the action sequence that was generated by the planner. After each action a_t, the simulator updates the environment state and generates a corresponding image frame I_t reflecting the current state. This process continues until the goal is reached, resulting in a sequence of frames $\mathcal{I} = \{I_1, I_2, \dots, I_T\}$. Throughout the simulation, we record object positions, inter-object relationships, and the executed actions to create structured ground-truth annotations.

Scene Rendering. While PDDLgym includes a built-in 2D renderer, the resulting images are overly simplistic and provide little visual challenge. To make the visual representations more advanced, we employ Blender, an open-source 3D graphics software, following a similar approach as [33]. We develop a custom rendering engine that translates the symbolic state representations from PDDLgym into 3D visualizations, producing images that more accurately reflect the environments.

Moreover, PDDLgym natively supports only discrete state rendering, where one image corresponds to one symbolic state. However, for our purposes, we require continuous image sequences that capture intermediate transitions between discrete states to introduce additional ambiguity and increase task complexity. To address this, we apply linear interpolation to the objects’ positions between consecutive states and render images for these intermediate configurations. To further enhance temporal realism, we introduce synthetic motion primitives — additional low-level actions not defined in the original PDDL domain — that model continuous transitions between symbolic states. An excerpt from a generated sequence is shown in Figure 4.2.



I_1 I_2 I_3 I_4

Figure 4.2: Example sequence of four frames generated during the dataset creation process for the Blocksworld domain. The blue block is being picked up by the robot in the first frame, moved to the left pile in the second and third frame, and finally placed on the white block in the fourth frame. The generated sequence comprises a total of 30 frames, the four shown frames are selected to illustrate the action sequence.

Annotation Processing. To enable supervised learning, it is essential to annotate each frame with ground-truth labels, including the corresponding DSG. For each frame, we store the objects' positions, the inter-object relationships, and the action executed since the previous frame, including the involved objects. The action labels and arguments are derived from the planned action sequence, while the bounding boxes of all objects and the relationship types are obtained from the PDDLgym simulation. This structured annotation enables effective model training and detailed evaluation of performance on the structured action classification task. Figure 4.3 provides an example of the generated annotations for frame 4 of the generated sequence from Figure 4.2.

This pipeline enables the scalable generation of diverse image sequences with controllable complexity and object arrangements, facilitating robust training and evaluation. In particular, the structured annotations, including action types and argument entities, serve as the ground-truth supervision signals for the training of both the DSG generation module and the action classification module in our supervised learning setup.

4.3.2 Overview of the Proposed Framework

After formally introducing the structured action classification problem in Section 4.1 and presenting our benchmark generation pipeline in Section 4.3.1, we now describe our proposed framework for addressing this task. As outlined in Chapter 1, the framework comprises three main components: object detection, DSG generation, and action classification. The object detection module is integrated into the DSG generation process (cf. Section 4.2) and is not the focus of this work. Instead, our contributions lie primarily in the action classification module, which will be detailed in the subsequent sections alongside the DSG generation step. An overview of the complete framework is depicted in Figure 4.4.

```
"frame_4.png": {
    "action_name": "stack",
    "args": [
        {
            "class": "block",
            "identifier": "b"
        },
        {
            "class": "block",
            "identifier": "a"
        }
    ],
    "metadata": {
        "set": "train"
    }
},


...
```


(a) Frame 4 action annotation


```
"frame_4.png": [
{
    "class": "block",
    "identifier": "b",
    "bbox": [
        64,
        305,
        78,
        320
    ],
    "binary_relationships": {
        "on": [
            "a"
        ]
    },
    "unary_relationships": [
        "clear"
    ],
    "metadata": {
        "set": "train",
    },
},
...
]
```


(b) Frame 4 DSG annotation

Figure 4.3: Example JSON annotations for frame 4 of the generated sequence. The action annotation from Figure 4.3a includes the action type and the involved objects for every frame. The DSG annotation from Figure 4.3b contains the bounding box, object class, identifier, and relationships for each object in the frame.

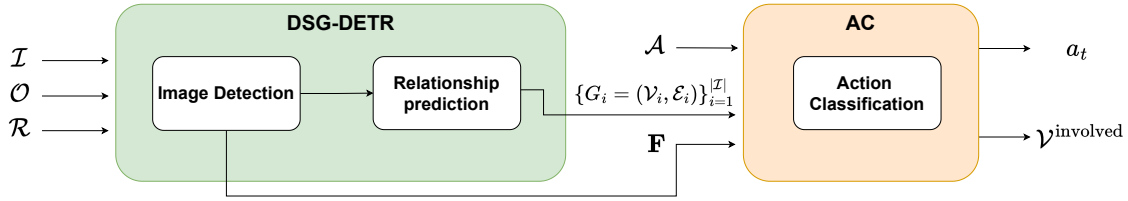


Figure 4.4: Overview of our proposed framework. Starting with the input image sequence, the object detection module detects objects and their bounding boxes in each frame. The DSG generation module constructs a scene graph for every frame based on the detected objects and their relationships. Finally, the action classification module leverages the generated DSG to classify the actions performed in the image sequence.

Dynamic Scene Graph Generation

The detections $D_{ij} = \langle \mathbf{b}_i, \mathbf{c}_i, \mathbf{f}_i \rangle$ from the object detection module already represent the nodes of the DSG for every frame i in the image sequence:

$$\mathcal{V}_i = \{v_j \mid D_{ij} \in D_i\}. \tag{4.12}$$

That is, for every detection in frame i, we create a node v_j representing this j-th detection in the i-th node set $\mathcal{V}_i$.

Then, the DSG-DETR predicts the relationships between the detected objects in the image sequence as described in Section 4.2.1. This results in a confidence score $r_{k,o,p} \in [0,1]$ for every pair of detections $D_{io}, D_{ip} \in D_i$ and relationship type $k \in [K]$ in frame i. The predicted relationships are then used to construct the scene graph for every frame in the image sequence. For this purpose, we define the edge set $\mathcal{E}_i$ for every frame i as follows:

$$\mathcal{E}_i = \{(v_o, k, v_p) \mid \text{ if } r_{k,o,p} \geq \theta \text{ for } D_{io}, D_{ip} \in D_i\}, \tag{4.13}$$

where $\theta \in [0,1]$ is a tunable parameter, serving as a threshold for the edges. The predicted DSG describing the input sequence $\mathcal{I}$ is then defined as $\mathcal{G}_{\text{dyn}} = \{G_i = (\mathcal{V}_i, \mathcal{E}_i)\}_{i=1}^{|\mathcal{I}|}$.

Formulating the DSG this way, we adapt the relationship prediction module to also consider object-object interactions (pairwise) and not only human-centered as proposed by Feng *et al.* [9]. This allows us to predict relationships between all detected objects in the image sequence, not just the human-object interactions.

Action Classification

To optimally leverage the constructed DSG for our action classification module, we parse it into an adjacency matrix sequence $\mathbf{A}_{\text{seq}} \in \{0,1\}^{T \times K \times |\mathcal{V}_i| \times |\mathcal{V}_i|}$ for every relationship type $k \in [K]$ as well as into a node feature matrix sequence $\mathbf{X}_{\text{seq}} \in \mathbb{R}^{T \times |\mathcal{V}_i| \times C}$ using every individual time step from $\mathcal{G}_{\text{dyn}} = \{G_i = (\mathcal{V}_i, \mathcal{E}_i)\}_{i=1}^{|\mathcal{I}|}$, and the object features $\mathbf{F} \in \mathbb{R}^{T \times |\mathcal{V}_i| \times C}$ from the image detector. To maintain consistent node ordering across frames in $\mathbf{X}_{\text{seq}}$ and $\mathbf{A}_{\text{seq}}$, we rely on the constructed tracklets from the DSG-DETR. Specifically, each object in a tracklet T_j is consistently assigned the same index j in every frame i in the ordering of $\mathcal{V}_i$. That is, for example, the third object in the fourth tracklet is represented by the third column of the fourth node feature matrix in the node feature matrix sequence $\mathbf{X}_{\text{seq}}$. This results in consistently ordered adjacency matrices $\mathbf{A}_{\text{seq}}$ and node feature matrices $\mathbf{X}_{\text{seq}}$, representing a mandatory property since the adjacency matrices are used for spatial convolution, aggregating information within the neighborhood of every node in the DSG, and not providing the correct graph structure of the DSG would lead to improper convolution. Moreover, we enforce a fixed tracklet length and pad the adjacency matrices and node feature matrices accordingly to ensure consistent dimensionality across all sequences. Figure 4.5 illustrates an example scene graph excerpt that is processed into the adjacency matrix and node feature matrix.

These consistently ordered adjacency matrices $\mathbf{A}_{\text{seq}}$ and node feature matrices $\mathbf{X}_{\text{seq}}$ can then be used as input for our action classification module, illustrated in Figure 4.6. Since our method is trained in a supervised manner, each input sequence is paired with annotated ground-truth labels that specify the action type and the involved entities for every transition between frames. After normalizing the node feature matrices $\mathbf{X}_{\text{seq}}$ using the BatchNorm2D, which stabilizes the

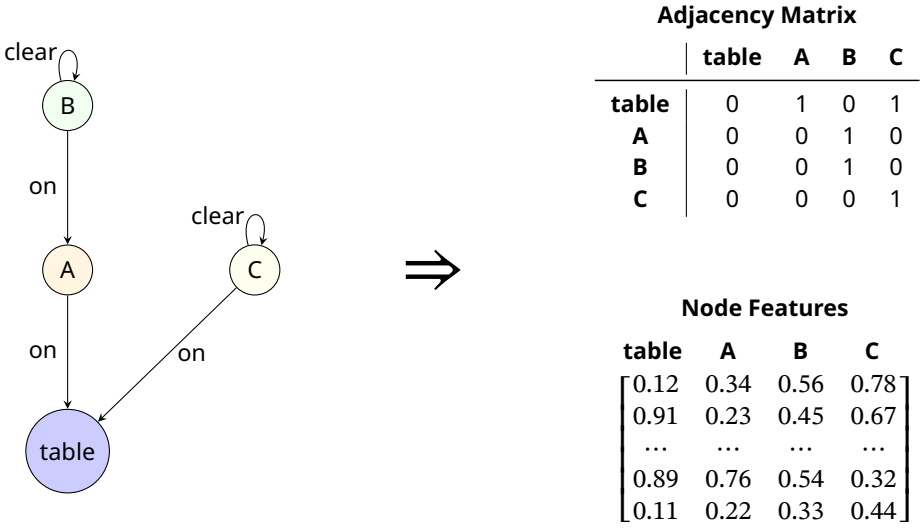


Figure 4.5: DSG processing pipeline. The scene graph is parsed into an adjacency matrix and a node feature matrix, which are then used as input for the action classification module. The adjacency matrix in the example combines all relationships into one matrix for simplicity, but in practice, we create one adjacency matrix for every relationship type. The node feature matrix is a 2D representation of the object features, where each column corresponds to a detected object and each row represents a feature dimension.

training process [34], we apply 10 layers of our adapted *dynamic* ST-GCN architecture, as this depth yielded the best empirical performance in our experiments.

Dynamic ST-GCN Layers. In comparison to the ST-GCN from [7], the graph representation in our dynamic formulation remains similar, while the key difference is that the adjacency matrices $\mathbf{A}_{\text{seq}}$ stemming from our generated DSGs are not fixed, but highly dynamic and altering throughout the sequence. Therefore, at each time step t, we use the corresponding adjacency matrix $\mathbf{A}_t$ to perform the spatial graph convolution specific to that time step. Furthermore, since the DSGs in our problem formulation have different kinds of edges and not just one type as in [7], we employ edge-type partitioning, creating one adjacency matrix not only for every time step t, but also for every edge type $k \in [K]$ as mentioned before during adjacency matrix synthesis. Finally, to enable the model to learn the difference between these edge types, we introduce learnable edge weights $\alpha_k^{(l)} \in \mathbb{R}^{K \times |\mathcal{V}| \times |\mathcal{V}|}$, ensuring the model attends to the different partitions. Our new spatial graph convolution is defined as follows:

$$\mathbf{X}_{t,v}^{(l+1)} = \sum_{k=1}^{K} \alpha_k^{(l)} \sum_{u=1}^{|\mathcal{V}|} \mathbf{Z}_{t,k,u}^{(l)} \cdot \mathbf{A}_{t,k,u,v}^{(l)}, \tag{4.14}$$

where $\mathbf{X}_{t,v}^{(l+1)}$ is the updated feature vector of node v at layer $l+1$ and timepoint t. Note that $\mathbf{Z}_{t,k,u}^{(l)}$ again represents the node feature matrix $\mathbf{X}_{t,u}^{(l)}$ at layer l, but projected into K separate

channels, one for every partition. It is computed as described in Equation (4.9). The adjacency matrix $\mathbf{A}_{t,k}^{(l)}$ is the k-th partition of the adjacency matrix $\mathbf{A}_t$ for the current frame t, which is used to perform the spatial graph convolution. This allows us to learn different patterns for different graph partitions, while also considering the dynamic nature of the DSG. After this dynamic frame-level spatial graph convolution, the temporal convolution, residual connection, and activation function are executed exactly as in the original ST-GCN formulation. This spatial-temporal convolution aggregates information among the neighborhood of each node in the DSG and captures the temporal dependencies between the frames in the image sequence. The output of this dynamic ST-GCN is a set of node feature matrices $\mathbf{X}^{(10)}$, which are then passed to the action classification head.

Dual-Head Classification. Following this dynamic ST-GCN, we propose to separate two custom classification heads for two varying downstream tasks considering every consecutive graph pair (G_t, G_{t+1}) in the DSG: action type classification and argument selection. That is, instead of only classifying a simple action label a_t among a flat set, the action classification is split into two parts. The action type classification head is similar to the one used in [7], where an MLP predicts the action class for every pair of consecutive graph-level embeddings, which is obtained by performing average pooling over the node-level embeddings, representing a multi-class classification task. The argument selection head, however, is novel and not found in [7]. It compares the unpooled node features of every pair of consecutive graphs, predicting a confidence score $k_v \in [0, 1]$ for every node v of the DSG that decides whether the respective node is involved in the action. To this end, the unpooled node features of each pair of consecutive graphs are first reduced into a dimension of 32 using a linear transformation, before being fed into a multi-head attention mechanism [21]. This dimension reduction is necessary to reduce the computational complexity of the attention mechanism, which is quadratic in the number of nodes. The attended node features are then passed through an MLP, in combination with the output of the action type classification head, and a sigmoid activation is applied to obtain probability scores in the range $[0, 1]$ for every node in the DSG, which represent the aforementioned confidence scores. The attention mechanism allows the model to learn which nodes are most relevant for the action classification task, while the MLP with the sigmoid activation function provides a final estimate of the confidence scores.

These confidence scores can be utilized to predict the involved entities for a classified action. Let q be the arity of the predicted action type a_t. Then, the predicted arguments are selected by choosing the q nodes with the highest confidence scores, corresponding to the arity of the predicted action type a_t. We simplify the ordering of the involved arguments by only considering actions of arity two and choosing exactly the involved object as the first argument that has the biggest node feature difference from frame t to $t + 1$. This heuristic is based on the hypothesis that more relevant objects are more likely to be affected visually by the performed

action, and can be replaced by more sophisticated approaches in the future. A summary of the action classification module architecture is shown in Figure 4.6.

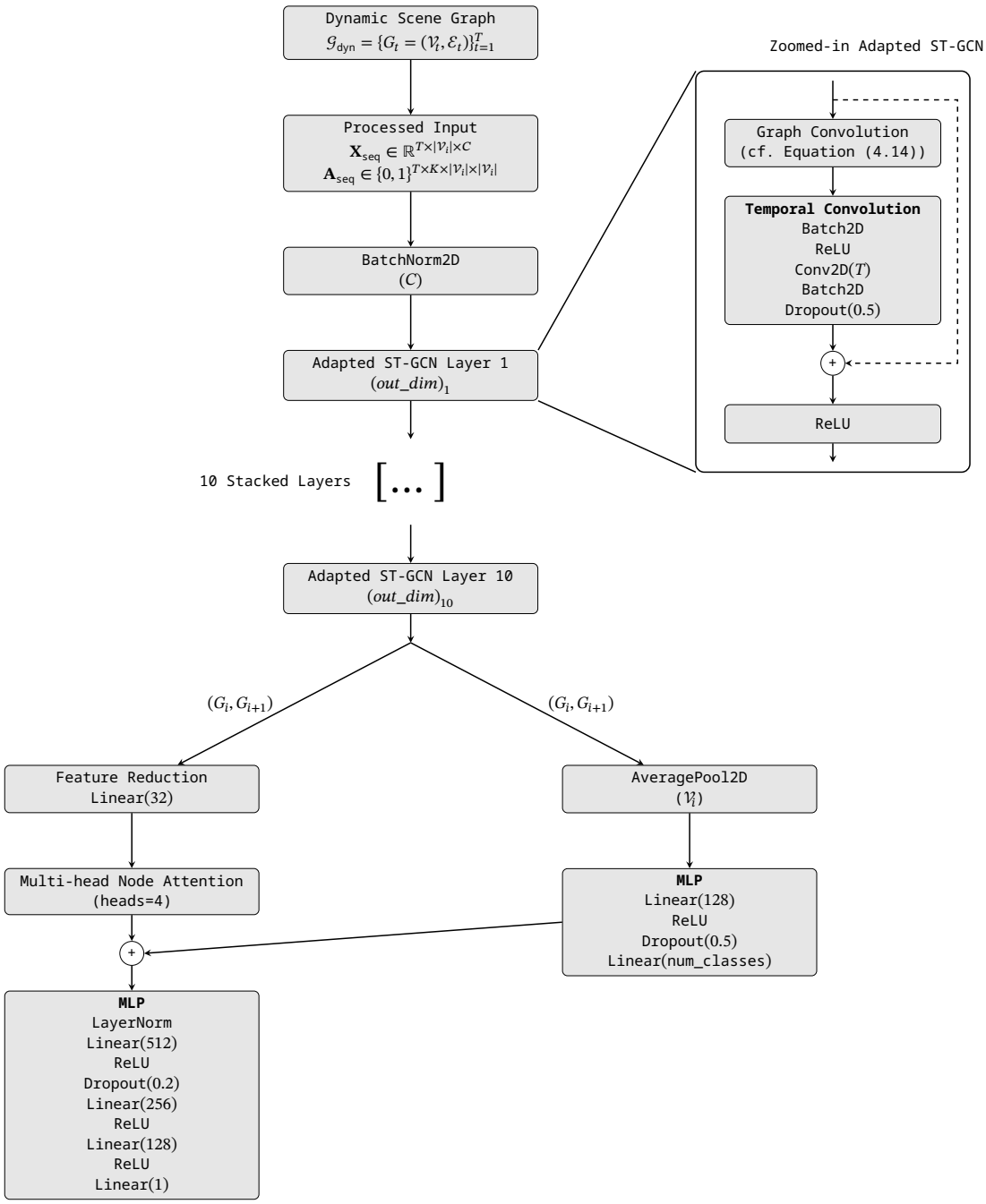


Figure 4.6: Architecture overview of our proposed action classification module. The branch on the left-hand side shows the argument selection head, which predicts the involved objects for the action. The branch on the right-hand side shows the action type classification head, which predicts the action class. The two branches are connected by a shared ST-GCN backbone, which processes the input DSG and aggregates information using spatial-temporal convolution.

Our reformulation of the ST-GCN is essential to capture both the dynamic structure of DSGs and the temporal dependencies between frames in the image sequence. Unlike the original ST-GCN formulation [7], which assumes a fixed adjacency matrix, our approach supports dynamically changing graph structures and multiple edge types over time. Additionally, by introducing a novel argument selection head, our action classification module can handle complex, multi-argument actions — a capability not supported in the original ST-GCN. Together, these innovations enable our model to reason over temporally evolving object relationships and to classify actions in structured, dynamic environments more effectively than prior methods.

Permutation Sensitivity Analysis: Invariance and Equivariance Properties. To further validate the soundness of our model, we analyze its behavior under node permutations — a critical property for graph-based models, as isomorphic graphs (i.e., graphs differing only in node ordering) should produce consistent outputs. Specifically, we aim to show that the predicted action type a is permutation *invariant*, while the selected arguments $\mathcal{V}^{\text{involved}}$ are permutation *equivariant*.

Formally, a function f is said to be *permutation invariant* if, for any permutation matrix $\mathbf{P}$ and input $\mathbf{X}$,

$$f(\mathbf{PX}) = f(\mathbf{X}). \tag{4.15}$$

Similarly, f is *permutation equivariant* if

$$f(\mathbf{PX}) = \mathbf{P} \cdot f(\mathbf{X}). \tag{4.16}$$

Theorem 1 (Permutation Behavior)**.** *Let $\mathbf{X}_1, \mathbf{X}_2 \in \mathbb{R}^{|\mathcal{V}| \times C}$ be the node feature matrices corresponding to two consecutive scene graphs G_1 and G_2, and let $\mathbf{A}_1, \mathbf{A}_2 \in \{0,1\}^{K \times |\mathcal{V}| \times |\mathcal{V}|}$ be the respective adjacency tensors encoding K edge types.*

Let further $f : (\mathbf{X}_1, \mathbf{X}_2, \mathbf{A}_1, \mathbf{A}_2) \mapsto \big(a, \mathcal{V}^{involved}\big)$ be the function implemented by our classification module that maps the pair (G_1, G_2) of scene graphs to an action type $a \in \mathcal{A}$ and a subset of involved node indices $\mathcal{V}^{involved} \subseteq \{1, \dots, |\mathcal{V}|\}$.

Then the model output satisfies:

$$f\big(\mathbf{PX}_1, \mathbf{PX}_2, \mathbf{PA}_1\mathbf{P}^\top, \mathbf{PA}_2\mathbf{P}^\top\big) = \big(a,\ \mathbf{P}\mathcal{V}^{involved}\big), \tag{4.17}$$

where $\mathbf{P} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ is a permutation matrix and the permutation of the adjacency tensors is defined as $\mathbf{PA}_i\mathbf{P}^\top := \{\mathbf{PA}_{i,k}\mathbf{P}^\top\}_{k=1}^{K}$ for $i \in \{1,2\}$. That means the predicted action type is permutation invariant*, and the predicted arguments are permutation* equivariant*.*

Proof. To prove the equivalence from Equation (4.17), we analyze the behavior of each component in the model under the application of a permutation matrix $\mathbf{P} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$. Let $\mathbf{X}_i \in \mathbb{R}^{|\mathcal{V}|\times C}$ and $\mathbf{A}_i \in \{0,1\}^{K\times|\mathcal{V}|\times|\mathcal{V}|}$ for $i \in \{1,2\}$.

1: *BatchNorm2D* [34]. This operation normalizes features channel-wise across nodes:

$$\mathrm{BN}(\mathbf{X})_{v,c} = \frac{\mathbf{X}_{v,c} - \mu_c}{\sigma_c}, \tag{4.18}$$

where $\mu_c$is the mean and $\sigma_c$is the standard deviation of $\mathbf{X}$. Since the mean and standard deviation are invariant under permutation of the node dimension (row-wise reordering), we have:

$$\mathrm{BN}(\mathbf{PX}) = \mathbf{P} \cdot \mathrm{BN}(\mathbf{X}). \tag{4.19}$$

Hence, BatchNorm2D is **equivariant** to node permutations.

2: *Dynamic Graph Convolution*. Each dynamic graph convolutional layer, as defined in Equation (4.14), is permutation equivariant. Let $\mathbf{X}_t^{(l)} \in \mathbb{R}^{|\mathcal{V}|\times C_l}$ denote the node feature matrix at time t and layer l, and let $\mathbf{A}_{t,k}^{(l)} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ be the adjacency matrix for edge type k. Applying a permutation matrix $\mathbf{P} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$ to both features and adjacency matrices, we obtain:

$$\sum_{c=1}^{C_l} \sum_{\tau=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor} \mathbf{W}_{k,c}^{(l)} \cdot \left(\mathbf{P}\mathbf{X}_{t+\tau}^{(l)}\right)_c = \mathbf{P} \cdot \left(\sum_{c=1}^{C_l} \sum_{\tau=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor} \mathbf{W}_{k,c}^{(l)} \cdot \mathbf{X}_{t+\tau,c}^{(l)} \right) = \mathbf{P}\mathbf{Z}_{t,k}^{(l)}, \tag{4.20}$$

which shows that the node feature projection into K edge-type-specific channels is equivariant under $\mathbf{P}$. Similarly, the spatial aggregation step satisfies:

$$\sum_{k=1}^{K} \alpha_k^{(l)} \sum_{u=1}^{|\mathcal{V}|} \left(\mathbf{P}\mathbf{Z}_{t,k}^{(l)}\right)_u \cdot \left(\mathbf{P}\mathbf{A}_{t,k}^{(l)}\mathbf{P}^\top\right)_{u,v} \cdot \left(\mathbf{P}\mathbf{X}_t^{(l+1)}\right)_v = \left[\mathbf{P}\mathbf{X}_t^{(l+1)}\right]_v. \tag{4.21}$$

where the reordered adjacency and feature matrices result in a consistent permutation of the output.

Together, the node feature projection and spatial aggregation steps demonstrate that a single dynamic graph convolutional layer, as defined in Equation (4.14), is permutation equivariant. This conclusion builds on the known result that standard graph convolution operations are themselves permutation equivariant [15]. Combined with the linearity and associativity of matrix multiplication and the orthogonality of permutation matrices (i.e., $\mathbf{P}^\top = \mathbf{P}^{-1}$), this guarantees that applying a permutation $\mathbf{P}$ to both the node features and adjacency matrices results in a consistent reordering of the output. Formally, if we denote the layer by $\mathrm{DGCN}\big(\mathbf{X}^{(l)}, \mathbf{A}^{(l)}\big) = \mathbf{X}^{(l+1)}$, then:

$$\mathrm{DGCN}\big(\mathbf{PX},\ \mathbf{PAP}^\top\big) = \mathbf{P} \cdot \mathrm{DGCN}(\mathbf{X}, \mathbf{A}), \tag{4.22}$$

confirming that the operation preserves **equivariance** under arbitrary node permutations.

3: *Temporal Convolution*. Temporal convolution, as defined in Equation (4.11), applies a 2D convolution over the temporal and feature dimensions, independently for each node. Since the convolution operates along the time axis and does not aggregate across nodes, permuting the node dimension (i.e., reordering the nodes consistently across all time steps) does not affect the computation. Formally, for a permutation matrix $\mathbf{P}$ applied to the node dimension:

$$\text{Conv2D}(\mathbf{PX}) = \mathbf{P} \cdot \text{Conv2D}(\mathbf{X}). \tag{4.23}$$

Thus, temporal convolution is permutation **equivariant** concerning node reordering.

4: *Average Pooling*. Average pooling is invariant to permutations of the node dimension because it aggregates by computing the mean over all nodes, regardless of their order. Specifically, the operation first sums the node features and then divides by the number of nodes. Since summation is order-independent, we have:

$$\text{AvgPool}(\mathbf{PX}) = \text{AvgPool}(\mathbf{X}). \tag{4.24}$$

As a result, the action type classification head, which applies an MLP to this pooled graph-level representation, is permutation **invariant**.

5: *Multi-head Attention*. In the argument selection head, node-level features are passed through multi-head attention as described by [21]:

$$\text{Attention}(\mathbf{X}) = \text{softmax}\left(\frac{\mathbf{QK}^\top}{\sqrt{d}}\right)\mathbf{V}, \tag{4.25}$$

where the queries $\mathbf{Q}$, keys $\mathbf{K}$, and values $\mathbf{V}$ are computed as learned linear projections of $\mathbf{X}$. When the input features are permuted by a matrix $\mathbf{P}$, the projected matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are permuted accordingly. Since the attention weights depend only on these projections, and all subsequent matrix multiplications preserve permutation structure, the output satisfies:

$$\text{Attention}(\mathbf{PX}) = \mathbf{P} \cdot \text{Attention}(\mathbf{X}). \tag{4.26}$$

This holds under the assumption that the attention mechanism is entirely determined by $\mathbf{X}$. Under this condition, multi-head attention is permutation **equivariant**.

6: *Argument Selection via MLP*. The MLP used to compute node-wise confidence scores is applied identically and independently across all nodes. This includes not only the linear layers, but also non-linear activations such as ReLU, and any dimensionality reduction layers applied to the node features. Since all these operations act row-wise and do not mix information between

nodes, they preserve the permutation structure. Hence, given permuted input features:

$$\mathrm{MLP}(\mathbf{PX}) = \mathbf{P} \cdot \mathrm{MLP}(\mathbf{X}), \tag{4.27}$$

the scores are permuted accordingly. Selecting the top-k nodes based on these scores yields:

$$\mathrm{argmax}(\mathbf{P} \cdot \mathrm{MLP}(\mathbf{X})) = \mathbf{P} \cdot \mathrm{argmax}(\mathrm{MLP}(\mathbf{X})), \tag{4.28}$$

so the final node selection is also **equivariant**.

Conclusion: Each component of the network maintains the appropriate permutation behavior. The action type output is invariant due to average pooling, while the argument selection is equivariant due to consistent permutation-aware transformations. Hence, the overall function f satisfies:

$$f\big(\mathbf{PX}_1, \mathbf{PX}_2, \mathbf{PA}_1\mathbf{P}^\top, \mathbf{PA}_2\mathbf{P}^\top\big) = \big(a,\ \mathbf{P}\mathcal{V}^{\text{involved}}\big). \tag{4.29}$$

□

4.3.3 Training and Testing Pipeline

Both the object detection module — a Faster R-CNN with a ResNet-101 backbone — and the adapted DSG-DETR module are pre-trained separately for each benchmark domain. Then, their parameters are frozen and the models are reused during the training of the action classification component. This modular setup enables a focused evaluation of the classification stage. All components are trained independently in a supervised manner using the ground-truth annotations provided by the synthetic dataset generation pipeline.

Each generated dataset is split into a training and test set using an 80/20 ratio. The action classification module is trained using the AdamW optimizer [35], which decouples weight decay from the gradient update step and has been shown to improve generalization performance in deep neural networks, particularly in structured settings such as dynamic graph models. We use a learning rate of $1 \cdot 10^{-3}$ and a weight decay of $1 \cdot 10^{-4}$, as this configuration yielded the best empirical results across all benchmark domains. The model is trained for a maximum of 50 epochs, with early stopping applied based on test set performance to prevent overfitting [36].

Since the structured action classification task comprises two sub-tasks — action type prediction and argument selection — we employ two separate loss functions. For action-type prediction, we use the categorical cross-entropy loss:

$$\mathcal{L}_{\text{type}} = -\sum_{c=1}^{|\mathcal{A}|} y_c \log(\hat{y}_c), \tag{4.30}$$

where $\mathcal{A}$ is the set of action classes, $y_c \in 0, 1$ is the one-hot encoded ground-truth label, and $\hat{y}_c$ is the predicted class probability for class c.

For argument selection, we use the binary cross-entropy loss with logits, defined as

$$\mathcal{L}_{\text{args}} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\sigma(\hat{y}_i)) + (1 - y_i)\log(1 - \sigma(\hat{y}_i))], \tag{4.31}$$

where N denotes the number of candidate nodes, $y_i \in \{0, 1\}$ is a binary indicator specifying whether node i is involved in the action, $\hat{y}_i \in \mathbb{R}$ is the predicted logit for node i, and $\sigma(\cdot)$ denotes the sigmoid activation function. This formulation enables multi-label classification by independently evaluating the likelihood of each node being part of the predicted argument set.

The total training objective is defined as the sum of the action type loss and the argument selection loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{type}} + \mathcal{L}_{\text{args}}. \tag{4.32}$$

Using this combined loss function, the model is evaluated on the test set after each epoch, and the checkpoint achieving the lowest test set loss is selected for reporting.

5 Evaluation

To evaluate our proposed method, we generate a set of synthetic datasets across various symbolic planning domains. Using these datasets, we perform experiments to assess how effectively our model can classify actions based on the generated DSGs. We evaluate the action classification module using accuracy as the primary metric, while the quality of the DSG generation is assessed using the F1 score. Additionally, we conduct ablation studies to analyze the contribution of individual components to the overall performance of the model.

5.1 Benchmarks

We use our PDDLgym-based dataset generator to construct two types of datasets: symbolic-only and interpolated. As described in Section 4.3.1, the main distinction lies in representation: symbolic-only datasets provide symbolic state and action labels with a single image per state, while interpolated datasets consist of interpolated visual representations capturing transitions between states. These datasets form the foundation for evaluating our model's ability to classify actions in image sequences.

To ensure a diverse and comprehensive assessment, we generate data across five symbolic planning domains, each designed to test different aspects of relational and temporal reasoning in visually grounded settings. Unless stated otherwise, visual observations are generated using the default PDDLgym renderer, which produces images from a fixed camera angle.

Blocksworld: We use the Blocksworld (BW) implementation provided by PDDLgym, which is based on the classical symbolic planning domain introduced by Edelkamp and Hoffmann [30]. The environment includes two object types: blocks and a robot arm. The robot can manipulate blocks using four symbolic actions: "pickup", "putdown", "stack", and "unstack". The task involves transforming an initial block configuration into a goal configuration, typically requiring the blocks to be stacked in a specific order.

Blocksworld-Realistic: This domain is symbolically equivalent to the standard Blocksworld environment but introduces greater visual complexity through a more realistic rendering pipeline, as detailed in Section 4.3.1. Unlike the default renderer, the realistic version generates images with varying camera angles, lighting conditions, and textures. This added visual diversity increases the perceptual difficulty of the task, while the underlying symbolic goal

— manipulating the blocks with the robot arm to reach a specified configuration — remains unchanged.

Ball Hiding: This domain extends the classic Blocksworld setting to include interactions among three object types: balls, boxes, and a robot arm. While the robot retains the standard Blocksworld actions (”pickup”, ”putdown”, ”stack”, and ”unstack”), it is also equipped with two additional primitives: ”hide” and ”unhide”, enabling it to place balls inside boxes and retrieve them. The objective is to match a target configuration by correctly hiding specific balls inside designated boxes. Each box can contain only one ball at a time, and stacking is permitted only when the boxes are empty. This setup introduces additional relational and temporal constraints not present in standard Blocksworld. Notably, because hidden balls are visually occluded, solving the task requires leveraging temporal reasoning over sequences of observations.

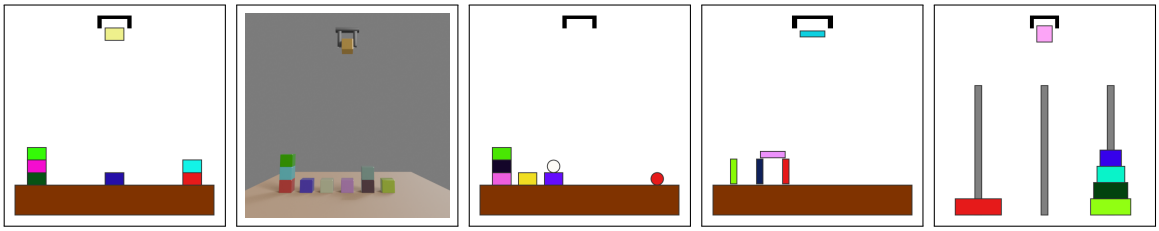
Dominoes: To introduce greater semantic complexity, we extend the Blocksworld domain by replacing blocks with dominoes. The domain includes two object types: dominoes and a robot arm. While the robot retains pick-and-place capabilities similar to Blocksworld, it also gains the ability to rotate dominoes. The objective is to arrange the dominoes into a specified goal configuration, requiring the robot to reason not only about object placement but also about orientation.

A key challenge in this domain arises from the richer set of spatial relationships between objects: the relative positioning and rotation of each domino significantly affect how they relate to one another. This increased relational complexity makes the domain particularly demanding for action classification, as models must interpret positional and orientational information to predict valid actions.

Towers of Hanoi (With Gripper): We build on the PDDLgym implementation of the classical Towers of Hanoi (ToH) problem [31], extending it to include a robot arm with a gripper. This modification expands the original action space — traditionally limited to a single ”move” action — to incorporate symbolic manipulation primitives analogous to those in the Blocksworld domain, such as ”stack” and ”unstack”. The task involves transferring a stack of disks from one peg to another, subject to the standard constraints: only one disk may be moved at a time, and no larger disk may be placed on top of a smaller one.

This domain presents unique challenges for action classification, as it requires reasoning over both spatial and size-based relationships. Crucially, these relationships are often non-local; for example, a valid move may depend on the relative sizes of disks that are not physically adjacent. This adds a layer of abstract, global reasoning not present in more localized domains like Blocksworld.

An example domain definition file for our custom domain, *Ball Hiding*, is provided in Appendix A.1.1[1]. Additionally, Figure 5.1 shows example images generated for each domain.



(a) Blocksworld (b) BW Realistic (c) Ball Hiding (d) Dominoes (e) ToH

Figure 5.1: Example images used in the benchmarks.

5.1.1 Symbolic-Only Sequences

For every domain, we generate 1000 sequences of symbolic actions. Each sequence consists of a series of images, where each image represents a state in the symbolic planning domain. Table 5.1 summarizes the generated datasets, reporting the number of object types N_O, relation types N_R, and action types N_A defined per domain. The range of object counts per frame[2], denoted by $[N_{\text{low}}, N_{\text{high}}]$, is sampled uniformly and fixed for each sequence. Additionally, we report the number of generated sequences N_{seq} and the total number of frames N_{tot} per dataset.

Table 5.1: Statistics of the generated symbolic-only datasets.

Domain	N_O	N_R	N_A	N_{low}	N_{high}	N_{seq}	N_{tot}
Ball Hiding	4	9	10	4	10	1000	7,568
Blocksworld	3	6	4	3	10	1000	7,265
BW-Realistic	3	6	4	3	10	1000	7,285
Dominoes	3	13	13	3	5	1000	7,178
Towers of Hanoi	3	6	2	5	8	1000	9,728

5.1.2 Interpolated Sequences

Similar to the symbolic-only sequences, we generate 1000 sequences of interpolated actions for the Blocksworld and Blocksworld-Realistic domains. Each sequence consists of a series of images, where each image represents a state in the symbolic planning domain. Table 5.2 provides a summary of the statistics for the generated datasets, including the same details as in Table 5.1. The essential difference is that these interpolated datasets include interpolated visual representations of the symbolic planning states, forcing the model to effectively analyze

[1]The PDDL domain files for the other domains are available in our repository at https://github.com/p-schulte/ac_dsg.

[2]Object counts refer exclusively to manipulatable entities, such as blocks or disks. Static elements like the robot and the table are excluded.

the node features instead of relying on the relationships between the nodes solely. To this end, we interpolate up to 5 frames between each pair of symbolic states.

Table 5.2: Statistics of the generated interpolated datasets.

Domain	N_O	N_R	N_A	N_{low}	N_{high}	N_{seq}	N_{tot}
Blocksworld	3	6	7	3	10	1000	26,136
BW-Realistic	3	6	7	3	10	1000	25,105

With these 7 diverse datasets at hand, we can evaluate the model's ability to classify actions in a variety of symbolic planning scenarios. The datasets cover a range of action types and object relationships, providing a comprehensive benchmark for assessing the effectiveness of our approach in structured action classification tasks.

5.2 Evaluation Metrics

To evaluate the performance of our full system, we report distinct metrics for each of the core components: object detection, DSG generation, and structured action classification. Specifically, we use the mean Average Precision (mAP) for evaluating the object detector, the *F1 score* for DSG generation, and a combination of class-balanced accuracy metrics for assessing the structured action classification model.

5.2.1 Object Detection and DSG Generation

For evaluating object detection performance, we calculate the mAP metric on the test set of each dataset, following the standard procedure outlined in [1]. The mAP summarizes detection quality by computing the area under the precision-recall curve for each class and averaging these values across all classes, thereby capturing classification performance. Formally, the mAP is defined as:

$$\text{mAP} = \frac{1}{|\mathcal{O}|} \sum_{c \in \mathcal{O}} \int_0^1 p_c(r) dr, \tag{5.1}$$

where $\mathcal{O}$ denotes the set of object classes, and $p_c(r)$ is the precision-recall curve for class c as a function of recall r. The integral measures the area under this curve, representing the Average Precision (AP) for class c, and mAP averages this over all classes.

Since DSG-generation builds upon the detected objects, we evaluate it with a separate metric that reflects the correctness of inter-object relationships. In contrast to previous works on DSG-generation [6, 9], we refrain from using the commonly employed recall-at-K ($R@K$) metric. While $R@K$ has proven useful in benchmarks with a limited and relatively fixed number of relationships per image, it becomes unsuitable for our synthetic datasets. These datasets often contain a large and variable number of ground-truth relationships, making $R@K$ inflexible

and potentially misleading — it truncates the evaluation to only the top-k predictions, ignoring the rest regardless of correctness.

To obtain a more insightful assessment, we instead report the F1 score, which balances precision and recall and better reflects performance in settings with high relationship density. It is defined as:

$$\mathrm{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \tag{5.2}$$

where precision is the proportion of correctly predicted edges among all predicted edges, and recall is the proportion of correctly predicted edges among all ground-truth edges.

5.2.2 Structured Action Classification

We evaluate our structured action classification model along two axes: predicting the correct action type and identifying the corresponding argument entities. These sub-tasks are assessed independently to analyze their individual contributions and aggregated into a combined score for holistic evaluation.

Action Type Classification. Following [7], we treat action type classification as a standard multi-class classification task and report the top-1 accuracy. This metric quantifies the proportion of samples for which the predicted action label exactly matches the ground-truth label. To account for class imbalance in the dataset, we compute per-class accuracy and report the class-averaged result.

Formally, the accuracy for class c is defined as:

$$\mathrm{ac}_{\mathrm{type}}^{(c)} = \frac{1}{N_c} \sum_{i=1}^{N_c} \mathbb{1}\big(y_i^c = \hat{y}_i^c\big), \tag{5.3}$$

where N_c denotes the number of samples belonging to class c, y_i^c is the ground-truth label, $\hat{y}_i^c$ is the predicted label, and $\mathbb{1}(\cdot)$ is the indicator function, returning 1 if the argument is true and 0 otherwise.

The final reported score, $\mathrm{ac}_{\mathrm{type}}$, is obtained by averaging the class-wise accuracies:

$$\mathrm{ac}_{\mathrm{type}} = \frac{1}{|\mathcal{A}|} \sum_{c \in \mathcal{A}} \mathrm{ac}_{\mathrm{type}}^{(c)}, \tag{5.4}$$

where $\mathcal{A}$ denotes the set of all action classes. This class-averaging strategy ensures that the evaluation is not dominated by frequently occurring actions, providing a more balanced assessment of the model's performance across all action types.

Argument Selection. Unlike action-type classification, argument selection is a multi-label classification problem. That is, each sample may have multiple correct labels (i.e., object arguments). To evaluate this, we adopt the Exact Match Ratio (EMR), as proposed in [37], which measures the proportion of predictions where the entire set of predicted labels matches the ground-truth label set exactly. We average this EMR score over all action type classes $\mathcal{A}$ to obtain the total argument selection accuracy ac_{args}:

$$\text{ac}_{\text{args}} = \frac{1}{|\mathcal{A}|} \sum_{c \in \mathcal{A}} \text{EMR}^{(c)} = \frac{1}{|\mathcal{A}|} \sum_{c \in \mathcal{A}} \left[\frac{1}{N_c} \sum_{i=1}^{N_c} \mathbb{1}\big(h^c(x_i) = \mathbf{l}_i^c\big) \right], \tag{5.5}$$

where $h^c(x_i)$ is the predicted label vector for input x_i and class c, and $\mathbf{l}_i^c$ is the corresponding ground-truth label vector for class c. This metric is strict, penalizing partial correctness, and thus provides a conservative estimate of performance.

Combined Score. To capture both sub-tasks in a single evaluation score, we report the average of ac_{type} and ac_{args}:

$$\text{ac}_{\text{total}} = \frac{1}{2}\big(\text{ac}_{\text{type}} + \text{ac}_{\text{args}}\big), \tag{5.6}$$

which provides a unified metric for evaluating structured action classification performance. During our experiments, we report all three accuracies in percentage points.

5.3 Results

As detailed in Section 4.3.3, we begin by pre-training both the object detection and DSG-DETR modules individually for each dataset. Their strong performance is essential to the overall pipeline: since the action classification module depends on accurate object detections and reliable scene graph representations, errors in early stages would propagate downstream and degrade classification performance. Leveraging these pre-trained components, we fine-tune the action classification module on the synthetic datasets introduced in Section 5.1, and evaluate each component using the metrics defined in Section 5.2.

Table 5.3 summarizes the performance of all three modules — object detector, DSG generator, and action classifier — across the evaluated domains, covering both symbolic-only and interpolated image sequences. The results demonstrate that our method consistently achieves high performance across all evaluated benchmarks. In the symbolic-only domains, the model excels at structured action classification, with total accuracies exceeding 91%. This indicates a robust ability to correctly identify both the action type and its arguments, even in the presence of class imbalance. Note that this strong performance is contingent on the high quality of the preceding modules: the object detector consistently achieves near-perfect mAP scores, and the

DSG generator maintains F1 scores above 0.98 across all datasets. These reliable inputs are essential for enabling accurate downstream action classification.

Table 5.3: Performance metrics across datasets.

		DSG-Generation		Action Classification		
	Dataset	mAP	F1	ac_{type}	ac_{args}	ac_{total}
Symbolic	Ball Hiding	100.00	0.99	99.84	96.59	98.22
	Blocksworld	100.00	1.00	100.00	97.19	98.60
	Blocksworld-Realistic	99.49	0.99	98.50	92.70	95.60
	Dominoes	100.00	0.99	92.49	90.90	91.70
	Towers of Hanoi	99.59	0.99	100.00	96.34	98.17
Interpo-lated	Blocksworld	100.00	1.00	99.96	99.87	99.91
	Blocksworld-Realistic	99.49	0.99	95.61	93.25	94.43

Interestingly, we observe a slight performance drop between the action type classification and argument selection subtasks. This is expected, as the latter involves a more challenging multi-label classification problem that demands fine-grained, object-centric relational reasoning rather than coarse scene-level interpretation. Despite this increased complexity, the model achieves argument selection accuracies exceeding 90% across all scenarios, demonstrating its ability to reliably associate predicted actions with the correct object arguments.

Among all datasets, the Dominoes domain shows the lowest performance among the symbolic-only datasets, with a total accuracy of 91.70%. This performance drop can be attributed to the domain's increased relational and semantic complexity. With 13 action and relation types, the model must operate over a larger and more fine-grained classification space. Notably, many actions in this domain have multiple, semantically similar variants — such as "unstack h1(·)" and "unstack h2(·)" — which differ subtly in spatial configuration[3]. As a result, the model is often required to distinguish between nearly identical actions based on minimal visual or relational cues. This increases the likelihood of confusion, as also evidenced by the misclassification patterns observed in the confusion matrix shown in Figure 5.2. The proximity of similar action types makes it more challenging to exploit the learned node features and relational structure for precise action classification. Nonetheless, the model still demonstrates strong and reliable performance, successfully classifying the vast majority of actions even in this more demanding setting.

[3]The distinction between "unstack h1(·)" and "unstack h2(·)" lies in their preconditions: "unstack h1(·)" is only applicable when the block to be picked up has no adjacent neighbors, while "unstack h2(·)" permits unstacking regardless of neighboring blocks.

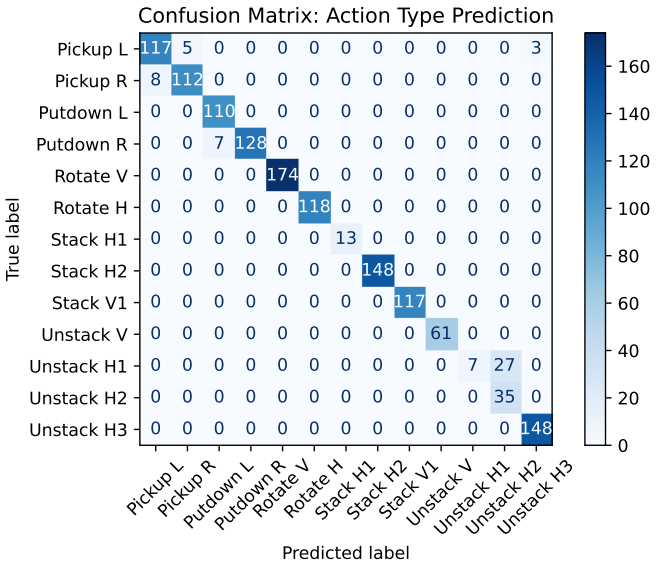
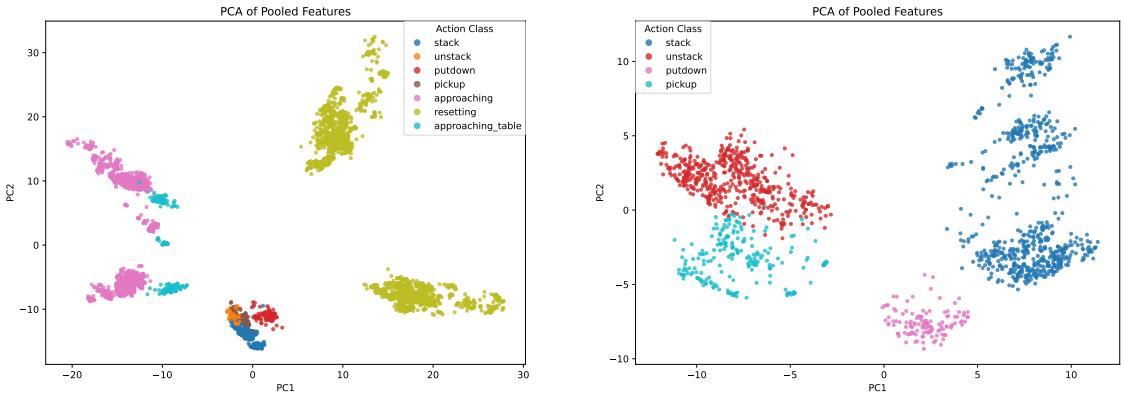


Figure 5.2: Confusion matrix of predicted action types in the Dominoes domain, illustrating partial misclassifications between semantically similar actions.

In comparison, the model maintains strong performance in the interpolated settings, achieving near-perfect scores on the Blocksworld domain and only a slight drop on the visually more complex Blocksworld-Realistic dataset. This highlights the model's capacity to generalize from symbolic representations to more ambiguous input sequences that use motion primitives. Overall, the consistently high values across all metrics confirm that the proposed architecture effectively handles the temporal dynamics hidden in DSGs across diverse action classification scenarios.

To gain deeper qualitative insights into the learned feature representations, we perform a Principal Component Analysis (PCA) on the aggregated node features extracted from pairs of consecutive time steps in the DSGs for both the interpolated Blocksworld-Realistic and symbolic Blocksworld datasets. This technique enables the visualization of the learned embeddings in a lower-dimensional space, allowing us to understand how well the model captures the underlying structure of the data. Figure 5.3 shows the results of this analysis.

In the PCA for the interpolated Blocksworld-Realistic dataset (Figure 5.3a), we observe that the model effectively separates the action types into distinct clusters. Notably, the symbolic actions "stack", "unstack", "putdown", and "pickup" are distinguishable, but also closely grouped together, indicating that the model has learned to differentiate between these symbolic actions and the artificially added motion primitives "resetting", "approaching", and "approaching_table". Furthermore, we observe that the three motion primitives each form two disconnected sub-clusters. This suggests that the model has learned to internally distinguish between two different modes of execution for the same primitive. A plausible explanation is that this



(a) Interpolated Blocksworld-Realistic dataset.

(b) Symbolic Blocksworld dataset.

Figure 5.3: PCA visualizations of the learned node features from the DSG in two variants of the Blocksworld dataset: (a) interpolated Blocksworld-Realistic and (b) symbolic Blocksworld.

separation corresponds to the direction of the robot's movement during the action — such as approaching the table from the left versus the right. The model therefore appears to encode not only the high-level action semantics but also low-level spatial dynamics tied to the robot's trajectory.

In the PCA visualization of the symbolic Blocksworld dataset (Figure 5.3b), a different clustering pattern emerges. Since this domain contains only symbolic actions, the resulting representation is sparser but still clearly structured. The action types ”stack”, ”unstack”, ”putdown”, and ”pickup” form distinct clusters, with ”stack”/”putdown” and ”unstack”/”pickup” each grouped closely together. This reflects the model's ability to capture semantic similarities between these action pairs — often used in related contexts — while still maintaining meaningful distinctions. Notably, the cluster corresponding to ”stack” appears more dispersed than the others. This is likely due to the higher intra-class variability of ”stack” instances, which involve more complex spatial arrangements, varying stack heights, or object combinations. These variations lead to greater diversity in the DSG representations, resulting in a less compact embedding space for this action class. The absence of motion primitives highlights the model's capacity to learn purely from the symbolic structure without relying on transitional cues.

Before turning to ablation studies, we illustrate the behavior of our model on an example image sequence from the test set. Figure 5.4 shows a series of frames along with the predicted action types and argument selections produced by the model. This example highlights how the model processes dynamic scenes using both visual and relational cues to infer structured action labels over time.

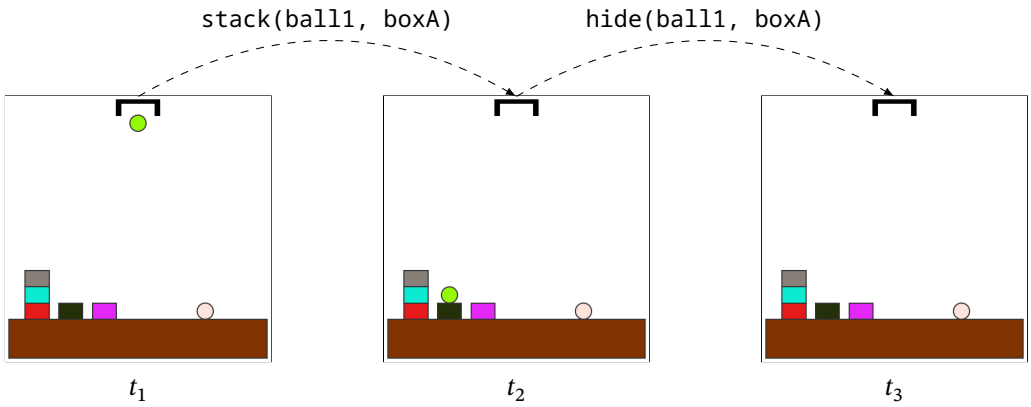


Figure 5.4: Example inference results on a test sequence from the Ball Hiding domain. The sequence shows three consecutive frames t_1, t_2, t_3, along with the action predictions made by the model. Arrows indicate the predicted structured actions, including both the action type and its object arguments. The visualized transitions demonstrate the model's ability to infer temporally grounded, object-centric interactions over time.

5.4 Ablation Studies

To better understand the individual components and design decisions of our proposed framework, we conduct a series of ablation studies on our dynamic ST-GCN model. These studies aim to isolate and quantify the contribution of key architectural elements and input representations. Specifically, we address the following research questions:

- **Q1:** What is the impact of temporal convolution on the overall classification performance?
- **Q2:** How critical are the learned node features extracted from the object detector for accurate action classification?
- **Q3:** How does independent per-frame scene graph generation affect classification compared to temporally coherent DSGs?

To address **Q1**, we conduct experiments on the interpolated Blocksworld-Realistic dataset. This dataset is particularly well-suited for evaluating the role of temporal convolution in the action classification module, as it includes motion primitives — fine-grained visual transitions that are not explicitly reflected in the symbolic state of the scene graph. In contrast to purely symbolic actions, which visibly alter the DSG structure (e.g., adding or removing edges), motion primitives often preserve the same high-level graph topology across frames. As a result, understanding these transitions requires the model to analyze the temporal evolution of visual and structural cues, making temporal convolution an essential component for accurate classification.

For **Q2**, we also leverage the interpolated Blocksworld-Realistic dataset, as it offers the most visually detailed and realistic node features due to its 3D-rendered imagery. These rich visual

representations are especially relevant in the presence of motion primitives, where spatial and semantic relationships between objects alone may be insufficient for reliable action inference. By exchanging the learned object-level features from the detector, we can rigorously evaluate their contribution to the downstream classification performance and determine how much the model depends on fine-grained visual encodings of individual objects.

Regarding **Q3**, we perform our experiments on the Ball Hiding dataset, which includes object occlusions throughout the sequences. This dataset is ideal for evaluating the necessity of temporally coherent DSGs, as occlusions often remove or obscure critical scene information in individual frames. Without temporal integration, static per-frame graph generation fails to capture the true dynamics of the scene, especially in cases where object visibility is inconsistent. In contrast, other datasets in our benchmark (e.g., standard Blocksworld) do not suffer from such occlusions, making static DSG generation largely sufficient there. Thus, Ball Hiding exposes the limitations of frame-wise disconnected scene graphs and highlights the importance of modeling temporal continuity in structure prediction.

Note that the difference between **Q1** and **Q3** lies in their respective focus: **Q1** examines the effect of temporal convolution on action classification performance under the assumption of temporally coherent DSGs, whereas **Q3** investigates whether temporal coherence in the DSGs is necessary in the first place. In other words, **Q1** isolates the benefit of explicitly modeling temporal dependencies within a coherent graph structure, while **Q3** evaluates the importance of that temporal coherence itself for achieving high classification accuracy.

5.4.1 Action Classification Without Temporal Convolution

To address **Q1** we evaluate the contribution of temporal dependencies by comparing our full ST-GCN model with an ablated version that omits the temporal convolutional layers, using only spatial graph convolutions over the DSGs.

As shown in Table 5.4, incorporating temporal convolution significantly improves overall performance. The full model achieves a total accuracy of 94.43%, outperforming the spatial-only variant by 3.02 percentage points. The most substantial gain is observed in argument selection, with an 4.79-point increase in accuracy — suggesting that temporal context plays a critical role in distinguishing object interactions within action sequences.

Table 5.4: Performance metrics on the interpolated Blocksworld-Realistic dataset with, and without temporal convolution.

Method	ac_{total}	ac_{type}	ac_{args}
ST-GCN	**94.43**	**95.61**	**93.25**
GCN	91.41	94.37	88.46

Notably, the change in action type classification accuracy is relatively minor, with only a 1.24-point difference. This suggests that action type prediction relies less on temporal context, making it a comparatively simpler subtask that offers a limited view of the scene's underlying dynamics. This observation highlights the need to incorporate argument selection, which necessitates fine-grained temporal and relational reasoning — and thereby enforces a deeper, more holistic scene understanding.

5.4.2 Handcrafted vs. Learned Features

To answer **Q2**, we compare our model's performance using learned node features from the object detector (Faster R-CNN) with two baseline variants. These include one model that uses handcrafted features, and one that uses constant features. The handcrafted features encode bounding box coordinates and predicted object classes, while the constant features contain no meaningful information. In this case, the model can only rely on the graph structure — that is, the relationships between nodes — since the node features themselves are uninformative.

Table 5.5 presents the results of this ablation study. The model with learned node features achieves the highest total accuracy of 94.43%, outperforming the handcrafted feature variant by 7.77 percentage points. The model using constant features performs significantly worse, with accuracies below 40%, as expected. This confirms that meaningful and learned node features are essential for high action classification performance.

Table 5.5: Performance metrics on the interpolated Blocksworld-Realistic dataset with different node features.

Method	ac_{total}	ac_{type}	ac_{args}	D^4
Faster R-CNN	**94.43**	**95.61**	**93.25**	2048
Handcrafted	86.66	95.10	78.22	9
Constant	39.83	50.97	28.69	100

Notably, the difference in accuracy between the model with handcrafted features and the one with learned features is particularly present in argument selection, where the learned features outperform the handcrafted ones by 15.03 percentage points. This suggests that the model benefits from the richer, learned representations that capture more detailed object characteristics and visual cues, rather than relying solely on basic bounding box coordinates and class labels.

5.4.3 Frame-Wise Scene Graph Generation

To address **Q3**, we assess the impact of temporal coherence in DSG generation by comparing the full DSG-DETR model with an ablated variant that omits the temporal transformer module.

[4]Dimension of the feature vectors.

This component is responsible for linking object instances across frames, thereby enabling consistent identity tracking and temporal reasoning. In the ablated version, scene graphs are generated independently for each frame, treating the input as a sequence of temporally disconnected static images. Importantly, both the full and ablated DSGs are processed by the same downstream ST-GCN-framework. This isolates the effect of temporal consistency during DSG generation and ensures a controlled comparison. Without the temporal transformer, object identities cannot be propagated over time, severely limiting the model's ability to reason about occlusions or infer the presence of temporarily hidden entities. This deficiency is especially problematic in domains like Ball Hiding, where effective action classification depends on maintaining object continuity across frames.

The results, summarized in Table 5.6, reveal a measurable performance drop when temporal modeling is removed. The full model achieves an overall classification accuracy of 98.22%, outperforming the static variant by 1.57 percentage points. The largest degradation is observed in the argument selection task, where accuracy drops from 96.59% to 93.56%. This indicates that argument selection relies more heavily on temporally consistent object tracking.

Table 5.6: Performance metrics of our ST-GCN framework on the Ball Hiding dataset using the standard DSG-DETR, and an ablated version that generates the scene graphs individually.

Method	ac_{total}	ac_{type}	ac_{args}
DSG-DETR	**98.22**	**99.84**	**96.59**
Static SGs	96.65	99.73	93.56

Interestingly, the impact on DSG generation itself is modest: the F1 score decreases only slightly (from 0.99 to 0.96), suggesting that many object relationships are still recoverable without temporal reasoning, likely because occlusion events affect only a subset of the scene. However, even small inconsistencies in DSGs can propagate through the pipeline and compound into downstream errors during classification. These results confirm our hypothesis: the temporal transformer is essential for maintaining object coherence across frames and enabling robust reasoning about dynamic interactions, especially in settings involving partial observability and occlusion.

6 Conclusion

In this thesis, we presented a novel approach to action classification in image sequences by leveraging DSGs in combination with ST-GCNs. Our method is designed to exploit the captured spatial relationships between objects and the temporal dynamics of actions in generated DSGs. To enable this, we extended the spatial graph convolution from ST-GCNs to operate on dynamic, heterogeneous graphs, processing the evolving structure of DSGs over time.

Extensive experiments and ablation studies confirm that learning these intermediate DSG representations substantially improves action classification performance. Our full model — integrating spatial and temporal convolutions — consistently outperforms ablated variants, particularly in domains with rich object interactions. These findings, detailed in Section 5.3 and Section 5.4, directly answer **RQ1**, demonstrating the effectiveness of our structured, temporally-aware action classification approach.

To enhance interpretability, we introduced a dual-head classification mechanism that separates action type prediction from argument selection. This design not only improves precision but also yields more structured and transparent outputs, making the model's decisions easier to analyze and understand. In doing so, it directly supports the objective of **RQ2**.

For systematic evaluation, we developed a synthetic dataset generator that produces visually and semantically rich image sequences grounded in symbolic planning domains. The resulting datasets cover a wide range of actions, object interactions, and scene configurations, providing a controlled yet challenging benchmark. This setup addresses **RQ3** by enabling consistent and rigorous evaluation of structured action classification methods.

Despite the strengths of our approach, several limitations remain. First, it relies on pre-trained object detectors and external DSG-generation methods, which may not generalize well to real-world settings where detection noise and domain shifts are common. Future work could explore end-to-end DSG construction methods — such as OED — that do not depend on a separate object detection pipeline. Second, while our current implementation of the argument selection mechanism allows for set prediction of arbitrary size, our proposed argument ordering heuristic is limited to actions of arity two. Extending this heuristic to handle actions with higher arity would enable it to address more complex domains and richer interaction patterns.

Third, although our synthetic benchmarks provide a solid foundation for controlled evaluation, they do not fully capture the complexity and variability of real-world environments. Developing more sophisticated datasets and evaluation frameworks — ideally grounded in realistic

perception and interaction scenarios — remains an important direction for future work. Lastly, further research should explicitly investigate the model's generalization capabilities, particularly its robustness to unseen domains, novel object configurations, and noisy or incomplete input representations.

A Appendix

A.1 Benchmarks

This section provides supporting materials for the benchmark domains used in our experiments, including PDDL definitions.

A.1.1 PDDL Files

This section presents the domain and problem definition files used for our custom environment, *Ball Hiding*. These files define the symbolic structure of the domain and include an example problem instance, all specified in PDDL format.

Listing A.1: PDDL domain file for the Ball Hiding domain.

```
(define (domain ball_hiding)
    (:requirements :strips :typing)
    (:types block ball robot)
    (:predicates
        (on ?x - block ?y - block)
        (on ?x - ball ?y - block)
        (ontable ?x - block)
        (ontable ?x - ball)
        (clear ?x - block)
        (handempty ?x - robot)
        (handfull ?x - robot)
        (holding ?x - block)
        (holding ?x - ball)
        (block_full ?x - block)
        (block_empty ?x - block)
        (in ?x - ball ?y - block)

        (pickup_block ?x - block)
        (pickup_ball ?x - ball)
        (putdown_block ?x - block)
        (putdown_ball ?x - ball)
        (stack_block ?x - block ?y - block)
        (stack_ball ?x - ball ?y - block)
        (unstack_block ?x - block)
        (unstack_ball ?x - ball)
        (hide_ball ?x - ball ?y - block)
        (unhide_ball ?x - ball ?y - block)
    )

```

```
30      ; (:actions pickup_block pickup_ball putdown_block putdown_ball
            stack_block stack_ball unstack_block unstack_ball hide_ball
            unhide_ball)
31
32      (:action pickup_block
33          :parameters (?x - block ?robot - robot)
34          :precondition (and
35              (pickup_block ?x)
36              (clear ?x)
37              (block_empty ?x)
38              (ontable ?x)
39              (handempty ?robot)
40          )
41          :effect (and
42              (not (ontable ?x))
43              (not (clear ?x))
44              (not (handempty ?robot))
45              (handfull ?robot)
46              (holding ?x)
47          )
48      )
49      (:action pickup_ball
50          :parameters (?x - ball ?robot - robot)
51          :precondition (and
52              (pickup_ball ?x)
53              (ontable ?x)
54              (handempty ?robot)
55          )
56          :effect (and
57              (not (ontable ?x))
58              (not (handempty ?robot))
59              (handfull ?robot)
60              (holding ?x)
61          )
62      )
63      (:action putdown_block
64          :parameters (?x - block ?robot - robot)
65          :precondition (and
66              (putdown_block ?x)
67              (holding ?x)
68              (handfull ?robot)
69          )
70          :effect (and
71              (not (holding ?x))
72              (clear ?x)
73              (handempty ?robot)
74              (not (handfull ?robot))
75              (ontable ?x))
76      )
77      (:action putdown_ball
78          :parameters (?x - ball ?robot - robot)
79          :precondition (and
80              (putdown_ball ?x)
```

```
            (holding ?x)
            (handfull ?robot)
        )
        :effect (and
            (not (holding ?x))
            (handempty ?robot)
            (not (handfull ?robot))
            (ontable ?x))
    )
    (:action stack_block
        :parameters (?x - block ?y - block ?robot - robot)
        :precondition (and
            (stack_block ?x ?y)
            (holding ?x)
            (clear ?y)
            (block_empty ?y)
            (handfull ?robot)
        )
        :effect (and
            (not (holding ?x))
            (not (clear ?y))
            (clear ?x)
            (handempty ?robot)
            (not (handfull ?robot))
            (on ?x ?y)
        )
    )
    (:action stack_ball
        :parameters (?x - ball ?y - block ?robot - robot)
        :precondition (and
            (stack_ball ?x ?y)
            (holding ?x)
            (handfull ?robot)
            (clear ?y)
            (block_empty ?y)
        )
        :effect (and
            (not (holding ?x))
            (not (clear ?y))
            (handempty ?robot)
            (not (handfull ?robot))
            (on ?x ?y)
        )
    )
    (:action unstack_block
        :parameters (?x - block ?y - block ?robot - robot)
        :precondition (and
            (unstack_block ?x)
            (on ?x ?y)
            (clear ?x)
            (block_empty ?x)
            (handempty ?robot)
        )
```

```
134         :effect (and
135             (holding ?x)
136             (clear ?y)
137             (not (clear ?x))
138             (not (handempty ?robot))
139             (handfull ?robot)
140             (not (on ?x ?y))
141         )
142     )
143     (:action unstack_ball
144         :parameters (?x - ball ?y - block ?robot - robot)
145         :precondition (and
146             (unstack_ball ?x)
147             (on ?x ?y)
148             (handempty ?robot)
149         )
150         :effect (and
151             (holding ?x)
152             (clear ?y)
153             (not (handempty ?robot))
154             (handfull ?robot)
155             (not (on ?x ?y))
156         )
157     )
158     (:action hide_ball
159         :parameters (?x - ball ?y - block ?robot - robot)
160         :precondition (and
161             (hide_ball ?x ?y)
162             (block_empty ?y)
163             (on ?x ?y)
164             (handempty ?robot)
165         )
166         :effect (and
167             (block_full ?y)
168             (not (block_empty ?y))
169             (in ?x ?y)
170             (not (on ?x ?y))
171         )
172     )
173     (:action unhide_ball
174         :parameters (?x - ball ?y - block ?robot - robot)
175         :precondition (and
176             (unhide_ball ?x ?y)
177             (in ?x ?y)
178             (block_full ?y)
179             (handempty ?robot)
180         )
181         :effect (and
182             (not (block_full ?y))
183             (block_empty ?y)
184             (not (in ?x ?y))
185             (on ?x ?y)
186         )
```

```
    )
)
```

Listing A.2: Example PDDL problem instance file for the Ball Hiding domain.

```
(define (problem random_problem)
    (:domain ball_hiding)
    (:objects
        e - block
        b - block
        d - block
        c - block
        a - block
        f - ball
        g - ball
        robot - robot)
    (:init
        (ontable d)
        (ontable b)
        (ontable a)
        (ontable e)
        (ontable f)
        (ontable g)
        (clear a)
        (clear e)
        (clear d)
        (clear b)
        (block_empty e)
        (block_empty b)
        (block_empty d)
        (block_empty c)
        (block_empty a)
        (holding c)
        (handfull robot)
    )
    (:goal (and
        (in g a)
        (in f c)
    ))
)
```

A.2 Additional Observations

This section presents supplementary insights that, while not central to the main evaluation, contribute to a deeper understanding of the proposed model design.

A.2.1 Action Classification Space

The dual-head architecture used in our action classification model separates the prediction of the action type from the selection of its arguments. This decomposition substantially reduces

the classification space compared to a single-head approach that would need to predict complete action-argument tuples jointly.

In a single-head setting, the model must classify all possible combinations of action types and argument permutations, resulting in a total of $|\mathcal{A}| \cdot |\mathcal{O}|^k$ classes, where $|\mathcal{A}|$ denotes the number of action types, $|\mathcal{O}|$ the number of objects, and k the number of arguments required per action. In contrast, the dual-head architecture decouples this prediction task into two simpler subproblems: first predicting the action type, and then selecting the k arguments individually. This yields a total classification space of $|\mathcal{A}| + k \cdot |\mathcal{O}|$, with the first term corresponding to action type prediction and the second to independent selection of arguments from the object pool.

This reduction is particularly advantageous in domains with a large number of objects, where the combinatorial explosion of possible argument tuples can quickly make the single-head formulation intractable. For instance, assuming $|\mathcal{A}| = 10$ action types, $|\mathcal{O}| = 15$ objects, and an action arity of $k = 3$, the single-head formulation would require the model to discriminate among $10 \cdot 15^3 = 33,750$ distinct classes. The dual-head approach, by contrast, reduces this to $10 + 3 \cdot 100 = 55$ classes — a reduction of several orders of magnitude.

Beyond its computational benefits, this decomposition also supports greater architectural modularity and interpretability. It enables independent improvements to the type and argument predictors and facilitates more robust training, especially in the presence of class imbalance or limited training data. Moreover, it offers a natural extension path for handling variable-arity actions and more structured argument constraints in future work.

List of Acronyms

AG	Action Genome
BW	Blocksworld
CNN	Convolutional Neural Network
DSG	dynamic scene graph
DSG-DETR	Dynamic Scene Graph Detection Transformer
EMR	Exact Match Ratio
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
IoU	Intersection-over-Union
LSTM	Long Short-Term Memory
mAP	mean Average Precision
MLP	Multi-Layer Perceptron
MOTIFNET	Stacked Motif Network
PCA	Principal Component Analysis
PDDL	Planning Domain Definition Language
RePN	Relation Proposal Network
RNN	Recurrent Neural Network
SGG	scene graph generation
ST-GCN	Spatial-Temporal Graph Convolutional Network
ToH	Towers of Hanoi

List of Symbols

General

$(\cdot)$	arbitrary (scalar) variable
$(\bullet)$	arbitrary vector variable
$\langle \cdot, \cdot, \cdot \rangle$	tuple of multiple variables
n	scalar value
$\mathbf{v}$	vector
$\mathbf{X}$	matrix or tensor
$\mathcal{V}$	set
$\mathbb{R}$	set of real numbers
$\mathbb{N}$	set of natural numbers
$[n]$	finite initial segment $\{0, 1, \dots, n\}$ of $\mathbb{N}$ for $n \in \mathbb{N}$
$\arg\min_a f(a)$	optimal value of a minimizing the expression $f(a)$
$\arg\max_a f(a)$	optimal value of a maximizing the expression $f(a)$

List of Figures

List of Tables

List of Algorithms

List of Listings

List of References

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2016.2577031. [Online]. Available: https://ieeexplore.ieee.org/document/7485869/?arnumber=7485869 (visited on 03/31/2025).

[2] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep Learning for Generic Object Detection: A Survey," *International Journal of Computer Vision*, vol. 128, no. 2, pp. 261–318, Feb. 2020, ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-019-01247-4. [Online]. Available: https://link.springer.com/10.1007/s11263-019-01247-4 (visited on 03/31/2025).

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 24, 2017, ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3065386. [Online]. Available: https://dl.acm.org/doi/10.1145/3065386 (visited on 05/14/2025).

[4] Y. Zhu, X. Li, C. Liu, M. Zolfaghari, Y. Xiong, C. Wu, Z. Zhang, J. Tighe, R. Manmatha, and M. Li. "A Comprehensive Study of Deep Video Action Recognition." arXiv: 2012.06567 [cs]. (Dec. 11, 2020), [Online]. Available: http://arxiv.org/abs/2012.06567 (visited on 03/31/2025), pre-published.

[5] J. Johnson, R. Krishna, M. Stark, L.-J. Li, D. A. Shamma, M. S. Bernstein, and L. Fei-Fei, "Image retrieval using scene graphs," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA: IEEE, Jun. 2015, pp. 3668–3678, ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298990. [Online]. Available: http://ieeexplore.ieee.org/document/7298990/ (visited on 03/31/2025).

[6] Y. Cong, W. Liao, H. Ackermann, B. Rosenhahn, and M. Y. Yang, "Spatial-Temporal Transformer for Dynamic Scene Graph Generation," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, QC, Canada: IEEE, Oct. 2021, pp. 16 352–16 362, ISBN: 978-1-6654-2812-5. DOI: 10.1109/ICCV48922.2021.01606. [Online]. Available: https://ieeexplore.ieee.org/document/9710181/ (visited on 04/01/2025).

[7] S. Yan, Y. Xiong, and D. Lin, "Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 27, 2018, ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaa

i.v32i1.12328. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/12328 (visited on 03/31/2025).

[8] J. Ji, R. Krishna, L. Fei-Fei, and J. C. Niebles, "Action Genome: Actions As Compositions of Spatio-Temporal Scene Graphs," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA: IEEE, Jun. 2020, pp. 10 233–10 244, ISBN: 978-1-7281-7168-5. DOI: 10.1109/CVPR42600.2020.01025. [Online]. Available: https://ieeexplore.ieee.org/document/9157115/ (visited on 05/05/2025).

[9] S. Feng, H. Mostafa, M. Nassar, S. Majumdar, and S. Tripathi, "Exploiting Long-Term Dependencies for Generating Dynamic Scene Graphs," in *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, Waikoloa, HI, USA: IEEE, Jan. 2023, pp. 5119–5128, ISBN: 978-1-6654-9346-8. DOI: 10.1109/WACV56688.2023.00510. [Online]. Available: https://ieeexplore.ieee.org/document/10030513/ (visited on 03/31/2025).

[10] T. Silver and R. Chitnis. "PDDLGym: Gym Environments from PDDL Problems." arXiv: 2002.06432 [cs]. (Sep. 15, 2020), [Online]. Available: http://arxiv.org/abs/2002.06432 (visited on 03/31/2025), pre-published.

[11] P. Kochakarn, D. D. Martini, D. Omeiza, and L. Kunze. "Explainable Action Prediction through Self-Supervision on Scene Graphs." arXiv: 2302.03477 [cs]. (Feb. 7, 2023), [Online]. Available: http://arxiv.org/abs/2302.03477 (visited on 04/01/2025), pre-published.

[12] W. L. Hamilton, R. Ying, and J. Leskovec. "Inductive Representation Learning on Large Graphs." arXiv: 1706.02216 [cs]. (Sep. 10, 2018), [Online]. Available: http://arxiv.org/abs/1706.02216 (visited on 05/29/2025), pre-published.

[13] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. "Graph Attention Networks." arXiv: 1710.10903 [stat]. (Feb. 4, 2018), [Online]. Available: http://arxiv.org/abs/1710.10903 (visited on 04/01/2025), pre-published.

[14] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. "Neural Message Passing for Quantum Chemistry." arXiv: 1704.01212 [cs]. (Jun. 12, 2017), [Online]. Available: http://arxiv.org/abs/1704.01212 (visited on 04/01/2025), pre-published.

[15] T. N. Kipf and M. Welling. "Semi-Supervised Classification with Graph Convolutional Networks." arXiv: 1609.02907 [cs]. (Feb. 22, 2017), [Online]. Available: http://arxiv.org/abs/1609.02907 (visited on 04/16/2025), pre-published.

[16] H. Li, G. Zhu, L. Zhang, Y. Jiang, Y. Dang, H. Hou, P. Shen, X. Zhao, S. A. A. Shah, and M. Bennamoun, "Scene Graph Generation: A comprehensive survey," *Neurocomputing*, vol. 566, p. 127 052, Jan. 2024, ISSN: 09252312. DOI: 10.1016/j.neucom.2023.127052. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S092523122301175X (visited on 04/16/2025).

[17] G. Wang, Z. Li, Q. Chen, and Y. Liu, "OED: Towards One-stage End-to-End Dynamic Scene Graph Generation," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA: IEEE, Jun. 16, 2024, pp. 27 938–27 947, ISBN: 979-8-3503-5300-6. DOI: 10.1109/CVPR52733.2024.02639. [Online]. Available: https://ieeexplore.ieee.org/document/10657340/ (visited on 04/16/2025).

[18] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei. "Scene Graph Generation by Iterative Message Passing." arXiv: 1701.02426 [cs]. (Apr. 12, 2017), [Online]. Available: http://arxiv.org/abs/1701.02426 (visited on 05/29/2025), pre-published.

[19] R. Zellers, M. Yatskar, S. Thomson, and Y. Choi, "Neural Motifs: Scene Graph Parsing with Global Context," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT: IEEE, Jun. 2018, pp. 5831–5840, ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00611. [Online]. Available: https://ieeexplore.ieee.org/document/8578709/ (visited on 05/16/2025).

[20] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh, "Graph R-CNN for Scene Graph Generation," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., vol. 11205, Cham: Springer International Publishing, 2018, pp. 690–706, ISBN: 978-3-030-01245-8 978-3-030-01246-5. DOI: 10.1007/978-3-030-01246-5_41. [Online]. Available: https://link.springer.com/10.1007/978-3-030-01246-5_41 (visited on 05/16/2025).

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention Is All You Need." arXiv: 1706.03762 [cs]. (Aug. 2, 2023), [Online]. Available: http://arxiv.org/abs/1706.03762 (visited on 05/29/2025), pre-published.

[22] Y. Teng, L. Wang, Z. Li, and G. Wu, "Target Adaptive Context Aggregation for Video Scene Graph Generation," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, QC, Canada: IEEE, Oct. 2021, pp. 13 668–13 677, ISBN: 978-1-6654-2812-5. DOI: 10.1109/ICCV48922.2021.01343. [Online]. Available: https://ieeexplore.ieee.org/document/9710834/ (visited on 04/16/2025).

[23] T. Pu, T. Chen, H. Wu, Y. Lu, and L. Lin, "Spatial–Temporal Knowledge-Embedded Transformer for Video Scene Graph Generation," *IEEE Transactions on Image Processing*, vol. 33, pp. 556–568, 2024, ISSN: 1941-0042. DOI: 10.1109/TIP.2023.3345652. [Online]. Available: https://ieeexplore.ieee.org/document/10375886/ (visited on 04/16/2025).

[24] K. Simonyan and A. Zisserman. "Two-Stream Convolutional Networks for Action Recognition in Videos." arXiv: 1406.2199 [cs]. (Nov. 12, 2014), [Online]. Available: http://arxiv.org/abs/1406.2199 (visited on 03/31/2025), pre-published.

[25] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, M. Suleyman, and A. Zisserman. "The Kinetics Human Action Video Dataset." arXiv: 1705.06950 [cs]. (May 19, 2017), [Online]. Available: http://arxiv.org/abs/1705.06950 (visited on 04/16/2025), pre-published.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770–778, ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90. [Online]. Available: http://ieeexplore.ieee.org/document/7780459/ (visited on 05/19/2025).

[27] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1–2, pp. 83–97, Mar. 1955, ISSN: 0028-1441, 1931-9193. DOI: 10.1002/nav.3800020109. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/nav.3800020109 (visited on 04/19/2025).

[28] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA: IEEE, Jun. 2019, pp. 658–666, ISBN: 978-1-7281-3293-8. DOI: 10.1109/CVPR.2019.00075. [Online]. Available: https://ieeexplore.ieee.org/document/8953982/ (visited on 04/19/2025).

[29] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, and D. Weld, "PDDL - the planning domain definition language," AIPS-98 Planning Competition Committee, Aug. 1998. [Online]. Available: https://www.cs.cmu.edu/~mmv/planning/readings/98aips-PDDL.pdf (visited on 05/29/2025).

[30] S. Edelkamp and J. Hoffmann, "The Deterministic Part of IPC-4: An Overview," *Journal of Artificial Intelligence Research*, vol. 24, pp. 519–579, Oct. 30, 2005, ISSN: 1076-9757. DOI: 10.1613/jair.1677. arXiv: 1109.5663 [cs]. [Online]. Available: http://arxiv.org/abs/1109.5663 (visited on 05/19/2025).

[31] J. Hoffmann, "Where 'Ignoring Delete Lists' Works: Local Search Topology in Planning Benchmarks," *Journal of Artificial Intelligence Research*, vol. 24, pp. 685–758, Nov. 27, 2005, ISSN: 1076-9757. DOI: 10.1613/jair.1747. [Online]. Available: https://jair.org/index.php/jair/article/view/10430 (visited on 05/19/2025).

[32] M. Helmert, "The Fast Downward Planning System," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, Jul. 12, 2006, ISSN: 1076-9757. DOI: 10.1613/jair.1705. [Online]. Available: https://jair.org/index.php/jair/article/view/10457 (visited on 04/28/2025).

[33] M. Asai. “Photo-Realistic Blocksworld Dataset.” arXiv: 1812.01818 [cs]. (Dec. 5, 2018), [Online]. Available: http://arxiv.org/abs/1812.01818 (visited on 04/28/2025), pre-published.

[34] S. Ioffe and C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” arXiv: 1502.03167 [cs]. (Mar. 2, 2015), [Online]. Available: http://arxiv.org/abs/1502.03167 (visited on 05/29/2025), pre-published.

[35] I. Loshchilov and F. Hutter. “Decoupled Weight Decay Regularization.” arXiv: 1711.05101 [cs]. (Jan. 4, 2019), [Online]. Available: http://arxiv.org/abs/1711.05101 (visited on 05/05/2025), pre-published.

[36] L. Prechelt, “Early stopping — but when?” In *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science, G. B. Orr and K.-R. Müller, Eds., vol. 1524, Berlin, Heidelberg: Springer, 1998, pp. 55–69, ISBN: 978-3-540-65311-8. DOI: 10.1007/3-540-49430-8_3. [Online]. Available: https://link.springer.com/content/pdf/10.1007/3-540-49430-8_3.pdf (visited on 05/29/2025).

[37] S. Nazmi, X. Yan, A. Homaifar, and E. Doucette. “Evolving Multi-label Classification Rules by Exploiting High-order Label Correlation.” arXiv: 2007.11609 [cs]. (Jul. 22, 2020), [Online]. Available: http://arxiv.org/abs/2007.11609 (visited on 05/20/2025), pre-published.